

# Graphics Gems, by Book

Listed by book, in order.

- [Graphics Gems](#)
- [Graphics Gems II](#)
- [Graphics Gems III](#)
- [Graphics Gems IV](#)
- [Graphics Gems V](#)

[return to main page](#)

---

## Graphics Gems

Glassner, Andrew, **Useful 2D Geometry**, p. 3-11.

Glassner, Andrew, **Useful Trigonometry**, p. 13-17.

Paeth, Alan W., **Trigonometric Functions at Select Points**, p. 18-19.

Goldman, Ronald, **Triangles**, p. 20-23.

Turk, Greg, **Generating Random Points in Triangles**, p. 24-28, code: p. 649-650, [TriPoints.c](#).

Shapira, Andrew, **Fast Line-Edge Intersections on a Uniform Grid**, p. 29-36, code: p. 651-653, [LineEdge.c](#).

Thompson, Kelvin, **Area of Intersection: Circle and a Half-Plane**, p. 38-39.

Thompson, Kelvin, **Area of Intersection: Circle and a Thick Line**, p. 40-42.

Thompson, Kelvin, **Area of Intersection: Two Circles**, p. 43-46.

Thompson, Kelvin, **Vertical Distance from a Point to a Line**, p. 47-48.

Paeth, Alan W., **A Fast 2D Point-on-line Test**, p. 49-50, code: p. 654-655, [PntOnLine.c](#).

Shaffer, Clifford A., **Fast Circle-Rectangle Intersection Checking**, p. 51-53, code: p. 656, [CircleRect.c](#).

Paeth, Alan W., **Circles of Integral Radius on Integer Lattices**, p. 57-60.

Heckbert, Paul S., **Nice Numbers for Graph Labels**, p. 61-63, code: p. 657-659, [Label.c](#).

Cychosz, Joseph M., **Efficient Generation of Sampling Jitter Using Look-up Tables**, p. 64-74, code: p. 660-661, [FastJitter.c](#).

Morrison, Jack C., **Fast Anti-Aliasing Polygon Scan Conversion**, p. 76-83, code: p. 662-666, [AAPolyScan.c](#).

Heckbert, Paul S., **Generic Convex Polygon Scan Conversion and Clipping**, p. 84-86, code: p. 667-680, [PolyScan/](#).

Heckbert, Paul S., **Concave Polygon Scan Conversion**, p. 87-91, code: p. 681-684, [ConcaveScan.c](#).

Wallis, Bob, **Fast Scan Conversion of Arbitrary Polygons**, p. 92-97.

Heckbert, Paul S., **Digital Line Drawing**, p. 99-100, code: p. 685, [DigitalLine.c](#).

Wyvill, Brian, **Symmetric Double Step Line Algorithm**, p. 101-104, code: p. 686-689, [DoubleLine.c](#).

Thompson, Kelvin, **Rendering Anti-Aliased Lines**, p. 105-106, code: p. 690-693, [AALines/](#).

Ritter, Jack, **An Algorithm for Filling in 2D Wide Line Bevel Joints**, p. 107-113.

- Wallis, Bob, **Rendering Fat Lines on a Raster Grid**, p. 114-120.
- Spoelder, Hans J.W., and Ullings, Fons H., **Two-Dimensional Clipping: A Vector-Based Approach**, p. 121-128, code: p. 694-710, [2DClip/](#).
- Lee, Greg, Penk, Mike, and Wallis, Bob, **Periodic Tilings of the Plane on a Raster Grid**, p. 129-139.
- Pavicic, Mark J., **Anti-Aliasing Filters that Minimize "Bumpy" Sampling**, p. 144-146.
- Turkowski, Ken, **Filters for Common Resampling Tasks**, p. 147-165.
- Olsen, John, **Smoothing Enlarged Monochrome Images**, p. 166-170.
- Paeth, Alan W., **Median Finding on a 3-by-3 Grid**, p. 171-175, code: p. 711-712, [Median.c](#).
- Hawley, Stephen, **Ordered Dithering**, p. 176-178, code: p. 713-714, [OrderDither.c](#).
- Paeth, Alan W., **A Fast Algorithm for General Raster Rotation**, p. 179-195.
- Schumacher, Dale A., **Useful 1-to-1 Pixel Transforms**, p. 196-209.
- Thompson, Kelvin, **Alpha Blending**, p. 210-211.
- Glassner, Andrew, **Frame Buffers and Color Maps**, p. 215-218.
- Paeth, Alan W., **Reading a Write-Only Write Mask**, p. 219-220.
- Morton, Mike, **A Digital "Dissolve" Effect**, p. 221-232, code: p. 715-717, [Dissolve.c](#).
- Paeth, Alan W., **Mapping RGB Triples Onto Four Bits**, p. 233-245, code: p. 718, [RGBTo4Bits.c](#).
- Heckbert, Paul S., **What Are the Coordinates of a Pixel?**, p. 246-248.
- Paeth, Alan W., **Proper Treatment of Pixels as Integers**, p. 249-256, code: p. 719, [PixelInteger.c](#).
- Glassner, Andrew, **Normal Coding**, p. 257-264.
- Heckbert, Paul S., **Recording Animation in Binary Order for Progressive Temporal Refinement**, p. 265-269, code: p. 720, [BinRec.c](#).
- Schumacher, Dale A., **1-to-1 Pixel Transforms Optimized Through Color-Map Manipulation**, p. 270-274.
- Heckbert, Paul S., **A Seed Fill Algorithm**, p. 275-277, code: p. 721-722, [SeedFill.c](#).
- Fishkin, Ken, **Filling a Region in a Frame Buffer**, p. 278-284.
- Wallace, Bill, **Precalculating Addresses for Fast Fills, Circles, and Lines**, p. 285-286.
- Gervautz, Michael, and Purgathofer, Werner, **A Simple Method for Color Quantization: Octree Quantization**, p. 287-293.
- Glassner, Andrew, **Useful 3D Geometry**, p. 297-300.
- Ritter, Jack, **An Efficient Bounding Sphere**, p. 301-303, code: p. 723-725, [BoundSphere.c](#).
- Goldman, Ronald, **Intersection of Two Lines in Three-Space**, p. 304.
- Goldman, Ronald, **Intersection of Three Planes**, p. 305.
- Paeth, Alan W., **Digital Cartography for Computer Graphics**, p. 307-320.
- Bame, Paul D., **Albers Equal-Area Conic Map Projection**, p. 321-325, code: p. 726-729, [Albers.c](#).
- Montani, Claudio, and Scopigno, Roberto, **Spheres-to-Voxels Conversion**, p. 327-334.
- Arvo, James, **A Simple Method for Box-Sphere Intersection Testing**, p. 335-339, code: p. 730-732, [BoxSphere.c](#).
- Wyvill, Brian, **3D Grid Hashing Function**, p. 343-345, code: p. 733-734, [Hash3D.c](#).
- Hultquist, Jeff, **Backface Culling**, p. 346-347.
- Lee, Mark E., **Fast Dot Products for Shading**, p. 348-360.
- Thompson, Kelvin, **Scanline Depth Gradient of a Z-Buffered Triangle**, p. 361-363.
- Glassner, Andrew, **Simulating Fog and Haze**, p. 364-365.
- Glassner, Andrew, **Interpretation of Texture Map Indices**, p. 366-375.
- Glassner, Andrew, **Multidimensional Sum Tables**, p. 376-381.

- Ritter, Jack, **A Simple Ray Rejection Test**, p. 385-386.
- Hultquist, Jeff, **Intersection of a Ray with a Sphere**, p. 388-389.
- Badouel, Didier, **An Efficient Ray-Polygon Intersection**, p. 390-393, code: p. 735, [RayPolygon.c](#).
- Woo, Andrew, **Fast Ray-Polygon Intersection**, p. 394.
- Woo, Andrew, **Fast Ray-Box Intersection**, p. 395-396, code: p. 736-737, [RayBox.c](#).
- Pearce, Andrew, **Shadow Attenuation for Ray Tracing Transparent Objects**, p. 397-399.
- Schwarze, Jochen, **Cubic and Quartic Roots**, p. 404-407, code: p. 738-786, [Roots3And4.c](#).
- Schneider, Philip J., **A Bézier Curve-Based Root-Finder**, p. 408-415, code: p. 787, [NearestPoint.c](#).
- Hook, D.G., and McAree, P.R., **Using Sturm Sequences To Bracket Real Roots of Polynomial Equations**, p. 416-423, code: p. 743-755, [Sturm/](#).
- Lalonde, Paul, and Dawson, Robert, **A High-Speed, Low Precision Square Root**, p. 424-426, code: p. 756-757, [SquareRoot.c](#).
- Paeth, Alan W., **A Fast Approximation To the Hypotenuse**, p. 427-431, code: p. 758, [HypotApprox.c](#).
- Ritter, Jack, **A Fast Approximation To 3D Euclidian Distance**, p. 432-433.
- Thompson, Kelvin, **Full-Precision Constants**, p. 434.
- Thompson, Kelvin, **Converting Between Bits and Digits**, p. 435.
- Wyvill, Brian, **Storage-free Swapping**, p. 436-437.
- Glassner, Andrew, **Generating Random Integers**, p. 438-439.
- Ritter, Jack, **Fast 2D-3D Rotation**, p. 440-441.
- Shoemake, Ken, **Bit Patterns for Encoding Angles**, p. 442.
- Shaffer, Clifford A., **Bit Interleaving for Quad- or Octrees**, p. 443-447, code: p. 759-762, [Interleave.c](#).
- Fishkin, Ken, **A Fast HSL-to-RGB Transform**, p. 448-449, code: p. 763-764, [HSLtoRGB.c](#).
- Thompson, Kelvin, **Matrix Identities**, p. 453-454.
- Thompson, Kelvin, **Transforming Axes**, p. 456-459.
- Thompson, Kelvin, **Fast Matrix Multiplication**, p. 460-461.
- Hultquist, Jeff, **A Virtual Trackball**, p. 462-463.
- Raible, Eric, **Matrix Orthogonalization**, p. 464, code: p. 765, [MatrixOrtho.c](#).
- Pique, Michael E., **Rotation Tools**, p. 465-469.
- Carling, Richard, **Matrix Inversion**, p. 470-471, code: p. 766-769, [MatrixInvert.c](#).
- Goldman, Ronald, **Matrices and Transformations**, p. 472-475.
- Cychosz, Joseph M., **Efficient Post-Concatenation of Transformation Matrices**, p. 476-481, code: p. 770-772, [MatrixPost.c](#).
- Greene, Ned, **Transformation Identities**, p. 485-493.
- Turkowski, Ken, **Fixed-Point Trigonometry with CORDIC Iterations**, p. 494-497, code: p. 773-774, [FixedTrig.c](#).
- Maillot, Patrick-Gilles, **Using Quaternions for Coding 3D Transformations**, p. 498-515, code: p. 775-777, [Quaternions.c](#).
- Cunningham, Steve, **3D Viewing and Rotation Using Orthonormal Bases**, p. 516-521, code: p. 778-779, [ViewTrans.c](#).
- Turkowski, Ken, **The Use of Coordinate Frames in Computer Graphics**, p. 522-532.
- Wallis, Bob, **Forms, Vectors, and Transforms**, p. 533-538, code: p. 780-784, [Forms.c](#).
- Turkowski, Ken, **Properties of Surface-Normal Transformations**, p. 539-547.
- Arvo, James, **Transforming Axis-Aligned Bounding Boxes**, p. 548-550, code: p. 785-786, [TransBox.c](#).

- Hall, Mark, **Defining Surfaces From Sampled Data**, p. 552-557.
- Hall, Mark, **Defining Surfaces From Contour Data**, p. 558-561.
- Glassner, Andrew, **Computing Surface Normals for 3D Models**, p. 562-566.
- Bloomenthal, Jules, **Calculation of Reference Frames Along a Space Curve**, p. 567-571.
- Glassner, Andrew, **Planar Cubic Curves**, p. 575-578.
- Rasala, Richard, **Explicit Cubic Spline Interpolation Formulas**, p. 579-584.
- Gomez, Julian, **Fast Spline Drawing**, p. 585-586.
- Goldman, Ronald, **Some Properties of Bézier Curves**, p. 587-593.
- Wallis, Bob, **Tutorial on Forward Differencing**, p. 594-603.
- Goldman, Ronald, **Integration of Bernstein Basis Functions**, p. 604-606.
- Schneider, Philip J., **Solving the Nearest-Point-on-Curve Problem**, p. 607-611, code: p. 787-796, [Nearest.c](#).
- Schneider, Philip J., **An Algorithm for Automatically Fitting Digitized Curves**, p. 612-626, code: p. 797-807, [FitCurves.c](#).
- Glassner, Andrew, **Graphics Gems Header File**, p. 629-632, code: p. 629-632, [GraphicsGems.h](#).
- Glassner, Andrew, **2D and 3D Vector C Library**, p. 633-642, code: p. 633-642, [GGVecLib.c](#) [GraphicsGems.h](#).
- Hultquist, Jeff, **Memory Allocation in C**, p. 643, code: p. 643.
- Raible, Eric, **Two Useful C Macros**, p. 644, code: p. 644.
- Thompson, Kelvin, **How to Build Circular Structures in C**, p. 645, code: p. 645.
- Thompson, Kelvin, **How to Use C Register Variables to Point to 2D Arrays**, p. 646, code: p. 646.

## Graphics Gems II

- Rokne, Jon, **The Area of a Simple Polygon**, p. 5-6.
- Prasad, Mukesh, **Intersection of Line Segments**, p. 7-9, code: p. 473-476, [xlines.c](#).
- Morrison, Jack C., **Distance From a Point To a Line**, p. 10-13.
- Rokne, Jon, **An Easy Bounding Circle**, p. 14-16.
- Rokne, Jon, **The Smallest Circle Containing the Intersection of Two Circles**, p. 17-18.
- Rokne, Jon, **Appolonius's 10th Problem**, p. 19-24.
- Musgrave, F. Kenton, **A Peano Curve Generation Algorithm**, p. 25, code: p. 477-484, [Peano/](#).
- Voorhies, Douglas, **Space-Filling Curves and a Measure of Coherence**, p. 26-30, code: p. 485-486, [Hilbert.c](#).
- Steinhart, Jonathan E., **Scanline Coherent Shape Algebra**, p. 31-45, code: p. 487-501.
- Schumacher, Dale A., **Image Smoothing and Sharpening by Discrete Convolution**, p. 50-56.
- Schumacher, Dale A., **A Comparison of Digital Halftoning Techniques**, p. 57-71, code: p. 502-508.
- Thomas, Spencer W., and Bogart, Rod G., **Color Dithering**, p. 72-77, code: p. 509-513, [dither/](#).
- Schumacher, Dale A., **Fast Anamorphic Image Scaling**, p. 78-79.
- Ward, Greg, **Real Pixels**, p. 80-83, code: [RealPixels/](#).
- Yap, Sue-Ken, **A Fast 90-Degree Bitmap Rotator**, p. 84-85, code: p. 514-515, [rotate8x8.c](#).
- Holt, Jeff, **Rotation of Run-Length Encoded Image Data**, p. 86-88, code: p. 516-524.
- Glassner, Andrew, **Adaptive Run-Length Encoding**, p. 89-92.
- Paeth, Alan W., **Image File Compression Made Easy**, p. 93-100.
- Max, Nelson L., **An Optimal Filter for Image Reconstruction**, p. 101-104.



Schlag, John, **Noise Thresholding in Edge Images**, p. 105-106.

Bieri, Hanspeter, and Kohler, Andreas, **Computing the Area, the Circumference, and the Genus of a Binary Digital Image**, p. 107-111, code: p. 525-527.

Thomas, Spencer W., **Efficient Inverse Color Map Computation**, p. 116-125, code: p. 528-535, [inv\\_cmap/](#).

Wu, Xiaolin, **Efficient Statistical Computations for Optimal Color Quantization**, p. 126-133, code: [quantizer.c](#).

Musgrave, F. Kenton, **A Random Color Map Animation Algorithm**, p. 134-137, code: p. 536-541, [ran\\_ramp.c](#).

Hall, Jim, and Lindgren, Terence, **A Fast Approach To PHIGS PLUS Pseudo Color**, p. 138-142.

Paeth, Alan W., **Mapping RGB Triples Onto 16 Distinct Values**, p. 143-146.

Martindale, David, and Paeth, Alan W., **Television Color Encoding and "Hot" Broadcast Colors**, p. 147-158, code: p. 542-549, [hot.c](#).

Meyer, Gary W., **An Inexpensive Method of Setting the Monitor White Point**, p. 159-162.

Musgrave, F. Kenton, **Some Tips for Making Color Hardcopy**, p. 163-165.

Goldman, Ronald, **Area of Planar Polygons and Volume of Polyhedra**, p. 170-171.

Shaffer, Clifford A., **Getting Around on a Sphere**, p. 172-173.

Paeth, Alan W., **Exact Dihedral Metrics for Common Polyhedra**, p. 174-178.

Glassner, Andrew, **A Simple Viewing Geometry**, p. 179-180.

Bogart, Rod G., **View Correlation**, p. 181-190, code: p. 550-562, [viewcorr/](#).

Glassner, Andrew, **Maintaining Winged-Edge Models**, p. 191-201.

Montani, Claudio, and Scopigno, Roberto, **Quadtree/Octree-to-Boundary Conversion**, p. 202-218.

Maillot, Patrick-Gilles, **Three-Dimensional Homogeneous Clipping of Triangle Strips**, p. 219-231, code: p. 563-570.

Thalmann, Nadia Magnenat, Thalmann, Daniel, and Minh, Hong Tong, **InterPhong Shading**, p. 232-241, code: p. 571-574, [InterPhong.c](#).

Haines, Eric, **Fast Ray-Convex Polyhedron Intersection**, p. 247-250, code: p. 575-576, [RayCPhdron.c](#).

Cychosz, Joseph M., **Intersecting a Ray with An Elliptical Torus**, p. 251-256, code: p. 577-580, [intersect/inttor.c](#).

Voorhies, Douglas, and Kirk, David, **Ray-Triangle Intersection Using Binary Recursive Subdivision**, p. 257-263.

Kirk, David, and Arvo, James, **Improved Ray Tagging for Voxel-Based Ray Tracing**, p. 264-266.

Haines, Eric, **Efficiency Improvements for Hierarchy Traversal in Ray Tracing**, p. 267-272.

Pearce, Andrew, **A Recursive Shadow Voxel Cache for Ray Tracing**, p. 273-274, code: p. 581-582, [VoxelCache.c](#).

Pearce, Andrew, **Avoiding Incorrect Shadow Intersections for Ray Tracing**, p. 275-276.

Lee, Mark E., and Uselton, Samuel P., **A Body Color Model: Absorption Through Translucent Media**, p. 277-282.

Lee, Mark E., and Uselton, Samuel P., **More Shadow Attenuation for Ray Tracing Transparent or Translucent Objects**, p. 283-289.

Chen, Shenchang Eric, **Implementing Progressive Radiosity with User-Provided Polygon Display Routines**, p. 295-298, code: p. 583-597, [radiosity/](#).

Beran-Koehn, Jeffrey C., and Pavicic, Mark J., **A Cubic Tetrahedral Adaptation of the Hemi-Cube Algorithm**, p. 299-302.

- Tampieri, Filippo, **Fast Vertex Radiosity Update**, p. 303-305, code: p. 598, [FastUpdate.c](#).
- Shirley, Peter, **Radiosity via Ray Tracing**, p. 306-310.
- Sillion, François, **Detection of Shadow Boundaries for Adaptive Meshing in Radiosity**, p. 311-315.
- Thomas, Spencer W., **Decomposing a Matrix Into Simple Transformations**, p. 320-323, code: p. 599-602, [unmatrix.c](#).
- Goldman, Ronald, **Recovering the Data From the Transformation Matrix**, p. 324-331.
- Goldman, Ronald, **Transformations as Exponentials**, p. 332-337.
- Goldman, Ronald, **More Matrices and Transforms: Shear and Pseudo-Perspective**, p. 338-341.
- Wu, Kevin, **Fast Matrix Inversion**, p. 342-350, code: p. 603-605, [inverse.c](#).
- Shoemake, Ken, **Quaternions and 4x4 Matrices**, p. 351-354.
- Arvo, James, **Random Rotation Matrices**, p. 355-356, code: p. 606-607, [rotate.c](#).
- Arvo, James, **Classifying Small Sparse Matrices**, p. 357-361, code: p. 608-609, [sparse.c](#).
- Shoemake, Ken, **Bit Picking**, p. 366-367.
- Shoemake, Ken, **Faster Fourier Transform**, p. 368-370.
- Paeth, Alan W., and Schilling, David, **Of Integers, Fields, and Bit Counting**, p. 371-376, code: p. 610-611, [BitCounting/](#).
- Schlag, John, **Using Geometric Constructions to Interpolate Orientation with Quaternions**, p. 377-380.
- Paeth, Alan W., **A Half-Angle Identity for Digital Computation: The Joys of the Halved Tangent**, p. 381-386.
- Musial, Christopher J., **An Integer Square Root Algorithm**, p. 387-388, code: p. 612.
- Capelli, Ron, **Fast Approximation To the Arctangent**, p. 389-391.
- Ritter, Jack, **Fast Sign of Cross Product Calculation**, p. 392-393, code: p. 613-614.
- Shoemake, Ken, **Interval Sampling**, p. 394-395.
- Ward, Greg, **A Recursive Implementation of the Perlin Noise Function**, p. 396-401, code: p. 615-616, [noise3.c](#).
- Moore, Doug, and Warren, Joseph, **Least-Squares Approximations To Bézier Curves and Surfaces**, p. 406-411.
- Shoemake, Ken, **Beyond Bézier Curves**, p. 412-416.
- Schlag, John, **A Simple Formulation for Curve Interpolation with Variable Control Point Approximation**, p. 417-419.
- Lindgren, Terence, **Symmetric Evaluation of Polynomials**, p. 420-423.
- Seidel, Hans-Peter, **Menelaus's Theorem**, p. 424-427.
- Seidel, Hans-Peter, **Geometrically Continuous Cubic Bézier Curves**, p. 428-434.
- Musial, Christopher J., **A Good Straight-Line Approximation of a Circular Arc**, p. 435-439, code: p. 617.
- Paeth, Alan W., **Great Circle Plotting**, p. 440-445.
- Wu, Xiaolin, **Fast Anti-Aliased Circle Generation**, p. 446-450.
- Glassner, Andrew, **Graphics Gems Header File**, p. 455-457, code: p. 629-632, [GraphicsGems.h](#).
- Glassner, Andrew, and Bogart, Rod G., **2D and 3D Vector C Library**, p. 458-466, code: p. 633-642, [GGVecLib.c](#) [GraphicsGems.h](#).
- Hollasch, Steve, **Useful C Macros for Vector Operations**, p. 467-469, code: p. 467-469, [vector.h](#).

# Graphics Gems III

- Möller, Tomas, **Fast Bitmap Stretching**, p. 4-7, code: p. 411-413, [fastBitmap.c](#).
- Schumacher, Dale A., **General Filtered Image Rescaling**, p. 8-16, code: p. 414-424, [filter.c](#) [filter\\_rcg.c](#).
- Schumacher, Dale A., **Optimization of Bitmap Scaling Operations**, p. 17-19, code: p. 425-428, [bitmap.c](#).
- Bragg, Dennis, **A Simple Color Reduction Filter**, p. 20-22, code: p. 429-431, [rgbvary.c](#) [rgbvaryW.c](#).
- Moore, Doug, and Warren, Joseph, **Compact Isocontours From Sampled Data**, p. 23-28.
- Feldman, Tim, **Generating Iso-value Contours From a Pixmap**, p. 29-33, code: p. 432-440, [contour.c](#).
- Salesin, David, and Barzel, Ronen, **Compositing Black-and-White Bitmaps**, p. 34-35.
- Scofield, Cary, **2-1/2-d Depth-of-Field Simulation for Computer Animation**, p. 36-38.
- Furman, Eric, **A Fast Boundary Generator for Composited Regions**, p. 39-43, code: p. 441-445, [scallops8.c](#).
- Hill, Steve, **IEEE Fast Square Root**, p. 48, code: p. 446-447, [sqrt.c](#).
- Hill, Steve, **A Simple Fast Memory Allocator**, p. 49-50, code: p. 448-451, [alloc/](#).
- Hanson, Andrew J., **The Rolling Ball**, p. 51-60, code: p. 452-453, [3d.c](#) [defs.h](#).
- Rokne, Jon, **Interval Arithmetic**, p. 61-66, code: p. 454-457, [interval.C](#).
- Paeth, Alan W., **Fast Generation of Cyclic Sequences**, p. 67-76, code: p. 458-459, [cyclic.c](#).
- Paeth, Alan W., **A Generic Pixel Selection Mechanism**, p. 77-79.
- Shirley, Peter, **Nonuniform Random Point Sets**, p. 80-83.
- Goldman, Ronald, **Cross Product in Four Dimensions and Beyond**, p. 84-88.
- Badouel, Didier, and Wuthrich, Charles A., **Face-Connected Line Segment Generation in an n-Dimensional Space**, p. 89-91, code: p. 460, [ndline.c](#).
- Morrison, Jack C., **Quaternion Interpolation with Extra Spins**, p. 96-97, code: p. 461-462, [quatspin.c](#).
- Goldman, Ronald, **Decomposing Projective Transformations**, p. 98-107.
- Goldman, Ronald, **Decomposing Linear and Affine Transformations**, p. 108-116.
- Arvo, James, **Fast Random Rotation Matrices**, p. 117-120, code: p. 463-464, [rand\\_rotation.c](#).
- Dana, Paul, **Issues and Techniques for Keyframing Transformations**, p. 121-123.
- Shoemake, Ken, **Uniform Random Rotations**, p. 124-132, code: p. 465-467, [urot.c](#).
- Elber, Gershon, **Interpolation Using Bézier Curves**, p. 133-136, code: p. 468-471, [bzrintp.c](#).
- Barr, A.H., **Physically Based Superquadrics**, p. 137-159, code: p. 472-477, [sqfinal.c](#).
- Van Aken, Jerry, and Simar, Ray, **A Parametric Elliptical Arc Algorithm**, p. 164-172, code: p. 478-479, [parelarc.c](#).
- Rosati, Claudio, **A Simple Connection Algorithm for 2-D Drawing**, p. 173-181, code: p. 480-486, [con2d.c](#).
- Srinivasan, Raman V., **A Fast Circle Clipping Algorithm**, p. 182-187, code: p. 487-490, [circlexc.c](#).
- Shaffer, Clifford A., and Feustel, Charles D., **Exact Computation of 2-D Intersections**, p. 188-192, code: p. 491-495, [Polyintr.c](#).
- Miller, Robert D., **Joining Two Lines with a Circular Arc Fillet**, p. 193-198, code: p. 496-499, [fillet.c](#).
- Antonio, Franklin, **Faster Line Segment Intersection**, p. 199-202, code: p. 500-501, [insectc.c](#).
- Sevici, Constantin A., **Solving the Problem of Apollonius and Other Related Problems**, p. 203-209.
- López-López, Fernando J., **Triangles Revisited**, p. 215-218.

- Chin, Norman, **Partitioning a 3-D Convex Polygon with an Arbitrary Plane**, p. 219-222, code: p. 502-510, [partition3d/](#).
- Georgiades, Príamos, **Signed Distance From Point To Plane**, p. 223-224, code: p. 511, [pt2plane.c](#).
- Salesin, David, and Tampieri, Filippo, **Grouping Nearly Coplanar Polygons Into Coplanar Sets**, p. 225-230, code: p. 512-516, [planeSets.c](#).
- Tampieri, Filippo, **Newell's Method for the Plane Equation of a Polygon**, p. 231-232, code: p. 517-518, [newell.c](#).
- Georgiades, Príamos, **Plane-to-Plane Intersection**, p. 233-235, code: p. 519-520, [pl2plane.c](#).
- Voorhies, Douglas, **Triangle-Cube Intersection**, p. 236-239, code: p. 521-526, [triangleCube.c](#).
- Wanger, Len, and Fusco, Mike, **Fast N-Dimensional Extent Overlap Testing**, p. 240-243, code: p. 527-533, [extttest/](#).
- Moore, Doug, **Subdividing Simplices**, p. 244-249, code: p. 534-535, [simplex/](#).
- Moore, Doug, **Understanding Simplicoids**, p. 250-255.
- Lischinski, Dani, **Converting Bézier Triangles Into Rectangular Patches**, p. 256-261, code: p. 536-537, [bezierTri.C](#).
- Lindgren, Terence, Sanchez, Juan, and Hall, Jim, **Curve Tessellation Criteria Thru Sampling**, p. 262-265.
- Sung, Kelvin, and Shirley, Peter, **Ray Tracing with the BSP Tree**, p. 271-274, code: p. 538-546, [bsp.c](#).
- Cychosz, Joseph M., and Waggenspack, Warren N., Jr., **Intersecting a Ray with a Quadric Surface**, p. 275-283, code: p. 547-550, [intqdr.c](#) [intell.c](#).
- Cychosz, Joseph M., **Use of Residency Masks and Object Space Partitioning to Eliminate Ray-Object Intersection Calculations**, p. 284-287.
- Musgrave, F. Kenton, **A Panoramic Virtual Screen for Ray Tracing**, p. 288-294, code: p. 551-554, [panorama.c](#).
- Trumbore, Ben, **Rectangular Bounding Volumes for Popular Primitives**, p. 295-300, code: p. 555-561, [bounding\\_volumes.c](#).
- Wu, Xiaolin, **A Linear-Time Simple Bounding Volume Algorithm**, p. 301-306.
- Wang, Changyaw, **Physically Correct Direct Lighting for Distribution Ray Tracing**, p. 307-313, code: p. 562-568, [luminaire/](#).
- Bian, Bumg, **Hemispherical Projection of a Triangle**, p. 314-317, code: p. 569-574, [hemis.c](#).
- Max, Nelson L., and Allison, Michael J., **Linear Radiosity Approximation Using Vertex-to-Vertex Form Factors**, p. 318-323.
- Beran-Koehn, Jeffrey C., and Pavicic, Mark J., **Delta Form-Factor Calculation for the Cubic Tetrahedral Algorithm**, p. 324-328, code: p. 575-576, [forfac.c](#).
- Tampieri, Filippo, **Accurate Form-Factor Computation**, p. 329-333, code: p. 577-581, [accForm.c](#).
- Woo, Andrew, **The Shadow Depth Map Revisited**, p. 338-342, code: p. 582, [zdepth.c](#).
- Cheng, Russell C.H., **Fast Linear Color Rendering**, p. 343-348, code: p. 583-585, [fastLinear.c](#).
- Cheng, Russell C.H., **Edge and Bit-Mask Calculations for Anti-Aliasing**, p. 349-354, code: p. 586-593, [edgeCalc.c](#).
- Grace, Thom, **Fast Span Conversion: Unrolling Short Loops**, p. 355-357, code: p. 594-596, [fastSpan.c](#).
- Hollasch, Steve, **Progressive Image Refinement via Gridded Sampling**, p. 358-361, code: p. 597-598, [PIR.c](#).



- Fleischer, Kurt, and Salesin, David, **Accurate Polygon Scan Conversion Using Half-Open Intervals**, p. 362-365, code: p. 599-605, [accurate\\_scan/](#).
- Glassner, Andrew, **Darklights**, p. 366-368.
- Glassner, Andrew, **Anti-Aliasing in Triangular Pixels**, p. 369-373.
- Snyder, John, Barzel, Ronen, and Gabriel, Steve, **Motion Blur on Graphics Workstations**, p. 374-382, code: p. 606-609, [motblur.c](#).
- Arvo, James, and Scofield, Cary, **The Shader Cache: A Rendering Pipeline Accelerator**, p. 383-389.
- Glassner, Andrew, **Graphics Gems Header File**, p. 455-457, code: p. 393-395, [GraphicsGems.h](#).
- Glassner, Andrew, **2-D and 3-D Vector C Library -- Corrected and Indexed**, p. 396-404, code: p. 396-404, [GraphicsGems.c](#).
- Hollasch, Steve, **Useful C Macros for Vector Operations**, p. 405-407, code: p. 405-407, [vector.h](#).

## Graphics Gems IV

- Bashein, Gerard, and Detmer, Paul R., **Centroid of a Polygon**, p. 3-6, code: p. 5, [centroid.c](#).
- Schorn, Peter, and Fisher, Frederick, **Testing the Convexity of a Polygon**, p. 7-15, code: p. 11-15, [convex\\_test/](#).
- Weiler, Kevin, **An Incremental Angle Point in Polygon Test**, p. 16-23, code: p. 17-22, [ptpoly\\_weiler/](#).
- Haines, Eric, **Point in Polygon Strategies**, p. 24-46, code: p. 34-45, [ptpoly\\_haines/](#).
- Lischinski, Dani, **Incremental Delaunay Triangulation**, p. 47-59, code: p. 51-58, [delaunay/](#).
- Glassner, Andrew, **Building Vertex Normals from an Unstructured Polygon List**, p. 60-73, code: p. 64-73, [vert\\_norm/](#).
- Greene, Ned, **Detecting Intersection of a Rectangular Solid and a Convex Polyhedron**, p. 74-82.
- Rabbitz, Rich, **Fast Collision Detection of Moving Convex Polyhedra**, p. 83-109, code: p. 91-108, [collide.c](#).
- Hart, John C., **Distance to an Ellipsoid**, p. 113-119.
- Ohashi, Yoshikazu, **Fast Linear Approximations of Euclidean Distance in Higher Dimensions**, p. 121-124, code: [dist\\_fast.c](#).
- Donovan, Walt, and Van Hook, Tim, **Direct Outcode Calculation for Faster Clip Testing**, p. 125-131, code: p. 127-131, [outcode/](#).
- Miller, Robert D., **Computing the Area of a Spherical Polygon**, p. 132-137, code: p. 135-136, [sph\\_poly.c](#).
- Hill, F. S., Jr., **The Pleasures of 'Perp Dot' Products**, p. 138-148.
- Hanson, Andrew J., **Geometry for N-Dimensional Graphics**, p. 149-170.
- Shoemake, Ken, **Arcball Rotation Control**, p. 175-192, code: p. 178-191, [arcball/](#).
- Cromwell, Robert L., **Efficient Eigenvalues for Visualization**, p. 193-198.
- Wu, Kevin, **Fast Inversion of Length- and Angle-Preserving Matrices**, p. 199-206, code: p. 204-206, [inv\\_fast.c](#).
- Shoemake, Ken, **Polar Matrix Decomposition**, p. 207-221, code: p. 211-220, [polar\\_decomp/](#).
- Shoemake, Ken, **Euler Angle Conversion**, p. 222-229, code: p. 225-228, [euler\\_angle/](#).
- Shoemake, Ken, **Fiber Bundle Twist Reduction**, p. 230-236.
- Eilers, Paul H. C., **Smoothing and Interpolation with Finite Differences**, p. 241-250, code: p. 246-249,

[data\\_smooth/](#).

Barry, Phillip, and Goldman, Ronald, **Knot Insertion using Forward Differences**, p. 251-255.

Bajaj, Chandrajit, and Xu, Guoliang, **Converting a Rational Curve to a Standard Rational Bernstein-Bézier Representation**, p. 256-260.

Klassen, R. Victor, **Intersecting Parametric Cubic Curves by Midpoint Subdivision**, p. 261-277, code: p. 266-276, [curve\\_isect/](#).

Lischinski, Dani, **Converting Rectangular Patches into Bézier Triangles**, p. 278-285, code: p. 281-285, [patch\\_conv.C](#).

Peterson, John W., **Tessellation of NURB Surfaces**, p. 286-320, code: p. 294-319, [nurb\\_polyg/](#).

Shene, Ching-Kuang, **Equations of Cylinders and Cones**, p. 321-323.

Bloomenthal, Jules, **An Implicit Surface Polygonizer**, p. 324-349, code: p. 334-349, [implicit.c](#).

Shene, Ching-Kuang, **Computing the Intersection of a Line and a Cylinder**, p. 353-355.

Cychosz, Joseph M., and Waggenspack, Warren N., Jr., **Intersecting a Ray with a Cylinder**, p. 356-365, code: p. 361-364, [ray\\_cyl.c](#).

Cohen, Daniel, **Voxel Traversal along a 3D Line**, p. 366-369, code: p. 368, [vox\\_traverse.c](#).

Chiu, Kenneth, Shirley, Peter, and Wang, Changyaw, **Multi-Jittered Sampling**, p. 370-374, code: p. 373-374, [multi\\_jitter/](#).

Heckbert, Paul S., **A Minimal Ray Tracer**, p. 375-381, code: p. 378-380, [minray/](#).

Schlick, Christophe, **A Fast Alternative to Phong's Specular Model**, p. 385-387.

Fisher, Frederick, and Woo, Andrew, **R.E versus N.H Specular Highlights**, p. 388-400.

Schlick, Christophe, **Fast Alternatives to Perlin's Bias and Gain Functions**, p. 401-403.

Behrens, Uwe, **Fence Shading**, p. 404-409.

Kopp, Manfred, and Gervautz, Michael, **XOR-Drawing with Guaranteed Contrast**, p. 413-414.

Ward, Greg, **A Contrast-Based Scalefactor for Luminance Display**, p. 415-421.

Schlick, Christophe, **High Dynamic Range Pixels**, p. 422-429, code: p. 425-428, [dyn\\_range/](#).

Schlag, John, **Fast Embossing Effects on Raster Image Data**, p. 433-437, code: p. 435-436, [emboss.c](#).

Heckbert, Paul S., **Bilinear Coons Patch Image Warping**, p. 438-446, code: p. 441-444, [coons\\_warp.c](#).

Wolberg, George, and Massalin, Henry, **Fast Convolution with Packed Lookup Tables**, p. 447-464, code: p. 455-463, [convolve.c](#).

Cychosz, Joseph M., **Efficient Binary Image Thinning using Neighborhood Maps**, p. 465-473, code: p. 470-472, [thin\\_image.c](#).

Zuiderveld, Karel, **Contrast Limited Adaptive Histogram Equalization**, p. 474-485, code: p. 479-484, [clahe.c](#).

Paeth, Alan W., **Ideal Tiles for Shading and Halftoning**, p. 486-492.

Christensen, Jon, Marks, Joe, and Shieber, Stuart, **Placing Text Labels on Maps and Diagrams**, p. 497-504.

Szirmay-Kalos, László, **Dynamic Layout Algorithm to Display General Graphs**, p. 505-517, code: p. 511-517, [graph\\_layout/](#).

Hill, Steve, **Tri-linear Interpolation**, p. 521-525, code: p. 523-524, [trilerp.c](#).

Eker, Steven, **Faster Linear Interpolation**, p. 526-533, code: p. 532-533, [interp\\_fast.c](#).

Doué, Jean-François, **C++ Vector and Matrix Algebra Routines**, p. 534-557, code: p. 535-557, [vec\\_mat/](#).

Glassner, Andrew, and Haines, Eric, **C Header File and Vector Library**, p. 558-570, code: p. 558-570,

# Graphics Gems V

- Herbison-Evans, Don, **Solving Quartics and Cubics for Graphics**, p. 3-15, code: [ch1-1/](#).
- Turkowski, Ken, **Computing the Inverse Square Root**, p. 16-21, code: p. 17-19, [ch1-2/](#).
- Turkowski, Ken, **Fixed Point Square Root**, p. 22-24, code: p. 23, [ch1-3/](#).
- Shoemake, Ken, **Rational Approximation**, p. 25-32, code: p. 29-31, [ch1-4/](#).
- Van Gelder, Allen, **Efficient Computation of Polygon Area and Polyhedron Volume**, p. 35-41.
- Carvalho, Paulo Cezar Pinto, and Cavalcanti, Paulo Roma, **Point in Polyhedron Testing Using Spherical Polygons**, p. 42-49, code: p. 46-49, [ch2-2/](#).
- Glassner, Andrew, **Clipping a Concave Polygon**, p. 50-54.
- Hanson, Andrew J., **Rotations for N-dimensional Graphics**, p. 55-64.
- Buckley, Robert, **Parallelhedra and Uniform Color Quantization**, p. 65-71.
- Hill, Kenneth J., **Matrix-based Ellipse Geometry**, p. 72-77, code: [ch2-6/](#).
- Paeth, Alan W., **Distance Approximations and Bounding Polyhedra**, p. 78-87, code: p. 85-86, [ch2-7/](#).
- Alciatore, David, and Miranda, Rick, **The Best Least-Squares Line Fit**, p. 91-97.
- Hill, Steve, and Roberts, Jonathan C., **Surface Models and the Resolution of N-Dimensional Cell Ambiguity**, p. 98-106.
- Arata, Louis K., **Tri-cubic Interpolation**, p. 107-110, code: p. 108-109, [ch3-3/](#).
- Miller, Robert D., **Transforming Coordinates From One Coordinate Plane To Another**, p. 111-120, code: p. 115-120, [ch3-4/](#).
- Chin, Norman, **A Walk Through BSP Trees**, p. 121-138, code: p. 131-138, [ch3-5/](#).
- Blanc, Carole, **Generic Implementation of Axial Deformation Techniques**, p. 139-145, code: p. 141-144, [ch3-6/](#).
- Goldman, Ronald, **Identities for the Univariate, Bivariate Bernstein Basis Fcns**, p. 149-162.
- Goldman, Ronald, **Identities for the B-Spline Basis Functions**, p. 163-167.
- Turkowski, Ken, **Circular Arc Subdivision**, p. 168-172, code: p. 170-171, [ch4-3/](#).
- de Figueiredo, Luiz Henrique, **Adaptive Sampling of Parametric Curves**, p. 173-178, code: p. 177, [ch4-4/](#).
- Ahn, Jaewoo, **Fast Generation of Ellipsoids**, p. 179-190, code: p. 185-190, [ch4-5/](#).
- Bajaj, Chandrajit, and Xu, Guoliang, **Sparse Smooth Connection Between Bézier/B-Spline Curves**, p. 191-198.
- Gravesen, Jens, **The Length of Bézier Curves**, p. 199-205, code: [ch4-7/](#).
- Miller, Robert D., **Quick and Simple Bézier Curve Drawing**, p. 206-209, code: p. 207-209, [ch4-8/](#).
- Shoemake, Ken, **Linear Form Curves**, p. 210-223, code: p. 220-222, [ch4-9/](#).
- Shene, Ching-Kuang, **Computing the Intersection of a Line and a Cone**, p. 227-231.
- Schlick, Christophe, and Subrenat, Gilles, **Ray Intersection of Tessellated Surfaces: Quad Vs Triangle**, p. 232-241, code: p. 237-240, [ch5-2/](#).
- Möller, Tomas, **Faster Ray Tracing Using Scanline Rejection**, p. 242-257, code: p. 249-257, [ch5-3/](#).
- Leipelt, Andreas, **Ray Tracing a Swept Sphere**, p. 258-267, code: p. 261-267, [ch5-4/](#).
- Márton, Gábor, **Acceleration of Ray Tracing via Voronoi-diagrams**, p. 268-284, code: p. 276-283,

[ch5-5/](#).

Zimmerman, Kurt, **Direct Lighting Models for Ray Tracing with Cylindrical Lamps**, p. 285-289.

Feda, Martin, **Improving Intermediate Radiosity Using Directional Light**, p. 290-293.

Purgathofer, Werner, Tobler, Robert F., and Geiler, Manfred, **Improved Threshold Matrices for Ordered Dithering**, p. 297-301.

Wong, Tien-tsin, and Hsu, Siu-chi, **Halftoning with Selective Precipitation and Adaptive Clustering**, p. 302-313, code: p. 306-312, [ch6-2/](#).

Eker, Steven, **Faster Pixel-Perfect Line Clipping**, p. 314-322, code: p. 319-322, [ch6-3/](#).

Doué, Jean-François, and Rubio, Ruben Gonzalez, **Efficient and Robust 2D Shape Vectorization**, p. 323-337, code: p. 329-336, [ch6-4/](#).

Hsu, Siu-chi, and Lee, I.H.H., **Reversible Straight Line Edge Reconstruction**, p. 338-354, code: p. 342-353, [ch6-5/](#).

Sharma, Rajesh, **Priority-based Adaptive Image Refinement**, p. 355-358.

Cross, Robert A., **Sampling Patterns Optimized for Uniform Distribution of Edges**, p. 359-363, code: p. 362, [ch6-7/](#).

Schlick, Christophe, **Wave Generators for Computer Graphics**, p. 367-374, code: p. 371-374, [ch7-1/](#).

Green, Daniel, and Hatch, Don, **Fast Polygon-Cube Intersection Testing**, p. 375-379, code: [ch7-2/](#).

Bouma, William, and Vanecek, George, Jr., **Velocity-based Collision Detection**, p. 380-385, code: p. 383-385, [ch7-3/](#).

Vanecek, George, Jr., **Spatial Partitioning of a Polygon by a Plane**, p. 386-393, code: p. 387-393, [ch7-4/](#).

Narkhede, Atul, and Manocha, Dinesh, **Fast Polygon Triangulation Based on Seidel's Algorithm**, p. 394-397, code: [ch7-5/](#).

Karinthi, Raghu, **Accurate Z-buffer Rendering**, p. 398-399, code: [ch7-6/](#).

Paeth, Alan W., Scheepers, Ferdi, and May, Stephen, **A Survey of Graphics Libraries**, p. 400-406, code: [ch7-7/](#).

---

Last change: *July 6, 2000* [Eric Haines](#), Gems archivist / [erich@acm.org](mailto:erich@acm.org)



# Graphics Gems Repository

**Announcement:** The easy to remember URL for this site is  
<http://www.graphicsgems.org/>.

This is the official on-line repository for the code from the *Graphics Gems* series of books (from [Academic Press](#)). This series focusses on short to medium length pieces of code which perform a wide variety of computer graphics related tasks. All code here can be used without restrictions. The code distributions here contain all known bug fixes and enhancements. We also provide errata listings for the text of each book. Please report any new errata or bugs to [Eric Haines](#).

The gems can be viewed [by category](#), [by book](#), or [by author](#). Gems code can be accessed in two ways: by viewing the code directly from these pages or by downloading a book's entire code base.

<a href="#">Graphics Gems Series Table of Contents</a>			
<a href="#">Graphics Gems</a>	<a href="#">Errata listing</a>	<a href="#">tar file</a>	<a href="#">zip file</a>
<a href="#">Graphics Gems II</a>	<a href="#">Errata listing</a>	<a href="#">tar file</a>	<a href="#">zip file</a>
<a href="#">Graphics Gems III</a>	<a href="#">Errata listing</a>	<a href="#">tar file</a>	<a href="#">zip file</a>
<a href="#">Graphics Gems IV</a>	<a href="#">Errata listing</a>	<a href="#">tar file</a>	<a href="#">zip file</a>
<a href="#">Graphics Gems V</a>	<a href="#">Errata listing</a>	<a href="#">tar file</a>	<a href="#">zip file</a>
Gems V image samples		<a href="#">tar file</a>	<a href="#">zip file</a>

Here is publication information for the books themselves:

- [Graphics Gems](#), *Andrew Glassner (editor)*, Academic Press, 1990, ISBN: 0122861663.
- [Graphics Gems II](#), *James Arvo (editor)*, Academic Press, 1991, ISBN: 0120644819.
- [Graphics Gems III](#), *David Kirk (editor)*, Academic Press, 1992, ISBN: 0124096735 ([Mac](#): 0124096727).
- [Graphics Gems IV](#), *Paul Heckbert (editor)*, Academic Press, 1994, ISBN: 0123361559 ([Mac](#): 0123361567).
- [Graphics Gems V](#), *Alan Paeth (editor)*, Academic Press, 1995, ISBN: 0125434553 ([Mac](#): 012543457X).

**Note:** the first three books in the series can be [bought as a package](#) directly from Morgan Kaufmann for considerably less money.

The ISBN numbers listed are for the books with IBM PC disks included, with Mac versions in parentheses. The archives at this site are newer than those in the books, as they include the most up-to-date code with all known bug fixes.

**Related Resources:** The *Graphics Gems* series has been ended by the founding editor, Andrew Glassner.

To allow continued presentation of new results, he started the [\*journal of graphics tools\*](#), a quarterly which includes Gems-like material. Their web site includes related code for a number of basic graphics operations (e.g. [triangle-triangle intersection](#), [gamma-corrected antialiased lines](#), [Hammersley and Halton sampling](#), etc.).

An excellent place for solid code for all sorts of basic graphics algorithms is Dave Eberly's [Magic Software](#) site.

Another useful site is the [3D Object Intersection page](#). Gems and other articles about various intersection algorithms are shown here.



go to [ACM TOG Software Page](#)

---

Last change: *January 3, 2001* [Eric Haines](#), Gems archivist / [erich@acm.org](mailto:erich@acm.org)

# Graphics Gems, by Category

Listed by category. Note that some gems may be listed more than once.

[2D Geometry](#)

[2D Rendering](#)

[3D Geometry](#)

[3D Rendering](#)

[C Utilities](#)

[Curves and Surfaces](#)

[Frame Buffer Techniques](#)

[Image Processing](#)

[Matrix Techniques](#)

[Modeling and Transformations](#)

[Numerical and Programming Techniques](#)

[Radiosity](#)

[Ray Tracing](#)

[return to main page](#)

---

## 2D Geometry

Glassner, Andrew, **Useful 2D Geometry**, *Graphics Gems*, p. 3-11.

Glassner, Andrew, **Useful Trigonometry**, *Graphics Gems*, p. 13-17.

Paeth, Alan W., **Trigonometric Functions at Select Points**, *Graphics Gems*, p. 18-19.

Goldman, Ronald, **Triangles**, *Graphics Gems*, p. 20-23.

Turk, Greg, **Generating Random Points in Triangles**, *Graphics Gems*, p. 24-28, code: p. 649-650, [TriPoints.c](#).

Shapira, Andrew, **Fast Line-Edge Intersections on a Uniform Grid**, *Graphics Gems*, p. 29-36, code: p. 651-653, [LineEdge.c](#).

Thompson, Kelvin, **Area of Intersection: Circle and a Half-Plane**, *Graphics Gems*, p. 38-39.

Thompson, Kelvin, **Area of Intersection: Circle and a Thick Line**, *Graphics Gems*, p. 40-42.

Thompson, Kelvin, **Area of Intersection: Two Circles**, *Graphics Gems*, p. 43-46.

Thompson, Kelvin, **Vertical Distance from a Point to a Line**, *Graphics Gems*, p. 47-48.

Paeth, Alan W., **A Fast 2D Point-on-line Test**, *Graphics Gems*, p. 49-50, code: p. 654-655, [PntOnLine.c](#).

Shaffer, Clifford A., **Fast Circle-Rectangle Intersection Checking**, *Graphics Gems*, p. 51-53, code: p. 656, [CircleRect.c](#).

Glassner, Andrew, **2D and 3D Vector C Library**, *Graphics Gems*, p. 633-642, code: p. 633-642, [GGVecLib.c](#) [GraphicsGems.h](#).

Rokne, Jon, **The Area of a Simple Polygon**, *Graphics Gems II*, p. 5-6.

- Prasad, Mukesh, **Intersection of Line Segments**, *Graphics Gems II*, p. 7-9, code: p. 473-476, [xlines.c](#).
- Morrison, Jack C., **Distance From a Point To a Line**, *Graphics Gems II*, p. 10-13.
- Rokne, Jon, **An Easy Bounding Circle**, *Graphics Gems II*, p. 14-16.
- Rokne, Jon, **The Smallest Circle Containing the Intersection of Two Circles**, *Graphics Gems II*, p. 17-18.
- Rokne, Jon, **Appolonius's 10th Problem**, *Graphics Gems II*, p. 19-24.
- Musgrave, F. Kenton, **A Peano Curve Generation Algorithm**, *Graphics Gems II*, p. 25, code: p. 477-484, [Peano/](#).
- Voorhies, Douglas, **Space-Filling Curves and a Measure of Coherence**, *Graphics Gems II*, p. 26-30, code: p. 485-486, [Hilbert.c](#).
- Steinhart, Jonathan E., **Scanline Coherent Shape Algebra**, *Graphics Gems II*, p. 31-45, code: p. 487-501.
- Goldman, Ronald, **Area of Planar Polygons and Volume of Polyhedra**, *Graphics Gems II*, p. 170-171.
- Glassner, Andrew, and Bogart, Rod G., **2D and 3D Vector C Library**, *Graphics Gems II*, p. 458-466, code: p. 633-642, [GGVecLib.c](#) [GraphicsGems.h](#).
- Van Aken, Jerry, and Simar, Ray, **A Parametric Elliptical Arc Algorithm**, *Graphics Gems III*, p. 164-172, code: p. 478-479, [parelarc.c](#).
- Rosati, Claudio, **A Simple Connection Algorithm for 2-D Drawing**, *Graphics Gems III*, p. 173-181, code: p. 480-486, [con2d.c](#).
- Srinivasan, Raman V., **A Fast Circle Clipping Algorithm**, *Graphics Gems III*, p. 182-187, code: p. 487-490, [circlexc.c](#).
- Shaffer, Clifford A., and Feustel, Charles D., **Exact Computation of 2-D Intersections**, *Graphics Gems III*, p. 188-192, code: p. 491-495, [Polyintr.c](#).
- Miller, Robert D., **Joining Two Lines with a Circular Arc Fillet**, *Graphics Gems III*, p. 193-198, code: p. 496-499, [fillet.c](#).
- Antonio, Franklin, **Faster Line Segment Intersection**, *Graphics Gems III*, p. 199-202, code: p. 500-501, [insectc.c](#).
- Sevici, Constantin A., **Solving the Problem of Apollonius and Other Related Problems**, *Graphics Gems III*, p. 203-209.
- Bashein, Gerard, and Detmer, Paul R., **Centroid of a Polygon**, *Graphics Gems IV*, p. 3-6, code: p. 5, [centroid.c](#).
- Schorn, Peter, and Fisher, Frederick, **Testing the Convexity of a Polygon**, *Graphics Gems IV*, p. 7-15, code: p. 11-15, [convex\\_test/](#).
- Weiler, Kevin, **An Incremental Angle Point in Polygon Test**, *Graphics Gems IV*, p. 16-23, code: p. 17-22, [ptpoly\\_weiler/](#).
- Haines, Eric, **Point in Polygon Strategies**, *Graphics Gems IV*, p. 24-46, code: p. 34-45, [ptpoly\\_haines/](#).
- Lischinski, Dani, **Incremental Delaunay Triangulation**, *Graphics Gems IV*, p. 47-59, code: p. 51-58, [delaunay/](#).
- Hill, F. S., Jr., **The Pleasures of 'Perp Dot' Products**, *Graphics Gems IV*, p. 138-148.
- Hanson, Andrew J., **Geometry for N-Dimensional Graphics**, *Graphics Gems IV*, p. 149-170.
- Glassner, Andrew, and Haines, Eric, **C Header File and Vector Library**, *Graphics Gems IV*, p. 558-570, code: p. 558-570, [GraphicsGems.c](#) [GraphicsGems.h](#).
- Van Gelder, Allen, **Efficient Computation of Polygon Area and Polyhedron Volume**, *Graphics Gems V*, p. 35-41.



Glassner, Andrew, **Clipping a Concave Polygon**, *Graphics Gems V*, p. 50-54.

Hanson, Andrew J., **Rotations for N-dimensional Graphics**, *Graphics Gems V*, p. 55-64.

Hill, Kenneth J., **Matrix-based Ellipse Geometry**, *Graphics Gems V*, p. 72-77, code: [ch2-6/](#).

Narkhede, Atul, and Manocha, Dinesh, **Fast Polygon Triangulation Based on Seidel's Algorithm**, *Graphics Gems V*, p. 394-397, code: [ch7-5/](#).

## 2D Rendering

Paeth, Alan W., **Circles of Integral Radius on Integer Lattices**, *Graphics Gems*, p. 57-60.

Heckbert, Paul S., **Nice Numbers for Graph Labels**, *Graphics Gems*, p. 61-63, code: p. 657-659, [Label.c](#).

Cychosz, Joseph M., **Efficient Generation of Sampling Jitter Using Look-up Tables**, *Graphics Gems*, p. 64-74, code: p. 660-661, [FastJitter.c](#).

Morrison, Jack C., **Fast Anti-Aliasing Polygon Scan Conversion**, *Graphics Gems*, p. 76-83, code: p. 662-666, [AAPolyScan.c](#).

Heckbert, Paul S., **Generic Convex Polygon Scan Conversion and Clipping**, *Graphics Gems*, p. 84-86, code: p. 667-680, [PolyScan/](#).

Heckbert, Paul S., **Concave Polygon Scan Conversion**, *Graphics Gems*, p. 87-91, code: p. 681-684, [ConcaveScan.c](#).

Wallis, Bob, **Fast Scan Conversion of Arbitrary Polygons**, *Graphics Gems*, p. 92-97.

Heckbert, Paul S., **Digital Line Drawing**, *Graphics Gems*, p. 99-100, code: p. 685, [DigitalLine.c](#).

Wyvill, Brian, **Symmetric Double Step Line Algorithm**, *Graphics Gems*, p. 101-104, code: p. 686-689, [DoubleLine.c](#).

Thompson, Kelvin, **Rendering Anti-Aliased Lines**, *Graphics Gems*, p. 105-106, code: p. 690-693, [AALines/](#).

Ritter, Jack, **An Algorithm for Filling in 2D Wide Line Bevel Joints**, *Graphics Gems*, p. 107-113.

Wallis, Bob, **Rendering Fat Lines on a Raster Grid**, *Graphics Gems*, p. 114-120.

Spoelder, Hans J.W., and Ullings, Fons H., **Two-Dimensional Clipping: A Vector-Based Approach**, *Graphics Gems*, p. 121-128, code: p. 694-710, [2DClip/](#).

Lee, Greg, Penk, Mike, and Wallis, Bob, **Periodic Tilings of the Plane on a Raster Grid**, *Graphics Gems*, p. 129-139.

Steinhart, Jonathan E., **Scanline Coherent Shape Algebra**, *Graphics Gems II*, p. 31-45, code: p. 487-501.

Cheng, Russell C.H., **Edge and Bit-Mask Calculations for Anti-Aliasing**, *Graphics Gems III*, p. 349-354, code: p. 586-593, [edgeCalc.c](#).

Grace, Thom, **Fast Span Conversion: Unrolling Short Loops**, *Graphics Gems III*, p. 355-357, code: p. 594-596, [fastSpan.c](#).

Hollasch, Steve, **Progressive Image Refinement via Gridded Sampling**, *Graphics Gems III*, p. 358-361, code: p. 597-598, [PIR.c](#).

Fleischer, Kurt, and Salesin, David, **Accurate Polygon Scan Conversion Using Half-Open Intervals**, *Graphics Gems III*, p. 362-365, code: p. 599-605, [accurate\\_scan/](#).

Glassner, Andrew, **Anti-Aliasing in Triangular Pixels**, *Graphics Gems III*, p. 369-373.

Donovan, Walt, and Van Hook, Tim, **Direct Outcode Calculation for Faster Clip Testing**, *Graphics*

*Gems IV*, p. 125-131, code: p. 127-131, [outcode/](#).

Christensen, Jon, Marks, Joe, and Shieber, Stuart, **Placing Text Labels on Maps and Diagrams**, *Graphics Gems IV*, p. 497-504.

Szirmay-Kalos, László, **Dynamic Layout Algorithm to Display General Graphs**, *Graphics Gems IV*, p. 505-517, code: p. 511-517, [graph\\_layout/](#).

Glassner, Andrew, **Clipping a Concave Polygon**, *Graphics Gems V*, p. 50-54.

## 3D Geometry

Glassner, Andrew, **Useful 3D Geometry**, *Graphics Gems*, p. 297-300.

Ritter, Jack, **An Efficient Bounding Sphere**, *Graphics Gems*, p. 301-303, code: p. 723-725, [BoundSphere.c](#).

Goldman, Ronald, **Intersection of Two Lines in Three-Space**, *Graphics Gems*, p. 304.

Goldman, Ronald, **Intersection of Three Planes**, *Graphics Gems*, p. 305.

Paeth, Alan W., **Digital Cartography for Computer Graphics**, *Graphics Gems*, p. 307-320.

Bame, Paul D., **Albers Equal-Area Conic Map Projection**, *Graphics Gems*, p. 321-325, code: p. 726-729, [Albers.c](#).

Montani, Claudio, and Scopigno, Roberto, **Spheres-to-Voxels Conversion**, *Graphics Gems*, p. 327-334.

Arvo, James, **A Simple Method for Box-Sphere Intersection Testing**, *Graphics Gems*, p. 335-339, code: p. 730-732, [BoxSphere.c](#).

Glassner, Andrew, **2D and 3D Vector C Library**, *Graphics Gems*, p. 633-642, code: p. 633-642, [GGVecLib.c](#) [GraphicsGems.h](#).

Goldman, Ronald, **Area of Planar Polygons and Volume of Polyhedra**, *Graphics Gems II*, p. 170-171.

Shaffer, Clifford A., **Getting Around on a Sphere**, *Graphics Gems II*, p. 172-173.

Paeth, Alan W., **Exact Dihedral Metrics for Common Polyhedra**, *Graphics Gems II*, p. 174-178.

Glassner, Andrew, **A Simple Viewing Geometry**, *Graphics Gems II*, p. 179-180.

Bogart, Rod G., **View Correlation**, *Graphics Gems II*, p. 181-190, code: p. 550-562, [viewcorr/](#).

Glassner, Andrew, **Maintaining Winged-Edge Models**, *Graphics Gems II*, p. 191-201.

Montani, Claudio, and Scopigno, Roberto, **Quadtree/Octree-to-Boundary Conversion**, *Graphics Gems II*, p. 202-218.

Glassner, Andrew, and Bogart, Rod G., **2D and 3D Vector C Library**, *Graphics Gems II*, p. 458-466, code: p. 633-642, [GGVecLib.c](#) [GraphicsGems.h](#).

López-López, Fernando J., **Triangles Revisited**, *Graphics Gems III*, p. 215-218.

Chin, Norman, **Partitioning a 3-D Convex Polygon with an Arbitrary Plane**, *Graphics Gems III*, p. 219-222, code: p. 502-510, [partition3d/](#).

Georgiades, Príamos, **Signed Distance From Point To Plane**, *Graphics Gems III*, p. 223-224, code: p. 511, [pt2plane.c](#).

Salesin, David, and Tampieri, Filippo, **Grouping Nearly Coplanar Polygons Into Coplanar Sets**, *Graphics Gems III*, p. 225-230, code: p. 512-516, [planeSets.c](#).

Tampieri, Filippo, **Newell's Method for the Plane Equation of a Polygon**, *Graphics Gems III*, p. 231-232, code: p. 517-518, [newell.c](#).

Georgiades, Príamos, **Plane-to-Plane Intersection**, *Graphics Gems III*, p. 233-235, code: p. 519-520, [pl2plane.c](#).

- Voorhies, Douglas, **Triangle-Cube Intersection**, *Graphics Gems III*, p. 236-239, code: p. 521-526, [triangleCube.c](#).
- Wanger, Len, and Fusco, Mike, **Fast N-Dimensional Extent Overlap Testing**, *Graphics Gems III*, p. 240-243, code: p. 527-533, [extttest/](#).
- Moore, Doug, **Subdividing Simplices**, *Graphics Gems III*, p. 244-249, code: p. 534-535, [simplex/](#).
- Moore, Doug, **Understanding Simplicoids**, *Graphics Gems III*, p. 250-255.
- Lischinski, Dani, **Converting Bézier Triangles Into Rectangular Patches**, *Graphics Gems III*, p. 256-261, code: p. 536-537, [bezierTri.C](#).
- Lindgren, Terence, Sanchez, Juan, and Hall, Jim, **Curve Tessellation Criteria Thru Sampling**, *Graphics Gems III*, p. 262-265.
- Glassner, Andrew, **Building Vertex Normals from an Unstructured Polygon List**, *Graphics Gems IV*, p. 60-73, code: p. 64-73, [vert\\_norm/](#).
- Greene, Ned, **Detecting Intersection of a Rectangular Solid and a Convex Polyhedron**, *Graphics Gems IV*, p. 74-82.
- Rabbitz, Rich, **Fast Collision Detection of Moving Convex Polyhedra**, *Graphics Gems IV*, p. 83-109, code: p. 91-108, [collide.c](#).
- Hart, John C., **Distance to an Ellipsoid**, *Graphics Gems IV*, p. 113-119.
- Ohashi, Yoshikazu, **Fast Linear Approximations of Euclidean Distance in Higher Dimensions**, *Graphics Gems IV*, p. 121-124, code: [dist\\_fast.c](#).
- Miller, Robert D., **Computing the Area of a Spherical Polygon**, *Graphics Gems IV*, p. 132-137, code: p. 135-136, [sph\\_poly.c](#).
- Hanson, Andrew J., **Geometry for N-Dimensional Graphics**, *Graphics Gems IV*, p. 149-170.
- Glassner, Andrew, and Haines, Eric, **C Header File and Vector Library**, *Graphics Gems IV*, p. 558-570, code: p. 558-570, [GraphicsGems.c](#) [GraphicsGems.h](#).
- Van Gelder, Allen, **Efficient Computation of Polygon Area and Polyhedron Volume**, *Graphics Gems V*, p. 35-41.
- Carvalho, Paulo Cezar Pinto, and Cavalcanti, Paulo Roma, **Point in Polyhedron Testing Using Spherical Polygons**, *Graphics Gems V*, p. 42-49, code: p. 46-49, [ch2-2/](#).
- Hanson, Andrew J., **Rotations for N-dimensional Graphics**, *Graphics Gems V*, p. 55-64.
- Buckley, Robert, **Parallelhedra and Uniform Color Quantization**, *Graphics Gems V*, p. 65-71.
- Paeth, Alan W., **Distance Approximations and Bounding Polyhedra**, *Graphics Gems V*, p. 78-87, code: p. 85-86, [ch2-7/](#).
- Green, Daniel, and Hatch, Don, **Fast Polygon-Cube Intersection Testing**, *Graphics Gems V*, p. 375-379, code: [ch7-2/](#).
- Bouma, William, and Vanecek, George, Jr., **Velocity-based Collision Detection**, *Graphics Gems V*, p. 380-385, code: p. 383-385, [ch7-3/](#).
- Vanecek, George, Jr., **Spatial Partitioning of a Polygon by a Plane**, *Graphics Gems V*, p. 386-393, code: p. 387-393, [ch7-4/](#).

# 3D Rendering

Heckbert, Paul S., **Generic Convex Polygon Scan Conversion and Clipping**, *Graphics Gems*, p. 84-86, code: p. 667-680, [PolyScan/](#).

Wyvill, Brian, **3D Grid Hashing Function**, *Graphics Gems*, p. 343-345, code: p. 733-734, [Hash3D.c](#).

Hultquist, Jeff, **Backface Culling**, *Graphics Gems*, p. 346-347.

Lee, Mark E., **Fast Dot Products for Shading**, *Graphics Gems*, p. 348-360.

Thompson, Kelvin, **Scanline Depth Gradient of a Z-Buffered Triangle**, *Graphics Gems*, p. 361-363.

Glassner, Andrew, **Simulating Fog and Haze**, *Graphics Gems*, p. 364-365.

Glassner, Andrew, **Interpretation of Texture Map Indices**, *Graphics Gems*, p. 366-375.

Glassner, Andrew, **Multidimensional Sum Tables**, *Graphics Gems*, p. 376-381.

Maillot, Patrick-Gilles, **Three-Dimensional Homogeneous Clipping of Triangle Strips**, *Graphics Gems II*, p. 219-231, code: p. 563-570.

Thalmann, Nadia Magnenat, Thalmann, Daniel, and Minh, Hong Tong, **InterPhong Shading**, *Graphics Gems II*, p. 232-241, code: p. 571-574, [InterPhong.c](#).

Ward, Greg, **A Recursive Implementation of the Perlin Noise Function**, *Graphics Gems II*, p. 396-401, code: p. 615-616, [noise3.c](#).

Woo, Andrew, **The Shadow Depth Map Revisited**, *Graphics Gems III*, p. 338-342, code: p. 582, [zdepth.c](#).

Cheng, Russell C.H., **Fast Linear Color Rendering**, *Graphics Gems III*, p. 343-348, code: p. 583-585, [fastLinear.c](#).

Hollasch, Steve, **Progressive Image Refinement via Gridded Sampling**, *Graphics Gems III*, p. 358-361, code: p. 597-598, [PIR.c](#).

Glassner, Andrew, **Darklights**, *Graphics Gems III*, p. 366-368.

Snyder, John, Barzel, Ronen, and Gabriel, Steve, **Motion Blur on Graphics Workstations**, *Graphics Gems III*, p. 374-382, code: p. 606-609, [motblur.c](#).

Arvo, James, and Scofield, Cary, **The Shader Cache: A Rendering Pipeline Accelerator**, *Graphics Gems III*, p. 383-389.

Donovan, Walt, and Van Hook, Tim, **Direct Outcode Calculation for Faster Clip Testing**, *Graphics Gems IV*, p. 125-131, code: p. 127-131, [outcode/](#).

Schlick, Christophe, **A Fast Alternative to Phong's Specular Model**, *Graphics Gems IV*, p. 385-387.

Fisher, Frederick, and Woo, Andrew, **R.E versus N.H Specular Highlights**, *Graphics Gems IV*, p. 388-400.

Schlick, Christophe, **Fast Alternatives to Perlin's Bias and Gain Functions**, *Graphics Gems IV*, p. 401-403.

Behrens, Uwe, **Fence Shading**, *Graphics Gems IV*, p. 404-409.

Schlick, Christophe, **Wave Generators for Computer Graphics**, *Graphics Gems V*, p. 367-374, code: p. 371-374, [ch7-1/](#).

Karinthi, Raghu, **Accurate Z-buffer Rendering**, *Graphics Gems V*, p. 398-399, code: [ch7-6/](#).



# C Utilities

- Glassner, Andrew, **Graphics Gems Header File**, *Graphics Gems*, p. 629-632, code: p. 629-632, [GraphicsGems.h](#).
- Glassner, Andrew, **2D and 3D Vector C Library**, *Graphics Gems*, p. 633-642, code: p. 633-642, [GGVecLib.c](#) [GraphicsGems.h](#).
- Hultquist, Jeff, **Memory Allocation in C**, *Graphics Gems*, p. 643, code: p. 643.
- Raible, Eric, **Two Useful C Macros**, *Graphics Gems*, p. 644, code: p. 644.
- Thompson, Kelvin, **How to Build Circular Structures in C**, *Graphics Gems*, p. 645, code: p. 645.
- Thompson, Kelvin, **How to Use C Register Variables to Point to 2D Arrays**, *Graphics Gems*, p. 646, code: p. 646.
- Glassner, Andrew, **Graphics Gems Header File**, *Graphics Gems II*, p. 455-457, code: p. 629-632, [GraphicsGems.h](#).
- Glassner, Andrew, and Bogart, Rod G., **2D and 3D Vector C Library**, *Graphics Gems II*, p. 458-466, code: p. 633-642, [GGVecLib.c](#) [GraphicsGems.h](#).
- Hollasch, Steve, **Useful C Macros for Vector Operations**, *Graphics Gems II*, p. 467-469, code: p. 467-469, [vector.h](#).
- Glassner, Andrew, **Graphics Gems Header File**, *Graphics Gems III*, p. 455-457, code: p. 393-395, [GraphicsGems.h](#).
- Glassner, Andrew, **2-D and 3-D Vector C Library -- Corrected and Indexed**, *Graphics Gems III*, p. 396-404, code: p. 396-404, [GraphicsGems.c](#).
- Hollasch, Steve, **Useful C Macros for Vector Operations**, *Graphics Gems III*, p. 405-407, code: p. 405-407, [vector.h](#).
- Doué, Jean-François, **C++ Vector and Matrix Algebra Routines**, *Graphics Gems IV*, p. 534-557, code: p. 535-557, [vec\\_mat/](#).
- Glassner, Andrew, and Haines, Eric, **C Header File and Vector Library**, *Graphics Gems IV*, p. 558-570, code: p. 558-570, [GraphicsGems.c](#) [GraphicsGems.h](#).
- Paeth, Alan W., Scheepers, Ferdi, and May, Stephen, **A Survey of Graphics Libraries**, *Graphics Gems V*, p. 400-406, code: [ch7-7/](#).

# Curves and Surfaces

- Glassner, Andrew, **Planar Cubic Curves**, *Graphics Gems*, p. 575-578.
- Rasala, Richard, **Explicit Cubic Spline Interpolation Formulas**, *Graphics Gems*, p. 579-584.
- Gomez, Julian, **Fast Spline Drawing**, *Graphics Gems*, p. 585-586.
- Goldman, Ronald, **Some Properties of Bézier Curves**, *Graphics Gems*, p. 587-593.
- Wallis, Bob, **Tutorial on Forward Differencing**, *Graphics Gems*, p. 594-603.
- Goldman, Ronald, **Integration of Bernstein Basis Functions**, *Graphics Gems*, p. 604-606.
- Schneider, Philip J., **Solving the Nearest-Point-on-Curve Problem**, *Graphics Gems*, p. 607-611, code: p. 787-796, [Nearest.c](#).
- Schneider, Philip J., **An Algorithm for Automatically Fitting Digitized Curves**, *Graphics Gems*, p. 612-626, code: p. 797-807, [FitCurves.c](#).
- Moore, Doug, and Warren, Joseph, **Least-Squares Approximations To Bézier Curves and Surfaces**,

*Graphics Gems II*, p. 406-411.

Shoemake, Ken, **Beyond Bézier Curves**, *Graphics Gems II*, p. 412-416.

Schlag, John, **A Simple Formulation for Curve Interpolation with Variable Control Point Approximation**, *Graphics Gems II*, p. 417-419.

Lindgren, Terence, **Symmetric Evaluation of Polynomials**, *Graphics Gems II*, p. 420-423.

Seidel, Hans-Peter, **Menelaus's Theorem**, *Graphics Gems II*, p. 424-427.

Seidel, Hans-Peter, **Geometrically Continuous Cubic Bézier Curves**, *Graphics Gems II*, p. 428-434.

Musial, Christopher J., **A Good Straight-Line Approximation of a Circular Arc**, *Graphics Gems II*, p. 435-439, code: p. 617.

Paeth, Alan W., **Great Circle Plotting**, *Graphics Gems II*, p. 440-445.

Wu, Xiaolin, **Fast Anti-Aliased Circle Generation**, *Graphics Gems II*, p. 446-450.

Eilers, Paul H. C., **Smoothing and Interpolation with Finite Differences**, *Graphics Gems IV*, p. 241-250, code: p. 246-249, [data\\_smooth/](#).

Barry, Phillip, and Goldman, Ronald, **Knot Insertion using Forward Differences**, *Graphics Gems IV*, p. 251-255.

Bajaj, Chandrajit, and Xu, Guoliang, **Converting a Rational Curve to a Standard Rational Bernstein-Bézier Representation**, *Graphics Gems IV*, p. 256-260.

Klassen, R. Victor, **Intersecting Parametric Cubic Curves by Midpoint Subdivision**, *Graphics Gems IV*, p. 261-277, code: p. 266-276, [curve\\_isect/](#).

Lischinski, Dani, **Converting Rectangular Patches into Bézier Triangles**, *Graphics Gems IV*, p. 278-285, code: p. 281-285, [patch\\_conv.C](#).

Peterson, John W., **Tessellation of NURB Surfaces**, *Graphics Gems IV*, p. 286-320, code: p. 294-319, [nurb\\_polyg/](#).

Shene, Ching-Kuang, **Equations of Cylinders and Cones**, *Graphics Gems IV*, p. 321-323.

Bloomenthal, Jules, **An Implicit Surface Polygonizer**, *Graphics Gems IV*, p. 324-349, code: p. 334-349, [implicit.c](#).

Goldman, Ronald, **Identities for the Univariate, Bivariate Bernstein Basis Fcns**, *Graphics Gems V*, p. 149-162.

Goldman, Ronald, **Identities for the B-Spline Basis Functions**, *Graphics Gems V*, p. 163-167.

Turkowski, Ken, **Circular Arc Subdivision**, *Graphics Gems V*, p. 168-172, code: p. 170-171, [ch4-3/](#).

de Figueiredo, Luiz Henrique, **Adaptive Sampling of Parametric Curves**, *Graphics Gems V*, p. 173-178, code: p. 177, [ch4-4/](#).

Ahn, Jaewoo, **Fast Generation of Ellipsoids**, *Graphics Gems V*, p. 179-190, code: p. 185-190, [ch4-5/](#).

Bajaj, Chandrajit, and Xu, Guoliang, **Sparse Smooth Connection Between Bézier/B-Spline Curves**, *Graphics Gems V*, p. 191-198.

Gravesen, Jens, **The Length of Bézier Curves**, *Graphics Gems V*, p. 199-205, code: [ch4-7/](#).

Miller, Robert D., **Quick and Simple Bézier Curve Drawing**, *Graphics Gems V*, p. 206-209, code: p. 207-209, [ch4-8/](#).

Shoemake, Ken, **Linear Form Curves**, *Graphics Gems V*, p. 210-223, code: p. 220-222, [ch4-9/](#).

# Frame Buffer Techniques

Glassner, Andrew, **Frame Buffers and Color Maps**, *Graphics Gems*, p. 215-218.

Paeth, Alan W., **Reading a Write-Only Write Mask**, *Graphics Gems*, p. 219-220.

Morton, Mike, **A Digital "Dissolve" Effect**, *Graphics Gems*, p. 221-232, code: p. 715-717, [Dissolve.c](#).

Paeth, Alan W., **Mapping RGB Triples Onto Four Bits**, *Graphics Gems*, p. 233-245, code: p. 718, [RGBTo4Bits.c](#).

Heckbert, Paul S., **What Are the Coordinates of a Pixel?**, *Graphics Gems*, p. 246-248.

Paeth, Alan W., **Proper Treatment of Pixels as Integers**, *Graphics Gems*, p. 249-256, code: p. 719, [PixelInteger.c](#).

Glassner, Andrew, **Normal Coding**, *Graphics Gems*, p. 257-264.

Heckbert, Paul S., **Recording Animation in Binary Order for Progressive Temporal Refinement**, *Graphics Gems*, p. 265-269, code: p. 720, [BinRec.c](#).

Schumacher, Dale A., **1-to-1 Pixel Transforms Optimized Through Color-Map Manipulation**, *Graphics Gems*, p. 270-274.

Heckbert, Paul S., **A Seed Fill Algorithm**, *Graphics Gems*, p. 275-277, code: p. 721-722, [SeedFill.c](#).

Fishkin, Ken, **Filling a Region in a Frame Buffer**, *Graphics Gems*, p. 278-284.

Wallace, Bill, **Precalculating Addresses for Fast Fills, Circles, and Lines**, *Graphics Gems*, p. 285-286.

Gervautz, Michael, and Purgathofer, Werner, **A Simple Method for Color Quantization: Octree Quantization**, *Graphics Gems*, p. 287-293.

Steinhart, Jonathan E., **Scanline Coherent Shape Algebra**, *Graphics Gems II*, p. 31-45, code: p. 487-501.

Thomas, Spencer W., **Efficient Inverse Color Map Computation**, *Graphics Gems II*, p. 116-125, code: p. 528-535, [inv\\_cmap/](#).

Wu, Xiaolin, **Efficient Statistical Computations for Optimal Color Quantization**, *Graphics Gems II*, p. 126-133, code: [quantizer.c](#).

Musgrave, F. Kenton, **A Random Color Map Animation Algorithm**, *Graphics Gems II*, p. 134-137, code: p. 536-541, [ran\\_ramp.c](#).

Hall, Jim, and Lindgren, Terence, **A Fast Approach To PHIGS PLUS Pseudo Color**, *Graphics Gems II*, p. 138-142.

Paeth, Alan W., **Mapping RGB Triples Onto 16 Distinct Values**, *Graphics Gems II*, p. 143-146.

Martindale, David, and Paeth, Alan W., **Television Color Encoding and "Hot" Broadcast Colors**, *Graphics Gems II*, p. 147-158, code: p. 542-549, [hot.c](#).

Meyer, Gary W., **An Inexpensive Method of Setting the Monitor White Point**, *Graphics Gems II*, p. 159-162.

Musgrave, F. Kenton, **Some Tips for Making Color Hardcopy**, *Graphics Gems II*, p. 163-165.

Kopp, Manfred, and Gervautz, Michael, **XOR-Drawing with Guaranteed Contrast**, *Graphics Gems IV*, p. 413-414.

Ward, Greg, **A Contrast-Based Scalefactor for Luminance Display**, *Graphics Gems IV*, p. 415-421.

Schlick, Christophe, **High Dynamic Range Pixels**, *Graphics Gems IV*, p. 422-429, code: p. 425-428, [dyn\\_range/](#).

Buckley, Robert, **Parallelohedra and Uniform Color Quantization**, *Graphics Gems V*, p. 65-71.

# Image Processing

- Pavlicic, Mark J., **Anti-Aliasing Filters that Minimize "Bumpy" Sampling**, *Graphics Gems*, p. 144-146.
- Turkowski, Ken, **Filters for Common Resampling Tasks**, *Graphics Gems*, p. 147-165.
- Olsen, John, **Smoothing Enlarged Monochrome Images**, *Graphics Gems*, p. 166-170.
- Paeth, Alan W., **Median Finding on a 3-by-3 Grid**, *Graphics Gems*, p. 171-175, code: p. 711-712, [Median.c](#).
- Hawley, Stephen, **Ordered Dithering**, *Graphics Gems*, p. 176-178, code: p. 713-714, [OrderDither.c](#).
- Paeth, Alan W., **A Fast Algorithm for General Raster Rotation**, *Graphics Gems*, p. 179-195.
- Schumacher, Dale A., **Useful 1-to-1 Pixel Transforms**, *Graphics Gems*, p. 196-209.
- Thompson, Kelvin, **Alpha Blending**, *Graphics Gems*, p. 210-211.
- Schumacher, Dale A., **Image Smoothing and Sharpening by Discrete Convolution**, *Graphics Gems II*, p. 50-56.
- Schumacher, Dale A., **A Comparison of Digital Halftoning Techniques**, *Graphics Gems II*, p. 57-71, code: p. 502-508.
- Thomas, Spencer W., and Bogart, Rod G., **Color Dithering**, *Graphics Gems II*, p. 72-77, code: p. 509-513, [dither/](#).
- Schumacher, Dale A., **Fast Anamorphic Image Scaling**, *Graphics Gems II*, p. 78-79.
- Ward, Greg, **Real Pixels**, *Graphics Gems II*, p. 80-83, code: [RealPixels/](#).
- Yap, Sue-Ken, **A Fast 90-Degree Bitmap Rotator**, *Graphics Gems II*, p. 84-85, code: p. 514-515, [rotate8x8.c](#).
- Holt, Jeff, **Rotation of Run-Length Encoded Image Data**, *Graphics Gems II*, p. 86-88, code: p. 516-524.
- Glassner, Andrew, **Adaptive Run-Length Encoding**, *Graphics Gems II*, p. 89-92.
- Paeth, Alan W., **Image File Compression Made Easy**, *Graphics Gems II*, p. 93-100.
- Max, Nelson L., **An Optimal Filter for Image Reconstruction**, *Graphics Gems II*, p. 101-104.
- Schlag, John, **Noise Thresholding in Edge Images**, *Graphics Gems II*, p. 105-106.
- Bieri, Hanspeter, and Kohler, Andreas, **Computing the Area, the Circumference, and the Genus of a Binary Digital Image**, *Graphics Gems II*, p. 107-111, code: p. 525-527.
- Möller, Tomas, **Fast Bitmap Stretching**, *Graphics Gems III*, p. 4-7, code: p. 411-413, [fastBitmap.c](#).
- Schumacher, Dale A., **General Filtered Image Rescaling**, *Graphics Gems III*, p. 8-16, code: p. 414-424, [filter.c](#) [filter\\_rcg.c](#).
- Schumacher, Dale A., **Optimization of Bitmap Scaling Operations**, *Graphics Gems III*, p. 17-19, code: p. 425-428, [bitmap.c](#).
- Bragg, Dennis, **A Simple Color Reduction Filter**, *Graphics Gems III*, p. 20-22, code: p. 429-431, [rgbvary.c](#) [rgbvaryW.c](#).
- Moore, Doug, and Warren, Joseph, **Compact Isocontours From Sampled Data**, *Graphics Gems III*, p. 23-28.
- Feldman, Tim, **Generating Iso-value Contours From a Pixmap**, *Graphics Gems III*, p. 29-33, code: p. 432-440, [contour.c](#).
- Salesin, David, and Barzel, Ronen, **Compositing Black-and-White Bitmaps**, *Graphics Gems III*, p. 34-35.
- Scofield, Cary, **2-1/2-d Depth-of-Field Simulation for Computer Animation**, *Graphics Gems III*, p.



36-38.

Furman, Eric, **A Fast Boundary Generator for Composited Regions**, *Graphics Gems III*, p. 39-43, code: p. 441-445, [scallops8.c](#).

Schlag, John, **Fast Embossing Effects on Raster Image Data**, *Graphics Gems IV*, p. 433-437, code: p. 435-436, [emboss.c](#).

Heckbert, Paul S., **Bilinear Coons Patch Image Warping**, *Graphics Gems IV*, p. 438-446, code: p. 441-444, [coons\\_warp.c](#).

Wolberg, George, and Massalin, Henry, **Fast Convolution with Packed Lookup Tables**, *Graphics Gems IV*, p. 447-464, code: p. 455-463, [convolve.c](#).

Cychosz, Joseph M., **Efficient Binary Image Thinning using Neighborhood Maps**, *Graphics Gems IV*, p. 465-473, code: p. 470-472, [thin\\_image.c](#).

Zuiderveld, Karel, **Contrast Limited Adaptive Histogram Equalization**, *Graphics Gems IV*, p. 474-485, code: p. 479-484, [clahe.c](#).

Paeth, Alan W., **Ideal Tiles for Shading and Halftoning**, *Graphics Gems IV*, p. 486-492.

Purgathofer, Werner, Tobler, Robert F., and Geiler, Manfred, **Improved Threshold Matrices for Ordered Dithering**, *Graphics Gems V*, p. 297-301.

Wong, Tien-tsin, and Hsu, Siu-chi, **Halftoning with Selective Precipitation and Adaptive Clustering**, *Graphics Gems V*, p. 302-313, code: p. 306-312, [ch6-2/](#).

Eker, Steven, **Faster Pixel-Perfect Line Clipping**, *Graphics Gems V*, p. 314-322, code: p. 319-322, [ch6-3/](#).

Doué, Jean-François, and Rubio, Ruben Gonzalez, **Efficient and Robust 2D Shape Vectorization**, *Graphics Gems V*, p. 323-337, code: p. 329-336, [ch6-4/](#).

Hsu, Siu-chi, and Lee, I.H.H., **Reversible Straight Line Edge Reconstruction**, *Graphics Gems V*, p. 338-354, code: p. 342-353, [ch6-5/](#).

Sharma, Rajesh, **Priority-based Adaptive Image Refinement**, *Graphics Gems V*, p. 355-358.

Cross, Robert A., **Sampling Patterns Optimized for Uniform Distribution of Edges**, *Graphics Gems V*, p. 359-363, code: p. 362, [ch6-7/](#).

## Matrix Techniques

Thompson, Kelvin, **Matrix Identities**, *Graphics Gems*, p. 453-454.

Thompson, Kelvin, **Transforming Axes**, *Graphics Gems*, p. 456-459.

Thompson, Kelvin, **Fast Matrix Multiplication**, *Graphics Gems*, p. 460-461.

Hultquist, Jeff, **A Virtual Trackball**, *Graphics Gems*, p. 462-463.

Raible, Eric, **Matrix Orthogonalization**, *Graphics Gems*, p. 464, code: p. 765, [MatrixOrtho.c](#).

Pique, Michael E., **Rotation Tools**, *Graphics Gems*, p. 465-469.

Carling, Richard, **Matrix Inversion**, *Graphics Gems*, p. 470-471, code: p. 766-769, [MatrixInvert.c](#).

Goldman, Ronald, **Matrices and Transformations**, *Graphics Gems*, p. 472-475.

Cychosz, Joseph M., **Efficient Post-Concatenation of Transformation Matrices**, *Graphics Gems*, p. 476-481, code: p. 770-772, [MatrixPost.c](#).

Thomas, Spencer W., **Decomposing a Matrix Into Simple Transformations**, *Graphics Gems II*, p. 320-323, code: p. 599-602, [unmatrix.c](#).

Goldman, Ronald, **Recovering the Data From the Transformation Matrix**, *Graphics Gems II*, p.



324-331.

Goldman, Ronald, **Transformations as Exponentials**, *Graphics Gems II*, p. 332-337.

Goldman, Ronald, **More Matrices and Transforms: Shear and Pseudo-Perspective**, *Graphics Gems II*, p. 338-341.

Wu, Kevin, **Fast Matrix Inversion**, *Graphics Gems II*, p. 342-350, code: p. 603-605, [inverse.c](#).

Shoemake, Ken, **Quaternions and 4x4 Matrices**, *Graphics Gems II*, p. 351-354.

Arvo, James, **Random Rotation Matrices**, *Graphics Gems II*, p. 355-356, code: p. 606-607, [rotate.c](#).

Arvo, James, **Classifying Small Sparse Matrices**, *Graphics Gems II*, p. 357-361, code: p. 608-609, [sparse.c](#).

Wu, Kevin, **Fast Inversion of Length- and Angle-Preserving Matrices**, *Graphics Gems IV*, p. 199-206, code: p. 204-206, [inv\\_fast.c](#).

Shoemake, Ken, **Polar Matrix Decomposition**, *Graphics Gems IV*, p. 207-221, code: p. 211-220, [polar\\_decomp/](#).

Shoemake, Ken, **Euler Angle Conversion**, *Graphics Gems IV*, p. 222-229, code: p. 225-228, [euler\\_angle/](#).

## Modeling and Transformations

Greene, Ned, **Transformation Identities**, *Graphics Gems*, p. 485-493.

Turkowski, Ken, **Fixed-Point Trigonometry with CORDIC Iterations**, *Graphics Gems*, p. 494-497, code: p. 773-774, [FixedTrig.c](#).

Maillot, Patrick-Gilles, **Using Quaternions for Coding 3D Transformations**, *Graphics Gems*, p. 498-515, code: p. 775-777, [Quaternions.c](#).

Cunningham, Steve, **3D Viewing and Rotation Using Orthonormal Bases**, *Graphics Gems*, p. 516-521, code: p. 778-779, [ViewTrans.c](#).

Turkowski, Ken, **The Use of Coordinate Frames in Computer Graphics**, *Graphics Gems*, p. 522-532.

Wallis, Bob, **Forms, Vectors, and Transforms**, *Graphics Gems*, p. 533-538, code: p. 780-784, [Forms.c](#).

Turkowski, Ken, **Properties of Surface-Normal Transformations**, *Graphics Gems*, p. 539-547.

Arvo, James, **Transforming Axis-Aligned Bounding Boxes**, *Graphics Gems*, p. 548-550, code: p. 785-786, [TransBox.c](#).

Hall, Mark, **Defining Surfaces From Sampled Data**, *Graphics Gems*, p. 552-557.

Hall, Mark, **Defining Surfaces From Contour Data**, *Graphics Gems*, p. 558-561.

Glassner, Andrew, **Computing Surface Normals for 3D Models**, *Graphics Gems*, p. 562-566.

Bloomenthal, Jules, **Calculation of Reference Frames Along a Space Curve**, *Graphics Gems*, p. 567-571.

Morrison, Jack C., **Quaternion Interpolation with Extra Spins**, *Graphics Gems III*, p. 96-97, code: p. 461-462, [quatspin.c](#).

Goldman, Ronald, **Decomposing Projective Transformations**, *Graphics Gems III*, p. 98-107.

Goldman, Ronald, **Decomposing Linear and Affine Transformations**, *Graphics Gems III*, p. 108-116.

Arvo, James, **Fast Random Rotation Matrices**, *Graphics Gems III*, p. 117-120, code: p. 463-464, [rand\\_rotation.c](#).

Dana, Paul, **Issues and Techniques for Keyframing Transformations**, *Graphics Gems III*, p. 121-123.

Shoemake, Ken, **Uniform Random Rotations**, *Graphics Gems III*, p. 124-132, code: p. 465-467, [urot.c](#).

- Elber, Gershon, **Interpolation Using Bézier Curves**, *Graphics Gems III*, p. 133-136, code: p. 468-471, [bzrintrp.c](#).
- Barr, A.H., **Physically Based Superquadrics**, *Graphics Gems III*, p. 137-159, code: p. 472-477, [sqfinal.c](#).
- Shoemake, Ken, **Arcball Rotation Control**, *Graphics Gems IV*, p. 175-192, code: p. 178-191, [arcball/](#).
- Cromwell, Robert L., **Efficient Eigenvalues for Visualization**, *Graphics Gems IV*, p. 193-198.
- Shoemake, Ken, **Fiber Bundle Twist Reduction**, *Graphics Gems IV*, p. 230-236.
- Alciatore, David, and Miranda, Rick, **The Best Least-Squares Line Fit**, *Graphics Gems V*, p. 91-97.
- Hill, Steve, and Roberts, Jonathan C., **Surface Models and the Resolution of N-Dimensional Cell Ambiguity**, *Graphics Gems V*, p. 98-106.
- Arata, Louis K., **Tri-cubic Interpolation**, *Graphics Gems V*, p. 107-110, code: p. 108-109, [ch3-3/](#).
- Miller, Robert D., **Transforming Coordinates From One Coordinate Plane To Another**, *Graphics Gems V*, p. 111-120, code: p. 115-120, [ch3-4/](#).
- Chin, Norman, **A Walk Through BSP Trees**, *Graphics Gems V*, p. 121-138, code: p. 131-138, [ch3-5/](#).
- Blanc, Carole, **Generic Implementation of Axial Deformation Techniques**, *Graphics Gems V*, p. 139-145, code: p. 141-144, [ch3-6/](#).

## Numerical and Programming Techniques

- Wyvill, Brian, **3D Grid Hashing Function**, *Graphics Gems*, p. 343-345, code: p. 733-734, [Hash3D.c](#).
- Schwarze, Jochen, **Cubic and Quartic Roots**, *Graphics Gems*, p. 404-407, code: p. 738-786, [Roots3And4.c](#).
- Schneider, Philip J., **A Bézier Curve-Based Root-Finder**, *Graphics Gems*, p. 408-415, code: p. 787, [NearestPoint.c](#).
- Hook, D.G., and McAree, P.R., **Using Sturm Sequences To Bracket Real Roots of Polynomial Equations**, *Graphics Gems*, p. 416-423, code: p. 743-755, [Sturm/](#).
- Lalonde, Paul, and Dawson, Robert, **A High-Speed, Low Precision Square Root**, *Graphics Gems*, p. 424-426, code: p. 756-757, [SquareRoot.c](#).
- Paeth, Alan W., **A Fast Approximation To the Hypotenuse**, *Graphics Gems*, p. 427-431, code: p. 758, [HypotApprox.c](#).
- Ritter, Jack, **A Fast Approximation To 3D Euclidian Distance**, *Graphics Gems*, p. 432-433.
- Thompson, Kelvin, **Full-Precision Constants**, *Graphics Gems*, p. 434.
- Thompson, Kelvin, **Converting Between Bits and Digits**, *Graphics Gems*, p. 435.
- Wyvill, Brian, **Storage-free Swapping**, *Graphics Gems*, p. 436-437.
- Glassner, Andrew, **Generating Random Integers**, *Graphics Gems*, p. 438-439.
- Ritter, Jack, **Fast 2D-3D Rotation**, *Graphics Gems*, p. 440-441.
- Shoemake, Ken, **Bit Patterns for Encoding Angles**, *Graphics Gems*, p. 442.
- Shaffer, Clifford A., **Bit Interleaving for Quad- or Octrees**, *Graphics Gems*, p. 443-447, code: p. 759-762, [Interleave.c](#).
- Fishkin, Ken, **A Fast HSL-to-RGB Transform**, *Graphics Gems*, p. 448-449, code: p. 763-764, [HSLtoRGB.c](#).
- Shoemake, Ken, **Bit Picking**, *Graphics Gems II*, p. 366-367.
- Shoemake, Ken, **Faster Fourier Transform**, *Graphics Gems II*, p. 368-370.

- Paeth, Alan W., and Schilling, David, **Of Integers, Fields, and Bit Counting**, *Graphics Gems II*, p. 371-376, code: p. 610-611, [BitCounting/](#).
- Schlag, John, **Using Geometric Constructions to Interpolate Orientation with Quaternions**, *Graphics Gems II*, p. 377-380.
- Paeth, Alan W., **A Half-Angle Identity for Digital Computation: The Joys of the Halved Tangent**, *Graphics Gems II*, p. 381-386.
- Musial, Christopher J., **An Integer Square Root Algorithm**, *Graphics Gems II*, p. 387-388, code: p. 612.
- Capelli, Ron, **Fast Approximation To the Arctangent**, *Graphics Gems II*, p. 389-391.
- Ritter, Jack, **Fast Sign of Cross Product Calculation**, *Graphics Gems II*, p. 392-393, code: p. 613-614.
- Shoemake, Ken, **Interval Sampling**, *Graphics Gems II*, p. 394-395.
- Ward, Greg, **A Recursive Implementation of the Perlin Noise Function**, *Graphics Gems II*, p. 396-401, code: p. 615-616, [noise3.c](#).
- Hill, Steve, **IEEE Fast Square Root**, *Graphics Gems III*, p. 48, code: p. 446-447, [sqrt.c](#).
- Hill, Steve, **A Simple Fast Memory Allocator**, *Graphics Gems III*, p. 49-50, code: p. 448-451, [alloc/](#).
- Hanson, Andrew J., **The Rolling Ball**, *Graphics Gems III*, p. 51-60, code: p. 452-453, [3d.c defs.h](#).
- Rokne, Jon, **Interval Arithmetic**, *Graphics Gems III*, p. 61-66, code: p. 454-457, [interval.C](#).
- Paeth, Alan W., **Fast Generation of Cyclic Sequences**, *Graphics Gems III*, p. 67-76, code: p. 458-459, [cyclic.c](#).
- Paeth, Alan W., **A Generic Pixel Selection Mechanism**, *Graphics Gems III*, p. 77-79.
- Shirley, Peter, **Nonuniform Random Point Sets**, *Graphics Gems III*, p. 80-83.
- Goldman, Ronald, **Cross Product in Four Dimensions and Beyond**, *Graphics Gems III*, p. 84-88.
- Badouel, Didier, and Wuthrich, Charles A., **Face-Connected Line Segment Generation in an n-Dimensional Space**, *Graphics Gems III*, p. 89-91, code: p. 460, [ndline.c](#).
- Donovan, Walt, and Van Hook, Tim, **Direct Outcode Calculation for Faster Clip Testing**, *Graphics Gems IV*, p. 125-131, code: p. 127-131, [outcode/](#).
- Hill, Steve, **Tri-linear Interpolation**, *Graphics Gems IV*, p. 521-525, code: p. 523-524, [trilerp.c](#).
- Eker, Steven, **Faster Linear Interpolation**, *Graphics Gems IV*, p. 526-533, code: p. 532-533, [interp\\_fast.c](#).
- Herbison-Evans, Don, **Solving Quartics and Cubics for Graphics**, *Graphics Gems V*, p. 3-15, code: [ch1-1/](#).
- Turkowski, Ken, **Computing the Inverse Square Root**, *Graphics Gems V*, p. 16-21, code: p. 17-19, [ch1-2/](#).
- Turkowski, Ken, **Fixed Point Square Root**, *Graphics Gems V*, p. 22-24, code: p. 23, [ch1-3/](#).
- Shoemake, Ken, **Rational Approximation**, *Graphics Gems V*, p. 25-32, code: p. 29-31, [ch1-4/](#).
- Schlick, Christophe, **Wave Generators for Computer Graphics**, *Graphics Gems V*, p. 367-374, code: p. 371-374, [ch7-1/](#).

## Radiosity

- Chen, Shenchang Eric, **Implementing Progressive Radiosity with User-Provided Polygon Display Routines**, *Graphics Gems II*, p. 295-298, code: p. 583-597, [radiosity/](#).
- Beran-Koehn, Jeffrey C., and Pavicic, Mark J., **A Cubic Tetrahedral Adaptation of the Hemi-Cube**

**Algorithm**, *Graphics Gems II*, p. 299-302.

Tampieri, Filippo, **Fast Vertex Radiosity Update**, *Graphics Gems II*, p. 303-305, code: p. 598, [FastUpdate.c](#).

Shirley, Peter, **Radiosity via Ray Tracing**, *Graphics Gems II*, p. 306-310.

Sillion, François, **Detection of Shadow Boundaries for Adaptive Meshing in Radiosity**, *Graphics Gems II*, p. 311-315.

Wang, Changyaw, **Physically Correct Direct Lighting for Distribution Ray Tracing**, *Graphics Gems III*, p. 307-313, code: p. 562-568, [luminaire/](#).

Bian, Bumng, **Hemispherical Projection of a Triangle**, *Graphics Gems III*, p. 314-317, code: p. 569-574, [hemis.c](#).

Max, Nelson L., and Allison, Michael J., **Linear Radiosity Approximation Using Vertex-to-Vertex Form Factors**, *Graphics Gems III*, p. 318-323.

Beran-Koehn, Jeffrey C., and Pavicic, Mark J., **Delta Form-Factor Calculation for the Cubic Tetrahedral Algorithm**, *Graphics Gems III*, p. 324-328, code: p. 575-576, [forfac.c](#).

Tampieri, Filippo, **Accurate Form-Factor Computation**, *Graphics Gems III*, p. 329-333, code: p. 577-581, [accForm.c](#).

## Ray Tracing

Ritter, Jack, **A Simple Ray Rejection Test**, *Graphics Gems*, p. 385-386.

Hultquist, Jeff, **Intersection of a Ray with a Sphere**, *Graphics Gems*, p. 388-389.

Badouel, Didier, **An Efficient Ray-Polygon Intersection**, *Graphics Gems*, p. 390-393, code: p. 735, [RayPolygon.c](#).

Woo, Andrew, **Fast Ray-Polygon Intersection**, *Graphics Gems*, p. 394.

Woo, Andrew, **Fast Ray-Box Intersection**, *Graphics Gems*, p. 395-396, code: p. 736-737, [RayBox.c](#).

Pearce, Andrew, **Shadow Attenuation for Ray Tracing Transparent Objects**, *Graphics Gems*, p. 397-399.

Haines, Eric, **Fast Ray-Convex Polyhedron Intersection**, *Graphics Gems II*, p. 247-250, code: p. 575-576, [RayCPhdron.c](#).

Cychosz, Joseph M., **Intersecting a Ray with An Elliptical Torus**, *Graphics Gems II*, p. 251-256, code: p. 577-580, [intersect/inttor.c](#).

Voorhies, Douglas, and Kirk, David, **Ray-Triangle Intersection Using Binary Recursive Subdivision**, *Graphics Gems II*, p. 257-263.

Kirk, David, and Arvo, James, **Improved Ray Tagging for Voxel-Based Ray Tracing**, *Graphics Gems II*, p. 264-266.

Haines, Eric, **Efficiency Improvements for Hierarchy Traversal in Ray Tracing**, *Graphics Gems II*, p. 267-272.

Pearce, Andrew, **A Recursive Shadow Voxel Cache for Ray Tracing**, *Graphics Gems II*, p. 273-274, code: p. 581-582, [VoxelCache.c](#).

Pearce, Andrew, **Avoiding Incorrect Shadow Intersections for Ray Tracing**, *Graphics Gems II*, p. 275-276.

Lee, Mark E., and Uselton, Samuel P., **A Body Color Model: Absorption Through Translucent Media**, *Graphics Gems II*, p. 277-282.

Lee, Mark E., and Uselton, Samuel P., **More Shadow Attenuation for Ray Tracing Transparent or**



**Translucent Objects**, *Graphics Gems II*, p. 283-289.

Sung, Kelvin, and Shirley, Peter, **Ray Tracing with the BSP Tree**, *Graphics Gems III*, p. 271-274, code: p. 538-546, [bsp.c](#).

Cychosz, Joseph M., and Waggenspack, Warren N., Jr., **Intersecting a Ray with a Quadric Surface**, *Graphics Gems III*, p. 275-283, code: p. 547-550, [intqdr.c](#) [intell.c](#).

Cychosz, Joseph M., **Use of Residency Masks and Object Space Partitioning to Eliminate Ray-Object Intersection Calculations**, *Graphics Gems III*, p. 284-287.

Musgrave, F. Kenton, **A Panoramic Virtual Screen for Ray Tracing**, *Graphics Gems III*, p. 288-294, code: p. 551-554, [panorama.c](#).

Trumbore, Ben, **Rectangular Bounding Volumes for Popular Primitives**, *Graphics Gems III*, p. 295-300, code: p. 555-561, [bounding\\_volumes.c](#).

Wu, Xiaolin, **A Linear-Time Simple Bounding Volume Algorithm**, *Graphics Gems III*, p. 301-306.

Wang, Changyaw, **Physically Correct Direct Lighting for Distribution Ray Tracing**, *Graphics Gems III*, p. 307-313, code: p. 562-568, [luminaire/](#).

Shene, Ching-Kuang, **Computing the Intersection of a Line and a Cylinder**, *Graphics Gems IV*, p. 353-355.

Cychosz, Joseph M., and Waggenspack, Warren N., Jr., **Intersecting a Ray with a Cylinder**, *Graphics Gems IV*, p. 356-365, code: p. 361-364, [ray\\_cyl.c](#).

Cohen, Daniel, **Voxel Traversal along a 3D Line**, *Graphics Gems IV*, p. 366-369, code: p. 368, [vox\\_traverse.c](#).

Chiu, Kenneth, Shirley, Peter, and Wang, Changyaw, **Multi-Jittered Sampling**, *Graphics Gems IV*, p. 370-374, code: p. 373-374, [multi\\_jitter/](#).

Heckbert, Paul S., **A Minimal Ray Tracer**, *Graphics Gems IV*, p. 375-381, code: p. 378-380, [minray/](#).

Shene, Ching-Kuang, **Computing the Intersection of a Line and a Cone**, *Graphics Gems V*, p. 227-231.

Schlick, Christophe, and Subrenat, Gilles, **Ray Intersection of Tessellated Surfaces: Quad Vs Triangle**, *Graphics Gems V*, p. 232-241, code: p. 237-240, [ch5-2/](#).

Möller, Tomas, **Faster Ray Tracing Using Scanline Rejection**, *Graphics Gems V*, p. 242-257, code: p. 249-257, [ch5-3/](#).

Leipelt, Andreas, **Ray Tracing a Swept Sphere**, *Graphics Gems V*, p. 258-267, code: p. 261-267, [ch5-4/](#).

Márton, Gábor, **Acceleration of Ray Tracing via Voronoi-diagrams**, *Graphics Gems V*, p. 268-284, code: p. 276-283, [ch5-5/](#).

Zimmerman, Kurt, **Direct Lighting Models for Ray Tracing with Cylindrical Lamps**, *Graphics Gems V*, p. 285-289.

Feda, Martin, **Improving Intermediate Radiosity Using Directional Light**, *Graphics Gems V*, p. 290-293.

---

Last change: July 6, 2000 [Eric Haines](#), Gems archivist / [erich@acm.org](mailto:erich@acm.org)



# Graphics Gems, by Author

Listed by author, in alphabetic order.

[A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [G](#) [H](#) [I](#) [J](#) [K](#) [L](#) [M](#) [N](#) [O](#) [P](#) [R](#) [S](#) [T](#) [U](#) [V](#) [W](#) [X](#) [Y](#) [Z](#)

[return to main page](#)

---

## Ahn, Jaewoo

Ahn, Jaewoo, **Fast Generation of Ellipsoids**, *Graphics Gems V*, p. 179-190, code: p. 185-190, [ch4-5/](#).

## Alciatore, David

Alciatore, David, and Miranda, Rick, **The Best Least-Squares Line Fit**, *Graphics Gems V*, p. 91-97.

## Allison, Michael J.

Max, Nelson L., and Allison, Michael J., **Linear Radiosity Approximation Using Vertex-to-Vertex Form Factors**, *Graphics Gems III*, p. 318-323.

## Antonio, Franklin

Antonio, Franklin, **Faster Line Segment Intersection**, *Graphics Gems III*, p. 199-202, code: p. 500-501, [insectc.c](#).

## Arata, Louis K.

Arata, Louis K., **Tri-cubic Interpolation**, *Graphics Gems V*, p. 107-110, code: p. 108-109, [ch3-3/](#).

## Arvo, James

Arvo, James, **A Simple Method for Box-Sphere Intersection Testing**, *Graphics Gems*, p. 335-339, code: p. 730-732, [BoxSphere.c](#).

Arvo, James, **Transforming Axis-Aligned Bounding Boxes**, *Graphics Gems*, p. 548-550, code: p. 785-786, [TransBox.c](#).

Kirk, David, and Arvo, James, **Improved Ray Tagging for Voxel-Based Ray Tracing**, *Graphics Gems II*, p. 264-266.

Arvo, James, **Random Rotation Matrices**, *Graphics Gems II*, p. 355-356, code: p. 606-607, [rotate.c](#).

Arvo, James, **Classifying Small Sparse Matrices**, *Graphics Gems II*, p. 357-361, code: p. 608-609, [sparse.c](#).

Arvo, James, **Fast Random Rotation Matrices**, *Graphics Gems III*, p. 117-120, code: p. 463-464, [rand\\_rotation.c](#).

Arvo, James, and Scofield, Cary, **The Shader Cache: A Rendering Pipeline Accelerator**, *Graphics Gems III*, p. 383-389.

## Badouel, Didier

Badouel, Didier, **An Efficient Ray-Polygon Intersection**, *Graphics Gems*, p. 390-393, code: p. 735, [RayPolygon.c](#).

Badouel, Didier, and Wuthrich, Charles A., **Face-Connected Line Segment Generation in an n-Dimensional Space**, *Graphics Gems III*, p. 89-91, code: p. 460, [ndline.c](#).

**Bajaj, Chandrajit**

Bajaj, Chandrajit, and Xu, Guoliang, **Converting a Rational Curve to a Standard Rational Bernstein-Bézier Representation**, *Graphics Gems IV*, p. 256-260.

Bajaj, Chandrajit, and Xu, Guoliang, **Sparse Smooth Connection Between Bézier/B-Spline Curves**, *Graphics Gems V*, p. 191-198.

**Bame, Paul D.**

Bame, Paul D., **Albers Equal-Area Conic Map Projection**, *Graphics Gems*, p. 321-325, code: p. 726-729, [Albers.c](#).

**Barr, A.H.**

Barr, A.H., **Physically Based Superquadrics**, *Graphics Gems III*, p. 137-159, code: p. 472-477, [sqfinal.c](#).

**Barry, Phillip**

Barry, Phillip, and Goldman, Ronald, **Knot Insertion using Forward Differences**, *Graphics Gems IV*, p. 251-255.

**Barzel, Ronen**

Salesin, David, and Barzel, Ronen, **Compositing Black-and-White Bitmaps**, *Graphics Gems III*, p. 34-35.

Snyder, John, Barzel, Ronen, and Gabriel, Steve, **Motion Blur on Graphics Workstations**, *Graphics Gems III*, p. 374-382, code: p. 606-609, [motblur.c](#).

**Bashein, Gerard**

Bashein, Gerard, and Detmer, Paul R., **Centroid of a Polygon**, *Graphics Gems IV*, p. 3-6, code: p. 5, [centroid.c](#).

**Behrens, Uwe**

Behrens, Uwe, **Fence Shading**, *Graphics Gems IV*, p. 404-409.

**Beran-Koehn, Jeffrey C.**

Beran-Koehn, Jeffrey C., and Pavicic, Mark J., **A Cubic Tetrahedral Adaptation of the Hemi-Cube Algorithm**, *Graphics Gems II*, p. 299-302.

Beran-Koehn, Jeffrey C., and Pavicic, Mark J., **Delta Form-Factor Calculation for the Cubic Tetrahedral Algorithm**, *Graphics Gems III*, p. 324-328, code: p. 575-576, [forfac.c](#).

**Bian, Buming**

Bian, Buming, **Hemispherical Projection of a Triangle**, *Graphics Gems III*, p. 314-317, code: p. 569-574, [hemis.c](#).

**Bieri, Hanspeter**

Bieri, Hanspeter, and Kohler, Andreas, **Computing the Area, the Circumference, and the Genus of a Binary Digital Image**, *Graphics Gems II*, p. 107-111, code: p. 525-527.

**Blanc, Carole**

Blanc, Carole, **Generic Implementation of Axial Deformation Techniques**, *Graphics Gems V*, p. 139-145, code: p. 141-144, [ch3-6/](#).

### **Bloomenthal, Jules**

Bloomenthal, Jules, **Calculation of Reference Frames Along a Space Curve**, *Graphics Gems*, p. 567-571.

Bloomenthal, Jules, **An Implicit Surface Polygonizer**, *Graphics Gems IV*, p. 324-349, code: p. 334-349, [implicit.c](#).

### **Bogart, Rod G.**

Thomas, Spencer W., and Bogart, Rod G., **Color Dithering**, *Graphics Gems II*, p. 72-77, code: p. 509-513, [dither/](#).

Bogart, Rod G., **View Correlation**, *Graphics Gems II*, p. 181-190, code: p. 550-562, [viewcorr/](#).

Glassner, Andrew, and Bogart, Rod G., **2D and 3D Vector C Library**, *Graphics Gems II*, p. 458-466, code: p. 633-642, [GGVecLib.c](#) [GraphicsGems.h](#).

### **Bouma, William**

Bouma, William, and Vanecek, George, Jr., **Velocity-based Collision Detection**, *Graphics Gems V*, p. 380-385, code: p. 383-385, [ch7-3/](#).

### **Bragg, Dennis**

Bragg, Dennis, **A Simple Color Reduction Filter**, *Graphics Gems III*, p. 20-22, code: p. 429-431, [rgbvary.c](#) [rgbvaryW.c](#).

### **Buckley, Robert**

Buckley, Robert, **Parallelhedra and Uniform Color Quantization**, *Graphics Gems V*, p. 65-71.

### **Capelli, Ron**

Capelli, Ron, **Fast Approximation To the Arctangent**, *Graphics Gems II*, p. 389-391.

### **Carling, Richard**

Carling, Richard, **Matrix Inversion**, *Graphics Gems*, p. 470-471, code: p. 766-769, [MatrixInvert.c](#).

### **Carvalho, Paulo Cezar Pinto**

Carvalho, Paulo Cezar Pinto, and Cavalcanti, Paulo Roma, **Point in Polyhedron Testing Using Spherical Polygons**, *Graphics Gems V*, p. 42-49, code: p. 46-49, [ch2-2/](#).

### **Cavalcanti, Paulo Roma**

Carvalho, Paulo Cezar Pinto, and Cavalcanti, Paulo Roma, **Point in Polyhedron Testing Using Spherical Polygons**, *Graphics Gems V*, p. 42-49, code: p. 46-49, [ch2-2/](#).

### **Chen, Shenchang Eric**

Chen, Shenchang Eric, **Implementing Progressive Radiosity with User-Provided Polygon Display Routines**, *Graphics Gems II*, p. 295-298, code: p. 583-597, [radiosity/](#).

### **Cheng, Russell C.H.**

Cheng, Russell C.H., **Fast Linear Color Rendering**, *Graphics Gems III*, p. 343-348, code: p. 583-585, [fastLinear.c](#).

Cheng, Russell C.H., **Edge and Bit-Mask Calculations for Anti-Aliasing**, *Graphics Gems III*, p. 349-354, code: p. 586-593, [edgeCalc.c](#).

### **Chin, Norman**

Chin, Norman, **Partitioning a 3-D Convex Polygon with an Arbitrary Plane**, *Graphics Gems III*, p. 219-222, code: p. 502-510, [partition3d/](#).

Chin, Norman, **A Walk Through BSP Trees**, *Graphics Gems V*, p. 121-138, code: p. 131-138, [ch3-5/](#).

### **Chiu, Kenneth**

Chiu, Kenneth, Shirley, Peter, and Wang, Changyaw, **Multi-Jittered Sampling**, *Graphics Gems IV*, p. 370-374, code: p. 373-374, [multi\\_jitter/](#).

### **Christensen, Jon**

Christensen, Jon, Marks, Joe, and Shieber, Stuart, **Placing Text Labels on Maps and Diagrams**, *Graphics Gems IV*, p. 497-504.

### **Cohen, Daniel**

Cohen, Daniel, **Voxel Traversal along a 3D Line**, *Graphics Gems IV*, p. 366-369, code: p. 368, [vox\\_traverse.c](#).

### **Cromwell, Robert L.**

Cromwell, Robert L., **Efficient Eigenvalues for Visualization**, *Graphics Gems IV*, p. 193-198.

### **Cross, Robert A.**

Cross, Robert A., **Sampling Patterns Optimized for Uniform Distribution of Edges**, *Graphics Gems V*, p. 359-363, code: p. 362, [ch6-7/](#).

### **Cunningham, Steve**

Cunningham, Steve, **3D Viewing and Rotation Using Orthonormal Bases**, *Graphics Gems*, p. 516-521, code: p. 778-779, [ViewTrans.c](#).

### **Cychosz, Joseph M.**

Cychosz, Joseph M., **Efficient Generation of Sampling Jitter Using Look-up Tables**, *Graphics Gems*, p. 64-74, code: p. 660-661, [FastJitter.c](#).

Cychosz, Joseph M., **Efficient Post-Concatenation of Transformation Matrices**, *Graphics Gems*, p. 476-481, code: p. 770-772, [MatrixPost.c](#).

Cychosz, Joseph M., **Intersecting a Ray with An Elliptical Torus**, *Graphics Gems II*, p. 251-256, code: p. 577-580, [intersect/inttor.c](#).

Cychosz, Joseph M., and Waggenspack, Warren N., Jr., **Intersecting a Ray with a Quadric Surface**, *Graphics Gems III*, p. 275-283, code: p. 547-550, [intqdr.c](#) [intell.c](#).

Cychosz, Joseph M., **Use of Residency Masks and Object Space Partitioning to Eliminate Ray-Object Intersection Calculations**, *Graphics Gems III*, p. 284-287.

Cychosz, Joseph M., and Waggenspack, Warren N., Jr., **Intersecting a Ray with a Cylinder**, *Graphics Gems IV*, p. 356-365, code: p. 361-364, [ray\\_cyl.c](#).

Cychosz, Joseph M., **Efficient Binary Image Thinning using Neighborhood Maps**, *Graphics Gems IV*, p. 465-473, code: p. 470-472, [thin\\_image.c](#).

### **Dana, Paul**

Dana, Paul, **Issues and Techniques for Keyframing Transformations**, *Graphics Gems III*, p. 121-123.

**Dawson, Robert**

Lalonde, Paul, and Dawson, Robert, **A High-Speed, Low Precision Square Root**, *Graphics Gems*, p. 424-426, code: p. 756-757, [SquareRoot.c](#).

**de Figueiredo, Luiz Henrique**

de Figueiredo, Luiz Henrique, **Adaptive Sampling of Parametric Curves**, *Graphics Gems V*, p. 173-178, code: p. 177, [ch4-4/](#).

**Detmer, Paul R.**

Bashein, Gerard, and Detmer, Paul R., **Centroid of a Polygon**, *Graphics Gems IV*, p. 3-6, code: p. 5, [centroid.c](#).

**Donovan, Walt**

Donovan, Walt, and Van Hook, Tim, **Direct Outcode Calculation for Faster Clip Testing**, *Graphics Gems IV*, p. 125-131, code: p. 127-131, [outcode/](#).

**Doué, Jean-François**

Doué, Jean-François, **C++ Vector and Matrix Algebra Routines**, *Graphics Gems IV*, p. 534-557, code: p. 535-557, [vec\\_mat/](#).

Doué, Jean-François, and Rubio, Ruben Gonzalez, **Efficient and Robust 2D Shape Vectorization**, *Graphics Gems V*, p. 323-337, code: p. 329-336, [ch6-4/](#).

**Eilers, Paul H. C.**

Eilers, Paul H. C., **Smoothing and Interpolation with Finite Differences**, *Graphics Gems IV*, p. 241-250, code: p. 246-249, [data\\_smooth/](#).

**Eker, Steven**

Eker, Steven, **Faster Linear Interpolation**, *Graphics Gems IV*, p. 526-533, code: p. 532-533, [interp\\_fast.c](#).

Eker, Steven, **Faster Pixel-Perfect Line Clipping**, *Graphics Gems V*, p. 314-322, code: p. 319-322, [ch6-3/](#).

**Elber, Gershon**

Elber, Gershon, **Interpolation Using Bézier Curves**, *Graphics Gems III*, p. 133-136, code: p. 468-471, [bzrintrp.c](#).

**Feda, Martin**

Feda, Martin, **Improving Intermediate Radiosity Using Directional Light**, *Graphics Gems V*, p. 290-293.

**Feldman, Tim**

Feldman, Tim, **Generating Iso-value Contours From a Pixmap**, *Graphics Gems III*, p. 29-33, code: p. 432-440, [contour.c](#).

**Feustel, Charles D.**

Shaffer, Clifford A., and Feustel, Charles D., **Exact Computation of 2-D Intersections**, *Graphics Gems III*, p. 188-192, code: p. 491-495, [Polyintr.c](#).



### **Fisher, Frederick**

Schorn, Peter, and Fisher, Frederick, **Testing the Convexity of a Polygon**, *Graphics Gems IV*, p. 7-15, code: p. 11-15, [convex\\_test/](#).

Fisher, Frederick, and Woo, Andrew, **R.E versus N.H Specular Highlights**, *Graphics Gems IV*, p. 388-400.

### **Fishkin, Ken**

Fishkin, Ken, **Filling a Region in a Frame Buffer**, *Graphics Gems*, p. 278-284.

Fishkin, Ken, **A Fast HSL-to-RGB Transform**, *Graphics Gems*, p. 448-449, code: p. 763-764, [HSLtoRGB.c](#).

### **Fleischer, Kurt**

Fleischer, Kurt, and Salesin, David, **Accurate Polygon Scan Conversion Using Half-Open Intervals**, *Graphics Gems III*, p. 362-365, code: p. 599-605, [accurate\\_scan/](#).

### **Furman, Eric**

Furman, Eric, **A Fast Boundary Generator for Compositing Regions**, *Graphics Gems III*, p. 39-43, code: p. 441-445, [scallop8.c](#).

### **Fusco, Mike**

Wanger, Len, and Fusco, Mike, **Fast N-Dimensional Extent Overlap Testing**, *Graphics Gems III*, p. 240-243, code: p. 527-533, [exttest/](#).

### **Gabriel, Steve**

Snyder, John, Barzel, Ronen, and Gabriel, Steve, **Motion Blur on Graphics Workstations**, *Graphics Gems III*, p. 374-382, code: p. 606-609, [motblur.c](#).

### **Geiler, Manfred**

Purgathofer, Werner, Tobler, Robert F., and Geiler, Manfred, **Improved Threshold Matrices for Ordered Dithering**, *Graphics Gems V*, p. 297-301.

### **Georgiades, Príamos**

Georgiades, Príamos, **Signed Distance From Point To Plane**, *Graphics Gems III*, p. 223-224, code: p. 511, [pt2plane.c](#).

Georgiades, Príamos, **Plane-to-Plane Intersection**, *Graphics Gems III*, p. 233-235, code: p. 519-520, [pl2plane.c](#).

### **Gervautz, Michael**

Gervautz, Michael, and Purgathofer, Werner, **A Simple Method for Color Quantization: Octree Quantization**, *Graphics Gems*, p. 287-293.

Kopp, Manfred, and Gervautz, Michael, **XOR-Drawing with Guaranteed Contrast**, *Graphics Gems IV*, p. 413-414.

### **Glassner, Andrew**

Glassner, Andrew, **Useful 2D Geometry**, *Graphics Gems*, p. 3-11.

Glassner, Andrew, **Useful Trigonometry**, *Graphics Gems*, p. 13-17.

Glassner, Andrew, **Frame Buffers and Color Maps**, *Graphics Gems*, p. 215-218.

Glassner, Andrew, **Normal Coding**, *Graphics Gems*, p. 257-264.

- Glassner, Andrew, **Useful 3D Geometry**, *Graphics Gems*, p. 297-300.
- Glassner, Andrew, **Simulating Fog and Haze**, *Graphics Gems*, p. 364-365.
- Glassner, Andrew, **Interpretation of Texture Map Indices**, *Graphics Gems*, p. 366-375.
- Glassner, Andrew, **Multidimensional Sum Tables**, *Graphics Gems*, p. 376-381.
- Glassner, Andrew, **Generating Random Integers**, *Graphics Gems*, p. 438-439.
- Glassner, Andrew, **Computing Surface Normals for 3D Models**, *Graphics Gems*, p. 562-566.
- Glassner, Andrew, **Planar Cubic Curves**, *Graphics Gems*, p. 575-578.
- Glassner, Andrew, **Graphics Gems Header File**, *Graphics Gems*, p. 629-632, code: p. 629-632, [GraphicsGems.h](http://GraphicsGems.h).
- Glassner, Andrew, **2D and 3D Vector C Library**, *Graphics Gems*, p. 633-642, code: p. 633-642, [GGVecLib.c](http://GGVecLib.c) [GraphicsGems.h](http://GraphicsGems.h).
- Glassner, Andrew, **Adaptive Run-Length Encoding**, *Graphics Gems II*, p. 89-92.
- Glassner, Andrew, **A Simple Viewing Geometry**, *Graphics Gems II*, p. 179-180.
- Glassner, Andrew, **Maintaining Winged-Edge Models**, *Graphics Gems II*, p. 191-201.
- Glassner, Andrew, **Graphics Gems Header File**, *Graphics Gems II*, p. 455-457, code: p. 629-632, [GraphicsGems.h](http://GraphicsGems.h).
- Glassner, Andrew, and Bogart, Rod G., **2D and 3D Vector C Library**, *Graphics Gems II*, p. 458-466, code: p. 633-642, [GGVecLib.c](http://GGVecLib.c) [GraphicsGems.h](http://GraphicsGems.h).
- Glassner, Andrew, **Darklights**, *Graphics Gems III*, p. 366-368.
- Glassner, Andrew, **Anti-Aliasing in Triangular Pixels**, *Graphics Gems III*, p. 369-373.
- Glassner, Andrew, **Graphics Gems Header File**, *Graphics Gems III*, p. 455-457, code: p. 393-395, [GraphicsGems.h](http://GraphicsGems.h).
- Glassner, Andrew, **2-D and 3-D Vector C Library -- Corrected and Indexed**, *Graphics Gems III*, p. 396-404, code: p. 396-404, [GraphicsGems.c](http://GraphicsGems.c).
- Glassner, Andrew, **Building Vertex Normals from an Unstructured Polygon List**, *Graphics Gems IV*, p. 60-73, code: p. 64-73, [vert\\_norm/](http://vert_norm/).
- Glassner, Andrew, and Haines, Eric, **C Header File and Vector Library**, *Graphics Gems IV*, p. 558-570, code: p. 558-570, [GraphicsGems.c](http://GraphicsGems.c) [GraphicsGems.h](http://GraphicsGems.h).
- Glassner, Andrew, **Clipping a Concave Polygon**, *Graphics Gems V*, p. 50-54.

## Goldman, Ronald

- Goldman, Ronald, **Triangles**, *Graphics Gems*, p. 20-23.
- Goldman, Ronald, **Intersection of Two Lines in Three-Space**, *Graphics Gems*, p. 304.
- Goldman, Ronald, **Intersection of Three Planes**, *Graphics Gems*, p. 305.
- Goldman, Ronald, **Matrices and Transformations**, *Graphics Gems*, p. 472-475.
- Goldman, Ronald, **Some Properties of Bézier Curves**, *Graphics Gems*, p. 587-593.
- Goldman, Ronald, **Integration of Bernstein Basis Functions**, *Graphics Gems*, p. 604-606.
- Goldman, Ronald, **Area of Planar Polygons and Volume of Polyhedra**, *Graphics Gems II*, p. 170-171.
- Goldman, Ronald, **Recovering the Data From the Transformation Matrix**, *Graphics Gems II*, p. 324-331.
- Goldman, Ronald, **Transformations as Exponentials**, *Graphics Gems II*, p. 332-337.
- Goldman, Ronald, **More Matrices and Transforms: Shear and Pseudo-Perspective**, *Graphics Gems II*, p. 338-341.
- Goldman, Ronald, **Cross Product in Four Dimensions and Beyond**, *Graphics Gems III*, p. 84-88.
- Goldman, Ronald, **Decomposing Projective Transformations**, *Graphics Gems III*, p. 98-107.

- Goldman, Ronald, **Decomposing Linear and Affine Transformations**, *Graphics Gems III*, p. 108-116.
- Barry, Phillip, and Goldman, Ronald, **Knot Insertion using Forward Differences**, *Graphics Gems IV*, p. 251-255.
- Goldman, Ronald, **Identities for the Univariate, Bivariate Bernstein Basis Fcns**, *Graphics Gems V*, p. 149-162.
- Goldman, Ronald, **Identities for the B-Spline Basis Functions**, *Graphics Gems V*, p. 163-167.

### **Gomez, Julian**

- Gomez, Julian, **Fast Spline Drawing**, *Graphics Gems*, p. 585-586.

### **Grace, Thom**

- Grace, Thom, **Fast Span Conversion: Unrolling Short Loops**, *Graphics Gems III*, p. 355-357, code: p. 594-596, [fastSpan.c](#).

### **Gravesen, Jens**

- Gravesen, Jens, **The Length of Bézier Curves**, *Graphics Gems V*, p. 199-205, code: [ch4-7/](#).

### **Green, Daniel**

- Green, Daniel, and Hatch, Don, **Fast Polygon-Cube Intersection Testing**, *Graphics Gems V*, p. 375-379, code: [ch7-2/](#).

### **Greene, Ned**

- Greene, Ned, **Transformation Identities**, *Graphics Gems*, p. 485-493.
- Greene, Ned, **Detecting Intersection of a Rectangular Solid and a Convex Polyhedron**, *Graphics Gems IV*, p. 74-82.

### **Haines, Eric**

- Haines, Eric, **Fast Ray-Convex Polyhedron Intersection**, *Graphics Gems II*, p. 247-250, code: p. 575-576, [RayCPhdron.c](#).
- Haines, Eric, **Efficiency Improvements for Hierarchy Traversal in Ray Tracing**, *Graphics Gems II*, p. 267-272.
- Haines, Eric, **Point in Polygon Strategies**, *Graphics Gems IV*, p. 24-46, code: p. 34-45, [ptpoly\\_haines/](#).
- Glassner, Andrew, and Haines, Eric, **C Header File and Vector Library**, *Graphics Gems IV*, p. 558-570, code: p. 558-570, [GraphicsGems.c](#) [GraphicsGems.h](#).

### **Hall, Jim**

- Hall, Jim, and Lindgren, Terence, **A Fast Approach To PHIGS PLUS Pseudo Color**, *Graphics Gems II*, p. 138-142.
- Lindgren, Terence, Sanchez, Juan, and Hall, Jim, **Curve Tessellation Criteria Thru Sampling**, *Graphics Gems III*, p. 262-265.

### **Hall, Mark**

- Hall, Mark, **Defining Surfaces From Sampled Data**, *Graphics Gems*, p. 552-557.
- Hall, Mark, **Defining Surfaces From Contour Data**, *Graphics Gems*, p. 558-561.

### **Hanson, Andrew J.**

- Hanson, Andrew J., **The Rolling Ball**, *Graphics Gems III*, p. 51-60, code: p. 452-453, [3d.c](#) [defs.h](#).
- Hanson, Andrew J., **Geometry for N-Dimensional Graphics**, *Graphics Gems IV*, p. 149-170.

Hanson, Andrew J., **Rotations for N-dimensional Graphics**, *Graphics Gems V*, p. 55-64.

**Hart, John C.**

Hart, John C., **Distance to an Ellipsoid**, *Graphics Gems IV*, p. 113-119.

**Hatch, Don**

Green, Daniel, and Hatch, Don, **Fast Polygon-Cube Intersection Testing**, *Graphics Gems V*, p. 375-379, code: [ch7-2/](#).

**Hawley, Stephen**

Hawley, Stephen, **Ordered Dithering**, *Graphics Gems*, p. 176-178, code: p. 713-714, [OrderDither.c](#).

**Heckbert, Paul S.**

Heckbert, Paul S., **Nice Numbers for Graph Labels**, *Graphics Gems*, p. 61-63, code: p. 657-659, [Label.c](#).

Heckbert, Paul S., **Generic Convex Polygon Scan Conversion and Clipping**, *Graphics Gems*, p. 84-86, code: p. 667-680, [PolyScan/](#).

Heckbert, Paul S., **Concave Polygon Scan Conversion**, *Graphics Gems*, p. 87-91, code: p. 681-684, [ConcaveScan.c](#).

Heckbert, Paul S., **Digital Line Drawing**, *Graphics Gems*, p. 99-100, code: p. 685, [DigitalLine.c](#).

Heckbert, Paul S., **What Are the Coordinates of a Pixel?**, *Graphics Gems*, p. 246-248.

Heckbert, Paul S., **Recording Animation in Binary Order for Progressive Temporal Refinement**, *Graphics Gems*, p. 265-269, code: p. 720, [BinRec.c](#).

Heckbert, Paul S., **A Seed Fill Algorithm**, *Graphics Gems*, p. 275-277, code: p. 721-722, [SeedFill.c](#).

Heckbert, Paul S., **A Minimal Ray Tracer**, *Graphics Gems IV*, p. 375-381, code: p. 378-380, [minray/](#).

Heckbert, Paul S., **Bilinear Coons Patch Image Warping**, *Graphics Gems IV*, p. 438-446, code: p. 441-444, [coons\\_warp.c](#).

**Herbison-Evans, Don**

Herbison-Evans, Don, **Solving Quartics and Cubics for Graphics**, *Graphics Gems V*, p. 3-15, code: [ch1-1/](#).

**Hill, F. S., Jr.**

Hill, F. S., Jr., **The Pleasures of 'Perp Dot' Products**, *Graphics Gems IV*, p. 138-148.

**Hill, Kenneth J.**

Hill, Kenneth J., **Matrix-based Ellipse Geometry**, *Graphics Gems V*, p. 72-77, code: [ch2-6/](#).

**Hill, Steve**

Hill, Steve, **IEEE Fast Square Root**, *Graphics Gems III*, p. 48, code: p. 446-447, [sqrt.c](#).

Hill, Steve, **A Simple Fast Memory Allocator**, *Graphics Gems III*, p. 49-50, code: p. 448-451, [alloc/](#).

Hill, Steve, **Tri-linear Interpolation**, *Graphics Gems IV*, p. 521-525, code: p. 523-524, [trilerp.c](#).

Hill, Steve, and Roberts, Jonathan C., **Surface Models and the Resolution of N-Dimensional Cell Ambiguity**, *Graphics Gems V*, p. 98-106.

**Hollasch, Steve**

Hollasch, Steve, **Useful C Macros for Vector Operations**, *Graphics Gems II*, p. 467-469, code: p.



467-469, [vector.h](#).

Hollasch, Steve, **Progressive Image Refinement via Gridded Sampling**, *Graphics Gems III*, p. 358-361, code: p. 597-598, [PIR.c](#).

Hollasch, Steve, **Useful C Macros for Vector Operations**, *Graphics Gems III*, p. 405-407, code: p. 405-407, [vector.h](#).

### **Holt, Jeff**

Holt, Jeff, **Rotation of Run-Length Encoded Image Data**, *Graphics Gems II*, p. 86-88, code: p. 516-524.

### **Hook, D.G.**

Hook, D.G., and McAree, P.R., **Using Sturm Sequences To Bracket Real Roots of Polynomial Equations**, *Graphics Gems*, p. 416-423, code: p. 743-755, [Sturm/](#).

### **Hsu, Siu-chi**

Wong, Tien-tsin, and Hsu, Siu-chi, **Halftoning with Selective Precipitation and Adaptive Clustering**, *Graphics Gems V*, p. 302-313, code: p. 306-312, [ch6-2/](#).

Hsu, Siu-chi, and Lee, I.H.H., **Reversible Straight Line Edge Reconstruction**, *Graphics Gems V*, p. 338-354, code: p. 342-353, [ch6-5/](#).

### **Hultquist, Jeff**

Hultquist, Jeff, **Backface Culling**, *Graphics Gems*, p. 346-347.

Hultquist, Jeff, **Intersection of a Ray with a Sphere**, *Graphics Gems*, p. 388-389.

Hultquist, Jeff, **A Virtual Trackball**, *Graphics Gems*, p. 462-463.

Hultquist, Jeff, **Memory Allocation in C**, *Graphics Gems*, p. 643, code: p. 643.

### **Karinthi, Raghu**

Karinthi, Raghu, **Accurate Z-buffer Rendering**, *Graphics Gems V*, p. 398-399, code: [ch7-6/](#).

### **Kirk, David**

Voorhies, Douglas, and Kirk, David, **Ray-Triangle Intersection Using Binary Recursive Subdivision**, *Graphics Gems II*, p. 257-263.

Kirk, David, and Arvo, James, **Improved Ray Tagging for Voxel-Based Ray Tracing**, *Graphics Gems II*, p. 264-266.

### **Klassen, R. Victor**

Klassen, R. Victor, **Intersecting Parametric Cubic Curves by Midpoint Subdivision**, *Graphics Gems IV*, p. 261-277, code: p. 266-276, [curve\\_isect/](#).

### **Kohler, Andreas**

Bieri, Hanspeter, and Kohler, Andreas, **Computing the Area, the Circumference, and the Genus of a Binary Digital Image**, *Graphics Gems II*, p. 107-111, code: p. 525-527.

### **Kopp, Manfred**

Kopp, Manfred, and Gervautz, Michael, **XOR-Drawing with Guaranteed Contrast**, *Graphics Gems IV*, p. 413-414.

### **Lalonde, Paul**

Lalonde, Paul, and Dawson, Robert, **A High-Speed, Low Precision Square Root**, *Graphics Gems*, p. 424-426, code: p. 756-757, [SquareRoot.c](#).

### **Lee, Greg**

Lee, Greg, Penk, Mike, and Wallis, Bob, **Periodic Tilings of the Plane on a Raster Grid**, *Graphics Gems*, p. 129-139.

### **Lee, I.H.H.**

Hsu, Siu-chi, and Lee, I.H.H., **Reversible Straight Line Edge Reconstruction**, *Graphics Gems V*, p. 338-354, code: p. 342-353, [ch6-5/](#).

### **Lee, Mark E.**

Lee, Mark E., **Fast Dot Products for Shading**, *Graphics Gems*, p. 348-360.

Lee, Mark E., and Uselton, Samuel P., **A Body Color Model: Absorption Through Translucent Media**, *Graphics Gems II*, p. 277-282.

Lee, Mark E., and Uselton, Samuel P., **More Shadow Attenuation for Ray Tracing Transparent or Translucent Objects**, *Graphics Gems II*, p. 283-289.

### **Leipelt, Andreas**

Leipelt, Andreas, **Ray Tracing a Swept Sphere**, *Graphics Gems V*, p. 258-267, code: p. 261-267, [ch5-4/](#).

### **Lindgren, Terence**

Hall, Jim, and Lindgren, Terence, **A Fast Approach To PHIGS PLUS Pseudo Color**, *Graphics Gems II*, p. 138-142.

Lindgren, Terence, **Symmetric Evaluation of Polynomials**, *Graphics Gems II*, p. 420-423.

Lindgren, Terence, Sanchez, Juan, and Hall, Jim, **Curve Tessellation Criteria Thru Sampling**, *Graphics Gems III*, p. 262-265.

### **Lischinski, Dani**

Lischinski, Dani, **Converting Bézier Triangles Into Rectangular Patches**, *Graphics Gems III*, p. 256-261, code: p. 536-537, [bezierTri.C](#).

Lischinski, Dani, **Incremental Delaunay Triangulation**, *Graphics Gems IV*, p. 47-59, code: p. 51-58, [delaunay/](#).

Lischinski, Dani, **Converting Rectangular Patches into Bézier Triangles**, *Graphics Gems IV*, p. 278-285, code: p. 281-285, [patch\\_conv.C](#).

### **López-López, Fernando J.**

López-López, Fernando J., **Triangles Revisited**, *Graphics Gems III*, p. 215-218.

### **Maillot, Patrick-Gilles**

Maillot, Patrick-Gilles, **Using Quaternions for Coding 3D Transformations**, *Graphics Gems*, p. 498-515, code: p. 775-777, [Quaternions.c](#).

Maillot, Patrick-Gilles, **Three-Dimensional Homogeneous Clipping of Triangle Strips**, *Graphics Gems II*, p. 219-231, code: p. 563-570.

### **Manocha, Dinesh**

Narkhede, Atul, and Manocha, Dinesh, **Fast Polygon Triangulation Based on Seidel's Algorithm**,

*Graphics Gems V*, p. 394-397, code: [ch7-5/](#).

### **Marks, Joe**

Christensen, Jon, Marks, Joe, and Shieber, Stuart, **Placing Text Labels on Maps and Diagrams**, *Graphics Gems IV*, p. 497-504.

### **Martindale, David**

Martindale, David, and Paeth, Alan W., **Television Color Encoding and "Hot" Broadcast Colors**, *Graphics Gems II*, p. 147-158, code: p. 542-549, [hot.c](#).

### **Massalin, Henry**

Wolberg, George, and Massalin, Henry, **Fast Convolution with Packed Lookup Tables**, *Graphics Gems IV*, p. 447-464, code: p. 455-463, [convolve.c](#).

### **Max, Nelson L.**

Max, Nelson L., **An Optimal Filter for Image Reconstruction**, *Graphics Gems II*, p. 101-104.

Max, Nelson L., and Allison, Michael J., **Linear Radiosity Approximation Using Vertex-to-Vertex Form Factors**, *Graphics Gems III*, p. 318-323.

### **May, Stephen**

Paeth, Alan W., Scheepers, Ferdi, and May, Stephen, **A Survey of Graphics Libraries**, *Graphics Gems V*, p. 400-406, code: [ch7-7/](#).

### **McAree, P.R.**

Hook, D.G., and McAree, P.R., **Using Sturm Sequences To Bracket Real Roots of Polynomial Equations**, *Graphics Gems*, p. 416-423, code: p. 743-755, [Sturm/](#).

### **Meyer, Gary W.**

Meyer, Gary W., **An Inexpensive Method of Setting the Monitor White Point**, *Graphics Gems II*, p. 159-162.

### **Miller, Robert D.**

Miller, Robert D., **Joining Two Lines with a Circular Arc Fillet**, *Graphics Gems III*, p. 193-198, code: p. 496-499, [fillet.c](#).

Miller, Robert D., **Computing the Area of a Spherical Polygon**, *Graphics Gems IV*, p. 132-137, code: p. 135-136, [sph\\_poly.c](#).

Miller, Robert D., **Transforming Coordinates From One Coordinate Plane To Another**, *Graphics Gems V*, p. 111-120, code: p. 115-120, [ch3-4/](#).

Miller, Robert D., **Quick and Simple Bézier Curve Drawing**, *Graphics Gems V*, p. 206-209, code: p. 207-209, [ch4-8/](#).

### **Minh, Hong Tong**

Thalmann, Nadia Magnenat, Thalmann, Daniel, and Minh, Hong Tong, **InterPhong Shading**, *Graphics Gems II*, p. 232-241, code: p. 571-574, [InterPhong.c](#).

### **Miranda, Rick**

Alciatore, David, and Miranda, Rick, **The Best Least-Squares Line Fit**, *Graphics Gems V*, p. 91-97.

### **Montani, Claudio**

Montani, Claudio, and Scopigno, Roberto, **Spheres-to-Voxels Conversion**, *Graphics Gems*, p. 327-334.  
Montani, Claudio, and Scopigno, Roberto, **Quadtree/Octree-to-Boundary Conversion**, *Graphics Gems II*, p. 202-218.

### **Moore, Doug**

Moore, Doug, and Warren, Joseph, **Least-Squares Approximations To Bézier Curves and Surfaces**, *Graphics Gems II*, p. 406-411.  
Moore, Doug, and Warren, Joseph, **Compact Isocontours From Sampled Data**, *Graphics Gems III*, p. 23-28.  
Moore, Doug, **Subdividing Simplices**, *Graphics Gems III*, p. 244-249, code: p. 534-535, [simplex/](#).  
Moore, Doug, **Understanding Simplicoids**, *Graphics Gems III*, p. 250-255.

### **Morrison, Jack C.**

Morrison, Jack C., **Fast Anti-Aliasing Polygon Scan Conversion**, *Graphics Gems*, p. 76-83, code: p. 662-666, [AAPolyScan.c](#).  
Morrison, Jack C., **Distance From a Point To a Line**, *Graphics Gems II*, p. 10-13.  
Morrison, Jack C., **Quaternion Interpolation with Extra Spins**, *Graphics Gems III*, p. 96-97, code: p. 461-462, [quatspin.c](#).

### **Morton, Mike**

Morton, Mike, **A Digital "Dissolve" Effect**, *Graphics Gems*, p. 221-232, code: p. 715-717, [Dissolve.c](#).

### **Musgrave, F. Kenton**

Musgrave, F. Kenton, **A Peano Curve Generation Algorithm**, *Graphics Gems II*, p. 25, code: p. 477-484, [Peano/](#).  
Musgrave, F. Kenton, **A Random Color Map Animation Algorithm**, *Graphics Gems II*, p. 134-137, code: p. 536-541, [ran\\_ramp.c](#).  
Musgrave, F. Kenton, **Some Tips for Making Color Hardcopy**, *Graphics Gems II*, p. 163-165.  
Musgrave, F. Kenton, **A Panoramic Virtual Screen for Ray Tracing**, *Graphics Gems III*, p. 288-294, code: p. 551-554, [panorama.c](#).

### **Musial, Christopher J.**

Musial, Christopher J., **An Integer Square Root Algorithm**, *Graphics Gems II*, p. 387-388, code: p. 612.  
Musial, Christopher J., **A Good Straight-Line Approximation of a Circular Arc**, *Graphics Gems II*, p. 435-439, code: p. 617.

### **Márton, Gábor**

Márton, Gábor, **Acceleration of Ray Tracing via Voronoi-diagrams**, *Graphics Gems V*, p. 268-284, code: p. 276-283, [ch5-5/](#).

### **Möller, Tomas**

Möller, Tomas, **Fast Bitmap Stretching**, *Graphics Gems III*, p. 4-7, code: p. 411-413, [fastBitmap.c](#).  
Möller, Tomas, **Faster Ray Tracing Using Scanline Rejection**, *Graphics Gems V*, p. 242-257, code: p. 249-257, [ch5-3/](#).



## **Narkhede, Atul**

Narkhede, Atul, and Manocha, Dinesh, **Fast Polygon Triangulation Based on Seidel's Algorithm**, *Graphics Gems V*, p. 394-397, code: [ch7-5/](#).

## **Ohashi, Yoshikazu**

Ohashi, Yoshikazu, **Fast Linear Approximations of Euclidean Distance in Higher Dimensions**, *Graphics Gems IV*, p. 121-124, code: [dist\\_fast.c](#).

## **Olsen, John**

Olsen, John, **Smoothing Enlarged Monochrome Images**, *Graphics Gems*, p. 166-170.

## **Paeth, Alan W.**

Paeth, Alan W., **Trigonometric Functions at Select Points**, *Graphics Gems*, p. 18-19.

Paeth, Alan W., **A Fast 2D Point-on-line Test**, *Graphics Gems*, p. 49-50, code: p. 654-655, [PntOnLine.c](#).

Paeth, Alan W., **Circles of Integral Radius on Integer Lattices**, *Graphics Gems*, p. 57-60.

Paeth, Alan W., **Median Finding on a 3-by-3 Grid**, *Graphics Gems*, p. 171-175, code: p. 711-712, [Median.c](#).

Paeth, Alan W., **A Fast Algorithm for General Raster Rotation**, *Graphics Gems*, p. 179-195.

Paeth, Alan W., **Reading a Write-Only Write Mask**, *Graphics Gems*, p. 219-220.

Paeth, Alan W., **Mapping RGB Triples Onto Four Bits**, *Graphics Gems*, p. 233-245, code: p. 718, [RGBTo4Bits.c](#).

Paeth, Alan W., **Proper Treatment of Pixels as Integers**, *Graphics Gems*, p. 249-256, code: p. 719, [PixelInteger.c](#).

Paeth, Alan W., **Digital Cartography for Computer Graphics**, *Graphics Gems*, p. 307-320.

Paeth, Alan W., **A Fast Approximation To the Hypotenuse**, *Graphics Gems*, p. 427-431, code: p. 758, [HypotApprox.c](#).

Paeth, Alan W., **Image File Compression Made Easy**, *Graphics Gems II*, p. 93-100.

Paeth, Alan W., **Mapping RGB Triples Onto 16 Distinct Values**, *Graphics Gems II*, p. 143-146.

Martindale, David, and Paeth, Alan W., **Television Color Encoding and "Hot" Broadcast Colors**, *Graphics Gems II*, p. 147-158, code: p. 542-549, [hot.c](#).

Paeth, Alan W., **Exact Dihedral Metrics for Common Polyhedra**, *Graphics Gems II*, p. 174-178.

Paeth, Alan W., and Schilling, David, **Of Integers, Fields, and Bit Counting**, *Graphics Gems II*, p. 371-376, code: p. 610-611, [BitCounting/](#).

Paeth, Alan W., **A Half-Angle Identity for Digital Computation: The Joys of the Halved Tangent**, *Graphics Gems II*, p. 381-386.

Paeth, Alan W., **Great Circle Plotting**, *Graphics Gems II*, p. 440-445.

Paeth, Alan W., **Fast Generation of Cyclic Sequences**, *Graphics Gems III*, p. 67-76, code: p. 458-459, [cyclic.c](#).

Paeth, Alan W., **A Generic Pixel Selection Mechanism**, *Graphics Gems III*, p. 77-79.

Paeth, Alan W., **Ideal Tiles for Shading and Halftoning**, *Graphics Gems IV*, p. 486-492.

Paeth, Alan W., **Distance Approximations and Bounding Polyhedra**, *Graphics Gems V*, p. 78-87, code: p. 85-86, [ch2-7/](#).

Paeth, Alan W., Scheepers, Ferdi, and May, Stephen, **A Survey of Graphics Libraries**, *Graphics Gems V*, p. 400-406, code: [ch7-7/](#).

### **Pavicic, Mark J.**

Pavicic, Mark J., **Anti-Aliasing Filters that Minimize "Bumpy" Sampling**, *Graphics Gems*, p. 144-146.

Beran-Koehn, Jeffrey C., and Pavicic, Mark J., **A Cubic Tetrahedral Adaptation of the Hemi-Cube Algorithm**, *Graphics Gems II*, p. 299-302.

Beran-Koehn, Jeffrey C., and Pavicic, Mark J., **Delta Form-Factor Calculation for the Cubic Tetrahedral Algorithm**, *Graphics Gems III*, p. 324-328, code: p. 575-576, [forfac.c](#).

### **Pearce, Andrew**

Pearce, Andrew, **Shadow Attenuation for Ray Tracing Transparent Objects**, *Graphics Gems*, p. 397-399.

Pearce, Andrew, **A Recursive Shadow Voxel Cache for Ray Tracing**, *Graphics Gems II*, p. 273-274, code: p. 581-582, [VoxelCache.c](#).

Pearce, Andrew, **Avoiding Incorrect Shadow Intersections for Ray Tracing**, *Graphics Gems II*, p. 275-276.

### **Penk, Mike**

Lee, Greg, Penk, Mike, and Wallis, Bob, **Periodic Tilings of the Plane on a Raster Grid**, *Graphics Gems*, p. 129-139.

### **Peterson, John W.**

Peterson, John W., **Tessellation of NURB Surfaces**, *Graphics Gems IV*, p. 286-320, code: p. 294-319, [nurb\\_polyg/](#).

### **Pique, Michael E.**

Pique, Michael E., **Rotation Tools**, *Graphics Gems*, p. 465-469.

### **Prasad, Mukesh**

Prasad, Mukesh, **Intersection of Line Segments**, *Graphics Gems II*, p. 7-9, code: p. 473-476, [xlines.c](#).

### **Purgathofer, Werner**

Gervautz, Michael, and Purgathofer, Werner, **A Simple Method for Color Quantization: Octree Quantization**, *Graphics Gems*, p. 287-293.

Purgathofer, Werner, Tobler, Robert F., and Geiler, Manfred, **Improved Threshold Matrices for Ordered Dithering**, *Graphics Gems V*, p. 297-301.

### **Rabbitz, Rich**

Rabbitz, Rich, **Fast Collision Detection of Moving Convex Polyhedra**, *Graphics Gems IV*, p. 83-109, code: p. 91-108, [collide.c](#).

### **Raible, Eric**

Raible, Eric, **Matrix Orthogonalization**, *Graphics Gems*, p. 464, code: p. 765, [MatrixOrtho.c](#).

Raible, Eric, **Two Useful C Macros**, *Graphics Gems*, p. 644, code: p. 644.

### **Rasala, Richard**

Rasala, Richard, **Explicit Cubic Spline Interpolation Formulas**, *Graphics Gems*, p. 579-584.

### **Ritter, Jack**

Ritter, Jack, **An Algorithm for Filling in 2D Wide Line Bevel Joints**, *Graphics Gems*, p. 107-113.

Ritter, Jack, **An Efficient Bounding Sphere**, *Graphics Gems*, p. 301-303, code: p. 723-725, [BoundSphere.c](#).

Ritter, Jack, **A Simple Ray Rejection Test**, *Graphics Gems*, p. 385-386.

Ritter, Jack, **A Fast Approximation To 3D Euclidian Distance**, *Graphics Gems*, p. 432-433.

Ritter, Jack, **Fast 2D-3D Rotation**, *Graphics Gems*, p. 440-441.

Ritter, Jack, **Fast Sign of Cross Product Calculation**, *Graphics Gems II*, p. 392-393, code: p. 613-614.

**Roberts, Jonathan C.**

Hill, Steve, and Roberts, Jonathan C., **Surface Models and the Resolution of N-Dimensional Cell Ambiguity**, *Graphics Gems V*, p. 98-106.

**Rokne, Jon**

Rokne, Jon, **The Area of a Simple Polygon**, *Graphics Gems II*, p. 5-6.

Rokne, Jon, **An Easy Bounding Circle**, *Graphics Gems II*, p. 14-16.

Rokne, Jon, **The Smallest Circle Containing the Intersection of Two Circles**, *Graphics Gems II*, p. 17-18.

Rokne, Jon, **Appolonius's 10th Problem**, *Graphics Gems II*, p. 19-24.

Rokne, Jon, **Interval Arithmetic**, *Graphics Gems III*, p. 61-66, code: p. 454-457, [interval.C](#).

**Rosati, Claudio**

Rosati, Claudio, **A Simple Connection Algorithm for 2-D Drawing**, *Graphics Gems III*, p. 173-181, code: p. 480-486, [con2d.c](#).

**Rubio, Ruben Gonzalez**

Doué, Jean-François, and Rubio, Ruben Gonzalez, **Efficient and Robust 2D Shape Vectorization**, *Graphics Gems V*, p. 323-337, code: p. 329-336, [ch6-4/](#).

**Salesin, David**

Salesin, David, and Barzel, Ronen, **Compositing Black-and-White Bitmaps**, *Graphics Gems III*, p. 34-35.

Salesin, David, and Tampieri, Filippo, **Grouping Nearly Coplanar Polygons Into Coplanar Sets**, *Graphics Gems III*, p. 225-230, code: p. 512-516, [planeSets.c](#).

Fleischer, Kurt, and Salesin, David, **Accurate Polygon Scan Conversion Using Half-Open Intervals**, *Graphics Gems III*, p. 362-365, code: p. 599-605, [accurate\\_scan/](#).

**Sanchez, Juan**

Lindgren, Terence, Sanchez, Juan, and Hall, Jim, **Curve Tessellation Criteria Thru Sampling**, *Graphics Gems III*, p. 262-265.

**Scheepers, Ferdi**

Paeth, Alan W., Scheepers, Ferdi, and May, Stephen, **A Survey of Graphics Libraries**, *Graphics Gems V*, p. 400-406, code: [ch7-7/](#).

**Schilling, David**

Paeth, Alan W., and Schilling, David, **Of Integers, Fields, and Bit Counting**, *Graphics Gems II*, p. 371-376, code: p. 610-611, [BitCounting/](#).

## **Schlag, John**

Schlag, John, **Noise Thresholding in Edge Images**, *Graphics Gems II*, p. 105-106.

Schlag, John, **Using Geometric Constructions to Interpolate Orientation with Quaternions**, *Graphics Gems II*, p. 377-380.

Schlag, John, **A Simple Formulation for Curve Interpolation with Variable Control Point Approximation**, *Graphics Gems II*, p. 417-419.

Schlag, John, **Fast Embossing Effects on Raster Image Data**, *Graphics Gems IV*, p. 433-437, code: p. 435-436, [emboss.c](#).

## **Schlick, Christophe**

Schlick, Christophe, **A Fast Alternative to Phong's Specular Model**, *Graphics Gems IV*, p. 385-387.

Schlick, Christophe, **Fast Alternatives to Perlin's Bias and Gain Functions**, *Graphics Gems IV*, p. 401-403.

Schlick, Christophe, **High Dynamic Range Pixels**, *Graphics Gems IV*, p. 422-429, code: p. 425-428, [dyn\\_range/](#).

Schlick, Christophe, and Subrenat, Gilles, **Ray Intersection of Tessellated Surfaces: Quad Vs Triangle**, *Graphics Gems V*, p. 232-241, code: p. 237-240, [ch5-2/](#).

Schlick, Christophe, **Wave Generators for Computer Graphics**, *Graphics Gems V*, p. 367-374, code: p. 371-374, [ch7-1/](#).

## **Schneider, Philip J.**

Schneider, Philip J., **A Bézier Curve-Based Root-Finder**, *Graphics Gems*, p. 408-415, code: p. 787, [NearestPoint.c](#).

Schneider, Philip J., **Solving the Nearest-Point-on-Curve Problem**, *Graphics Gems*, p. 607-611, code: p. 787-796, [Nearest.c](#).

Schneider, Philip J., **An Algorithm for Automatically Fitting Digitized Curves**, *Graphics Gems*, p. 612-626, code: p. 797-807, [FitCurves.c](#).

## **Schorn, Peter**

Schorn, Peter, and Fisher, Frederick, **Testing the Convexity of a Polygon**, *Graphics Gems IV*, p. 7-15, code: p. 11-15, [convex\\_test/](#).

## **Schumacher, Dale A.**

Schumacher, Dale A., **Useful 1-to-1 Pixel Transforms**, *Graphics Gems*, p. 196-209.

Schumacher, Dale A., **1-to-1 Pixel Transforms Optimized Through Color-Map Manipulation**, *Graphics Gems*, p. 270-274.

Schumacher, Dale A., **Image Smoothing and Sharpening by Discrete Convolution**, *Graphics Gems II*, p. 50-56.

Schumacher, Dale A., **A Comparison of Digital Halftoning Techniques**, *Graphics Gems II*, p. 57-71, code: p. 502-508.

Schumacher, Dale A., **Fast Anamorphic Image Scaling**, *Graphics Gems II*, p. 78-79.

Schumacher, Dale A., **General Filtered Image Rescaling**, *Graphics Gems III*, p. 8-16, code: p. 414-424, [filter.c](#) [filter\\_rcg.c](#).

Schumacher, Dale A., **Optimization of Bitmap Scaling Operations**, *Graphics Gems III*, p. 17-19, code: p. 425-428, [bitmap.c](#).



### **Schwarze, Jochen**

Schwarze, Jochen, **Cubic and Quartic Roots**, *Graphics Gems*, p. 404-407, code: p. 738-786, [Roots3And4.c](#).

### **Scofield, Cary**

Scofield, Cary, **2-1/2-d Depth-of-Field Simulation for Computer Animation**, *Graphics Gems III*, p. 36-38.

Arvo, James, and Scofield, Cary, **The Shader Cache: A Rendering Pipeline Accelerator**, *Graphics Gems III*, p. 383-389.

### **Scopigno, Roberto**

Montani, Claudio, and Scopigno, Roberto, **Spheres-to-Voxels Conversion**, *Graphics Gems*, p. 327-334.

Montani, Claudio, and Scopigno, Roberto, **Quadtree/Octree-to-Boundary Conversion**, *Graphics Gems II*, p. 202-218.

### **Seidel, Hans-Peter**

Seidel, Hans-Peter, **Menelaus's Theorem**, *Graphics Gems II*, p. 424-427.

Seidel, Hans-Peter, **Geometrically Continuous Cubic Bézier Curves**, *Graphics Gems II*, p. 428-434.

### **Sevici, Constantin A.**

Sevici, Constantin A., **Solving the Problem of Apollonius and Other Related Problems**, *Graphics Gems III*, p. 203-209.

### **Shaffer, Clifford A.**

Shaffer, Clifford A., **Fast Circle-Rectangle Intersection Checking**, *Graphics Gems*, p. 51-53, code: p. 656, [CircleRect.c](#).

Shaffer, Clifford A., **Bit Interleaving for Quad- or Octrees**, *Graphics Gems*, p. 443-447, code: p. 759-762, [Interleave.c](#).

Shaffer, Clifford A., **Getting Around on a Sphere**, *Graphics Gems II*, p. 172-173.

Shaffer, Clifford A., and Feustel, Charles D., **Exact Computation of 2-D Intersections**, *Graphics Gems III*, p. 188-192, code: p. 491-495, [Polyintr.c](#).

### **Shapira, Andrew**

Shapira, Andrew, **Fast Line-Edge Intersections on a Uniform Grid**, *Graphics Gems*, p. 29-36, code: p. 651-653, [LineEdge.c](#).

### **Sharma, Rajesh**

Sharma, Rajesh, **Priority-based Adaptive Image Refinement**, *Graphics Gems V*, p. 355-358.

### **Shene, Ching-Kuang**

Shene, Ching-Kuang, **Equations of Cylinders and Cones**, *Graphics Gems IV*, p. 321-323.

Shene, Ching-Kuang, **Computing the Intersection of a Line and a Cylinder**, *Graphics Gems IV*, p. 353-355.

Shene, Ching-Kuang, **Computing the Intersection of a Line and a Cone**, *Graphics Gems V*, p. 227-231.

### **Shieber, Stuart**

Christensen, Jon, Marks, Joe, and Shieber, Stuart, **Placing Text Labels on Maps and Diagrams**,

*Graphics Gems IV*, p. 497-504.

### **Shirley, Peter**

Shirley, Peter, **Radiosity via Ray Tracing**, *Graphics Gems II*, p. 306-310.

Shirley, Peter, **Nonuniform Random Point Sets**, *Graphics Gems III*, p. 80-83.

Sung, Kelvin, and Shirley, Peter, **Ray Tracing with the BSP Tree**, *Graphics Gems III*, p. 271-274, code: p. 538-546, [bsp.c](#).

Chiu, Kenneth, Shirley, Peter, and Wang, Changyaw, **Multi-Jittered Sampling**, *Graphics Gems IV*, p. 370-374, code: p. 373-374, [multi\\_jitter/](#).

### **Shoemake, Ken**

Shoemake, Ken, **Bit Patterns for Encoding Angles**, *Graphics Gems*, p. 442.

Shoemake, Ken, **Quaternions and 4x4 Matrices**, *Graphics Gems II*, p. 351-354.

Shoemake, Ken, **Bit Picking**, *Graphics Gems II*, p. 366-367.

Shoemake, Ken, **Faster Fourier Transform**, *Graphics Gems II*, p. 368-370.

Shoemake, Ken, **Interval Sampling**, *Graphics Gems II*, p. 394-395.

Shoemake, Ken, **Beyond Bézier Curves**, *Graphics Gems II*, p. 412-416.

Shoemake, Ken, **Uniform Random Rotations**, *Graphics Gems III*, p. 124-132, code: p. 465-467, [urot.c](#).

Shoemake, Ken, **Arcball Rotation Control**, *Graphics Gems IV*, p. 175-192, code: p. 178-191, [arcball/](#).

Shoemake, Ken, **Polar Matrix Decomposition**, *Graphics Gems IV*, p. 207-221, code: p. 211-220, [polar\\_decomp/](#).

Shoemake, Ken, **Euler Angle Conversion**, *Graphics Gems IV*, p. 222-229, code: p. 225-228, [euler\\_angle/](#).

Shoemake, Ken, **Fiber Bundle Twist Reduction**, *Graphics Gems IV*, p. 230-236.

Shoemake, Ken, **Rational Approximation**, *Graphics Gems V*, p. 25-32, code: p. 29-31, [ch1-4/](#).

Shoemake, Ken, **Linear Form Curves**, *Graphics Gems V*, p. 210-223, code: p. 220-222, [ch4-9/](#).

### **Sillion, François**

Sillion, François, **Detection of Shadow Boundaries for Adaptive Meshing in Radiosity**, *Graphics Gems II*, p. 311-315.

### **Simar, Ray**

Van Aken, Jerry, and Simar, Ray, **A Parametric Elliptical Arc Algorithm**, *Graphics Gems III*, p. 164-172, code: p. 478-479, [parelarc.c](#).

### **Snyder, John**

Snyder, John, Barzel, Ronen, and Gabriel, Steve, **Motion Blur on Graphics Workstations**, *Graphics Gems III*, p. 374-382, code: p. 606-609, [motblur.c](#).

### **Spoelder, Hans J.W.**

Spoelder, Hans J.W., and Ullings, Fons H., **Two-Dimensional Clipping: A Vector-Based Approach**, *Graphics Gems*, p. 121-128, code: p. 694-710, [2DClip/](#).

### **Srinivasan, Raman V.**

Srinivasan, Raman V., **A Fast Circle Clipping Algorithm**, *Graphics Gems III*, p. 182-187, code: p. 487-490, [circlexc.c](#).

## **Steinhart, Jonathan E.**

Steinhart, Jonathan E., **Scanline Coherent Shape Algebra**, *Graphics Gems II*, p. 31-45, code: p. 487-501.

## **Subrenat, Gilles**

Schlick, Christophe, and Subrenat, Gilles, **Ray Intersection of Tessellated Surfaces: Quad Vs Triangle**, *Graphics Gems V*, p. 232-241, code: p. 237-240, [ch5-2/](#).

## **Sung, Kelvin**

Sung, Kelvin, and Shirley, Peter, **Ray Tracing with the BSP Tree**, *Graphics Gems III*, p. 271-274, code: p. 538-546, [bsp.c](#).

## **Szirmay-Kalos, László**

Szirmay-Kalos, László, **Dynamic Layout Algorithm to Display General Graphs**, *Graphics Gems IV*, p. 505-517, code: p. 511-517, [graph\\_layout/](#).

## **Tampieri, Filippo**

Tampieri, Filippo, **Fast Vertex Radiosity Update**, *Graphics Gems II*, p. 303-305, code: p. 598, [FastUpdate.c](#).

Salesin, David, and Tampieri, Filippo, **Grouping Nearly Coplanar Polygons Into Coplanar Sets**, *Graphics Gems III*, p. 225-230, code: p. 512-516, [planeSets.c](#).

Tampieri, Filippo, **Newell's Method for the Plane Equation of a Polygon**, *Graphics Gems III*, p. 231-232, code: p. 517-518, [newell.c](#).

Tampieri, Filippo, **Accurate Form-Factor Computation**, *Graphics Gems III*, p. 329-333, code: p. 577-581, [accForm.c](#).

## **Thalmann, Daniel**

Thalmann, Nadia Magnenat, Thalmann, Daniel, and Minh, Hong Tong, **InterPhong Shading**, *Graphics Gems II*, p. 232-241, code: p. 571-574, [InterPhong.c](#).

## **Thalmann, Nadia Magnenat**

Thalmann, Nadia Magnenat, Thalmann, Daniel, and Minh, Hong Tong, **InterPhong Shading**, *Graphics Gems II*, p. 232-241, code: p. 571-574, [InterPhong.c](#).

## **Thomas, Spencer W.**

Thomas, Spencer W., and Bogart, Rod G., **Color Dithering**, *Graphics Gems II*, p. 72-77, code: p. 509-513, [dither/](#).

Thomas, Spencer W., **Efficient Inverse Color Map Computation**, *Graphics Gems II*, p. 116-125, code: p. 528-535, [inv\\_cmap/](#).

Thomas, Spencer W., **Decomposing a Matrix Into Simple Transformations**, *Graphics Gems II*, p. 320-323, code: p. 599-602, [unmatrix.c](#).

## **Thompson, Kelvin**

Thompson, Kelvin, **Area of Intersection: Circle and a Half-Plane**, *Graphics Gems*, p. 38-39.

Thompson, Kelvin, **Area of Intersection: Circle and a Thick Line**, *Graphics Gems*, p. 40-42.

Thompson, Kelvin, **Area of Intersection: Two Circles**, *Graphics Gems*, p. 43-46.

Thompson, Kelvin, **Vertical Distance from a Point to a Line**, *Graphics Gems*, p. 47-48.

Thompson, Kelvin, **Rendering Anti-Aliased Lines**, *Graphics Gems*, p. 105-106, code: p. 690-693, [AALines/](#).

Thompson, Kelvin, **Alpha Blending**, *Graphics Gems*, p. 210-211.

Thompson, Kelvin, **Scanline Depth Gradient of a Z-Buffered Triangle**, *Graphics Gems*, p. 361-363.

Thompson, Kelvin, **Full-Precision Constants**, *Graphics Gems*, p. 434.

Thompson, Kelvin, **Converting Between Bits and Digits**, *Graphics Gems*, p. 435.

Thompson, Kelvin, **Matrix Identities**, *Graphics Gems*, p. 453-454.

Thompson, Kelvin, **Transforming Axes**, *Graphics Gems*, p. 456-459.

Thompson, Kelvin, **Fast Matrix Multiplication**, *Graphics Gems*, p. 460-461.

Thompson, Kelvin, **How to Build Circular Structures in C**, *Graphics Gems*, p. 645, code: p. 645.

Thompson, Kelvin, **How to Use C Register Variables to Point to 2D Arrays**, *Graphics Gems*, p. 646, code: p. 646.

### **Tobler, Robert F.**

Purgathofer, Werner, Tobler, Robert F., and Geiler, Manfred, **Improved Threshold Matrices for Ordered Dithering**, *Graphics Gems V*, p. 297-301.

### **Trumbore, Ben**

Trumbore, Ben, **Rectangular Bounding Volumes for Popular Primitives**, *Graphics Gems III*, p. 295-300, code: p. 555-561, [bounding\\_volumes.c](#).

### **Turk, Greg**

Turk, Greg, **Generating Random Points in Triangles**, *Graphics Gems*, p. 24-28, code: p. 649-650, [TriPoints.c](#).

### **Turkowski, Ken**

Turkowski, Ken, **Filters for Common Resampling Tasks**, *Graphics Gems*, p. 147-165.

Turkowski, Ken, **Fixed-Point Trigonometry with CORDIC Iterations**, *Graphics Gems*, p. 494-497, code: p. 773-774, [FixedTrig.c](#).

Turkowski, Ken, **The Use of Coordinate Frames in Computer Graphics**, *Graphics Gems*, p. 522-532.

Turkowski, Ken, **Properties of Surface-Normal Transformations**, *Graphics Gems*, p. 539-547.

Turkowski, Ken, **Computing the Inverse Square Root**, *Graphics Gems V*, p. 16-21, code: p. 17-19, [ch1-2/](#).

Turkowski, Ken, **Fixed Point Square Root**, *Graphics Gems V*, p. 22-24, code: p. 23, [ch1-3/](#).

Turkowski, Ken, **Circular Arc Subdivision**, *Graphics Gems V*, p. 168-172, code: p. 170-171, [ch4-3/](#).

### **Ullings, Fons H.**

Spoelder, Hans J.W., and Ullings, Fons H., **Two-Dimensional Clipping: A Vector-Based Approach**, *Graphics Gems*, p. 121-128, code: p. 694-710, [2DClip/](#).

### **Uselton, Samuel P.**

Lee, Mark E., and Uselton, Samuel P., **A Body Color Model: Absorption Through Translucent Media**, *Graphics Gems II*, p. 277-282.

Lee, Mark E., and Uselton, Samuel P., **More Shadow Attenuation for Ray Tracing Transparent or Translucent Objects**, *Graphics Gems II*, p. 283-289.

### **Van Aken, Jerry**



Van Aken, Jerry, and Simar, Ray, **A Parametric Elliptical Arc Algorithm**, *Graphics Gems III*, p. 164-172, code: p. 478-479, [parelarc.c](#).

### **Van Gelder, Allen**

Van Gelder, Allen, **Efficient Computation of Polygon Area and Polyhedron Volume**, *Graphics Gems V*, p. 35-41.

### **Van Hook, Tim**

Donovan, Walt, and Van Hook, Tim, **Direct Outcode Calculation for Faster Clip Testing**, *Graphics Gems IV*, p. 125-131, code: p. 127-131, [outcode/](#).

### **Vanecek, George, Jr.**

Bouma, William, and Vanecek, George, Jr., **Velocity-based Collision Detection**, *Graphics Gems V*, p. 380-385, code: p. 383-385, [ch7-3/](#).

Vanecek, George, Jr., **Spatial Partitioning of a Polygon by a Plane**, *Graphics Gems V*, p. 386-393, code: p. 387-393, [ch7-4/](#).

### **Voorhies, Douglas**

Voorhies, Douglas, **Space-Filling Curves and a Measure of Coherence**, *Graphics Gems II*, p. 26-30, code: p. 485-486, [Hilbert.c](#).

Voorhies, Douglas, and Kirk, David, **Ray-Triangle Intersection Using Binary Recursive Subdivision**, *Graphics Gems II*, p. 257-263.

Voorhies, Douglas, **Triangle-Cube Intersection**, *Graphics Gems III*, p. 236-239, code: p. 521-526, [triangleCube.c](#).

### **Waggenpack, Warren N., Jr.**

Cychosz, Joseph M., and Waggenpack, Warren N., Jr., **Intersecting a Ray with a Quadric Surface**, *Graphics Gems III*, p. 275-283, code: p. 547-550, [intqdr.c](#) [intell.c](#).

Cychosz, Joseph M., and Waggenpack, Warren N., Jr., **Intersecting a Ray with a Cylinder**, *Graphics Gems IV*, p. 356-365, code: p. 361-364, [ray\\_cyl.c](#).

### **Wallace, Bill**

Wallace, Bill, **Precalculating Addresses for Fast Fills, Circles, and Lines**, *Graphics Gems*, p. 285-286.

### **Wallis, Bob**

Wallis, Bob, **Fast Scan Conversion of Arbitrary Polygons**, *Graphics Gems*, p. 92-97.

Wallis, Bob, **Rendering Fat Lines on a Raster Grid**, *Graphics Gems*, p. 114-120.

Lee, Greg, Penk, Mike, and Wallis, Bob, **Periodic Tilings of the Plane on a Raster Grid**, *Graphics Gems*, p. 129-139.

Wallis, Bob, **Forms, Vectors, and Transforms**, *Graphics Gems*, p. 533-538, code: p. 780-784, [Forms.c](#).

Wallis, Bob, **Tutorial on Forward Differencing**, *Graphics Gems*, p. 594-603.

### **Wang, Changyaw**

Wang, Changyaw, **Physically Correct Direct Lighting for Distribution Ray Tracing**, *Graphics Gems III*, p. 307-313, code: p. 562-568, [luminaire/](#).

Chiu, Kenneth, Shirley, Peter, and Wang, Changyaw, **Multi-Jittered Sampling**, *Graphics Gems IV*, p. 370-374, code: p. 373-374, [multi\\_jitter/](#).

## **Wanger, Len**

Wanger, Len, and Fusco, Mike, **Fast N-Dimensional Extent Overlap Testing**, *Graphics Gems III*, p. 240-243, code: p. 527-533, [exttest/](#).

## **Ward, Greg**

Ward, Greg, **Real Pixels**, *Graphics Gems II*, p. 80-83, code: [RealPixels/](#).

Ward, Greg, **A Recursive Implementation of the Perlin Noise Function**, *Graphics Gems II*, p. 396-401, code: p. 615-616, [noise3.c](#).

Ward, Greg, **A Contrast-Based Scalefactor for Luminance Display**, *Graphics Gems IV*, p. 415-421.

## **Warren, Joseph**

Moore, Doug, and Warren, Joseph, **Least-Squares Approximations To Bézier Curves and Surfaces**, *Graphics Gems II*, p. 406-411.

Moore, Doug, and Warren, Joseph, **Compact Isocontours From Sampled Data**, *Graphics Gems III*, p. 23-28.

## **Weiler, Kevin**

Weiler, Kevin, **An Incremental Angle Point in Polygon Test**, *Graphics Gems IV*, p. 16-23, code: p. 17-22, [ptpoly\\_weiler/](#).

## **Wolberg, George**

Wolberg, George, and Massalin, Henry, **Fast Convolution with Packed Lookup Tables**, *Graphics Gems IV*, p. 447-464, code: p. 455-463, [convolve.c](#).

## **Wong, Tien-tsin**

Wong, Tien-tsin, and Hsu, Siu-chi, **Halftoning with Selective Precipitation and Adaptive Clustering**, *Graphics Gems V*, p. 302-313, code: p. 306-312, [ch6-2/](#).

## **Woo, Andrew**

Woo, Andrew, **Fast Ray-Polygon Intersection**, *Graphics Gems*, p. 394.

Woo, Andrew, **Fast Ray-Box Intersection**, *Graphics Gems*, p. 395-396, code: p. 736-737, [RayBox.c](#).

Woo, Andrew, **The Shadow Depth Map Revisited**, *Graphics Gems III*, p. 338-342, code: p. 582, [zdepth.c](#).

Fisher, Frederick, and Woo, Andrew, **R.E versus N.H Specular Highlights**, *Graphics Gems IV*, p. 388-400.

## **Wu, Kevin**

Wu, Kevin, **Fast Matrix Inversion**, *Graphics Gems II*, p. 342-350, code: p. 603-605, [inverse.c](#).

Wu, Kevin, **Fast Inversion of Length- and Angle-Preserving Matrices**, *Graphics Gems IV*, p. 199-206, code: p. 204-206, [inv\\_fast.c](#).

## **Wu, Xiaolin**

Wu, Xiaolin, **Efficient Statistical Computations for Optimal Color Quantization**, *Graphics Gems II*, p. 126-133, code: [quantizer.c](#).

Wu, Xiaolin, **Fast Anti-Aliased Circle Generation**, *Graphics Gems II*, p. 446-450.

Wu, Xiaolin, **A Linear-Time Simple Bounding Volume Algorithm**, *Graphics Gems III*, p. 301-306.

**Wuthrich, Charles A.**

Badouel, Didier, and Wuthrich, Charles A., **Face-Connected Line Segment Generation in an n-Dimensional Space**, *Graphics Gems III*, p. 89-91, code: p. 460, [ndline.c](#).

**Wyvill, Brian**

Wyvill, Brian, **Symmetric Double Step Line Algorithm**, *Graphics Gems*, p. 101-104, code: p. 686-689, [DoubleLine.c](#).

Wyvill, Brian, **3D Grid Hashing Function**, *Graphics Gems*, p. 343-345, code: p. 733-734, [Hash3D.c](#).

Wyvill, Brian, **Storage-free Swapping**, *Graphics Gems*, p. 436-437.

**Xu, Guoliang**

Bajaj, Chandrajit, and Xu, Guoliang, **Converting a Rational Curve to a Standard Rational Bernstein-Bézier Representation**, *Graphics Gems IV*, p. 256-260.

Bajaj, Chandrajit, and Xu, Guoliang, **Sparse Smooth Connection Between Bézier/B-Spline Curves**, *Graphics Gems V*, p. 191-198.

**Yap, Sue-Ken**

Yap, Sue-Ken, **A Fast 90-Degree Bitmap Rotator**, *Graphics Gems II*, p. 84-85, code: p. 514-515, [rotate8x8.c](#).

**Zimmerman, Kurt**

Zimmerman, Kurt, **Direct Lighting Models for Ray Tracing with Cylindrical Lamps**, *Graphics Gems V*, p. 285-289.

**Zuiderveld, Karel**







Zuiderveld, Karel, **Contrast Limited Adaptive Histogram Equalization**, *Graphics Gems IV*, p. 474-485, code: p. 479-484, [clahe.c](#).

---

Last change: July 6, 2000 [Eric Haines](#), Gems archivist / [erich@acm.org](mailto:erich@acm.org)

























# Index of














## /pubs/tog/GraphicsGems/gemsv/ch4-5/

Name	Last modified	Size	Description
 <a href="#">_Parent Directory</a>			
 <a href="#">_Makefile</a>	29-Jun-00 08:23	1K	
 <a href="#">_ellihelp.c</a>	29-Jun-00 08:23	7K	
 <a href="#">_ellipsoid.c</a>	29-Jun-00 08:23	9K	
 <a href="#">_ellipsoid.h</a>	29-Jun-00 08:23	1K	
 <a href="#">_timing.c</a>	29-Jun-00 08:23	5K	



# Index of /pubs/tog/GraphicsGems/gemsv/

Name	Last modified	Size	Description
 <a href="#">Parent Directory</a>			
 <a href="#">AllGems.TOC</a>	29-Jun-00 08:25	26K	
 <a href="#">Errata.GraphicsGemsV</a>	03-Jan-01 10:54	2K	
 <a href="#">GemsV.TOC</a>	29-Jun-00 08:25	3K	
 <a href="#">ch1-1/</a>	29-Jun-00 08:22	1K	
 <a href="#">ch1-2/</a>	29-Jun-00 08:22	1K	
 <a href="#">ch1-3/</a>	29-Jun-00 08:22	1K	
 <a href="#">ch1-4/</a>	03-Jan-01 10:51	1K	
 <a href="#">ch2-2/</a>	29-Jun-00 08:22	1K	
 <a href="#">ch2-6/</a>	29-Jun-00 08:22	1K	
 <a href="#">ch2-7/</a>	29-Jun-00 08:22	1K	
 <a href="#">ch3-3/</a>	29-Jun-00 08:22	1K	
 <a href="#">ch3-4/</a>	29-Jun-00 08:22	1K	
 <a href="#">ch3-5/</a>	29-Jun-00 08:22	1K	
 <a href="#">ch3-6/</a>	29-Jun-00 08:22	1K	
 <a href="#">ch4-3/</a>	29-Jun-00 08:23	1K	
 <a href="#">ch4-4/</a>	29-Jun-00 08:23	1K	
 <a href="#">ch4-5/</a>	29-Jun-00 08:23	1K	
 <a href="#">ch4-7/</a>	29-Jun-00 08:23	1K	
 <a href="#">ch4-8/</a>	29-Jun-00 08:23	1K	
 <a href="#">ch4-9/</a>	29-Jun-00 08:23	1K	
 <a href="#">ch5-2/</a>	29-Jun-00 08:23	1K	
 <a href="#">ch5-3/</a>	29-Jun-00 08:23	1K	
 <a href="#">ch5-4/</a>	29-Jun-00 08:23	1K	

 <a href="#">ch5-5/</a>	29-Jun-00 08:24	1K
 <a href="#">ch6-2/</a>	29-Jun-00 08:24	1K
 <a href="#">ch6-3/</a>	29-Jun-00 08:24	1K
 <a href="#">ch6-4/</a>	29-Jun-00 08:24	1K
 <a href="#">ch6-5/</a>	29-Jun-00 08:24	1K
 <a href="#">ch6-7/</a>	29-Jun-00 08:24	1K
 <a href="#">ch7-1/</a>	29-Jun-00 08:24	1K
 <a href="#">ch7-2/</a>	29-Jun-00 08:24	1K
 <a href="#">ch7-3/</a>	29-Jun-00 08:24	1K
 <a href="#">ch7-4/</a>	29-Jun-00 08:24	1K
 <a href="#">ch7-5/</a>	29-Jun-00 08:24	1K
 <a href="#">ch7-6/</a>	29-Jun-00 08:25	1K
 <a href="#">ch7-7/</a>	29-Jun-00 08:25	1K

```
timing:
    cc -O -pca -c ellipsoid_par_help.c
    cc -O -c ellipsoid.c
    cc -O -c timing.c
    cc -O -pca -o timing ellipsoid_par_help.o ellipsoid.o timing.o -lsphere -lmalloc
-lgl_s -lm -s
```

```
/* ellipsoid_par_help.c */

#include "ellipsoid.h"

#define SetP(p,px,py,pz) (p).x=(px), (p).y=(py), (p).z=(pz)
#define SetV(v,px,py,pz,nx,ny,nz) SetP((v)->p,px,py,pz), SetP((v)->n,nx,ny,nz)
#define SetF(f,i0,i1,i2) (f)->v0 = i0, (f)->v1 = i1, (f)->v2 = i2

void ellipsoid_par_help (int n, int nv, int nf, vertex *ev, face *ef, vertex *octant)
{
    int i, j, vv, ww, vv0, ww0;
    vertex *v, *o;
    face *f;

    /* Generate vertices and triangular faces of the ellipsoid.    */
    /* Note that each of the 18 statements below (par_0 ~ par_17) */
    /* are independent of each other.                               */

#pragma parallel
#pragma local(i,j,v,o,f,vv,ww,vv0,ww0)
#pragma byvalue(n,nv,nf,ev,ef,octant)
/** #pragma distinct(*v,*ev) **/
/** #pragma distinct(*o,*octant) **/
/** #pragma distinct(*f,*ef) **/
/* parallel */ {
#pragma independent
    /* par_0 */ { /* the north pole */
        v = ev, o = octant;
        SetV (v, o->p.x, o->p.y, o->p.z, o->n.x, o->n.y, o->n.z);
    }
}
#pragma independent
    /* par_1 */ { /* vertices on the 1st octant */
        v = ev+1, o = octant+1;
        for (i = 1;;) {
            for (j = i; --j >= 0; o++, v++)
                SetV (v, o->p.x, o->p.y, o->p.z, o->n.x, o->n.y, o->n.z);
            if (i == n) break;
            v += 3*i, i++, o++;
        }
        ; /* This empty statement is necessary because of the compiler-bug(?). */
    }
}
#pragma independent
    /* par_2 */ { /* vertices on the 2nd octant */
        v = ev+2, o = octant+2;
        for (i = 1;;) {
            for (j = i; --j >= 0; o--, v++)
                SetV (v, -o->p.x, o->p.y, o->p.z, -o->n.x, o->n.y, o->n.z);
            if (i == n) break;
            v += 3*i+1, i++, o += 2*i;
        }
    }
}
#pragma independent
    /* par_3 */ { /* vertices on the 3rd octant */
        v = ev+3, o = octant+1;
        for (i = 1;;) {
            for (j = i; --j >= 0; o++, v++)
                SetV (v, -o->p.x, -o->p.y, o->p.z, -o->n.x, -o->n.y, o->n.z);
            if (i == n) break;
            v += 3*i+2, i++, o++;
        }
    }
}
```

```
    }
    ;
}

#pragma independent
/* par_4 */ { /* vertices on the 4th octant */
    v = ev+4, o = octant+2;
    for (i = 1;;) {
        for (j = i; --j >= 0; o--, v++)
            SetV (v, o->p.x, o->p.y, o->p.z, o->n.x, o->n.y, o->n.z);
        if (i == n) break;
        i++, v += 3*i, o += 2*i;
    }
    ;
}

#pragma independent
/* par_5 */ { /* vertices on the 5th octant */
    v = ev+nv-5, o = octant+1;
    for (i = 1;;) {
        for (j = i; --j >= 0; o++, v++)
            SetV (v, o->p.x, o->p.y, o->p.z, o->n.x, o->n.y, o->n.z);
        if (i == n) break;
        v -= 5*i+4, i++, o++;
    }
    ;
}

#pragma independent
/* par_6 */ { /* vertices on the 6th octant */
    v = ev+nv-4, o = octant+2;
    for (i = 1;;) {
        for (j = i; --j >= 0; o--, v++)
            SetV (v, o->p.x, o->p.y, o->p.z, o->n.x, o->n.y, o->n.z);
        if (i == n) break;
        v -= 5*i+3, i++, o += 2*i;
    }
    ;
}

#pragma independent
/* par_7 */ { /* vertices on the 7th octant */
    v = ev+nv-3, o = octant+1;
    for (i = 1;;) {
        for (j = i; --j >= 0; o++, v++)
            SetV (v, o->p.x, o->p.y, o->p.z, o->n.x, o->n.y, o->n.z);
        if (i == n) break;
        v -= 5*i+2, i++, o++;
    }
    ;
}

#pragma independent
/* par_8 */ { /* vertices on the 8th octant */
    v = ev+nv-2, o = octant+2;
    for (i = 1;;) {
        for (j = i; --j >= 0; o--, v++)
            SetV (v, o->p.x, o->p.y, o->p.z, o->n.x, o->n.y, o->n.z);
        if (i == n) break;
        v -= 5*i+1, i++, o += 2*i;
    }
    ;
}

#pragma independent
/* par_9 */ { /* the south pole */
    v = ev+nv-1, o = octant;
```



```
        SetV (v, -o->p.x, -o->p.y, -o->p.z, -o->n.x, -o->n.y, -o->n.z);
    };
}
#pragma independent
/* par_10 */ { /* faces on the 1st octant */
    f = ef; vv = 0, ww = 1;
    for (i = 1;;) {
        for (j = i;;) {
            SetF (f, vv, ww, ++ww), f++;
            if (--j == 0) break;
            SetF (f, vv, ww, ++vv), f++;
        }
        if (i == n) break;
        f += 6*i-3; vv = ww-i; ww += 3*i; i++;
    }
}
#pragma independent
/* par_11 */ { /* faces on the 2nd octant */
    f = ef+1; vv = 0, ww = 2;
    for (i = 1;;) {
        for (j = i;;) {
            SetF (f, vv, ww, ++ww), f++;
            if (--j == 0) break;
            SetF (f, vv, ww, ++vv), f++;
        }
        if (i == n) break;
        f += 6*i-1; vv = ww-i; ww += 3*i+1; i++;
    }
}
#pragma independent
/* par_12 */ { /* faces on the 3rd octant */
    f = ef+2; vv = 0, ww = 3;
    for (i = 1;;) {
        for (j = i;;) {
            SetF (f, vv, ww, ++ww), f++;
            if (--j == 0) break;
            SetF (f, vv, ww, ++vv), f++;
        }
        if (i == n) break;
        f += 6*i+1; vv = ww-i; ww += 3*i+2; i++;
    }
}
#pragma independent
/* par_13 */ { /* faces on the 4th octant */
    f = ef+3; vv = 0, ww = 4, vv0 = 0, ww0 = 1;
    for (i = 1;;) {
        for (j = i;;) {
            if (--j == 0) { SetF (f, vv0, ww, ww0), vv++, ww++, f++; break; }
            SetF (f, vv, ww, ++ww), f++;
            if (j == 1) SetF (f, vv, ww, vv0), f++;
            else SetF (f, vv, ww, ++vv), f++;
        }
        if (i == n) break;
        vv0 = ww0; ww0 += 4*i;
        f += 6*i+3; vv = ww-i; i++; ww += 3*i;
    }
}
```

```
#pragma independent
/* par_14 */ { /* faces on the 5th octant */
  f = ef+nf-4; vv = nv-5, ww = nv-1;
  for (i = 1;;) {
    for (j = i;;) {
      SetF (f, vv, ww, ++vv), f++;
      if (--j == 0) break;
      SetF (f, vv, ww, ++ww), f++;
    }
    if (i == n) break;
    f -= 10*i+3; ww = vv-i; i++; vv -= 5*i-1;
  }
}

#pragma independent
/* par_15 */ { /* faces on the 6th octant */
  f = ef+nf-3; vv = nv-4, ww = nv-1;
  for (i = 1;;) {
    for (j = i;;) {
      SetF (f, vv, ww, ++vv), f++;
      if (--j == 0) break;
      SetF (f, vv, ww, ++ww), f++;
    }
    if (i == n) break;
    f -= 10*i+1; ww = vv-i; i++; vv -= 5*i-2;
  }
}

#pragma independent
/* par_16 */ { /* faces on the 7th octant */
  f = ef+nf-2; vv = nv-3, ww = nv-1;
  for (i = 1;;) {
    for (j = i;;) {
      SetF (f, vv, ww, ++vv), f++;
      if (--j == 0) break;
      SetF (f, vv, ww, ++ww), f++;
    }
    if (i == n) break;
    f -= 10*i-1; ww = vv-i; i++; vv -= 5*i-3;
  }
}

#pragma independent
/* par_17 */ { /* faces on the 8th octant */
  f = ef+nf-1; vv = nv-2, ww = nv-1, vv0 = nv-5, ww0 = nv-1;
  for (i = 1;;) {
    for (j = i;;) {
      if (--j == 0) { SetF (f, vv, ww0, vv0), vv++, ww++, f++; break; }
      SetF (f, vv, ww, ++vv), f++;
      if (j == 1) SetF (f, vv, ww, ww0), f++;
      else SetF (f, vv, ww, ++ww), f++;
    }
    if (i == n) break;
    f -= 10*i-3; ww = vv-i; i++; vv -= 5*i-4;
    ww0 = vv0; vv0 -= 4*i;
  }
}

/* parallel */ }
```

```
/* ellipsoid.c */

#include <stdio.h>
#include <math.h>
#include <malloc.h>
#include "ellipsoid.h"

#ifndef M_PI_2
#define M_PI_2  1.57079632679489661923
#endif

typedef struct slot { float cos, sin; enum { None, Only, Done } flag; } slot;

static int n_max = 0;          /* current maximum degree of subdivision */
static slot *table = NULL;     /* an array of slots */
static vertex *octant = NULL; /* the base octant of the ellipsoid */

#define SetP(p,px,py,pz) (p).x=(px), (p).y=(py), (p).z=(pz)
#define SetV(v,px,py,pz,nx,ny,nz) SetP((v)->p,px,py,pz), SetP((v)->n,nx,ny,nz)
#define SetF(f,i0,i1,i2) (f)->v0 = i0, (f)->v1 = i1, (f)->v2 = i2

/*
// Compute the necessary cosine and sine values for generating ellipsoids
// with the degree of subdivision n, and initialize the array table[].
// The largest n becomes n_max, and calls with n <= n_max return immediately.
// The memory for the base octant is allocated to cope with any n <= n_max.
*/
void ellipsoid_init (int n)
{
    int n_table, i, j, k, l, m, h, d;
    slot *t0, *t1, *t2;
    float theta;

    if (n > n_max) {
        n_max = n;
        if (table) free (table);
        if ((n_table = ((n-1)*n)/2) == 0) table = NULL;
        else table = (slot *) malloc (n_table * sizeof(slot));
        if (octant) free (octant);
        octant = (vertex *) malloc (((n+1)*(n+2))/2 * sizeof(vertex));

        for (t0 = table, k = n_table; k > 0; k--, t0++) t0->flag = None;
        for (t0 = table, k = 0, l = 1, m = 3, i = 2; i <= n_max; i++) {
            l += m, m += 2, h = n_max / i - 1;
            for (t1 = t0+i - 2, j = 1; j < i; j++, k++, t0++, t1--) {
                if (t0->flag == None) {
                    theta = (M_PI_2 * j) / i;
                    t0->cos = t1->sin = cos (theta);
                    t0->sin = t1->cos = sin (theta);
                    t0->flag = t1->flag = Only;
                }
                if (t0->flag == Only) {
                    t0->flag = Done;
                    for (d = k+1, t2 = t0; h > 0; h--) {
                        t2 += d, d += 1;
                        t2->cos = t0->cos;
                        t2->sin = t0->sin;
                        t2->flag = Done;
                    }
                }
            }
        }
    }
}
```

```
    }  
  }  
}  
  
/*  
// Construct the base octant of the ellipsoid whose parameters are a, b, and c,  
// with the degree of subdivision n using the cosine and sine values in table[].  
// It is assumed that n <= n_max.  
*/  
static void ellipsoid_octant (int n, float a, float b, float c)  
{  
  int i, j;  
  float a_1, b_1, c_1;  
  float cos_ph, sin_ph, px, py, pz, nx, ny, nz, nznz, rnorm, tmp;  
  vertex *o = octant;  
  slot *table_th, *table_ph;  
  
  a_1 = 1.0 / a; b_1 = 1.0 / b; c_1 = 1.0 / c;  
  o = octant;  
  table_th = table;  
  table_ph = table + ((n-1)*(n-2))/2;  
  
  SetV (o, 0.0, 0.0, c, 0.0, 0.0, 1.0), o++;          /* i = 0, j = 0 */  
  for (i = 1; i < n; i++, table_ph++) {  
    cos_ph = table_ph->cos;  
    sin_ph = table_ph->sin;  
    pz = cos_ph * c;  
    nz = cos_ph * c_1;  
    nznz = nz * nz;  
  
    px = sin_ph * a;  
    nx = sin_ph * a_1;  
    rnorm = 1.0 / sqrt (nx*nx + nznz);          /* 0 < i < n, j = 0 */  
    SetV (o, px, 0.0, pz, nx*rnorm, 0.0, nz*rnorm), o++;  
    for (j = i; --j > 0; table_th++) {  
      tmp = table_th->cos * sin_ph;  
      px = tmp * a;  
      nx = tmp * a_1;  
      tmp = table_th->sin * sin_ph;  
      py = tmp * b;  
      ny = tmp * b_1;  
      rnorm = 1.0 / sqrt (nx*nx + ny*ny + nznz); /* 0 < i < n, 0 < j < i */  
      SetV (o, px, py, pz, nx*rnorm, ny*rnorm, nz*rnorm), o++;  
    }  
    py = sin_ph * b;  
    ny = sin_ph * b_1;  
    rnorm = 1.0 / sqrt (ny*ny + nznz);          /* 0 < i < n, j = i */  
    SetV (o, 0.0, py, pz, 0.0, ny*rnorm, nz*rnorm), o++;  
  }  
  SetV (o, a, 0.0, 0.0, 1.0, 0.0, 0.0), o++;          /* i = n, j = 0 */  
  for (j = i; --j > 0; table_th++) {  
    tmp = table_th->cos;  
    px = tmp * a;  
    nx = tmp * a_1;  
    tmp = table_th->sin;  
    py = tmp * b;  
    ny = tmp * b_1;  
    rnorm = 1.0 / sqrt (nx*nx + ny*ny);          /* i = n, 0 < j < i */  
    SetV (o, px, py, 0.0, nx*rnorm, ny*rnorm, 0.0), o++;  
  }  
  SetV (o, 0.0, b, 0.0, 0.0, 1.0, 0.0);          /* i = n, j = i */  
}
```

```

}

/*
// Note the following conventions in ellipsoid_seq() and ellipsoid_par():
// the north pole:      th = 0,      ph = 0,
// the 1st octant:      0 <= th < 90,  0 < ph <= 90,
// the 2nd octant:      90 <= th < 180, 0 < ph <= 90,
// the 3rd octant:      180 <= th < 270, 0 < ph <= 90,
// the 4th octant:      270 <= th < 360, 0 < ph <= 90,
// the 5th octant:      0 <= th < 90, 90 < ph <= 180,
// the 6th octant:      90 <= th < 180, 90 < ph <= 180,
// the 7th octant:      180 <= th < 270, 90 < ph <= 180,
// the 8th octant:      270 <= th < 360, 90 < ph <= 180, and
// the south pole:      th = 0,      ph = 180.
*/

/*
// Generate the vertices for the ellipsoid with parameters a, b, and c
// with the degree of subdivision n, by reflecting the base octant.
// Also generate triangular faces of the ellipsoid with vertices ordered
// counterclockwise when viewed from the outside.
*/

/* sequential version */
void ellipsoid_seq (object *ellipsoid, int n, float a, float b, float c)
{
    vertex *v, *o;
    face *f;
    int i, j, ko, kv, kw, kv0, kw0;

    /* Check parameters for validity. */
    if (n <= 0 || n_max < n || a <= 0.0 || b <= 0.0 || c <= 0.0) {
        ellipsoid->nv = 0; ellipsoid->v = NULL;
        ellipsoid->nf = 0; ellipsoid->f = NULL;
        return;
    }

    /* Initialize the base octant. */
    ellipsoid_octant (n, a, b, c);

    /* Allocate memories for vertices and faces. */
    ellipsoid->nv = 4*n*n + 2;
    ellipsoid->nf = 8*n*n;
    ellipsoid->v = (vertex *) malloc (ellipsoid->nv * sizeof(vertex));
    ellipsoid->f = (face *) malloc (ellipsoid->nf * sizeof(face));

    /* Generate vertices of the ellipsoid from octant[]. */
    v = ellipsoid->v;
    o = octant;
#define op o->p
#define on o->n
    SetV (v, op.x, op.y, op.z, on.x, on.y, on.z), v++; /* the north pole */
    for (i = 0; ++i <= n; ) {
        o += i;
        for (j = i; --j >= 0; o++, v++) /* 1st octant */
            SetV (v, op.x, op.y, op.z, on.x, on.y, on.z);
        for (j = i; --j >= 0; o--, v++) /* 2nd octant */
            SetV (v, -op.x, op.y, op.z, -on.x, on.y, on.z);
        for (j = i; --j >= 0; o++, v++) /* 3rd octant */
            SetV (v, -op.x, -op.y, op.z, -on.x, -on.y, on.z);
        for (j = i; --j >= 0; o--, v++) /* 4th octant */

```



```
        SetV (v,  op.x, -op.y,  op.z,  on.x, -on.y,  on.z);
    }
    for (; --i > 1;) {
        o -= i;
        for (j = i; --j > 0; o++, v++)                /* 5th octant */
            SetV (v,  op.x,  op.y, -op.z,  on.x,  on.y, -on.z);
        for (j = i; --j > 0; o--, v++)                /* 6th octant */
            SetV (v, -op.x,  op.y, -op.z, -on.x,  on.y, -on.z);
        for (j = i; --j > 0; o++, v++)                /* 7th octant */
            SetV (v, -op.x, -op.y, -op.z, -on.x, -on.y, -on.z);
        for (j = i; --j > 0; o--, v++)                /* 8th octant */
            SetV (v,  op.x, -op.y, -op.z,  on.x, -on.y, -on.z);
    }
    o--, SetV (v, -op.x, -op.y, -op.z, -on.x, -on.y, -on.z); /* the south pole */
#undef op
#undef on

/* Generate triangular faces of the ellipsoid. */
f = ellipsoid->f;
kv = 0, kw = 1;
for (i = 0; i < n; i++) {
    kv0 = kv, kw0 = kw;
    for (ko = 1; ko <= 3; ko++)                /* the 1st, 2nd, 3rd octants */
        for (j = i;; j--) {
            SetF (f, kv, kw, ++kw), f++;
            if (j == 0) break;
            SetF (f, kv, kw, ++kv), f++;
        }
    for (j = i;; j--) {                        /* the 4th octant */
        if (j == 0) { SetF (f, kv0, kw, kw0), kv++, kw++, f++; break; }
        SetF (f, kv, kw, ++kw), f++;
        if (j == 1) SetF (f, kv, kw, kv0), f++;
        else SetF (f, kv, kw, ++kv), f++;
    }
}
for (; --i >= 0;) {
    kv0 = kv, kw0 = kw;
    for (ko = 5; ko <= 7; ko++)                /* the 5th, 6th, 7th octants */
        for (j = i;; j--) {
            SetF (f, kv, kw, ++kv), f++;
            if (j == 0) break;
            SetF (f, kv, kw, ++kw), f++;
        }
    for (j = i;; j--) {                        /* the 8th octant */
        if (j == 0) { SetF (f, kv, kw0, kv0), kv++, kw++, f++; break; }
        SetF (f, kv, kw, ++kv), f++;
        if (j == 1) SetF (f, kv, kw, kw0), f++;
        else SetF (f, kv, kw, ++kw), f++;
    }
}
}

/* parallel version */
void ellipsoid_par (object *ellipsoid, int n, float a, float b, float c)
{
    int nv, nf;
    vertex *ev;
    face *ef;

    /* Check parameters for validity. */
    if (n <= 0 || n_max < n || a <= 0.0 || b <= 0.0 || c <= 0.0) {
```

```
    ellipsoid->nv = 0; ellipsoid->v = NULL;
    ellipsoid->nf = 0; ellipsoid->f = NULL;
    return;
}

/* Initialize the base octant. */
ellipsoid_octant (n, a, b, c);

/* Allocate memories for vertices and faces. */
nv = ellipsoid->nv = 4*n*n + 2;
nf = ellipsoid->nf = 8*n*n;
ev = ellipsoid->v = (vertex *) malloc (nv * sizeof(vertex));
ef = ellipsoid->f = (face *) malloc (nf * sizeof(face));

ellipsoid_par_help (n, nv, nf, ev, ef, octant);
}

/* A utility function that simply outputs the ellipsoid data. */
void ellipsoid_print (object *ellipsoid)
{
    int i;
    vertex *v;
    face *f;

    for (v = ellipsoid->v, i = 0; i < ellipsoid->nv; i++, v++)
        printf ("v %g %g %g %g %g %g\n",
            v->p.x, v->p.y, v->p.z, v->n.x, v->n.y, v->n.z);

    for (f = ellipsoid->f, i = 0; i < ellipsoid->nf; i++, f++)
        printf ("f %d %d %d\n", f->v0, f->v1, f->v2);
}

/* undo of ellipsoid_init () */
void ellipsoid_done (void)
{
    n_max = 0;
    if (table) free (table), table = NULL;
    if (octant) free (octant), octant = NULL;
}

/* undo of ellipsoid_seq () */
void ellipsoid_free (object *ellipsoid)
{
    free (ellipsoid->v), ellipsoid->nv = 0, ellipsoid->v = NULL;
    free (ellipsoid->f), ellipsoid->nf = 0, ellipsoid->f = NULL;
}
```

```
/* ellipsoid.h */

#ifndef ellipsoid_H
#define ellipsoid_H

typedef struct point { float x, y, z; } point;
typedef struct vertex {
    point p, n;          /* point and unit normal */
} vertex;
typedef struct face {
    int v0, v1, v2;      /* indices of vertex array for a triangular face */
} face;
typedef struct object {
    int nv, nf;          /* numbers of elements in v and f */
    vertex *v; face *f; /* arrays of vertices and faces */
} object;

void ellipsoid_init (int n);
void ellipsoid_seq (object *ellipsoid, int n, float a, float b, float c);
void ellipsoid_par (object *ellipsoid, int n, float a, float b, float c);
void ellipsoid_print (object *ellipsoid);
void ellipsoid_done (void);
void ellipsoid_free (object *ellipsoid);

#endif /* ellipsoid_H */
```

```
/*
// timing.c
//
//
// Performance test of the suggested ellipsoid generation method.
// The fastest execution time among 100 runs of ellipsoid generation method
// subtracted by the fastest execution time among 100 runs of gettimeofday
// system call is output.
//
// A comparison is made to the IRIS GL sphere library.
//
// Here, depth denote the degree of subdivision.
*/

#include "ellipsoid.h"

#include <gl/gl.h>
#include <gl/sphere.h>

#include <stdio.h>
#include <sys/time.h>
struct timeval t0, t1;
long microsec;

#define max_depth 105
#define max_run 100

struct {
    long init;
    long seq;
    long par;
    long libsphere;
} timing[max_depth];

int main ()
{
    object ellipsoid;
    int depth;
    int run;

    long min_syscall = -1;
    long min_init = -1;
    long min_seq = -1;
    long min_par = -1;
    long min_libsphere_draw = -1;
    long min_libsphere = -1;

    /*
    // Find the fastest execution time among 100 runs
    // of gettimeofday system call.
    */
    for (run = 1; run <= max_run; run++) {
        gettimeofday (&t0);
        gettimeofday (&t1);

        microsec = (t1.tv_sec-t0.tv_sec)*1000000 + (t1.tv_usec-t0.tv_usec);
        if (min_syscall > microsec || min_syscall < 0)
            min_syscall = microsec;
    }

    /*
```

```
// Find the fastest execution time among 100 runs
// of ellipsoid initialization for each depth.
*/
for (depth = 1; depth <= max_depth; depth++) {
    for (run = 1; run <= max_run; run++) {
        gettimeofday (&t0);
        ellipsoid_init (depth);
        gettimeofday (&t1);
        ellipsoid_done ();

        microsec = (t1.tv_sec-t0.tv_sec)*1000000 + (t1.tv_usec-t0.tv_usec);
        if (min_init > microsec || min_init < 0)
            min_init = microsec;
    }
    min_init -= min_syscall;
    timing[depth-1].init = min_init;
    min_init = -1;
}

/*
// Find the fastest execution time among 100 runs
// of ellipsoid generation by ellipsoid_seq() for each depth.
*/
ellipsoid_init (max_depth);
for (depth = 1; depth <= max_depth; depth++) {
    for (run = 1; run <= max_run; run++) {
        gettimeofday (&t0);
        ellipsoid_seq (&ellipsoid, depth, 1.0, 2.0, 3.0);
        gettimeofday (&t1);
        ellipsoid_free (&ellipsoid);

        microsec = (t1.tv_sec-t0.tv_sec)*1000000 + (t1.tv_usec-t0.tv_usec);
        if (min_seq > microsec || min_seq < 0)
            min_seq = microsec;
    }
    min_seq -= min_syscall;
    timing[depth-1].seq = min_seq;
    min_seq = -1;
}

/*
// Find the fastest execution time among 100 runs
// of ellipsoid generation by ellipsoid_par() for each depth.
*/
ellipsoid_init (max_depth);
for (depth = 1; depth <= max_depth; depth++) {
    for (run = 1; run <= max_run; run++) {
        gettimeofday (&t0);
        ellipsoid_par (&ellipsoid, depth, 1.0, 2.0, 3.0);
        gettimeofday (&t1);
        ellipsoid_free (&ellipsoid);

        microsec = (t1.tv_sec-t0.tv_sec)*1000000 + (t1.tv_usec-t0.tv_usec);
        if (min_par > microsec || min_par < 0)
            min_par = microsec;
    }
    min_par -= min_syscall;
    timing[depth-1].par = min_par;
    min_par = -1;
}
```



```
/*
// Find the fastest execution time among 100 runs
// of sphere generation by IRIS GL sphere library for each depth.
*/
foreground ();
noport ();
winopen ("");
sphmode (SPH_TESS, SPH_OCT); /* octahedral subdivision */
sphmode (SPH_PRIM, SPH_POLY);
for (depth = 1; depth <= SPH_MAXDEPTH; depth++) {
    static float sphparams[] = { 0.0, 0.0, 0.0, 1.0 };

    /*
    // Find the fastest execution time among 100 runs
    // of sphere drawing by IRIS GL sphere library.
    */
    sphmode (SPH_DEPTH, depth);
    sphdraw (sphparams); /* generation and drawing of the sphere */
    for (run = 1; run <= max_run; run++) {
        finish (); /* block until the geometry pipeline is empty */
        gettimeofday (&t0);
        sphdraw (sphparams); /* drawing only without generation */
        gettimeofday (&t1);

        microsec = (t1.tv_sec-t0.tv_sec)*1000000 + (t1.tv_usec-t0.tv_usec);
        if (min_libsphere_draw > microsec || min_libsphere_draw < 0)
            min_libsphere_draw = microsec;
    }
    sphfree ();
    min_libsphere_draw -= min_syscall;

    /*
    // Find the fastest execution time among 100 runs
    // of sphere generation by IRIS GL sphere library.
    */
    for (run = 1; run <= max_run; run++) {
        sphmode (SPH_DEPTH, depth);
        finish ();
        gettimeofday (&t0);
        sphdraw (sphparams); /* generation and drawing of the sphere */
        gettimeofday (&t1);
        sphfree ();

        microsec = (t1.tv_sec-t0.tv_sec)*1000000 + (t1.tv_usec-t0.tv_usec);
        if (min_libsphere > microsec || min_libsphere < 0)
            min_libsphere = microsec;
    }
    min_libsphere -= min_libsphere_draw + min_syscall;
    timing[depth-1].libsphere = min_libsphere;
    min_libsphere = -1;
}

for (depth = 1; depth <= max_depth; depth++) {
    printf ("depth = %3d: nv = %8d, nf = %8d, init = %8d, seq = %8d, par = %8d",
        depth, 4*depth*depth + 2, 8*depth*depth,
        timing[depth-1].init, timing[depth-1].seq, timing[depth-1].par);
    if (depth <= SPH_MAXDEPTH)
        printf (" , libsphere = %8d", timing[depth-1].libsphere);
    printf ("\n");
}
```

```
    return 0;
```

```
}
```

```
/* Faster Line Segment Intersection */
/* Franklin Antonio */

/* return values */
#define DONT_INTERSECT 0
#define DO_INTERSECT 1
#define PARALLEL 2

/* The SAME_SIGNS macro assumes arithmetic where the exclusive-or
/* operation will work on sign bits. This works for twos-complement,
/* and most other machine arithmetic.
#define SAME_SIGNS( a, b ) \
    (((long) ((unsigned long) a ^ (unsigned long) b)) >= 0 )

/* The use of some short working variables allows this code to run
/* faster on 16-bit computers, but is not essential. It should not
/* affect operation on 32-bit computers. The short working variables
/* to not restrict the range of valid input values, as these were
/* constrained in any case, due to algorithm restrictions.

int lines_intersect(x1,y1,x2,y2,x3,y3,x4,y4,x,y)
long x1,y1,x2,y2,x3,y3,x4,y4,*x,*y;
{

long Ax,Bx,Cx,Ay,By,Cy,d,e,f,num,offset;
short x1lo,x1hi,y1lo,y1hi;

Ax = x2-x1;
Bx = x3-x4;

if(Ax<0) {
    x1lo=(short)x2; x1hi=(short)x1;
} else {
    x1hi=(short)x2; x1lo=(short)x1;
}
if(Bx>0) {
    if(x1hi < (short)x4 || (short)x3 < x1lo) return DONT_INTERSECT;
} else {
    if(x1hi < (short)x3 || (short)x4 < x1lo) return DONT_INTERSECT;
}

Ay = y2-y1;
By = y3-y4;

if(Ay<0) {
    y1lo=(short)y2; y1hi=(short)y1;
} else {
    y1hi=(short)y2; y1lo=(short)y1;
}
if(By>0) {
    if(y1hi < (short)y4 || (short)y3 < y1lo) return DONT_INTERSECT;
} else {
    if(y1hi < (short)y3 || (short)y4 < y1lo) return DONT_INTERSECT;
}

Cx = x1-x3;
Cy = y1-y3;
```

```
d = By*Cx - Bx*Cy; /* alpha numerator*/
f = Ay*Bx - Ax*By; /* both denominator*/
if(f>0) { /* alpha tests*/
    if(d<0 || d>f) return DONT_INTERSECT;
    } else {
    if(d>0 || d<f) return DONT_INTERSECT;
    }

e = Ax*Cy - Ay*Cx; /* beta numerator*/
if(f>0) { /* beta tests*/
    if(e<0 || e>f) return DONT_INTERSECT;
    } else {
    if(e>0 || e<f) return DONT_INTERSECT;
    }

/*compute intersection coordinates*/



if(f==0) return PARALLEL;
num = d*Ax; /* numerator */
offset = SAME_SIGNS(num,f) ? f/2 : -f/2; /* round direction*/
*x = x1 + (num+offset) / f; /* intersection x */

num = d*Ay;
offset = SAME_SIGNS(num,f) ? f/2 : -f/2;
*y = y1 + (num+offset) / f; /* intersection y */

return DO_INTERSECT;
}
```

# Index of

## /pubs/tog/GraphicsGems/gemsv/ch3-3/

Name	Last modified	Size	Description
 <a href="#">_Parent Directory</a>			
 <a href="#">_tricubic.c</a>	29-Jun-00 08:22	2K	



```
typedef struct
{
    float          x, y, z;
}
    Point;

/*
 * TriCubic - tri-cubic interpolation at point, p.
 * inputs:
 *     p - the interpolation point.
 *     volume - a pointer to the float volume data, stored in x,
 *             y, then z order (x index increasing fastest).
 *     xDim, yDim, zDim - dimensions of the array of volume data.
 * returns:
 *     the interpolated value at p.
 * note:
 *     NO range checking is done in this function.
 */

float          TriCubic (Point p, float *volume, int xDim, int yDim, int zDim)
{
    int          x, y, z;
    register int  i, j, k;
    float        dx, dy, dz;
    register float *pv;
    float        u[4], v[4], w[4];
    float        r[4], q[4];
    float        vox = 0;
    int          xyDim;

    xyDim = xDim * yDim;

    x = (int) p.x, y = (int) p.y, z = (int) p.z;
    if (x < 0 || x >= xDim || y < 0 || y >= yDim || z < 0 || z >= zDim)
        return (0);

    dx = p.x - (float) x, dy = p.y - (float) y, dz = p.z - (float) z;
    pv = volume + (x - 1) + (y - 1) * xDim + (z - 1) * xyDim;

#define CUBE(x)    ((x) * (x) * (x))
#define SQR(x)    ((x) * (x))
/*
#define DOUBLE(x) ((x) + (x))
#define HALF(x)    ...
 *
 * may also be used to reduce the number of floating point
 * multiplications. The IEEE standard allows for DOUBLE/HALF
 * operations.
 */

    /* factors for Catmull-Rom interpolation */

    u[0] = -0.5 * CUBE (dx) + SQR (dx) - 0.5 * dx;
    u[1] = 1.5 * CUBE (dx) - 2.5 * SQR (dx) + 1;
    u[2] = -1.5 * CUBE (dx) + 2 * SQR (dx) + 0.5 * dx;
    u[3] = 0.5 * CUBE (dx) - 0.5 * SQR (dx);

    v[0] = -0.5 * CUBE (dy) + SQR (dy) - 0.5 * dy;
    v[1] = 1.5 * CUBE (dy) - 2.5 * SQR (dy) + 1;
    v[2] = -1.5 * CUBE (dy) + 2 * SQR (dy) + 0.5 * dy;
    v[3] = 0.5 * CUBE (dy) - 0.5 * SQR (dy);
```

```
w[0] = -0.5 * CUBE (dz) + SQR (dz) - 0.5 * dz;  
w[1] = 1.5 * CUBE (dz) - 2.5 * SQR (dz) + 1;  
w[2] = -1.5 * CUBE (dz) + 2 * SQR (dz) + 0.5 * dz;  
w[3] = 0.5 * CUBE (dz) - 0.5 * SQR (dz);
```

```
for (k = 0; k < 4; k++)  
{  
    q[k] = 0;  
    for (j = 0; j < 4; j++)  
    {  
        r[j] = 0;  
        for (i = 0; i < 4; i++)  
        {  
            r[j] += u[i] * *pv;  
            pv++;  
        }  
        q[k] += v[j] * r[j];  
        pv += xDim - 4;  
    }  
    vox += w[k] * q[k];  
    pv += xyDim - 4 * xDim;  
}  
return (vox < 0 ? 0.0 : vox);  
}
```

```
/*
A Simple Method for Box-Sphere Intersection Testing
by Jim Arvo
from "Graphics Gems", Academic Press, 1990
*/

#include "GraphicsGems.h"

/*
 * This routine tests for intersection between an n-dimensional
 * axis-aligned box and an n-dimensional sphere. The mode argument
 * indicates whether the objects are to be regarded as surfaces or
 * solids. The values are:
 *
 * mode
 *
 * 0    Hollow Box, Hollow Sphere
 * 1    Hollow Box, Solid Sphere
 * 2    Solid Box, Hollow Sphere
 * 3    Solid Box, Solid Sphere
 */
int Box_Sphere_Intersect( n, Bmin, Bmax, C, r, mode )
int    n;          /* The dimension of the space. */
float  Bmin[];     /* The minimum of the box for each axis. */
float  Bmax[];     /* The maximum of the box for each axis. */
float  C[];        /* The sphere center in n-space. */
float  r;          /* The radius of the sphere. */
int    mode;       /* Selects hollow or solid. */
{
    float  a, b;
    float  dmin, dmax;
    float  r2 = SQR( r );
    int    i, face;

    switch( mode )
    {
        case 0: /* Hollow Box - Hollow Sphere */
            dmin = 0;
            dmax = 0;
            face = FALSE;
            for( i = 0; i < n; i++ ) {
                a = SQR( C[i] - Bmin[i] );
                b = SQR( C[i] - Bmax[i] );
                dmax += MAX( a, b );
                if( C[i] < Bmin[i] ) {
                    face = TRUE;
                    dmin += a;
                }
                else if( C[i] > Bmax[i] ) {
                    face = TRUE;
                    dmin += b;
                }
                else if( MIN( a, b ) <= r2 ) face = TRUE;
            }
            if( face && ( dmin <= r2 ) && ( r2 <= dmax ) ) return(TRUE);
            break;

        case 1: /* Hollow Box - Solid Sphere */
            dmin = 0;
    }
}
```

```
    face = FALSE;
    for( i = 0; i < n; i++ ) {
        if( C[i] < Bmin[i] ) {
            face = TRUE;
            dmin += SQR( C[i] - Bmin[i] );
        }
        else if( C[i] > Bmax[i] ) {
            face = TRUE;
            dmin += SQR( C[i] - Bmax[i] );
        }
        else if( C[i] - Bmin[i] <= r ) face = TRUE;
        else if( Bmax[i] - C[i] <= r ) face = TRUE;
    }
    if( face && ( dmin <= r2 ) ) return( TRUE );
    break;

case 2: /* Solid Box - Hollow Sphere */
    dmax = 0;
    dmin = 0;
    for( i = 0; i < n; i++ ) {
        a = SQR( C[i] - Bmin[i] );
        b = SQR( C[i] - Bmax[i] );
        dmax += MAX( a, b );
        if( C[i] < Bmin[i] ) dmin += a; else
        if( C[i] > Bmax[i] ) dmin += b;
    }
    if( dmin <= r2 && r2 <= dmax ) return( TRUE );
    break;

case 3: /* Solid Box - Solid Sphere */
    dmin = 0;
    for( i = 0; i < n; i++ ) {
        if( C[i] < Bmin[i] ) dmin += SQR( C[i] - Bmin[i] ); else
        if( C[i] > Bmax[i] ) dmin += SQR( C[i] - Bmax[i] );
    }
    if( dmin <= r2 ) return( TRUE );
    break;

} /* end switch */

return( FALSE );
}
```

```
/*
Transforming Axis-Aligned Bounding Boxes
by Jim Arvo
from "Graphics Gems", Academic Press, 1990
*/

#include "GraphicsGems.h"
/* Transforms a 3D axis-aligned box via a 3x3 matrix and a translation
 * vector and returns an axis-aligned box enclosing the result. */

void Transform_Box( M, T, A, B )
Matrix3  M;      /* Transform matrix.          */
Vector3  T;      /* Translation matrix.          */
Box3     A;      /* The original bounding box.    */
Box3     *B;     /* The transformed bounding box. */
{
    float  a, b;
    float  Amin[3], Amax[3];
    float  Bmin[3], Bmax[3];
    int    i, j;

    /*Copy box A into a min array and a max array for easy reference.*/

    Amin[0] = A.min.x;  Amax[0] = A.max.x;
    Amin[1] = A.min.y;  Amax[1] = A.max.y;
    Amin[2] = A.min.z;  Amax[2] = A.max.z;

    /* Take care of translation by beginning at T. */

    Bmin[0] = Bmax[0] = T.x;
    Bmin[1] = Bmax[1] = T.y;
    Bmin[2] = Bmax[2] = T.z;

    /* Now find the extreme points by considering the product of the */
    /* min and max with each component of M.  */

    for( i = 0; i < 3; i++ )
    for( j = 0; j < 3; j++ )
    {
        a = M.element[i][j] * Amin[j];
        b = M.element[i][j] * Amax[j];
        if( a < b )

            {
                Bmin[i] += a;
                Bmax[i] += b;
            }
        else
            {
                Bmin[i] += b;
                Bmax[i] += a;
            }
    }

    /* Copy the result into the new box. */

    B->min.x = Bmin[0];  B->max.x = Bmax[0];
    B->min.y = Bmin[1];  B->max.y = Bmax[1];
    B->min.z = Bmin[2];  B->max.z = Bmax[2];
}
```

```
#include <math.h>
#include "GraphicsGems.h"

/*=====
 *   Author: Jim Arvo
 *
 *   This routine maps three values (x[0], x[1], x[2]) to a 3x3 rotation
 *   matrix, M.  If x0, x1, and x2 are uniformly distributed random numbers
 *   in [0,1], then M will be a random rotation matrix.
 *
 *   NOTE: This function will not produce UNIFORMLY distributed rotation
 *   matrices as claimed in Gems II.  Watch for a better version in
 *   Gems III.
 *=====*/
void Rand_rotation( x, M )
float x[];
Matrix3 *M;
{
    float  a, b, c, d, s;
    float  z, r, theta, omega;
    float  bb, cc, dd;
    float  ab, ac, ad;
    float  bc, bd, cd;

    /* Use the random variables x[0] and x[1] to determine the axis of
    /* rotation in cylindrical coordinates and the random variable x[2]
    /* to determine the amount of rotation, omega, about this axis.

    z = x[0];
    r = sqrt( 1 - z * z );
    theta = 2.0 * PI * x[1];
    omega = PI * x[2];

    /* Compute the unit quaternion (a,b,c,d) where a is the cosine of
    /* half the rotation angle and the axis vector (b,c,d) is determined
    /* by "r", "theta" and "z" computed above.

    s = sin( omega );
    a = cos( omega );
    b = s * cos( theta ) * r;
    c = s * sin( theta ) * r;
    d = s * z;

    /* Compute all the pairwise products of a, b, c, and d, except a * a.

    bb = b * b;    cc = c * c;    dd = d * d;
    ab = a * b;    ac = a * c;    ad = a * d;
    bc = b * c;    bd = b * d;    cd = c * d;

    /* Construct an orthogonal matrix corresponding to
    /* the unit quaternion (a,b,c,d).

    M->element[0][0] = 1.0 - 2.0 * ( cc + dd );
    M->element[0][1] =      2.0 * ( bc + ad );
    M->element[0][2] =      2.0 * ( bd - ac );

    M->element[1][0] =      2.0 * ( bc - ad );
    M->element[1][1] = 1.0 - 2.0 * ( bb + dd );
    M->element[1][2] =      2.0 * ( cd + ab );
```



```
M->element[2][0] =      2.0 * ( bd + ac );  
M->element[2][1] =      2.0 * ( cd - ab );  
M->element[2][2] = 1.0 - 2.0 * ( bb + cc );  
  
} /* Rand_rotation */
```

```
/*
 *   Author: Jim Arvo
 *
 *   Corrected 11/05/91 to remove a redundant form.  The "C1" form as it
 *   appeared in GemsII was unnecessary.
 */

#include "GraphicsGems.h"

#define P1 (1<< 0)
#define P2 (1<< 1)
#define P3 (1<< 2)
#define P4 (1<< 3)
#define P5 (1<< 4)
#define P6 (1<< 5)
#define RX (1<< 6)
#define RY (1<< 7)
#define RZ (1<< 8)
#define C1 (1<< 9)
#define C2 (1<<10)
#define C3 (1<<11)
#define C4 (1<<12)
#define C5 (1<<13)
#define C6 (1<<14)

/*-----*
 *
 * This function classifies a 3x3 matrix according to its zero structure.
 * It returns an unsigned integer in which each bit signifies a zero
 * structure that describes the given matrix.  If all bits are zero it
 * means the matrix is dense or does not fit any of these 15 forms.
 *
 *
 * Permutations:
 *
 * * 0 0      0 * 0      0 0 *      0 * 0      * 0 0      0 0 *
 * 0 * 0      0 0 *      * 0 0      * 0 0      0 0 *      0 * 0
 * 0 0 *      * 0 0      0 * 0      0 0 *      0 * 0      * 0 0
 *
 * P1          P2          P3          P4          P5          P6
 *
 *
 * Simple Rotations:
 *
 * * 0 0      * 0 *      * * 0
 * 0 * *      0 * 0      * * 0
 * 0 * *      * 0 *      0 0 *
 *
 * RX          RY          RZ
 *
 *
 * Permutations of the simple rotations:
 *
 * 0 0 *      0 * *      0 * 0      * * 0      * 0 *      0 * *
 * * * 0      0 * *      * 0 *      0 0 *      * 0 *      * 0 0
 * * * 0      * 0 0      * 0 *      * * 0      0 * 0      0 * *
 *
 * C1          C2          C3          C4          C5          C6
 *
 *-----*/
unsigned int classify_matrix( M )
```

```
Matrix3 M;
{
    unsigned int form = 0xFFFF;

    /* Classify based on the diagonal elements. */

    if( M.element[0][0] != 0 ) form &= P1 | P5 | RX | RY | RZ | C4 | C5;
    if( M.element[1][1] != 0 ) form &= P1 | P6 | RX | RY | RZ | C1 | C2;
    if( M.element[2][2] != 0 ) form &= P1 | P4 | RX | RY | RZ | C3 | C6;

    /* Classify based on the upper triangular elements. */

    if( M.element[0][1] != 0 ) form &= P2 | P4 | RZ | C2 | C3 | C4 | C6;
    if( M.element[0][2] != 0 ) form &= P3 | P6 | RY | C1 | C2 | C5 | C6;
    if( M.element[1][2] != 0 ) form &= P2 | P5 | RX | C2 | C3 | C4 | C5;

    /* Classify based on the lower triangular elements. */

    if( M.element[1][0] != 0 ) form &= P3 | P4 | RZ | C1 | C3 | C5 | C6;
    if( M.element[2][0] != 0 ) form &= P2 | P6 | RY | C1 | C2 | C3 | C4;
    if( M.element[2][1] != 0 ) form &= P3 | P5 | RX | C1 | C4 | C5 | C6;

    /* If multiple classifications apply, return only one. Doing this */
    /* makes it possible to do a "switch" on the returned value. */

    if( form & P1 ) return( P1 );
    if( form & P2 ) return( P2 );
    if( form & P3 ) return( P3 );
    if( form & P4 ) return( P4 );
    if( form & P5 ) return( P5 );
    if( form & P6 ) return( P6 );
    if( form & RX ) return( RX );
    if( form & RY ) return( RY );
    if( form & RZ ) return( RZ );
    if( form & C1 ) return( C1 );
    if( form & C2 ) return( C2 );
    if( form & C3 ) return( C3 );
    if( form & C4 ) return( C4 );
    if( form & C5 ) return( C5 );
    if( form & C6 ) return( C6 );
    return( form );
}

/*-----*
 *
 * This routine transforms a vector "v" by a matrix "M" whose structure
 * is specified by "form". The result is returned in the vector "w".
 * Note that "v" and "w" cannot be the same vector.
 *
 * All zero elements must still be stored in the matrix "M"; the form
 * argument simply indicates which ones to use. Thus, no storage is
 * saved in this implementation -- just time.
 *-----*/
void sparse_transform( M, form, v, w )
Matrix3 M;
unsigned int form;
Vector3 *v;
Vector3 *w;
{
    switch( form )
```

```
{
case P1:
    w->x = M.element[0][0] * v->x;
    w->y = M.element[1][1] * v->y;
    w->z = M.element[2][2] * v->z;
    break;
case P2:
    w->x = M.element[0][1] * v->x;
    w->y = M.element[1][2] * v->y;
    w->z = M.element[2][0] * v->z;
    break;
case P3:
    w->x = M.element[0][2] * v->x;
    w->y = M.element[1][0] * v->y;
    w->z = M.element[2][1] * v->z;
    break;
case P4:
    w->x = M.element[0][1] * v->x;
    w->y = M.element[1][0] * v->y;
    w->z = M.element[2][2] * v->z;
    break;
case P5:
    w->x = M.element[0][0] * v->x;
    w->y = M.element[1][2] * v->y;
    w->z = M.element[2][1] * v->z;
    break;
case P6:
    w->x = M.element[0][2] * v->x;
    w->y = M.element[1][1] * v->y;
    w->z = M.element[2][0] * v->z;
    break;
case RX:
    w->x = M.element[0][0] * v->x;
    w->y = M.element[1][1] * v->y + M.element[1][2] * v->z;
    w->z = M.element[2][1] * v->y + M.element[2][2] * v->z;
    break;
case RY:
    w->x = M.element[0][0] * v->x + M.element[0][2] * v->z;
    w->y = M.element[1][1] * v->y;
    w->z = M.element[2][0] * v->x + M.element[2][2] * v->z;
    break;
case RZ:
    w->x = M.element[0][0] * v->x + M.element[0][1] * v->y;
    w->y = M.element[1][0] * v->x + M.element[1][1] * v->y;
    w->z = M.element[2][2] * v->z;
    break;
case C1:
    w->x = M.element[0][2] * v->z;
    w->y = M.element[1][0] * v->x + M.element[1][1] * v->y;
    w->z = M.element[2][0] * v->x + M.element[2][1] * v->y;
    break;
case C2:
    w->x = M.element[0][1] * v->y + M.element[0][2] * v->z;
    w->y = M.element[1][1] * v->y + M.element[1][2] * v->z;
    w->z = M.element[2][0] * v->x;
    break;
case C3:
    w->x = M.element[0][1] * v->y;
    w->y = M.element[1][0] * v->x + M.element[1][2] * v->z;
    w->z = M.element[2][0] * v->x + M.element[2][2] * v->z;
    break;
```

```
case C4:
    w->x = M.element[0][0] * v->x + M.element[0][1] * v->y;
    w->y = M.element[1][2] * v->z;
    w->z = M.element[2][0] * v->x + M.element[2][1] * v->y;
    break;
case C5:
    w->x = M.element[0][0] * v->x + M.element[0][2] * v->z;
    w->y = M.element[1][0] * v->x + M.element[1][2] * v->z;
    w->z = M.element[2][1] * v->y;
    break;
case C6:
    w->x = M.element[0][1] * v->y + M.element[0][2] * v->z;
    w->y = M.element[1][0] * v->x;
    w->z = M.element[2][1] * v->y + M.element[2][2] * v->z;
    break;
default:
    w->x = M.element[0][0] * v->x + M.element[0][1] * v->y + M.element[0][2] *
v->z;
    w->y = M.element[1][0] * v->x + M.element[1][1] * v->y + M.element[1][2] *
v->z;
    w->z = M.element[2][0] * v->x + M.element[2][1] * v->y + M.element[2][2] *
v->z;
    break;
} /* switch */

} /* sparse_transform */
```

```
#include <math.h>
#include "GraphicsGems.h"

/*=====
 *  R A N D _ R O T A T I O N      Author: Jim Arvo, 1991
 *
 *  This routine maps three values (x[0], x[1], x[2]) in the range [0,1]
 *  into a 3x3 rotation matrix, M.  Uniformly distributed random variables
 *  x0, x1, and x2 create uniformly distributed random rotation matrices.
 *  To create small uniformly distributed "perturbations", supply
 *  samples in the following ranges
 *
 *      x[0] in [ 0, d ]
 *      x[1] in [ 0, 1 ]
 *      x[2] in [ 0, d ]
 *
 *  where 0 < d < 1 controls the size of the perturbation.  Any of the
 *  random variables may be stratified (or "jittered") for a slightly more
 *  even distribution.
 *=====
void rand_rotation( float x[], Matrix3 *M )
{
    float theta = x[0] * PITIMES2; /* Rotation about the pole (Z).      */
    float phi   = x[1] * PITIMES2; /* For direction of pole deflection. */
    float z     = x[2] * 2.0;      /* For magnitude of pole deflection. */

    /* Compute a vector V used for distributing points over the sphere */
    /* via the reflection I - V Transpose(V).  This formulation of V   */
    /* will guarantee that if x[1] and x[2] are uniformly distributed,  */
    /* the reflected points will be uniform on the sphere.  Note that V */
    /* has length sqrt(2) to eliminate the 2 in the Householder matrix. */

    float r  = sqrt( z );
    float Vx = sin( phi ) * r;
    float Vy = cos( phi ) * r;
    float Vz = sqrt( 2.0 - z );

    /* Compute the row vector S = Transpose(V) * R, where R is a simple */
    /* rotation by theta about the z-axis.  No need to compute Sz since */
    /* it's just Vz.
    */

    float st = sin( theta );
    float ct = cos( theta );
    float Sx = Vx * ct - Vy * st;
    float Sy = Vx * st + Vy * ct;

    /* Construct the rotation matrix ( V Transpose(V) - I ) R, which */
    /* is equivalent to V S - R.
    */

    M->element[0][0] = Vx * Sx - ct;
    M->element[0][1] = Vx * Sy - st;
    M->element[0][2] = Vx * Vz;

    M->element[1][0] = Vy * Sx + st;
    M->element[1][1] = Vy * Sy - ct;
    M->element[1][2] = Vy * Vz;

    M->element[2][0] = Vz * Sx;
    M->element[2][1] = Vz * Sy;
```



```
M->element[2][2] = 1.0 - z;    /* This equals Vz * Vz - 1.0 */  
}
```

```
/*
An Efficient Ray/Polygon Intersection
by Didier Badouel
from "Graphics Gems", Academic Press, 1990

just code, not a procedure.
*/

/* the value of t is computed.
 * i1 and i2 come from the polygon description.
 * V is the vertex table for the polygon and N the
 * associated normal vectors.
 */
P[0] = ray.O[0] + ray.D[0]*t;
P[1] = ray.O[1] + ray.D[1]*t;
P[2] = ray.O[2] + ray.D[2]*t;
u0 = P[i1] - V[0][i1]; v0 = P[i2] - V[0][i2];
inter = FALSE; i = 2;
do {
    /* The polygon is viewed as (n-2) triangles. */
    u1 = V[i-1][i1] - V[0][i1]; v1 = V[i-1][i2] - V[0][i2];
    u2 = V[i ][i1] - V[0][i1]; v2 = V[i ][i2] - V[0][i2];

    if (u1 == 0) {
        beta = u0/u2;
        if ((beta >= 0.) && (beta <= 1.)) {
            alpha = (v0 - beta*v2)/v1;
            inter = ((alpha >= 0.) && (alpha+beta) <= 1.);
        }
    } else {
        beta = (v0*u1 - u0*v1)/(v2*u1 - u2*v1);
        if ((beta >= 0.) && (beta <= 1.)) {
            alpha = (u0 - beta*u2)/u1;
            inter = ((alpha >= 0) && ((alpha+beta) <= 1.));
        }
    }
} while ((!inter) && (++i < poly.n));

if (inter) {
    /* Storing the intersection point. */
    ray.P[0] = P[0]; ray.P[1] = P[1]; ray.P[2] = P[2];
    /* the normal vector can be interpolated now or later. */
    if (poly.interpolate) {
        gamma = 1 - (alpha+beta);
        ray.normal[0] = gamma * N[0][0] + alpha * N[i-1][0] +
            beta * N[i][0];
        ray.normal[1] = gamma * N[0][1] + alpha * N[i-1][1] +
            beta * N[i][1];
        ray.normal[2] = gamma * N[0][2] + alpha * N[i-1][2] +
            beta * N[i][2];
    }
}
return (inter);
```

```
/* Badouel / Wuthrich "Face Connected Line Segment Generation in an
   N-dimensional Space"
*/
/*****
Data structure for n-dimensional line segment generation. Initialized by the
procedure Init() and used by the procedure Incr().
*****/

#define DIM      4                      /* number of dimensions          */

typedef struct {
    int  D[DIM];          /* counter for each of the N dimensions */
    int  N[DIM];          /* increment for each of the N dimensions */
    int  S[DIM];          /* orientation for each of the N dimensions */
    int  cm;              /* common multiple                    */
} Nline;

/*****
Init(): initializes the data structure in order to generate the discrete
path between the point P and the point Q in an n-dimensional space.

This procedure should be called once before using Incr().

Entry:  P      - origin point
        Q      - destination point
        line - line segment data structure
*****/

void Init (P, Q, line)
    int  *P, *Q;
    Nline *line;
{
    int i, v;

    line->cm = 1;
    for (i=0; i<DIM; i++) {
        v = Q[i] - P[i];
        if (v < 0) {
            line->S[i] = -1; line->N[i] = -v;
        } else {
            line->S[i] = 1; line->N[i] = v;
        }
        if (line->N[i] != 0) line->cm *= line->N[i];
    }
    for (i=0; i<DIM; i++) {
        if (line->N[i] == 0)
            line->D[i] = 2*line->cm;
        else {
            line->D[i] = line->cm/line->N[i];
            line->N[i] = 2*line->D[i];
        }
    }
}

/*****
Incr(): generates one step of a discrete segment line in an n-dimensional
space. Indicate the end of the generation with the returned value -1 for
the direction.
*****/
```

The procedure Init() must be called once before using Incr().

Entry: line - line segment data structure

Exit: d - current step direction

s - current step orientation

\*\*\*\*\*/

```
int Incr (line, d, s)
    Nline *line;
    int *d, *s;
{
    int i, v = 2*line->cm;

    *d = -1;
    for (i=0; i<DIM; i++) {
        if (line->D[i] < v) {
            v = line->D[i];
            *d = i;
        }
    }
    line->D[*d] += line->N[*d];
    *s = line->S[*d];
    return *d;
}
```

```
/*
Albers Equal-Area Conic Map Projection
by Paul Bame
from "Graphics Gems", Academic Press, 1990
*/

/*
 * Albers Conic Equal-Area Projection
 * Formulae taken from "Map Projections Used by the U.S.
 * Geological Survey" Bulletin #1532
 *
 * Equation reference numbers and some variable names taken
 * from the reference.
 * To use, call albers setup() once and then albers_project()
 * for each coordinate pair of interest.
*/

#include "GraphicsGems.h"
#include <stdio.h>
#include <math.h>

/*
 * This is the Clarke 1866 Earth spheroid data which is often
 * used by the USGS - there are other spheroids however - see the
 * book.
*/

/*
 * Earth radii in different units */
#define CONST_EradiusKM (6378.2064) /* Kilometers */
#define CONST_EradiusMI (CONST_EradiusKM/1.609) /* Miles */
#define CONST_Ec (0.082271854) /* Eccentricity */
#define CONST_Ecsq (0.006768658) /* Eccentricity squared */

/*
 * To keep things simple, assume Earth radius is 1.0. Projected
 * coordinates (X and Y obtained from albers project ()) are
 * dimensionless and may be multiplied by any desired radius
 * to convert to desired units (usually Kilometers or Miles).
*/
#define CONST_Eradius 1.0

/* Pre-computed variables */
static double middlelon; /* longitude at center of map */
static double bigC, cone_const, r0; /* See the reference */

static
calc_q_msq(lat, qp, msqp)
double lat;
double *qp;
double *msqp;
/*
 * Given latitude, calculate 'q' [eq 3-12]
 * if msqp is != NULL, m^2 [eq. 12-15].
*/
{
    double s, c, es;

    s = sin(lat);
```

```
    es = s * CONST_Ec;

    *qp = (1.0 - CONST_Ecsq) * ((s / (1 - es * es)) -
        (1 / (2 * CONST_Ec)) * log((1 - es) / (1 + es)));

    if (msqp != NULL)
    {
        c = cos(lat);
        *msqp = c * c / (1 - es * es);
    }
}

albers_setup(southlat, northlat, originlon, originlat)
double southlat, northlat;
double originlon;
double originlat;
/*
 * Pre-compute a bunch of variables which are used by the
 * albers_project()
 *
 * southlat      Southern latitude for Albers projection
 * northlat      Northern latitude for Albers projection
 * originlon      Longitude for origin of projected map
 * originlat      Latitude for origin of projected map -
 *                often (northlat + southlat) / 2
 */
{
    double q1, q2, q0;
    double m1sq, m2sq;

    middlelon = originlon;

    calc_q_msq(southlat, &q1, &m1sq);
    calc_q_msq(northlat, &q2, &m2sq);
    calc_q_msq(originlat, &q0, (double *)NULL);

    cone_const = (m1sq - m2sq) / (q2 - q1);
    bigC = m1sq + cone_const * q1;
    r0 = CONST_Eradius * sqrt(bigC - cone_const * q0) / cone_const;
}

/*****/

albers_project(lon, lat, xp, yp)
double lon, lat;
double *xp, *yp;
/*
 * Project lon/lat (in radians) according to albers_setup and
 * return the results via xp, yp. Units of *xp and *yp are same
 * as the units used by CONST_Eradius.
 */
{
    double q, r, theta;
    calc_q_msq(lat, &q, (double *)NULL);
    theta = cone_const * (lon - middlelon);
    r = CONST_Eradius * sqrt(bigC - cone_const * q) / cone_const;
    *xp = r * sin(theta);
    *yp = r0 - r * cos(theta);
}
```



```
}

#ifdef TESTPROGRAM

/*
 * Test value from the USGS book. Because of roundoff, the
 * actual values are printed for visual inspection rather
 * than guessing what constitutes "close enough".
 */
/* Convert a degrees, minutes pair to radians */
#define DEG_MIN2RAD(degrees, minutes) \
    ((double) ((degrees + minutes / 60.0) * M_PI / 180.0))

#define Nlat DEG_MIN2RAD(29, 30)          /* 29 degrees, 30' North Lat */
#define Slat DEG_MIN2RAD(45, 30)
#define Originlat DEG_MIN2RAD(23, 0)
#define Originlon DEG_MIN2RAD(-96, 0) /* West longitude is negative */

#define Testlat DEG_MIN2RAD(35, 0)
#define Testlon DEG_MIN2RAD(-75, 0)

#define TestX 1885.4727
#define TestY 1535.9250

main()
{
    int i;
    double x, y;

    /* Setup is also from USGS book test set */
    albers_setup(Slat, Nlat, Originlon, Originlat);

    albers_project(Testlon, Testlat, &x, &y);
    printf("%.41f, %.41f == %.41f, %.41f/n",
        x * CONST_EradiusKM, y * CONST_EradiusKM,
        TestX, TestY);
}
#endif          /* TESTPROGRAM */
```

```
/*
 * This code calculates the volume, mass, and inertia tensors of
 * superquadric ellipsoids and toroids. The code includes methods
 * to numerically compute gamma functions and beta functions
 */

#include <stdio.h>
#include <math.h>

/*
 * The following function, abgam() is based on a continued fraction numerical
 * method found in Abremowitz and Stegun, Handbook of Mathematical Functions
 */
double abgam (x)
double x;
{
    double gam[10],
           temp;

    gam[0] = 1./ 12.;
    gam[1] = 1./ 30.;
    gam[2] = 53./ 210.;
    gam[3] = 195./ 371.;
    gam[4] = 22999./ 22737.;
    gam[5] = 29944523./ 19733142.;
    gam[6] = 109535241009./ 48264275462.;
    temp = 0.5*log (2*M_PI) - x + (x - 0.5)*log (x)
          + gam[0]/(x + gam[1]/(x + gam[2]/(x + gam[3]/(x + gam[4] /
              (x + gam[5]/(x + gam[6]/x))))));

    return temp;
}

/*
 * A method to compute the gamma() function.
 */
double gamma (x)
double x;
{
    double result,
           abgam ();
    result = exp (abgam (x + 5))/(x*(x + 1)*(x + 2)*(x + 3)*(x + 4));
    return result;
}

/*
 * A method to compute the beta() function.
 */
double beta (m, n)
double m,
        n;
{
    double gamma ();
    return (gamma (m)*gamma (n)/gamma (m + n));
}
```

```
/*
 * A method to compute the volume of a superquadric ellipsoid
 * with axis lengths a1, a2, a3, north-south exponent n
 * and east-west exponent e
 */
double sqellipvol (a1, a2, a3, n, e)
double a1, /* x radius */
a2, /* y radius */
a3, /* z radius */
n, /* north-south param */
e; /* east-west param */
{
    double beta ();
    return ((2./ 3.)*a1*a2*a3*e*n*beta (e/2., e/2.)*beta (n, n/2.));
}

/*
 * A method to compute the volume of a superquadric toroid
 * with axis lengths a1, a2, a3, north-south exponent n
 * east-west exponent e, and hole parameter alpha
 */
double sqtoroidvol (a1, a2, a3, n, e, alpha)
double a1, /* x radius */
a2, /* y radius */
a3, /* z radius */
n, /* north-south param */
e, /* east-west param */
alpha; /* torus hole-size parameter */
{
    return (2.*a1*a2*a3*alpha*e*n*beta (e/2., e/2.)*beta (n/2., n/2.));
}

/*
 * A procedure to print the inertia tensor of a canonical superquadric
 * ellipsoid in its body coordinate system.
 */
void sq_ellipsoid_tensor (a1, a2, a3, e, n)
double a1,
a2,
a3,
e,
n;
{
    double iellip[3][3],
i1E,
i2E,
i3E;
i1E = (2./ 5.)* a1*a1*a1*a2*a3*e*n*beta(3.* e/2., e/2.)*beta(2.* n, n/2.);
i2E = (2./ 5.)* a1*a2*a2*a2*a3*e*n*beta(e/2., 3.* e/2.)*beta(2.* n, n/2.);
i3E = (2./ 5.)* a1*a2*a3*a3*a3*e*n*beta(e/2., e/2.)*beta(n, 3.* n/2.);

iellip[0][0] = 0;
iellip[1][0] = 0;
iellip[2][0] = 0;
iellip[0][1] = 0;
iellip[1][1] = 0;
iellip[2][1] = 0;
```

```
iellip[0][2] = 0;
iellip[1][2] = 0;
iellip[2][2] = 0;

iellip[0][0] = i2E + i3E;
iellip[1][1] = i1E + i3E;
iellip[2][2] = i1E + i2E;

printf ("ellipsoid inertia tensor in body coordinates\n");

printf ("  iellip1 = %f %f %f \n", iellip[0][0], iellip[1][0], iellip[2][0]);
printf ("  iellip2 = %f %f %f \n", iellip[0][1], iellip[1][1], iellip[2][1]);
printf ("  iellip3 = %f %f %f \n", iellip[0][2], iellip[1][2], iellip[2][2]);
}

/*
 * A procedure to print the inertia tensor of a canonical superquadric
 * toroid in its body coordinate system.
 */
void sq_toroid_tensor ( a1, a2, a3, e, n, alpha)
double  a1,
        a2,
        a3,
        e,
        n,
        alpha;
{
    double  itor[3][3],
            i1T,
            i2T,
            i3T;
    i1T = a1*a1*a1*a2*a3*alpha*e*n*beta(3*e/2,e/2)*(2*alpha*alpha*beta(n/2,n/2)+
        3*beta(3*n/2,n/2));
    i2T = a1*a2*a2*a2*a3*alpha*e*n*beta(e/2,3*e/2)*(2*alpha*alpha*beta(n/2,n/2)+
        3*beta(3*n/2,n/2));
    i3T = a1*a2*a3*a3*a3*alpha*e*n*beta(e/2,e/2)*beta(n/2,3*n/2);

    itor[0][0] = 0;
    itor[1][0] = 0;
    itor[2][0] = 0;
    itor[0][1] = 0;
    itor[1][1] = 0;
    itor[2][1] = 0;
    itor[0][2] = 0;
    itor[1][2] = 0;
    itor[2][2] = 0;
    itor[0][0] = i2T + i3T;
    itor[1][1] = i1T + i3T;
    itor[2][2] = i1T + i2T;
    printf ("toroid inertia tensor in body coordinates\n");
    printf ("  itor1  = %f %f %f \n", itor[0][0], itor[1][0], itor[2][0]);
    printf ("  itor2  = %f %f %f \n", itor[0][1], itor[1][1], itor[2][1]);
    printf ("  itor3  = %f %f %f \n", itor[0][2], itor[1][2], itor[2][2]);
}

/*
 * A procedure to print the inertia tensor components in world coordinates,
 * given an inertia tensor in body coordinates, and the 3x3 rotation matrix
 * which rotates body vectors into world coordinates.
```

```
*/
void iworld (Ibody, R)
double Ibody[3][3],
       R[3][3];
{
    double Iworld[3][3];
    Iworld[0][0] =
        (R[0][0]*Ibody[0][0]+R[0][1]*Ibody[1][0]+R[0][2]*Ibody[2][0])*R[0][0] +
        (R[0][0]*Ibody[0][1]+R[0][1]*Ibody[1][1]+R[0][2]*Ibody[2][1])*R[0][1] +
        (R[0][0]*Ibody[0][2]+R[0][1]*Ibody[1][2]+R[0][2]*Ibody[2][2])*R[0][2];

    Iworld[1][0] =
        (R[1][0]*Ibody[0][0]+R[1][1]*Ibody[1][0]+R[1][2]*Ibody[2][0])*R[0][0]+
        (R[1][0]*Ibody[0][1]+R[1][1]*Ibody[1][1]+R[1][2]*Ibody[2][1])*R[0][1]+
        (R[1][0]*Ibody[0][2]+R[1][1]*Ibody[1][2]+R[1][2]*Ibody[2][2])*R[0][2];

    Iworld[2][0] =
        (R[2][0]*Ibody[0][0]+R[2][1]*Ibody[1][0]+R[2][2]*Ibody[2][0])*R[0][0]+
        (R[2][0]*Ibody[0][1]+R[2][1]*Ibody[1][1]+R[2][2]*Ibody[2][1])*R[0][1]+
        (R[2][0]*Ibody[0][2]+R[2][1]*Ibody[1][2]+R[2][2]*Ibody[2][2])*R[0][2];

    Iworld[0][1] =
        (R[0][0]*Ibody[0][0]+R[0][1]*Ibody[1][0]+R[0][2]*Ibody[2][0])*R[1][0]+
        (R[0][0]*Ibody[0][1]+R[0][1]*Ibody[1][1]+R[0][2]*Ibody[2][1])*R[1][1]+
        (R[0][0]*Ibody[0][2]+R[0][1]*Ibody[1][2]+R[0][2]*Ibody[2][2])*R[1][2];

    Iworld[1][1] =
        (R[1][0]*Ibody[0][0]+R[1][1]*Ibody[1][0]+R[1][2]*Ibody[2][0])*R[1][0]+
        (R[1][0]*Ibody[0][1]+R[1][1]*Ibody[1][1]+R[1][2]*Ibody[2][1])*R[1][1]+
        (R[1][0]*Ibody[0][2]+R[1][1]*Ibody[1][2]+R[1][2]*Ibody[2][2])*R[1][2];

    Iworld[2][1] =
        (R[2][0]*Ibody[0][0]+R[2][1]*Ibody[1][0]+R[2][2]*Ibody[2][0])*R[1][0]+
        (R[2][0]*Ibody[0][1]+R[2][1]*Ibody[1][1]+R[2][2]*Ibody[2][1])*R[1][1]+
        (R[2][0]*Ibody[0][2]+R[2][1]*Ibody[1][2]+R[2][2]*Ibody[2][2])*R[1][2];

    Iworld[0][2] =
        (R[0][0]*Ibody[0][0]+R[0][1]*Ibody[1][0]+R[0][2]*Ibody[2][0])*R[2][0]+
        (R[0][0]*Ibody[0][1]+R[0][1]*Ibody[1][1]+R[0][2]*Ibody[2][1])*R[2][1]+
        (R[0][0]*Ibody[0][2]+R[0][1]*Ibody[1][2]+R[0][2]*Ibody[2][2])*R[2][2];

    Iworld[1][2] =
        (R[1][0]*Ibody[0][0]+R[1][1]*Ibody[1][0]+R[1][2]*Ibody[2][0])*R[2][0]+
        (R[1][0]*Ibody[0][1]+R[1][1]*Ibody[1][1]+R[1][2]*Ibody[2][1])*R[2][1]+
        (R[1][0]*Ibody[0][2]+R[1][1]*Ibody[1][2]+R[1][2]*Ibody[2][2])*R[2][2];

    Iworld[2][2] =
        (R[2][0]*Ibody[0][0]+R[2][1]*Ibody[1][0]+R[2][2]*Ibody[2][0])*R[2][0]+
        (R[2][0]*Ibody[0][1]+R[2][1]*Ibody[1][1]+R[2][2]*Ibody[2][1])*R[2][1]+
        (R[2][0]*Ibody[0][2]+R[2][1]*Ibody[1][2]+R[2][2]*Ibody[2][2])*R[2][2];

    printf ("toroid inertia tensor in body coordinates\n");
    printf (" Iworld1 = %f %f %f \n", Iworld[0][0], Iworld[1][0], Iworld[2][0]);
    printf (" Iworld2 = %f %f %f \n", Iworld[0][1], Iworld[1][1], Iworld[2][1]);
    printf (" Iworld3 = %f %f %f \n", Iworld[0][2], Iworld[1][2], Iworld[2][2]);
}

/*
* sgn(x) returns -1.0 or 1.0 for speed.
* sgn(x) can return 0.0 on zero if you wish, depending on
* your convention
```

```
*/
double sgn(x)
double x;
{
    if (x <= 0.0) return (-1.0);
    else return(1.0);
}

/*
 * computes position on the surface of a superquadric ellipsoid
 * v goes from -Pi/2 to Pi/2; u goes from -Pi to Pi.
 *
 */
void sqellipsoidposn(a1,a2,a3,n,e,alpha,u,v)
double a1,a2,a3,n,e,alpha,u,v;
{
    double cu, su, cv, sv, x,y,z;
    cu = cos(u);
    su = sin(u);
    cv = cos(v);
    sv = sin(v);

    x = a1*(alpha + pow(cv,n))*pow(cu,e)*sgn(cu)*sgn(cv);
    y = a2*(alpha + pow(cv,n))*pow(su,e)*sgn(su)*sgn(cv);
    z = a3*pow(sv,n)*sgn(sv);
}

/*
 * computes position on the surface of a superquadric toroid
 * u and v go from -Pi to Pi
 */
void sqtoroidposn(a1,a2,a3,n,e,u,v)
double a1,a2,a3,n,e,u,v;
{
    double cu, su, cv, sv, x,y,z;
    cu = cos(u);
    su = sin(u);
    cv = cos(v);
    sv = sin(v);

    x = a1*pow(cv,n)*pow(cu,e)*sgn(cu)*sgn(cv);
    y = a2*pow(cv,n)*pow(su,e)*sgn(su)*sgn(cv);
    z = a3*pow(sv,n)*sgn(sv);
}

/*
 * A procedure to test some of the above code
 *
 */
main () {
    printf (" gamma(1)= 1.0 = %12.10lf\n", gamma (1.0));
    printf (" gamma(1/2)^2= Pi =%12.10lf\n", gamma (0.5)*gamma (0.5));
    printf (" gamma(2)= 1.0 = %12.10lf\n", gamma (2.0));
    printf (" gamma(3)= 2.0 = %12.10lf\n", gamma (3.0));
    printf (" gamma(4)= 6.0 = %12.10lf\n", gamma (4.0));
    printf("\n");
    printf ("beta(1,1)= 1.0 = %12.10lf\n", beta (1.0, 1.0));
}
```



```
printf ("beta(1,1/2)= 2.0 = %12.10lf\n", beta (1.0, 0.5));
printf ("beta(1/2,1/2)= Pi = %12.10lf\n", beta (0.5, 0.5));
printf("\n");
printf ("sq ellipsoid volume/pi= 4/3 = %12.10lf\n",
        sqellipvol(1., 1., 1., 1., 1.)/M_PI);
printf ("sq toroid volume/pi^2 = 2.0 = %12.10lf\n",
        sqtoroidvol (1., 1., 1., 1., 1., 1.)/M_PI/M_PI);
printf("\n");
sq_ellipsoid_tensor (1., 1., 1., 1., 1.);
sq_toroid_tensor (1., 1., 1., 1., 1., 1.);
```

}

```
#include <string.h>
/*****
The following code implements motion blur on a HP9000 Series 800 computer,
including computing on fields and compensation for lack of subpixel positioning.
The anti-flicker filter discussed in Section 4 is not included.
In the interests of brevity and clarity, the code presented here is
not the most efficient possible, but note:
```

1. Rather than initializing the image accumulation buffer to 0 before each frame in `mb_frame()`, the buffer can be initialized to the value of the first subframe.
2. If the number of subframes per field is a power of two, then shifting the accumulated result is faster than dividing.
3. If more processing must be done to the image in `output()`, such as conversion from RGB to YUV, it is most efficient to do the processing within the `mb_average()` routine, rather than to make another pass over the same memory.
4. Results of the averaging are stored in place in a single accumulation buffer. Depending on the additional processing that must be done by the `output()` procedure, it may be advantageous to store the final result in a different buffer instead, such as an array of unsigned char rather than unsigned short.

```
*****/
#define FB_STRIDE 2048    /* framebuffer stride -- pixels in a scan line */
#define VID_XRES  640    /* video x resolution */
#define VID_YRES  486    /* video y resolution */

/* framebuffer address -- pixels stored in unsigned ints */
/* we assume frame is initialized to point to the memory-mapped framebuffer */
static unsigned int *frame;
#define BB(v)  ( ((v) <<24)>>24 )    /* extract blue byte from fb pixel */
#define BG(v)  ( ((v) <<16)>>24 )    /* extract green byte from fb pixel */
#define BR(v)  ( ((v) << 8)>>24 )    /* extract red byte from fb pixel */

/* accumulation buffer -- pixels stored as r,g,b, r,g,b,... */
static unsigned short acc[VID_XRES*VID_YRES*3];

void mb_average(), mb_accumulate();

/*****
Compute and output a motion blurred frame.
```

External Procedures:

```
void shift_image(int x,int y) - adjusts the viewing transformation to
                               render the image shifted by an integral
                               number of pixels in x and y

void draw(double t)           - renders a subframe at time t

void output(unsigned short *acc,int w,int h) - outputs an image (e.g.
                                              records a frame of the
                                              animation onto the output
                                              device)
```

Entry:  
t0            - frame start time

```
    delta    - shutter open time
    n        - number of subframes
*****/
void mb_frame(t0,delta,n)
double t0;
double delta;
int n;
{
    int i;
    int nfield = n/2; /* number of subframes per field */
    int field = 1;    /* current field; first 1, then 0 */

    /* pixel offsets for 16 subframes/field, gives triangle filter in x and y */
    static int x_off[] = {0,1,0,1, -1,0,-1,0,  0, 1, 0,1, -1, 0,-1,0};
    static int y_off[] = {0,1,1,0,  0,1, 1,0,  0,-1,-1,0,  0,-1,-1,0};
#define NOFFS ( sizeof(x_off)/sizeof(x_off[0]) )

    /* clear accumulation buffer */
    memset((void *)acc,0,sizeof(acc));

    for (i = 0; i < n; i++) {
        shift_image(x_off[(i%nfield)%NOFFS],y_off[(i%nfield)%NOFFS]);
        draw(t0 + i*delta/(n-1));
        if (i == nfield-1 || i == n-1) {
            mb_average(nfield,field);
            field = !field;
        } else {
            mb_accumulate(field);
        }
    }

    /* output final image */
    output(acc,VID_XRES,VID_YRES);
}
```

\*\*\*\*\*  
Add another subframe to the accumulation buffer, computing on fields.

```
Entry:
    field    - field number (0 or 1)
*****/
void mb_accumulate(field)
int field;
{
    register unsigned int *iptr;
    register unsigned short *optr;
    register int i,j;

    /* shift input and output pointers by one scan line if field 1 */
    if (field) {
        iptr = frame + FB_STRIDE;
        optr = acc + 3*VID_XRES;
    } else {
        iptr = frame;
        optr = acc;
    }

    /* add in field to accumulation buffer */
    for (i = 0; i < VID_YRES; i+=2) {
        for (j = 0; j < VID_XRES; j+=4) {
            /* process 4 pixels, skipping every other one */
```

```
    register unsigned int v1 = iptr[0];
    register unsigned int v3 = iptr[2];
    register unsigned int v5 = iptr[4];
    register unsigned int v7 = iptr[6];
```

```
    optr[0] += BR(v1); optr[1] += BG(v1); optr[2] += BB(v1);
    optr[3] += BR(v3); optr[4] += BG(v3); optr[5] += BB(v3);
    optr[6] += BR(v5); optr[7] += BG(v5); optr[8] += BB(v5);
    optr[9] += BR(v7); optr[10] += BG(v7); optr[11] += BB(v7);
```

```
    iptr += 8;
    optr += 12;
```

```
    }
    iptr += 4*FB_STRIDE-VID_XRES*2; /* process every fourth scan line */
    optr += 3*VID_XRES;             /* skip to next field scan line */
}
```

\*\*\*\*\*  
Add another subframe and divide by the number of field subframes.  
The result is placed back into the accumulation buffer.

Entry:  
    n           - number of subframes in a field  
    field       - field number (0 or 1)  
\*\*\*\*\*/

void mb\_average(n,field)

```
int n;
int field;
{
    register unsigned int *iptr;
    register unsigned short *optr;
    register int i,j;

    /* shift input and output pointers by one scan line if field 1 */
    if (field) {
        iptr = frame + FB_STRIDE;
        optr = acc + 3*VID_XRES;
    } else {
        iptr = frame;
        optr = acc;
    }
}
```

```
/* add in field to accumulation buffer and divide */
for (i = 0; i < VID_YRES; i+=2) {
    for (j = 0; j < VID_XRES; j+=4) {
        /* process 4 pixels, skipping every other one */
        register unsigned int v1 = iptr[0];
        register unsigned int v3 = iptr[2];
        register unsigned int v5 = iptr[4];
        register unsigned int v7 = iptr[6];
```

```
        optr[0] = (optr[0]+BR(v1))/n;
        optr[1] = (optr[1]+BG(v1))/n;
        optr[2] = (optr[2]+BB(v1))/n;
```

```
        optr[3] = (optr[3]+BR(v3))/n;
        optr[4] = (optr[4]+BG(v3))/n;
        optr[5] = (optr[5]+BB(v3))/n;
```

```
        optr[6] = (optr[6]+BR(v5))/n;
```

```
        optr[7] = (optr[7]+BG(v5))/n;
        optr[8] = (optr[8]+BB(v5))/n;

        optr[9] = (optr[9]+BR(v7))/n;
        optr[10] = (optr[10]+BG(v7))/n;
        optr[11] = (optr[11]+BB(v7))/n;

        iptr += 8;
        optr += 12;
    }
    iptr += 4*FB_STRIDE-VID_XRES*2; /* process every fourth scan line */
    optr += 3*VID_XRES;             /* skip to next field scan line */
}
```

```
/*
 * ANSI C code from the article
 * "Centroid of a Polygon"
 * by Gerard Bashein and Paul R. Detmer,
 *   (gb@locke.hs.washington.edu, pdetmer@u.washington.edu)
 * in "Graphics Gems IV", Academic Press, 1994
 */

/*****
polyCentroid: Calculates the centroid (xCentroid, yCentroid) and area
of a polygon, given its vertices (x[0], y[0]) ... (x[n-1], y[n-1]). It
is assumed that the contour is closed, i.e., that the vertex following
(x[n-1], y[n-1]) is (x[0], y[0]). The algebraic sign of the area is
positive for counterclockwise ordering of vertices in x-y plane;
otherwise negative.

Returned values:  0 for normal execution;  1 if the polygon is
degenerate (number of vertices < 3);  and 2 if area = 0 (and the
centroid is undefined).
*****/
int polyCentroid(double x[], double y[], int n,
                 double *xCentroid, double *yCentroid, double *area)
{
    register int i, j;
    double ai, atmp = 0, xtmp = 0, ytmp = 0;
    if (n < 3) return 1;
    for (i = n-1, j = 0; j < n; i = j, j++)
    {
        ai = x[i] * y[j] - x[j] * y[i];
        atmp += ai;
        xtmp += (x[j] + x[i]) * ai;
        ytmp += (y[j] + y[i]) * ai;
    }
    *area = atmp / 2;
    if (atmp != 0)
    {
        *xCentroid = xtmp / (3 * atmp);
        *yCentroid = ytmp / (3 * atmp);
        return 0;
    }
    return 2;
}
```

```

/*****
This program illustrates the calculation of delta form-factors for the cubic tetrahedral
algorithm. The delta form-factor of each shaded cell in Fig. 2
is computed and displayed.
*****/

#include <math.h>

#define MIN    -2.0          /* Minimum value of a ct coordinate */
#define MAX     1.0          /* Maximum value of a ct coordinate */
#define SUBDIV  8            /* Number of cell subdivisions */

/* Calculate the form-factor of a cell centered at (u,v) with area a. */

float formFactor(u, v, a)
    float u, v, a;
{
    float r = u*u + v*v + 1;
    return (a * (u + v + 1) / (M_PI * r*r * sqrt(3.0)));
}

main ()
{
    int    left, right, top, bottom,          /* Cell index boundaries */
           row, column;                      /* Current cell indices */

    float   delta, halfDelta,                /* Cell sizes */
           area, halfArea,                  /* Cell areas */
           y, z;                            /* Cell center location */

    /* Initialize index values */

    left = 1;  right = SUBDIV;
    top  = 1;  bottom = (SUBDIV + 1) / 2;
    row  = 1;  column = 1;

    /* Initialize cell values */

    delta = (MAX - MIN) / SUBDIV;  halfDelta = delta / 2.0;
    area  = delta * delta;          halfArea  = area / 2.0;
    y = z = MAX - halfDelta;

    /* Calculate and display delta form factors */

    for (row = top; row <= bottom; row++) {
        for (column = left; column < right; column++) {
            printf("Cell(%0d,%0d) = %f\n", row, column, formFactor(y, z, area));
            y -= delta;
        }
        printf("Cell(%0d,%0d) = %f\n", row, column,
            formFactor(y+halfDelta, z+halfDelta, halfArea));
        left++;  right--;
        y = z -= delta;
    }
}
```

```
/*  
Hemispherical Projection of a Triangle
```

Buming Bian

UT System---Center for High Performance Computing

Austin, Texas

```
*/  
#include <math.h>  
#include <stdio.h>  
  
#include "GraphicsGems.h"  
#include "GraphicsGems.c"  
  
#define HOMON 4  
#define PONPATCH 3  
#define ERR 1.0e-10  
#define ONE 1 - ERR  
/* types of structures */  
typedef struct {  
    Point3 pt[PONPATCH]; /* Vertices coordinates */  
    double ref; /* Reflectivity of the patch */  
    double trs; /* transmittance of the patch */  
} Patch;  
  
/* return a patch normal. */  
Vector3 *patchnorm(suf)  
Patch *suf;  
{  
    Vector3 *vt1 = NEWTYPE(Vector3);  
    Vector3 *vt2 = NEWTYPE(Vector3);  
    Vector3 *norm = NEWTYPE(Vector3);  
    vt1 = V3Sub(&suf->pt[1], &suf->pt[0], vt1);  
    vt2 = V3Sub(&suf->pt[2], &suf->pt[0], vt2);  
    norm = V3Normalize(V3Cross(vt1, vt2, norm));  
    free(vt1); free(vt2);  
    return(norm);  
}  
/*****  
* Compute the form factor. Homogeneous transformation is applied to the two *  
* input Patches such that the center of patch suf1 located at origin with *  
* its normal coinciding with z axis. A hemisphere is constructed above the xy*  
* plane, patch suf2 is projected on the hemisphere to form a spherical *  
* triangle. This spherical triangle is again projected on the xy plane and *  
* the projected area is the form factor of patch suf2 to patch suf1. *  
*****/  
double hemisphere(suf1, suf2)  
Patch *suf1, *suf2;  
{  
    int i;  
    double out, hm[HOMON][HOMON], projectarea();  
    Vector3 *vect[PONPATCH];  
    void patchtransf(), ghmgm();  
  
    /* use patch suf1 to form the homogeneous transformation matrix */  
  
    ghmgm(suf1, hm);  
    /* apply the transformation to patch suf2 */
```



```
    patchtransf(hm, suf2);
/* represent the vertices with vectors */
    for (i = 0; i < PONPATCH; i++)
        vect[i] = V3New(suf2->pt[i].x, suf2->pt[i].y, suf2->pt[i].z);
/* Normalize the form factor so the all sub-total of the form factor */
    out = projectarea(vect);
    for (i = 0; i < PONPATCH; i++)
        free(vect[i]);
    return(ABS(out/PI));
}
/* Calculate the form factor by adding up all the ellipse sectors formed by two
 * neighbor points */
double projectarea(vect)
Vector3 *vect[PONPATCH];
{
    int i;
    double out= 0, twopntarea();
    for (i = 0; i < PONPATCH; i++)
        out += twopntarea(vect[i], vect[(i+1)%PONPATCH]);
    return(out);
}
/* Calculate the area of the ellipsoid triangle */
double twopntarea(vect1, vect2)
Vector3 *vect1, *vect2;
{
    double a, b;
    double sum, cos_sita, halvesita, halfa, d1, d2;
    double arccos(), arctan();
    Vector3 *major, *norm, *v;
/* Find out the normal vector of the great circle */
    major = V3New(0.0, 0.0, 1.0);
    norm = V3Duplicate(vect1);
    v = NEWTYPE(Vector3);
    norm = V3Normalize(V3Cross(norm, vect2, v));
    cos_sita = ABS(V3Dot(major, norm));
    if (cos_sita < ERR){
/* if the normal vector is perpendicular to z axis, form factor is zero */
        sum = 0.0;
        free(norm);
        free(major);
        return(sum);
    }
    halvesita = cos_sita / 2.0;
    halfa = PI * halvesita;
    if (cos_sita < ONE){
/* project the great circle on to the equatorial plane and calculate the form factor */
        major = V3Normalize(V3Cross(major, norm, v));
        norm->x *= -1; norm->y *= -1; norm->z = 0;
        d1 = sqrt(V3Dot(vect1, vect1));
        d2 = sqrt(V3Dot(vect2, vect2));
        a = V3Dot(major, vect1) / d1;
        if (ABS(a) > 1.0) a /= ABS(a);
        b = V3Dot(norm, vect1);
        sum = arccos(a, b);
        a = V3Dot(major, vect2) / d2;
        if (ABS(a) > 1.0) a /= ABS(a);
        b = V3Dot(norm, vect2);
        sum -= arccos(a, b);
    }
    else{
/* if the normal vector is parallel to z axis, form factor is maximum */
```

```
        sum = arctan(vect1->x, vect1->y);
        sum -= arctan(vect2->x, vect2->y);
    }
    sum *= halfsita;
    free(norm); free(major); free(v);
    if (sum > halfa) sum -= 2.0 * halfa;
    else if (sum < -halfa) sum += 2.0 * halfa;
    return(sum);
}

double arccos(x, y)
double x, y;
{
    double angle;
    angle = acos(x);
    if (y < 0) angle = 2 * PI - angle;
    return(angle);
}

double arctan(x, y)
double x, y;
{
    double angle;
    angle = atan2(y, x);
    if (y < 0) angle = 2 * PI + angle;
    return(angle);
}

/* Derive the homogeneous transformation matrix for the given patch */
void ghmgn(suf, hm)
Patch *suf;
double hm[HOMON][HOMON];
{
    int i, j;
    Vector3 *v1;
    Point3 *pnt = NEWTYPE(Vector3);
    double d;
    double tt[HOMON][HOMON], tr1[HOMON][HOMON];
    double tr2[HOMON][HOMON], rr[HOMON][HOMON];
    void mtxprd();

    pnt->x = (suf->pt[0].x+suf->pt[1].x+suf->pt[2].x)/PONPATCH;
    pnt->y = (suf->pt[0].y+suf->pt[1].y+suf->pt[2].y)/PONPATCH;
    pnt->z = (suf->pt[0].z+suf->pt[1].z+suf->pt[2].z)/PONPATCH;
    for (i = 0; i < HOMON; i++)
        for (j = 0; j < HOMON; j++)
            if (i == j){
                tt[i][j] = 1.0; tr1[i][j] = 1.0; tr2[i][j] = 1.0; }
            else{
                tt[i][j] = 0; tr1[i][j] = 0; tr2[i][j] = 0; }
    v1 = patchnorm(suf);
    tt[3][0] = -pnt->x; tt[3][1] = -pnt->y; tt[3][2] = -pnt->z;
    free(pnt);
    d = sqrt(SQR(v1->y)+SQR(v1->z));
    if (d > ERR){
        tr1[0][0] = 1;
        tr1[3][3] = 1;
        tr1[1][1] = v1->z/d;
        tr1[2][2] = v1->z/d;
        tr1[2][1] = -v1->y/d;
        tr1[1][2] = v1->y/d;
```

```
    }
    mtxprd(tt, tr1, rr);
    tr2[0][0] = d; tr2[2][2] = d; tr2[2][0] = -v1->x; tr2[0][2] = v1->x;
    free(v1);
    mtxprd(rr, tr2, hm);
}

/* Transform a given patch */
void patchtransf(hm, suf)
double hm[HOMON][HOMON];
Patch *suf;
{
    int i;
    void pntttransf();
    for (i = 0; i < PONPATCH; i++)
        pntttransf(hm, &suf->pt[i]);
}

/* Transform a given point */
void pntttransf(hm, pnt)
double hm[HOMON][HOMON];
Point3 *pnt;
{
    double hi[HOMON], ho[HOMON];
    void transform();
    hi[0] = pnt->x; hi[1] = pnt->y; hi[2] = pnt->z; hi[3] = 1.;
    transform(hm, hi, ho);
    pnt->x = ho[0] / ho[3]; pnt->y = ho[1] / ho[3]; pnt->z = ho[2] / ho[3];
}

/* Transform a given vector */
void vectttransf(hm, vect)
double hm[HOMON][HOMON];
Vector3 *vect;
{
    double hi[HOMON], ho[HOMON];
    void transform();
    hi[0] = vect->x; hi[1] = vect->y; hi[2] = vect->z; hi[3] = 0.;
    transform(hm, hi, ho);
    vect->x = ho[0]; vect->y = ho[1]; vect->z = ho[2];
}





/* Transform a given vector */
void transform(hm, hi, ho)
double hm[HOMON][HOMON];
double hi[HOMON], ho[HOMON];
{
    int j, k;
    for (j = 0; j < HOMON; j++){
        ho[j] = 0;
        for (k = 0; k < HOMON; k++)
            ho[j] += hi[k] * hm[k][j];
    }
}

/* Multiple two matrices */
void mtxprd(h1, h2, h3)
double h1[HOMON][HOMON], h2[HOMON][HOMON], h3[HOMON][HOMON];
{
    int i, j, k;
```

```
for (i = 0; i < HOMON; i++)
  for (j = 0; j < HOMON; j++){
    h3[i][j] = 0;
    for (k = 0; k < HOMON; k++)
      h3[i][j] += h1[i][k] * h2[k][j];
  }
}
```

# Index of

## /pubs/tog/GraphicsGems/gemsv/ch3-6/

Name	Last modified	Size	Description
 <a href="#">_Parent Directory</a>			
 <a href="#">_README</a>	29-Jun-00 08:22	1K	
 <a href="#">_axd.c</a>	29-Jun-00 08:22	4K	
 <a href="#">_axd.h</a>	29-Jun-00 08:22	1K	

NOTE: These routines use the include files found in  
../ch7-7/mactbox/real.h  
../ch7-7/mactbox/mat.h

```
/* ----- */
AXD.C :

This package provides an implementation of 6 different algorithms
for doing procedural axial deformations.

by Carole Blanc (4 June 1994)

"A Generic Implementation of Axial Deformation Techniques"
in Graphics Gems V (edited by A. Paeth), Academic Press
/* ----- */

/*
** This package uses the "Toolbox of Macros Functions for Computer Graphics"
** which provides files : tool.h, real.h, uint.h, sint.h, vec?.h and mat?.h
*/

#include "axd.h"
#include "../ch7-7/mactbox/mat3.h"

/*
** Each "local_" routines inputs/outputs the following arguments
**
** Input:  Point = coordinates of the point in the local frame
**         Shape = shape function of the deformation
**         Ampli = amplitude of the deformation
** Output: Point = coordinates of the deformed point in the local frame
**
** Each "world_" routines inputs/outputs the following arguments
**
** Input:  Point = coordinates of the point in the world frame
**         Frame = local frame in which the deformation is applied
**         Shape = shape function of the deformation
**         Ampli = amplitude of the deformation
** Output: Point = coordinates of the deformed point in the world frame
**
** Note: The "Frame" argument must be initialized by MAKE_FRAME3 (see "mat3.h")
*/

/*
** pinch : Scale the x coordinate of the object according to z
*/

void local_pinch (realvec3 *Point, shape Shape, real Ampli)
{
    Point->x *= 1.0 - Ampli * Shape (Point->z);
}

void world_pinch (realvec3 *Point, frame3 Frame, shape Shape, real Ampli)
{
    realvec3 Tmp;

    LOCAL_FRAME3 (Tmp, Frame, *Point);
    local_pinch (&Tmp, Shape, Ampli);
    WORLD_FRAME3 (*Point, Frame, Tmp);
}

/*
** taper : Scale the polar radius of the object according to z
*/
```

```
void local_taper (realvec3 *Point, shape Shape, real Ampli)
{
    register real Tmp;

    Tmp = 1.0 - Ampli * Shape (Point->z); Point->x *= Tmp; Point->y *= Tmp;
}

void world_taper (realvec3 *Point, frame3 Frame, shape Shape, real Ampli)
{
    realvec3 Tmp;

    LOCAL_FRAME3 (Tmp, Frame, *Point);
    local_taper (&Tmp, Shape, Ampli);
    WORLD_FRAME3 (*Point, Frame, Tmp);
}

/*
** mould : Scale the polar radius of the object according to the polar angle
*/

void local_mould (realvec3 *Point, shape Shape, real Ampli)
{
    register real Tmp;

    Tmp = atan2 (Point->y, Point->x) / PI;
    Tmp = 1.0 - Ampli * Shape (Tmp); Point->x *= Tmp; Point->y *= Tmp;
}

void world_mould (realvec3 *Point, frame3 Frame, shape Shape, real Ampli)
{
    realvec3 Tmp;

    LOCAL_FRAME3 (Tmp, Frame, *Point);
    local_mould (&Tmp, Shape, Ampli);
    WORLD_FRAME3 (*Point, Frame, Tmp);
}

/*
** twist : Scale the polar angle of the object according to z
*/

void local_twist (realvec3 *Point, shape Shape, real Ampli)
{
    register real Tmp, Cos, Sin;

    Tmp = PI * Ampli * Shape (Point->z);
    Cos = cos (Tmp); Sin = sin (Tmp); Tmp = Point->x;
    Point->x = Cos * Tmp - Sin * Point->y;
    Point->y = Sin * Tmp + Cos * Point->y;
}

void world_twist (realvec3 *Point, frame3 Frame, shape Shape, real Ampli)
{
    realvec3 Tmp;

    LOCAL_FRAME3 (Tmp, Frame, *Point);
    local_twist (&Tmp, Shape, Ampli);
    WORLD_FRAME3 (*Point, Frame, Tmp);
}

/*
```



```
/** shear : Translate the z axis of the object along x according to z
 */
```

```
void local_shear (realvec3 *Point, shape Shape, real Ampli)
{
    Point->x += Ampli * Shape (Point->z);
}
```

```
void world_shear (realvec3 *Point, frame3 Frame, shape Shape, real Ampli)
{
    realvec3 Tmp;

    LOCAL_FRAME3 (Tmp, Frame, *Point);
    local_shear (&Tmp, Shape, Ampli);
    WORLD_FRAME3 (*Point, Frame, Tmp);
}
```

```
/*
** bend : Rotate the z axis of the object around y according to z
*/
```

```
void local_bend (realvec3 *Point, shape Shape, real Ampli)
{
    register real Tmp, Cos, Sin;

    Tmp = PI * Ampli * Shape (Point->z);
    Cos = cos (Tmp); Sin = sin (Tmp); Tmp = Point->z;
    Point->z = Cos * Tmp - Sin * Point->x;
    Point->x = Sin * Tmp + Cos * Point->x;
}
```

```
void world_bend (realvec3 *Point, frame3 Frame, shape Shape, real Ampli)
{
    realvec3 Tmp;

    LOCAL_FRAME3 (Tmp, Frame, *Point);
    local_bend (&Tmp, Shape, Ampli);
    WORLD_FRAME3 (*Point, Frame, Tmp);
}
```

```
/* ----- */
```

```
/* ----- */
AXD.H :

This package provides an implementation of 6 different algorithms
for doing procedural axial deformations.

by Carole Blanc (4 June 1994)

"A Generic Implementation of Axial Deformation Techniques"
in Graphics Gems V (edited by A. Paeth), Academic Press
/* ----- */

#ifndef _AXD_
#define _AXD_

/*
** This package uses the "Toolbox of Macros Functions for Computer Graphics"
** which provides files : tool.h, real.h, uint.h, sint.h, vec?.h and mat?.h
*/

#include "../ch7-7/mactbox/real.h"

typedef real (*shape) (real);

extern void local_pinch (rv3 *Point, shape Shape, real Ampli);
extern void world_pinch (rv3 *Point, frame3 Frame, shape Shape, real Ampli);
extern void local_taper (rv3 *Point, shape Shape, real Ampli);
extern void world_taper (rv3 *Point, frame3 Frame, shape Shape, real Ampli);
extern void local_mould (rv3 *Point, shape Shape, real Ampli);
extern void world_mould (rv3 *Point, frame3 Frame, shape Shape, real Ampli);
extern void local_twist (rv3 *Point, shape Shape, real Ampli);
extern void world_twist (rv3 *Point, frame3 Frame, shape Shape, real Ampli);
extern void local_shear (rv3 *Point, shape Shape, real Ampli);
extern void world_shear (rv3 *Point, frame3 Frame, shape Shape, real Ampli);
extern void local_bend (rv3 *Point, shape Shape, real Ampli);
extern void world_bend (rv3 *Point, frame3 Frame, shape Shape, real Ampli);

#endif

/* ----- */
```

```
/*
 * C code from the article
 * "An Implicit Surface Polygonizer"
 * by Jules Bloomenthal, jbloom@beauty.gmu.edu
 * in "Graphics Gems IV", Academic Press, 1994
 */

/* implicit.c
 * an implicit surface polygonizer, translated from Mesa
 * applications should call polygonize()
 *
 * to compile a test program for ASCII output:
 * cc implicit.c -o implicit -lm
 *
 * to compile a test program for display on an SGI workstation:
 * cc -DSGIGFX implicit.c -o implicit -lgl_s -lm
 *
 * Authored by Jules Bloomenthal, Xerox PARC.
 * Copyright (c) Xerox Corporation, 1991. All rights reserved.
 * Permission is granted to reproduce, use and distribute this code for
 * any and all purposes, provided that this notice appears in all copies. */

#include <stdlib.h>
#include <math.h>
#include <stdio.h>
#include <sys/types.h>

#define TET 0 /* use tetrahedral decomposition */
#define NOTET 1 /* no tetrahedral decomposition */

#define RES 10 /* # converge iterations */

#define L 0 /* left direction: -x, -i */
#define R 1 /* right direction: +x, +i */
#define B 2 /* bottom direction: -y, -j */
#define T 3 /* top direction: +y, +j */
#define N 4 /* near direction: -z, -k */
#define F 5 /* far direction: +z, +k */
#define LBN 0 /* left bottom near corner */
#define LBF 1 /* left bottom far corner */
#define LTN 2 /* left top near corner */
#define LTF 3 /* left top far corner */
#define RBN 4 /* right bottom near corner */
#define RBF 5 /* right bottom far corner */
#define RTN 6 /* right top near corner */
#define RTF 7 /* right top far corner */

/* the LBN corner of cube (i, j, k), corresponds with location
 * (start.x+(i-.5)*size, start.y+(j-.5)*size, start.z+(k-.5)*size) */

#define RAND() ((rand()&32767)/32767.) /* random number between 0 and 1 */
#define HASHBIT (5)
#define HASHSIZE (size_t)(1<<(3*HASHBIT)) /* hash table size (32768) */
#define MASK ((1<<HASHBIT)-1)
#define HASH(i,j,k) (((((i)&MASK)<<HASHBIT)|((j)&MASK))<<HASHBIT)|((k)&MASK))
#define BIT(i, bit) (((i)>>(bit))&1)
#define FLIP(i,bit) ((i)^1<<(bit)) /* flip the given bit of i */

typedef struct point { /* a three-dimensional point */
    double x, y, z; /* its coordinates */
} POINT;
```

```
typedef struct test {                /* test the function for a signed value */
    POINT p;                        /* location of test */
    double value;                   /* function value at p */
    int ok;                         /* if value is of correct sign */
} TEST;

typedef struct vertex {              /* surface vertex */
    POINT position, normal;         /* position and surface normal */
} VERTEX;

typedef struct vertices {            /* list of vertices in polygonization */
    int count, max;                 /* # vertices, max # allowed */
    VERTEX *ptr;                    /* dynamically allocated */
} VERTICES;

typedef struct corner {              /* corner of a cube */
    int i, j, k;                    /* (i, j, k) is index within lattice */
    double x, y, z, value;          /* location and function value */
} CORNER;

typedef struct cube {                /* partitioning cell (cube) */
    int i, j, k;                    /* lattice location of cube */
    CORNER *corners[8];             /* eight corners */
} CUBE;

typedef struct cubes {               /* linked list of cubes acting as stack */
    CUBE cube;                      /* a single cube */
    struct cubes *next;             /* remaining elements */
} CUBES;

typedef struct centerlist {          /* list of cube locations */
    int i, j, k;                    /* cube location */
    struct centerlist *next;        /* remaining elements */
} CENTERLIST;

typedef struct cornerlist {          /* list of corners */
    int i, j, k;                    /* corner id */
    double value;                   /* corner value */
    struct cornerlist *next;        /* remaining elements */
} CORNERLIST;

typedef struct edgelist {            /* list of edges */
    int i1, j1, k1, i2, j2, k2;    /* edge corner ids */
    int vid;                        /* vertex id */
    struct edgelist *next;          /* remaining elements */
} EDGELIST;

typedef struct intlist {             /* list of integers */
    int i;                          /* an integer */
    struct intlist *next;           /* remaining elements */
} INTLIST;

typedef struct intlists {            /* list of list of integers */
    INTLIST *list;                  /* a list of integers */
    struct intlists *next;          /* remaining elements */
} INTLISTS;

typedef struct process {             /* parameters, function, storage */
    double (*function)();           /* implicit surface function */
    int (*triproc)();               /* triangle output function */
}
```

```
double size, delta;           /* cube size, normal delta */
int bounds;                   /* cube range within lattice */
POINT start;                  /* start point on surface */
CUBES *cubes;                 /* active cubes */
VERTICES vertices;           /* surface vertices */
CENTERLIST **centers;        /* cube center hash table */
CORNERLIST **corners;        /* corner value hash table */
EDGELIST **edges;            /* edge and vertex id hash table */
} PROCESS;

void *calloc();
char *mycalloc();

/**** A Test Program ****/

/* torus: a torus with major, minor radii = 0.5, 0.1, try size = .05 */

double torus (x, y, z)
double x, y, z;
{
    double x2 = x*x, y2 = y*y, z2 = z*z;
    double a = x2+y2+z2+(0.5*0.5)-(0.1*0.1);
    return a*a-4.0*(0.5*0.5)*(y2+z2);
}

/* sphere: an inverse square function (always positive) */

double sphere (x, y, z)
double x, y, z;
{
    double rsq = x*x+y*y+z*z;
    return 1.0/(rsq < 0.00001? 0.00001 : rsq);
}

/* blob: a three-pole blend function, try size = .1 */

double blob (x, y, z)
double x, y, z;
{
    return 4.0-sphere(x+1.0,y,z)-sphere(x,y+1.0,z)-sphere(x,y,z+1.0);
}

#ifdef SGIGFX /*****
#include "gl.h"

/* triangle: called by polygonize() for each triangle; set SGI lines */

triangle (i1, i2, i3, vertices)
int i1, i2, i3;
VERTICES vertices;
{
    float v[3];
    int i, ids[3];
    ids[0] = i1;
    ids[1] = i2;
```

```
    ids[2] = i3;
    bgnclosedline();
    for (i = 0; i < 3; i++) {
        POINT *p = &vertices.ptr[ids[i]].position;
        v[0] = p->x; v[1] = p->y; v[2] = p->z;
        v3f(v);
    }
    endclosedline();
    return 1;
}

/* main: call polygonize() with torus function
 * display lines on SGI */

main ()
{
    char *err, *polygonize();

    keepaspect(1, 1);
    winopen("implicit");
    doublebuffer();
    gconfig();
    perspective(450, 1.0/1.0, 0.1, 10.0);
    color(7);
    clear();
    swapbuffers();
    makeobj(1);
    if ((err = polygonize(torus, .05, 20, 0.,0.,0., triangle, TET)) != NULL) {
        fprintf(stderr, "%s\n", err);
        exit(1);
    }
    closeobj();
    translate(0.0, 0.0, -2.0);
    pushmatrix();
    while(1) { /* spin the object */
        reshapeviewport();
        color(7);
        clear();
        color(0);
        callobj(1);
        rot(0.8, 'x');
        rot(0.3, 'y');
        rot(0.1, 'z');
        swapbuffers();
    }
}

#else /*****

int gntris;          /* global needed by application */
VERTICES gvertices; /* global needed by application */

/* triangle: called by polygonize() for each triangle; write to stdout */

triangle (i1, i2, i3, vertices)
int i1, i2, i3;
VERTICES vertices;
{

```

```
    gvertices = vertices;
    gntris++;
    fprintf(stdout, "%d %d %d\n", i1, i2, i3);
    return 1;
}

/* main: call polygonize() with torus function
 * write points-polygon formatted data to stdout */

main ()
{
    int i;
    char *err, *polygonize();
    gntris = 0;
    fprintf(stdout, "triangles\n\n");
    if ((err = polygonize(torus, .05, 20, 0.,0.,0., triangle, TET)) != NULL) {
        fprintf(stdout, "%s\n", err);
        exit(1);
    }
    fprintf(stdout, "\n%d triangles, %d vertices\n", gntris, gvertices.count);
    fprintf(stdout, "\nvertices\n\n");
    for (i = 0; i < gvertices.count; i++) {
        VERTEX v;
        v = gvertices.ptr[i];
        fprintf(stdout, "%f %f %f\t%f %f %f\n",
            v.position.x, v.position.y, v.position.z,
            v.normal.x, v.normal.y, v.normal.z);
    }
    fprintf(stderr, "%d triangles, %d vertices\n", gntris, gvertices.count);
    exit(0);
}

#endif /*****

/**** An Implicit Surface Polygonizer ****/

/* polygonize: polygonize the implicit surface function
 * arguments are:
 *     double function (x, y, z)
 *         double x, y, z (an arbitrary 3D point)
 *         the implicit surface function
 *         return negative for inside, positive for outside
 *     double size
 *         width of the partitioning cube
 *     int bounds
 *         max. range of cubes (+/- on the three axes) from first cube
 *     double x, y, z
 *         coordinates of a starting point on or near the surface
 *         may be defaulted to 0., 0., 0.
 *     int triproc (i1, i2, i3, vertices)
 *         int i1, i2, i3 (indices into the vertex array)
 *         VERTICES vertices (the vertex array, indexed from 0)
 *         called for each triangle
 *         the triangle coordinates are (for i = i1, i2, i3):
 *             vertices.ptr[i].position.x, .y, and .z
 *         vertices are ccw when viewed from the out (positive) side
 *         in a left-handed coordinate system
 *         vertex normals point outwards
```

```
*          return 1 to continue, 0 to abort
*      int mode
*          TET: decompose cube and polygonize six tetrahedra
*          NOTET: polygonize cube directly
*      returns error or NULL
*/
```

```
char *polygonize (function, size, bounds, x, y, z, triproc, mode)
double (*function)(), size, x, y, z;
int bounds, (*triproc)(), mode;
{
    PROCESS p;
    int n, noabort;
    CORNER *setcorner();
    TEST in, out, find();

    p.function = function;
    p.triproc = triproc;
    p.size = size;
    p.bounds = bounds;
    p.delta = size/(double)(RES*RES);

    /* allocate hash tables and build cube polygon table: */
    p.centers = (CENTERLIST **) mycalloc(HASHSIZE, sizeof(CENTERLIST *));
    p.corners = (CORNERLIST **) mycalloc(HASHSIZE, sizeof(CORNERLIST *));
    p.edges = (EDGELIST **) mycalloc(2*HASHSIZE, sizeof(EDGELIST *));
    makecubetable();

    /* find point on surface, beginning search at (x, y, z): */
    srand(1);
    in = find(1, &p, x, y, z);
    out = find(0, &p, x, y, z);
    if (!in.ok || !out.ok) return "can't find starting point";
    converge(&in.p, &out.p, in.value, p.function, &p.start);

    /* push initial cube on stack: */
    p.cubes = (CUBES *) mycalloc(1, sizeof(CUBES)); /* list of 1 */
    p.cubes->cube.i = p.cubes->cube.j = p.cubes->cube.k = 0;
    p.cubes->next = NULL;

    /* set corners of initial cube: */
    for (n = 0; n < 8; n++)
        p.cubes->cube.corners[n] = setcorner(&p, BIT(n,2), BIT(n,1), BIT(n,0));

    p.vertices.count = p.vertices.max = 0; /* no vertices yet */
    p.vertices.ptr = NULL;

    setcenter(p.centers, 0, 0, 0);

    while (p.cubes != NULL) { /* process active cubes till none left */
        CUBE c;
        CUBES *temp = p.cubes;
        c = p.cubes->cube;

        noabort = mode == TET?
            /* either decompose into tetrahedra and polygonize: */
            dotet(&c, LBN, LTN, RBN, LBF, &p) &&
            dotet(&c, RTN, LTN, LBF, RBN, &p) &&
            dotet(&c, RTN, LTN, LTF, LBF, &p) &&
            dotet(&c, RTN, RBN, LBF, RBF, &p) &&
            dotet(&c, RTN, LBF, LTF, RBF, &p) &&
```



```
        dotet(&c, RTN, LTF, RTF, RBF, &p)
        :
        /* or polygonize the cube directly: */
        docube(&c, &p);
    if (!noabort) return "aborted";

    /* pop current cube from stack */
    p.cubes = p.cubes->next;
    free((char *) temp);
    /* test six face directions, maybe add to stack: */
    testface(c.i-1, c.j, c.k, &c, L, LBN, LBF, LTN, LTF, &p);
    testface(c.i+1, c.j, c.k, &c, R, RBN, RBF, RTN, RTF, &p);
    testface(c.i, c.j-1, c.k, &c, B, LBN, LBF, RBN, RBF, &p);
    testface(c.i, c.j+1, c.k, &c, T, LTN, LTF, RTN, RTF, &p);
    testface(c.i, c.j, c.k-1, &c, N, LBN, LTN, RBN, RTN, &p);
    testface(c.i, c.j, c.k+1, &c, F, LBF, LTF, RBF, RTF, &p);
}
return NULL;
}
```

```
/* testface: given cube at lattice (i, j, k), and four corners of face,
 * if surface crosses face, compute other four corners of adjacent cube
 * and add new cube to cube stack */
```

```
testface (i, j, k, old, face, c1, c2, c3, c4, p)
CUBE *old;
PROCESS *p;
int i, j, k, face, c1, c2, c3, c4;
{
    CUBE new;
    CUBES *oldcubes = p->cubes;
    CORNER *setcorner();
    static int facebit[6] = {2, 2, 1, 1, 0, 0};
    int n, pos = old->corners[c1]->value > 0.0 ? 1 : 0, bit = facebit[face];

    /* test if no surface crossing, cube out of bounds, or already visited: */
    if ((old->corners[c2]->value > 0) == pos &&
        (old->corners[c3]->value > 0) == pos &&
        (old->corners[c4]->value > 0) == pos) return;
    if (abs(i) > p->bounds || abs(j) > p->bounds || abs(k) > p->bounds) return;
    if (setcenter(p->centers, i, j, k)) return;

    /* create new cube: */
    new.i = i;
    new.j = j;
    new.k = k;
    for (n = 0; n < 8; n++) new.corners[n] = NULL;
    new.corners[FLIP(c1, bit)] = old->corners[c1];
    new.corners[FLIP(c2, bit)] = old->corners[c2];
    new.corners[FLIP(c3, bit)] = old->corners[c3];
    new.corners[FLIP(c4, bit)] = old->corners[c4];
    for (n = 0; n < 8; n++)
        if (new.corners[n] == NULL)
            new.corners[n] = setcorner(p, i+BIT(n,2), j+BIT(n,1), k+BIT(n,0));

    /*add cube to top of stack: */
    p->cubes = (CUBES *) mycalloc(1, sizeof(CUBES));
    p->cubes->cube = new;
    p->cubes->next = oldcubes;
}
```

```
/* setcorner: return corner with the given lattice location
   set (and cache) its function value */
```

```
CORNER *setcorner (p, i, j, k)
int i, j, k;
PROCESS *p;
{
    /* for speed, do corner value caching here */
    CORNER *c = (CORNER *) mycalloc(1, sizeof(CORNER));
    int index = HASH(i, j, k);
    CORNERLIST *l = p->corners[index];
    c->i = i; c->x = p->start.x+((double)i-.5)*p->size;
    c->j = j; c->y = p->start.y+((double)j-.5)*p->size;
    c->k = k; c->z = p->start.z+((double)k-.5)*p->size;
    for (; l != NULL; l = l->next)
        if (l->i == i && l->j == j && l->k == k) {
            c->value = l->value;
            return c;
        }
    l = (CORNERLIST *) mycalloc(1, sizeof(CORNERLIST));
    l->i = i; l->j = j; l->k = k;
    l->value = c->value = p->function(c->x, c->y, c->z);
    l->next = p->corners[index];
    p->corners[index] = l;
    return c;
}
```

```
/* find: search for point with value of given sign (0: neg, 1: pos) */
```

```
TEST find (sign, p, x, y, z)
int sign;
PROCESS *p;
double x, y, z;
{
    int i;
    TEST test;
    double range = p->size;
    test.ok = 1;
    for (i = 0; i < 10000; i++) {
        test.p.x = x+range*(RAND()-0.5);
        test.p.y = y+range*(RAND()-0.5);
        test.p.z = z+range*(RAND()-0.5);
        test.value = p->function(test.p.x, test.p.y, test.p.z);
        if (sign == (test.value > 0.0)) return test;
        range = range*1.0005; /* slowly expand search outwards */
    }
    test.ok = 0;
    return test;
}
```

```
/**** Tetrahedral Polygonization ****/
```

```
/* dotet: triangulate the tetrahedron
   * b, c, d should appear clockwise when viewed from a
   * return 0 if client aborts, 1 otherwise */
```

```
int dotet (cube, c1, c2, c3, c4, p)
CUBE *cube;
int c1, c2, c3, c4;
PROCESS *p;
{
    CORNER *a = cube->corners[c1];
    CORNER *b = cube->corners[c2];
    CORNER *c = cube->corners[c3];
    CORNER *d = cube->corners[c4];
    int index = 0, apos, bpos, cpos, dpos, e1, e2, e3, e4, e5, e6;
    if (apos = (a->value > 0.0)) index += 8;
    if (bpos = (b->value > 0.0)) index += 4;
    if (cpos = (c->value > 0.0)) index += 2;
    if (dpos = (d->value > 0.0)) index += 1;
    /* index is now 4-bit number representing one of the 16 possible cases */
    if (apos != bpos) e1 = vertid(a, b, p);
    if (apos != cpos) e2 = vertid(a, c, p);
    if (apos != dpos) e3 = vertid(a, d, p);
    if (bpos != cpos) e4 = vertid(b, c, p);
    if (bpos != dpos) e5 = vertid(b, d, p);
    if (cpos != dpos) e6 = vertid(c, d, p);
    /* 14 productive tetrahedral cases (0000 and 1111 do not yield polygons */
    switch (index) {
        case 1: return p->triproc(e5, e6, e3, p->vertices);
        case 2: return p->triproc(e2, e6, e4, p->vertices);
        case 3: return p->triproc(e3, e5, e4, p->vertices) &&
                p->triproc(e3, e4, e2, p->vertices);
        case 4: return p->triproc(e1, e4, e5, p->vertices);
        case 5: return p->triproc(e3, e1, e4, p->vertices) &&
                p->triproc(e3, e4, e6, p->vertices);
        case 6: return p->triproc(e1, e2, e6, p->vertices) &&
                p->triproc(e1, e6, e5, p->vertices);
        case 7: return p->triproc(e1, e2, e3, p->vertices);
        case 8: return p->triproc(e1, e3, e2, p->vertices);
        case 9: return p->triproc(e1, e5, e6, p->vertices) &&
                p->triproc(e1, e6, e2, p->vertices);
        case 10: return p->triproc(e1, e3, e6, p->vertices) &&
                p->triproc(e1, e6, e4, p->vertices);
        case 11: return p->triproc(e1, e5, e4, p->vertices);
        case 12: return p->triproc(e3, e2, e4, p->vertices) &&
                p->triproc(e3, e4, e5, p->vertices);
        case 13: return p->triproc(e6, e2, e4, p->vertices);
        case 14: return p->triproc(e5, e3, e6, p->vertices);
    }
    return 1;
}
```

/\*\*\*\* Cubical Polygonization (optional) \*\*\*\*/

```
#define LB      0  /* left bottom edge */
#define LT      1  /* left top edge */
#define LN      2  /* left near edge */
#define LF      3  /* left far edge */
#define RB      4  /* right bottom edge */
#define RT      5  /* right top edge */
#define RN      6  /* right near edge */
#define RF      7  /* right far edge */
#define BN      8  /* bottom near edge */
#define BF      9  /* bottom far edge */
```

```
#define TN      10 /* top near edge */
#define TF      11 /* top far edge */

static INTLISTS *cubetable[256];

/*
    edge: LB, LT, LN, LF, RB, RT, RN, RF, BN, BF, TN, TF */
static int corner1[12] = {LBN,LTN,LBN,LBF,RBN,RTN,RBN,RBF,LBN,LBF,LTN,LTF};
static int corner2[12] = {LBF,LTF,LTN,LTF,RBF,RTF,RTN,RTF,RBN,RBF,RTN,RTF};
static int leftface[12] = {B, L, L, F, R, T, N, R, N, B, T, F};
/* face on left when going corner1 to corner2 */
static int rightface[12] = {L, T, N, L, B, R, R, F, B, F, N, T};
/* face on right when going corner1 to corner2 */

/* docube: triangulate the cube directly, without decomposition */

int docube (cube, p)
CUBE *cube;
PROCESS *p;
{
    INTLISTS *polys;
    int i, index = 0;
    for (i = 0; i < 8; i++) if (cube->corners[i]->value > 0.0) index += (1<<i);
    for (polys = cubetable[index]; polys; polys = polys->next) {
        INTLIST *edges;
        int a = -1, b = -1, count = 0;
        for (edges = polys->list; edges; edges = edges->next) {
            CORNER *c1 = cube->corners[corner1[edges->i]];
            CORNER *c2 = cube->corners[corner2[edges->i]];
            int c = vertid(c1, c2, p);
            if (++count > 2 && ! p->triproc(a, b, c, p->vertices)) return 0;
            if (count < 3) a = b;
            b = c;
        }
    }
    return 1;
}

/* nextcwedge: return next clockwise edge from given edge around given face */

int nextcwedge (edge, face)
int edge, face;
{
    switch (edge) {
        case LB: return (face == L)? LF : BN;
        case LT: return (face == L)? LN : TF;
        case LN: return (face == L)? LB : TN;
        case LF: return (face == L)? LT : BF;
        case RB: return (face == R)? RN : BF;
        case RT: return (face == R)? RF : TN;
        case RN: return (face == R)? RT : BN;
        case RF: return (face == R)? RB : TF;
        case BN: return (face == B)? RB : LN;
        case BF: return (face == B)? LB : RF;
        case TN: return (face == T)? LT : RN;
        case TF: return (face == T)? RT : LF;
    }
}
```

```
/* otherface: return face adjoining edge that is not the given face */

int otherface (edge, face)
int edge, face;
{
    int other = leftface[edge];
    return face == other? rightface[edge] : other;
}

/* makecubetable: create the 256 entry table for cubical polygonization */

makecubetable ()
{
    int i, e, c, done[12], pos[8];
    for (i = 0; i < 256; i++) {
        for (e = 0; e < 12; e++) done[e] = 0;
        for (c = 0; c < 8; c++) pos[c] = BIT(i, c);
        for (e = 0; e < 12; e++)
            if (!done[e] && (pos[corner1[e]] != pos[corner2[e]])) {
                INTLIST *ints = 0;
                INTLISTS *lists = (INTLISTS *) mycalloc(1, sizeof(INTLISTS));
                int start = e, edge = e;
                /* get face that is to right of edge from pos to neg corner: */
                int face = pos[corner1[e]]? rightface[e] : leftface[e];
                while (1) {
                    edge = nextc wedge(edge, face);
                    done[edge] = 1;
                    if (pos[corner1[edge]] != pos[corner2[edge]]) {
                        INTLIST *tmp = ints;
                        ints = (INTLIST *) mycalloc(1, sizeof(INTLIST));
                        ints->i = edge;
                        ints->next = tmp; /* add edge to head of list */
                        if (edge == start) break;
                        face = otherface(edge, face);
                    }
                }
                lists->list = ints; /* add ints to head of table entry */
                lists->next = cubetable[i];
                cubetable[i] = lists;
            }
    }
}

/***** Storage *****/

/* mycalloc: return successful calloc or exit program */

char *mycalloc (nitems, nbytes)
int nitems, nbytes;
{
    char *ptr = calloc(nitems, nbytes);
    if (ptr != NULL) return ptr;
    fprintf(stderr, "can't calloc %d bytes\n", nitems*nbytes);
    exit(1);
}

/* setcenter: set (i,j,k) entry of table[]
```

```
* return 1 if already set; otherwise, set and return 0 */
```

```
int setcenter(table, i, j, k)
CENTERLIST *table[];
int i, j, k;
{
    int index = HASH(i, j, k);
    CENTERLIST *new, *l, *q = table[index];
    for (l = q; l != NULL; l = l->next)
        if (l->i == i && l->j == j && l->k == k) return 1;
    new = (CENTERLIST *) mycalloc(1, sizeof(CENTERLIST));
    new->i = i; new->j = j; new->k = k; new->next = q;
    table[index] = new;
    return 0;
}
```

```
/* setedge: set vertex id for edge */
```

```
setedge (table, i1, j1, k1, i2, j2, k2, vid)
EDGEELIST *table[];
int i1, j1, k1, i2, j2, k2, vid;
{
    unsigned int index;
    EDGEELIST *new;
    if (i1>i2 || (i1==i2 && (j1>j2 || (j1==j2 && k1>k2)))) {
        int t=i1; i1=i2; i2=t; t=j1; j1=j2; j2=t; t=k1; k1=k2; k2=t;
    }
    index = HASH(i1, j1, k1) + HASH(i2, j2, k2);
    new = (EDGEELIST *) mycalloc(1, sizeof(EDGEELIST));
    new->i1 = i1; new->j1 = j1; new->k1 = k1;
    new->i2 = i2; new->j2 = j2; new->k2 = k2;
    new->vid = vid;
    new->next = table[index];
    table[index] = new;
}
```

```
/* getedge: return vertex id for edge; return -1 if not set */
```

```
int getedge (table, i1, j1, k1, i2, j2, k2)
EDGEELIST *table[];
int i1, j1, k1, i2, j2, k2;
{
    EDGEELIST *q;
    if (i1>i2 || (i1==i2 && (j1>j2 || (j1==j2 && k1>k2)))) {
        int t=i1; i1=i2; i2=t; t=j1; j1=j2; j2=t; t=k1; k1=k2; k2=t;
    };
    q = table[HASH(i1, j1, k1)+HASH(i2, j2, k2)];
    for (; q != NULL; q = q->next)
        if (q->i1 == i1 && q->j1 == j1 && q->k1 == k1 &&
            q->i2 == i2 && q->j2 == j2 && q->k2 == k2)
            return q->vid;
    return -1;
}
```

```
/* Vertices */
```

```
/* vertid: return index for vertex on edge:
```

```
* c1->value and c2->value are presumed of different sign
* return saved index if any; else compute vertex and save */
```

```
int vertid (c1, c2, p)
CORNER *c1, *c2;
PROCESS *p;
{
    VERTEX v;
    POINT a, b;
    int vid = getedge(p->edges, c1->i, c1->j, c1->k, c2->i, c2->j, c2->k);
    if (vid != -1) return vid; /* previously computed */
    a.x = c1->x; a.y = c1->y; a.z = c1->z;
    b.x = c2->x; b.y = c2->y; b.z = c2->z;
    converge(&a, &b, c1->value, p->function, &v.position); /* position */
    vnormal(&v.position, p, &v.normal); /* normal */
    addtovertices(&p->vertices, v); /* save vertex */
    vid = p->vertices.count-1;
    setedge(p->edges, c1->i, c1->j, c1->k, c2->i, c2->j, c2->k, vid);
    return vid;
}
```

```
/* addtovertices: add v to sequence of vertices */
```

```
addtovertices (vertices, v)
VERTICES *vertices;
VERTEX v;
{
    if (vertices->count == vertices->max) {
        int i;
        VERTEX *new;
        vertices->max = vertices->count == 0 ? 10 : 2*vertices->count;
        new = (VERTEX *) mycalloc(vertices->max, sizeof(VERTEX));
        for (i = 0; i < vertices->count; i++) new[i] = vertices->ptr[i];
        if (vertices->ptr != NULL) free((char *) vertices->ptr);
        vertices->ptr = new;
    }
    vertices->ptr[vertices->count++] = v;
}
```

```
/* vnormal: compute unit length surface normal at point */
```

```
vnormal (point, p, v)
POINT *point, *v;
PROCESS *p;
{
    double f = p->function(point->x, point->y, point->z);
    v->x = p->function(point->x+p->delta, point->y, point->z)-f;
    v->y = p->function(point->x, point->y+p->delta, point->z)-f;
    v->z = p->function(point->x, point->y, point->z+p->delta)-f;
    f = sqrt(v->x*v->x + v->y*v->y + v->z*v->z);
    if (f != 0.0) {v->x /= f; v->y /= f; v->z /= f;}
}
```





```
/* converge: from two points of differing sign, converge to zero crossing */
```

```
converge (p1, p2, v, function, p)
double v;
double (*function)();
```





















```
POINT *p1, *p2, *p;
{
    int i = 0;
    POINT pos, neg;
    if (v < 0) {
        pos.x = p2->x; pos.y = p2->y; pos.z = p2->z;
        neg.x = p1->x; neg.y = p1->y; neg.z = p1->z;
    }
    else {
        pos.x = p1->x; pos.y = p1->y; pos.z = p1->z;
        neg.x = p2->x; neg.y = p2->y; neg.z = p2->z;
    }
    while (1) {
        p->x = 0.5*(pos.x + neg.x);
        p->y = 0.5*(pos.y + neg.y);
        p->z = 0.5*(pos.z + neg.z);
        if (i++ == RES) return;
        if ((function(p->x, p->y, p->z)) > 0.0)
            {pos.x = p->x; pos.y = p->y; pos.z = p->z;}
        else {neg.x = p->x; neg.y = p->y; neg.z = p->z;}
    }
}
```














# Index of /pubs/tog/GraphicsGems/gemsii/dither/

Name	Last modified	Size	Description
 <a href="#">_Parent Directory</a>			
 <a href="#">_Makefile</a>	29-Jun-00 08:14	1K	
 <a href="#">_dither.3</a>	29-Jun-00 08:14	3K	
 <a href="#">_dither.c</a>	29-Jun-00 08:14	8K	

# Index of /pubs/tog/GraphicsGems/gemsii/

Name	Last modified	Size	Description
 <a href="#">Parent Directory</a>			
 <a href="#">AllGems.TOC</a>	29-Jun-00 08:22	26K	
 <a href="#">BitCounting/</a>	29-Jun-00 08:14	1K	
 <a href="#">Errata.GraphicsGemsII</a>	03-Jan-01 10:48	8K	
 <a href="#">FastUpdate.c</a>	29-Jun-00 08:15	1K	
 <a href="#">GGVecLib.c</a>	29-Jun-00 08:15	10K	
 <a href="#">GraphicsGems.h</a>	29-Jun-00 08:15	3K	
 <a href="#">Hilbert.c</a>	29-Jun-00 08:15	3K	
 <a href="#">InterPhong.c</a>	29-Jun-00 08:15	5K	
 <a href="#">Makefile</a>	29-Jun-00 08:15	2K	
 <a href="#">Peano/</a>	29-Jun-00 08:14	1K	
 <a href="#">README</a>	29-Jun-00 08:15	3K	
 <a href="#">RayCPhdron.c</a>	29-Jun-00 08:15	3K	
 <a href="#">RealPixels/</a>	29-Jun-00 08:15	1K	
 <a href="#">TOC</a>	29-Jun-00 08:15	7K	
 <a href="#">VoxelCache.c</a>	29-Jun-00 08:15	3K	
 <a href="#">c_format.c</a>	29-Jun-00 08:15	10K	
 <a href="#">dither/</a>	29-Jun-00 08:14	1K	
 <a href="#">hot.c</a>	29-Jun-00 08:15	10K	
 <a href="#">intersect/</a>	29-Jun-00 08:14	1K	
 <a href="#">inv_cmap/</a>	29-Jun-00 08:14	1K	
 <a href="#">inverse.c</a>	29-Jun-00 08:15	5K	
 <a href="#">noise3.c</a>	29-Jun-00 08:15	4K	

 <a href="#">__quantizer.c</a>	29-Jun-00 08:15	11K
 <a href="#">__radiosity/</a>	29-Jun-00 08:15	1K
 <a href="#">__ran_ramp.c</a>	29-Jun-00 08:15	6K
 <a href="#">__rotate.c</a>	29-Jun-00 08:15	2K
 <a href="#">__rotate8x8.c</a>	29-Jun-00 08:15	1K
 <a href="#">__sparse.c</a>	29-Jun-00 08:15	8K
 <a href="#">__unmatrix.c</a>	03-Jan-01 10:46	5K
 <a href="#">__unmatrix.h</a>	29-Jun-00 08:15	1K
 <a href="#">__vector.h</a>	29-Jun-00 08:15	3K
 <a href="#">__viewcorr/</a>	29-Jun-00 08:15	1K
 <a href="#">__xlines.c</a>	29-Jun-00 08:15	4K

CFLAGS = -g

dither.o: dither.c  
cc \$(CFLAGS) -c dither.c -o dither.o

clean:  
/bin/rm -f dither.o

.\" Copyright (c) 1986, 1987, University of Utah

.TH DITHER 3 2/2/87 3

.UC 4

.SH NAME

.HP

dithermap, bwdithermap, make\_square, dithergb, ditherbw \- functions for dithering color or black and white images.

.SH SYNOPSIS

.na

.sp

.B

dithermap( levels, gamma, rgbmap, divN, modN, magic )

.br

.B

int levels;

.br

.B

double gamma;

.br

.B

int rgbmap[][3], divN[256], modN[256], magic[16][16];

.sp

.B

bwdithermap( levels, gamma, bwmap, divN, modN, magic )

.br

.B

int levels;

.br

.B

double gamma;

.br

.B

int bwmap[], int divN[256], modN[256], magic[16][16];

.sp

.B

make\_square( N, divN, modN, magic )

.br

.B

double N;

.br

.B

int divN[256], modN[256], magic[16][16];

.sp

.B

dithergb( x, y, r, g, b, levels, divN, modN, magic )

.br

.B

int x, y, r, g, b, levels;

.br

.B

int divN[256], modN[256], magic[16][16];

.sp

.B

ditherbw( x, y, val, divN, modN, magic )

.br

.B

int x, y, val, divN[256], modN[256], magic[16][16];

.ad b

.SH DESCRIPTION

These functions provide a common set of routines for dithering a full color or gray scale image into a lower resolution color map.

.I Dithermap  
computes a color map and some auxiliary parameters for dithering a full color (24 bit) image to fewer bits. The argument  
.I levels  
tells how many different intensity levels per primary color should be computed. To get maximum use of a 256 entry color map, use  
.IR levels =6.  
The computed map uses  $\backslash fIlevels^3 \backslash fP$  entries.  
The  
.I gamma  
argument provides for gamma compensation of the generated color map (that is, the values in the map will be adjusted to give a linear intensity variation on a display with the given gamma).  
The computed color map will be returned in the array  
.IR rgbmap .  
.I divN  
and  
.I modN  
are auxiliary arrays for computing the dithering pattern (see below),  
and  
.I magic  
is the magic square dither pattern.  
.PP  
To compute a color map for dithering a black and white image to fewer intensity levels, use  
.IR bwdithermap .  
The arguments are as for  
.IR dithermap ,  
but only a single channel color map is computed. The value of  $\backslash fIlevels \backslash fP$  can be larger than for  $\backslash fIdithermap \backslash fP$ , as the computed map only has  $\backslash fIlevels \backslash fP$  entries.  
.PP  
To just build the magic square and other parameters, use  
.IR make\_square .  
The argument  
.I N  
should be equal to 255.0 divided by the desired number of intensity levels less one (i.e.,  $\backslash fIN = 255.0 / (levels - 1) \backslash fP$ ). The other arguments are filled in as above.  
.PP  
The color map index for a dithered full color pixel is computed by  
.IR dithergb .  
Since the pattern depends on the screen location, the first two arguments  
.IR x  
and  
.IR y ,  
specify that location. The true color of the pixel at that location is given by the triple  
.IR r ,  
.IR g ,  
and  
.IR b .  
The number of intensity  
.I levels  
and the dithering parameter matrices computed by  
.I dithermap  
are also passed to  
.IR dithergb .  
.PP

The color map index for a dithered gray scale pixel is computed by  
.IR ditherbw .

Again, the screen position is specified, and the intensity value of  
the pixel is supplied in

.IR val .

The dithering parameters must also be supplied.

.PP

Alternatively, the dithering may be done in line instead of incurring  
the extra overhead of a function call, which can be significant when  
repeated a million times. The computation is as follows:

.nf

```
.ta .5i 1.0i 1.5i
        row = y % 16;
        col = x % 16;
#define DMAP(v,col,row) (divN[v] + (modN[v]>magic[col][row] ? 1 : 0))
        pix = DMAP(r,col,row) + DMAP(g,col,row)*levels +
                DMAP(b,col,row)*levels*levels;
```

.fi

For a gray scale image, it is a little simpler:

.nf

```
.ta .5i 1.0i 1.5i
        pix = DMAP(val,row,col);
```

.fi

And on a single bit display (assuming a 1 means white):

.nf

```
.ta .5i 1.0i
        pix = divN[val] > magic[col][row] ? 1 : 0
```

.fi

.SH SEE ALSO

.IR rgb\_to\_bw (3),

.IR librle (3),

.IR RLE (5).

.SH AUTHOR

Spencer W. Thomas

.br

University of Utah

```
/*
 * This software is copyrighted as noted below.  It may be freely copied,
 * modified, and redistributed, provided that the copyright notice is
 * preserved on all copies.
 *
 * There is no warranty or other guarantee of fitness for this software,
 * it is provided solely "as is".  Bug reports or fixes may be sent
 * to the author, who may or may not act on them as he desires.
 *
 * You may not include this software in a program or other software product
 * without supplying the source, or without informing the end-user that the
 * source is available for no extra charge.
 *
 * If you modify this software, you should include a notice giving the
 * name of the person performing the modification, the date of modification,
 * and the reason for such modification.
 *
 * Modified at BRL 16-May-88 by Mike Muuss to avoid Alliant STDC desire
 * to have all "void" functions so declared.
 */
/*
 * dither.c - Functions for RGB color dithering.
 *
 * Author:      Spencer W. Thomas
 *              Computer Science Dept.
 *              University of Utah
 * Date:        Mon Feb  2 1987
 * Copyright (c) 1987, University of Utah
 */
```

```
#include <math.h>
```

```
#ifdef USE_PROTOTYPES
void      make_square( double, int [256], int [256], int [16][16] );
#else
void      make_square();
#endif
```

```
static int magic4x4[4][4] = {
    0, 14,  3, 13,
    11,  5,  8,  6,
    12,  2, 15,  1,
    7,  9,  4, 10
};
```

```
/* basic dithering macro */
#define DMAP(v,x,y)      (modN[v]>magic[x][y] ? divN[v] + 1 : divN[v])
```

```
/******
 * TAG( dithermap )
 *
 * Create a color dithering map with a specified number of intensity levels.
 * Inputs:
 *     levels:      Intensity levels per primary.
 *     gamma:       Display gamma value.
 * Outputs:
 *     rgbmap:      Generated color map.
 *     divN:        "div" function for dithering.
 *     modN:        "mod" function for dithering.
 * Assumptions:
```



```
*      rgbmap will hold levels^3 entries.
* Algorithm:
*      Compute gamma compensation map.
*      N = 255.0 / (levels - 1) is number of pixel values per level.
*      Compute rgbmap with red ramping fastest, green slower, and blue
*      slowest (treat it as if it were rgbmap[levels][levels][levels][3]).
*      Call make_square to get divN, modN, and magic
*
* Note:
*      Call dithergb( x, y, r, g, b, levels, divN, modN, magic ) to get index
*      into rgbmap for a given color/location pair, or use
*          row = y % 16; col = x % 16;
*          DMAP(v,col,row) = def (divN[v] + (modN[v]>magic[col][row] ? 1 : 0))
*          DMAP(r,col,row) + DMAP(g,col,row)*levels + DMAP(b,col,row)*levels^2
*      if you don't want function call overhead.
*/
```

```
void
dithermap( levels, gamma, rgbmap, divN, modN, magic )
double gamma;
int rgbmap[][3];
int divN[256];
int modN[256];
int magic[16][16];
{
    double N;
    register int i;
    int levelsq, levelsc;
    int gammamap[256];

    for ( i = 0; i < 256; i++ )
        gammamap[i] = (int)(0.5 + 255 * pow( i / 255.0, 1.0/gamma ));

    levelsq = levels*levels;    /* squared */
    levelsc = levels*levelsc;  /* and cubed */

    N = 255.0 / (levels - 1);    /* Get size of each step */

    /*
     * Set up the color map entries.
     */
    for(i = 0; i < levelsc; i++) {
        rgbmap[i][0] = gammamap[(int)(0.5 + (i%levels) * N)];
        rgbmap[i][1] = gammamap[(int)(0.5 + ((i/levels)%levels) * N)];
        rgbmap[i][2] = gammamap[(int)(0.5 + ((i/levelsq)%levels) * N)];
    }

    make_square( N, divN, modN, magic );
}
```

```
/******
* TAG( bwdithermap )
*
* Create a color dithering map with a specified number of intensity levels.
* Inputs:
*      levels:      Intensity levels.
*      gamma:      Display gamma value.
* Outputs:
*      bwmap:      Generated black & white map.
*      divN:      "div" function for dithering.
*      modN:      "mod" function for dithering.
```

\* Assumptions:

\*     bwmap will hold levels entries.

\* Algorithm:

\*     Compute gamma compensation map.

\*      $N = 255.0 / (\text{levels} - 1)$  is number of pixel values per level.

\*     Compute bwmap for levels entries.

\*     Call make\_square to get divN, modN, and magic.

\* Note:

\*     Call ditherbw( x, y, val, divN, modN, magic ) to get index into

\*     bwmap for a given color/location pair, or use

\*     row = y % 16; col = x % 16;

\*     divN[val] + (modN[val]>magic[col][row] ? 1 : 0)

\*     if you don't want function call overhead.

\*     On a 1-bit display, use

\*     divN[val] > magic[col][row] ? 1 : 0

\*/

void

bwdithermap( levels, gamma, bwmap, divN, modN, magic )

double gamma;

int bwmap[];

int divN[256];

int modN[256];

int magic[16][16];

{

    double N;

    register int i;

    int gammamap[256];

    for ( i = 0; i < 256; i++ )

        gammamap[i] = (int)(0.5 + 255 \* pow( i / 255.0, 1.0/gamma ));

    N = 255.0 / (levels - 1);     /\* Get size of each step \*/

    /\*

    \* Set up the color map entries.

    \*/

    for(i = 0; i < levels; i++)

        bwmap[i] = gammamap[(int)(0.5 + i \* N)];

    make\_square( N, divN, modN, magic );

}

/\*\*\*\*\*\*

\* TAG( make\_square )

\*

\* Build the magic square for a given number of levels.

\* Inputs:

\*     N:                     Pixel values per level (255.0 / levels).

\* Outputs:

\*     divN:                 Integer value of pixval / N

\*     modN:                 Integer remainder between pixval and divN[pixval]\*N

\*     magic:                Magic square for dithering to N sublevels.

\* Assumptions:

\*

\* Algorithm:

\*     divN[pixval] = (int)(pixval / N) maps pixval to its appropriate level.

\*     modN[pixval] = pixval - (int)(N \* divN[pixval]) maps pixval to

\*     its sublevel, and is used in the dithering computation.

\*     The magic square is computed as the (modified) outer product of

```
*      a 4x4 magic square with itself.
*      magic[4*k + i][4*l + j] = (magic4x4[i][j] + magic4x4[k][l]/16.0)
*      multiplied by an appropriate factor to get the correct dithering
*      range.
*/
void
make_square( N, divN, modN, magic )
double N;
int divN[256];
int modN[256];
int magic[16][16] ;
{
    register int i, j, k, l;
    double magicfact;

    for ( i = 0; i < 256; i++ )
    {
        divN[i] = (int)(i / N);
        modN[i] = i - (int)(N * divN[i]);
    }
    modN[255] = 0;                /* always */

    /*
     * Expand 4x4 dither pattern to 16x16.  4x4 leaves obvious patterning,
     * and doesn't give us full intensity range (only 17 sublevels).
     *
     * magicfact is (N - 1)/16 so that we get numbers in the matrix from 0 to
     * N - 1: mod N gives numbers in 0 to N - 1, don't ever want all
     * pixels incremented to the next level (this is reserved for the
     * pixel value with mod N == 0 at the next level).
     */
    magicfact = (N - 1) / 16.;
    for ( i = 0; i < 4; i++ )
        for ( j = 0; j < 4; j++ )
            for ( k = 0; k < 4; k++ )
                for ( l = 0; l < 4; l++ )
                    magic[4*k+i][4*l+j] =
                        (int)(0.5 + magic4x4[i][j] * magicfact +
                            (magic4x4[k][l] / 16.) * magicfact);
}

/*****
 * TAG( dithergb )
 *
 * Return dithered RGB value.
 * Inputs:
 *     x:                X location on screen of this pixel.
 *     y:                Y location on screen of this pixel.
 *     r, g, b:         Color at this pixel (0 - 255 range).
 *     levels:          Number of levels in this map.
 *     divN, modN:      From dithermap.
 *     magic:           Magic square from dithermap.
 * Outputs:
 *     Returns color map index for dithered pixelv value.
 * Assumptions:
 *     divN, modN, magic were set up properly.
 * Algorithm:
 *     see "Note:" in dithermap comment.
 */
dithergb( x, y, r, g, b, levels, divN, modN, magic )
```








```
int divN[256];
int modN[256];
int magic[16][16];
{
    int col = x % 16, row = y % 16;

    return DMAP(r, col, row) +
        DMAP(g, col, row) * levels +
        DMAP(b, col, row) * levels*levels;
}

/*****
 * TAG( ditherbw )
 *
 * Return dithered black & white value.
 * Inputs:
 *     x:          X location on screen of this pixel.
 *     y:          Y location on screen of this pixel.
 *     val:        Intensity at this pixel (0 - 255 range).
 *     divN, modN: From dithermap.
 *     magic:      Magic square from dithermap.
 * Outputs:
 *     Returns color map index for dithered pixel value.
 * Assumptions:
 *     divN, modN, magic were set up properly.
 * Algorithm:
 *     see "Note:" in bwdithermap comment.
 */
ditherbw( x, y, val, divN, modN, magic )
int divN[256];
int modN[256];
int magic[16][16];
{
    int col = x % 16, row = y % 16;

    return DMAP(val, col, row);
}
```

# Index of /pubs/tog/GraphicsGems/gemsii/viewcorr/

Name	Last modified	Size	Description
 <a href="#">_ Parent Directory</a>			
 <a href="#">_ Makefile</a>	29-Jun-00 08:15	1K	
 <a href="#">_ matrix.c</a>	29-Jun-00 08:15	4K	
 <a href="#">_ matrix.h</a>	29-Jun-00 08:15	1K	
 <a href="#">_ viewcorr.c</a>	29-Jun-00 08:15	8K	
 <a href="#">_ viewcorr.h</a>	29-Jun-00 08:15	1K	
 <a href="#">_ viewfind.c</a>	29-Jun-00 08:15	5K	

CFLAGS = -g -I..

all: matrix.o viewcorr.o viewfind.o

matrix.o: matrix.o matrix.h  
cc \$(CFLAGS) -c matrix.c -o matrix.o

viewcorr.o: viewcorr.o matrix.h viewcorr.h  
cc \$(CFLAGS) -c viewcorr.c -o viewcorr.o

viewfind.o: viewfind.o matrix.h viewcorr.h  
cc \$(CFLAGS) -c viewfind.c -o viewfind.o

clean:  
/bin/rm -f matrix.o viewcorr.o viewfind.o

```
/*
 * matrix.c - Simple routines for general sized matrices.
 *
 */

#include <stdio.h>
#include <math.h>
#include "matrix.h"

double
InvertMatrix(mat,actual_size)
Matrix mat; /* Holds the original and inverse */
int actual_size; /* Actual size of matrix in use, (high_subscript+1)*/
{
    int i,j,k; /* Locations of pivot elements */

    int *pvt_i, *pvt_j;
    double pvt_val; /* Value of current pivot element */
    double hold; /* Temporary storage */
    double determ; /* Determinant */

    determ = 1.0;

    pvt_i = (int *) malloc(actual_size * sizeof(int));
    pvt_j = (int *) malloc(actual_size * sizeof(int));

    for (k = 0; k < actual_size; k++)
    {
        /* Locate k'th pivot element */
        pvt_val = mat[k][k]; /* Initialize for search */
        pvt_i[k] = k;
        pvt_j[k] = k;
        for (i = k; i < actual_size; i++)
            for (j = k; j < actual_size; j++)
                if (fabs(mat[i][j]) > fabs(pvt_val))
                {
                    pvt_i[k] = i;
                    pvt_j[k] = j;
                    pvt_val = mat[i][j];
                }
        /* Product of pivots, gives determinant when finished */
        determ *= pvt_val;
        if (determ == 0.0) {
            /* Matrix is singular (zero determinant). */
            free(pvt_i);
            free(pvt_j);
            return (0.0);
        }

        /* "Interchange" rows (with sign change stuff) */
        i = pvt_i[k];
        if (i != k) /* If rows are different */
            for (j = 0; j < actual_size; j++)
            {
                hold = -mat[k][j];
                mat[k][j] = mat[i][j];
                mat[i][j] = hold;
            }

        /* "Interchange" columns */
    }
}
```

```
j = pvt_j[k];
if (j != k) /* If columns are different */
    for (i = 0; i < actual_size; i++)
    {
        hold = -mat[i][k];
        mat[i][k] = mat[i][j];
        mat[i][j] = hold;
    }
/* Divide column by minus pivot value */
for (i = 0; i < actual_size; i++)
    if (i != k) /* Don't touch the pivot entry */
        mat[i][k] /= ( -pvt_val) ; /* (Tricky C syntax for division) */

/* Reduce the matrix */
for (i = 0; i < actual_size; i++)
{
    hold = mat[i][k];
    for (j = 0; j < actual_size; j++)
        if ( i != k && j != k ) /* Don't touch pivot. */
            mat[i][j] += hold * mat[k][j];
}

/* Divide row by pivot */
for (j = 0; j < actual_size; j++)
    if (j != k) /* Don't touch the pivot! */
        mat[k][j] /= pvt_val;

/* Replace pivot by reciprocal (at last we can touch it). */
mat[k][k] = 1.0/pvt_val;
}

/* That was most of the work, one final pass of row/column interchange */
/* to finish */
for (k = actual_size-2; k >= 0; k--) /* Don't need to work with 1 by 1 */
    /* corner */
{
    i = pvt_j[k]; /* Rows to swap correspond to pivot COLUMN */
    if (i != k) /* If rows are different */
        for(j = 0; j < actual_size; j++)
        {
            hold = mat[k][j];
            mat[k][j] = -mat[i][j];
            mat[i][j] = hold;
        }

    j = pvt_i[k]; /* Columns to swap correspond to pivot ROW */
    if (j != k) /* If columns are different */
        for (i = 0; i < actual_size; i++)
        {
            hold = mat[i][k];
            mat[i][k] = -mat[i][j];
            mat[i][j] = hold;
        }
}

free(pvt_i);
free(pvt_j);
return(determ);
}
```

Matrix



```
NewMatrix(cols, rows)
int cols,rows;
{
    int i;
    Matrix newM;
    newM = (double **) malloc(rows * sizeof(double *));
    for(i = 0; i < rows; i++)
        newM[i] = (double *) malloc(cols * sizeof(double));
    return newM;
}

FreeMatrix(mat, rows)
Matrix mat;
int rows;
{
    int i;
    for(i = 0; i < rows; i++)
        free(mat[i]);
    free(mat);
}

TransposeMatrix(inM, outM, cols, rows)
Matrix inM, outM;
int cols,rows;
{
    int tempI, tempJ;
    for(tempI=0; tempI < rows; tempI++)
        for(tempJ=0; tempJ < cols; tempJ++)
            outM[tempI][tempJ] = inM[tempJ][tempI];
}

MultMatrix(firstM, secondM, outM, firstrows, cols, secondcols)
Matrix firstM, secondM, outM;
int firstrows, cols, secondcols;
{
    int i,j,k;
    double sum;

    for(i=0; i < secondcols; i++)
        for(j=0; j < firstrows; j++)
        {
            sum = 0.0;
            for(k=0; k < cols; k++)
                sum += firstM[j][k] * secondM[k][i];
            outM[j][i] = sum;
        }
}
```

```
/* matrix.h
 *      The type and externs for matrix routines.
 */

typedef double ** Matrix;

extern double InvertMatrix();
extern Matrix NewMatrix();
```

```
/*
 * viewcorr.c - Iterate the view parameters.
 *
 *
 * Author:      Rod G. Bogart
 * Date:       Oct 15 1990
 * Copyright (c) 1990, University of Michigan
 *
 */
#include <stdio.h>
#include <math.h>
#include "GraphicsGems.h"
#include "matrix.h"
#include "viewcorr.h"

iterate_view_parms( datapts, view_parms, num_iterations )
ViewData *datapts;
ViewParms *view_parms;
int num_iterations;
{
    Matrix errors, jacobian, corrections;
    int i,j;
    double rootmeansqr, last_rootmeansqr;

    /* allocate Matrix stuff */
    errors = NewMatrix(datapts->numpts*2, 1);
    jacobian = NewMatrix(datapts->numpts*2, NUM_VIEW_PARMS);
    corrections = NewMatrix(NUM_VIEW_PARMS, 1);

    if (num_iterations <= 0) {
        num_iterations = 10000;
    }
    last_rootmeansqr = 0.0;
    for (i = 0; i < num_iterations; i++)
    {
        measure_errors( datapts, view_parms, errors, &rootmeansqr );
        if (ABS(rootmeansqr - last_rootmeansqr) < 1E-8) {
            /* quit when rootmeansqr stays the same */
            break;
        }
        last_rootmeansqr = rootmeansqr;
        if (rootmeansqr > (0.1 * view_parms->halfx))
            /* When the error terms are large, the corrections become too
             * extreme, and knock the whole thing into outer space.  Sooo,
             * shrink the error terms, to cause the corrections to happen
             * a small amount at a time.  Note: dividing by the rootmeansqr
             * may be a little extreme, but it does slow down the erratic
             * behaviour.
             */
            for (j = 0; j < datapts->numpts; j++)
            {
                errors[0][j*2] /= rootmeansqr;
                errors[0][j*2+1] /= rootmeansqr;
            }
        build_jacobian( datapts, view_parms, jacobian );
        find_corrections( datapts, jacobian, errors, corrections );
        apply_corrections( corrections, view_parms );
    }
    FreeMatrix(errors, 1);
    FreeMatrix(jacobian, NUM_VIEW_PARMS);
    FreeMatrix(corrections, 1);
}
```

```
}

measure_errors( datapts, view_parms, errors, rootmeansqr )
ViewData *datapts;
Matrix errors;
ViewParms *view_parms;
double *rootmeansqr;
{
    int i;
    double sqrs=0.0;
    Point2 screenpt;

    for (i = 0; i < datapts->numpts; i++)
    {
        screen_project( &datapts->pts[i], view_parms, &screenpt );
        errors[0][i*2 + 0] = screenpt.x - datapts->scripts[i].x;
        errors[0][i*2 + 1] = screenpt.y - datapts->scripts[i].y;
        sqrs += SQR(errors[0][i*2 + 0]) + SQR(errors[0][i*2 + 1]);
    }
    *rootmeansqr = sqrt( sqrs / (datapts->numpts*2.0) );
}

build_jacobian( datapts, view_parms, jacobian )
ViewData *datapts;
Matrix jacobian;
ViewParms *view_parms;
{
    int i;
    Point3 xyz, eR;

    /* The jacobian matrix has at least 10 columns (u and v for 5 pts)
     * and 10 rows (10 iteration parameters). The iteration parameters will
     * be ordered: eRx eRy eRz phi_x phi_y phi_z ds xcenter ycenter aspect
     * from the top down.
     */

    V3MulPointByMatrix(&view_parms->eye, &view_parms->view, &eR);
    for (i = 0; i < datapts->numpts; i++)
    {
        V3MulPointByMatrix(&datapts->pts[i], &view_parms->view, &xyz);

        store_upartials( &xyz, &eR, view_parms, i, jacobian );
        store_vpartials( &xyz, &eR, view_parms, i, jacobian );
    }
}

store_upartials( xyz, eR, view_parms, ptnum, jacobian )
Point3 *xyz, *eR;
Matrix jacobian;
ViewParms *view_parms;
int ptnum;
{
    double x_min_eR, z_min_eR;
    int i2;

    i2 = ptnum*2;
    x_min_eR = xyz->x - eR->x;
    z_min_eR = xyz->z - eR->z;

    jacobian[0][i2] = view_parms->d_over_s * view_parms->halfx / z_min_eR;
    jacobian[1][i2] = 0.0;
}
```

```
jacobian[2][i2] = - view_parms->d_over_s * view_parms->halfx * x_min_eR
    / SQR( z_min_eR );

jacobian[3][i2] = view_parms->d_over_s * view_parms->halfx * xyz->y
    * x_min_eR / SQR( z_min_eR );
jacobian[4][i2] = - (view_parms->d_over_s * view_parms->halfx * xyz->z
    / z_min_eR)
    - (view_parms->d_over_s * view_parms->halfx * xyz->x * x_min_eR
    / SQR( z_min_eR ));
jacobian[5][i2] = view_parms->d_over_s * view_parms->halfx * xyz->y
    / z_min_eR;

jacobian[6][i2] = - view_parms->halfx * x_min_eR / z_min_eR;
jacobian[7][i2] = 1.0;
jacobian[8][i2] = 0.0;
#ifdef ITERATE_ASPECT_RATIO
    jacobian[9][i2] = 0.0;
#endif
}

store_v_partials( xyz, eR, view_parms, ptnum, jacobian )
Point3 *xyz, *eR;
Matrix jacobian;
ViewParms *view_parms;
int ptnum;
{
    double y_min_eR, z_min_eR, d_over_s;
    int i2;

    i2 = ptnum*2 + 1;
    y_min_eR = xyz->y - eR->y;
    z_min_eR = xyz->z - eR->z;
    d_over_s = view_parms->d_over_s * view_parms->aspect;

    jacobian[0][i2] = 0.0;
    jacobian[1][i2] = d_over_s * view_parms->halfx / z_min_eR;
    jacobian[2][i2] = - d_over_s * view_parms->halfx * y_min_eR
        / SQR( z_min_eR );

    jacobian[3][i2] = (d_over_s * view_parms->halfx * xyz->z
        / z_min_eR)
        + (d_over_s * view_parms->halfx * xyz->y * y_min_eR
        / SQR( z_min_eR ));
    jacobian[4][i2] = - d_over_s * view_parms->halfx * xyz->x
        * y_min_eR / SQR( z_min_eR );
    jacobian[5][i2] = - d_over_s * view_parms->halfx * xyz->x
        / z_min_eR;

    jacobian[6][i2] = - view_parms->aspect * view_parms->halfx * y_min_eR
        / z_min_eR;
    jacobian[7][i2] = 0.0;
    jacobian[8][i2] = 1.0;
#ifdef ITERATE_ASPECT_RATIO
    jacobian[9][i2] = - view_parms->d_over_s * view_parms->halfx * y_min_eR
        / z_min_eR;
#endif
}

find_corrections( datapts, jacobian, errors, corrections )
ViewData *datapts;
Matrix jacobian, errors, corrections;
```

```

{
    Matrix jacobian_transpose, combo_inverse, error_J_transpose;
    int i;

    /* The corrections matrix is the error matrix times the inverse
     * of the Jacobian. Since the Jacobian may not be square, the
     * pseudo-inverse is used:
     *
     *      -1      T      T -1
     *      C = E J  = E J (J J )
     */
    for (i = 0; i < NUM_VIEW_PARMS; i++)
        corrections[0][i] = 0.0;

    jacobian_transpose = NewMatrix(NUM_VIEW_PARMS, datapts->numpts*2);
    combo_inverse = NewMatrix(NUM_VIEW_PARMS, NUM_VIEW_PARMS);

    TransposeMatrix( jacobian, jacobian_transpose,
                     NUM_VIEW_PARMS, datapts->numpts*2 );
    MultMatrix( jacobian, jacobian_transpose, combo_inverse,
                NUM_VIEW_PARMS, datapts->numpts*2, NUM_VIEW_PARMS );
    if (InvertMatrix( combo_inverse, NUM_VIEW_PARMS ) == 0.0)
    {
        fprintf(stderr, "Could not invert matrix in iteration!!!\n");
        FreeMatrix(jacobian_transpose, datapts->numpts*2);
        FreeMatrix(combo_inverse, NUM_VIEW_PARMS);
        return;
    }

    error_J_transpose = NewMatrix(NUM_VIEW_PARMS, 1);
    MultMatrix( errors, jacobian_transpose, error_J_transpose, 1,
                datapts->numpts*2, NUM_VIEW_PARMS );
    MultMatrix( error_J_transpose, combo_inverse, corrections, 1,
                NUM_VIEW_PARMS, NUM_VIEW_PARMS );
    FreeMatrix(jacobian_transpose, datapts->numpts*2);
    FreeMatrix(combo_inverse, NUM_VIEW_PARMS);
    FreeMatrix(error_J_transpose, 1);
}

apply_corrections( corrections, view_parms )
Matrix corrections;
ViewParms *view_parms;
{
    ViewParms current_parms;
    Point3 eR;
    Matrix3 inc_rotate;

    current_parms = *view_parms;

    build_rotate(&inc_rotate,
                -corrections[0][3], -corrections[0][4], -corrections[0][5]);
    V2MatMul(&current_parms.view, &inc_rotate, &view_parms->view);
    propagate_rotate_change( view_parms );

    V3MulPointByMatrix(&current_parms.eye, &current_parms.view, &eR);
    eR.x -= corrections[0][0];
    eR.y -= corrections[0][1];
    eR.z -= corrections[0][2];
    V3MulPointByMatrix(&eR, &view_parms->viewinv, &view_parms->eye);
    view_parms->d_over_s -= corrections[0][6];
    view_parms->xcenter -= corrections[0][7];
    view_parms->ycenter -= corrections[0][8];
}

```

```
#ifdef ITERATE_ASPECT_RATIO
    view_parms->aspect -= corrections[0][9];
#endif
    return;
}

propagate_rotate_change( view_parms )
ViewParms *view_parms;
{
    /* inverse is just the transpose of a rotate matrix */
    TransposeMatrix3(&view_parms->view, &view_parms->viewinv);
}

rotate_mat(m, rot_angle, pos_sin_index, neg_sin_index)
Matrix3 *m;
double rot_angle;
int pos_sin_index, neg_sin_index;
{
    double cos_theta, sin_theta;
    int i,j;

    cos_theta = cos( rot_angle );    sin_theta = sin( rot_angle );
    for (i = 0; i < 3; i++)
        for (j = 0; j < 3; j++)
            m->element[i][j] = (i==j) ? 1.0 : 0.0;
    m->element[pos_sin_index][pos_sin_index] = cos_theta;
    m->element[neg_sin_index][neg_sin_index] = cos_theta;
    m->element[pos_sin_index][neg_sin_index] = sin_theta;
    m->element[neg_sin_index][pos_sin_index] = -sin_theta;
}

build_rotate( m, rot_x, rot_y, rot_z )
Matrix3 *m;
double rot_x, rot_y, rot_z;
{
    Matrix3 tmpmat, rotate;

    /* m = Xrotate
     * tmpmat = [m] [Yrotate]
     * m = [tmpmat] [Zrotate]
     */

    rotate_mat(m, rot_x, 1, 2);
    rotate_mat(&rotate, rot_y, 2, 0);
    V2MatMul(m, &rotate, &tmpmat);
    rotate_mat(&rotate, rot_z, 0, 1);
    V2MatMul(&tmpmat, &rotate, m);
}

screen_project(datapt, view_parms, screenpt)
Point3 *datapt;
ViewParms *view_parms;
Point2 *screenpt;
{
    Point3 xyz, data_minus_eye;

    V3Sub(datapt, &view_parms->eye, &data_minus_eye);
    V3MulPointByMatrix(&data_minus_eye, &view_parms->view, &xyz);
    screenpt->x = - (view_parms->d_over_s * view_parms->halfx * xyz.x
                    / xyz.z) + view_parms->xcenter;
    screenpt->y = - (view_parms->aspect * view_parms->d_over_s
```

```
    * view_parms->halfx * xyz.y  
    / xyz.z) + view_parms->ycenter;
```

```
}
```



```
/* viewcorr.h
 *   The global types for view correlation routines.
 */

typedef struct ViewParmsStruct {
    Point3 eye;                /* projection point */
    Matrix3 view;              /* 3x3 rotation matrix */
    Matrix3 viewinv;           /* 3x3 inverse rotation matrix */
    double d_over_s;           /* distance to screen / half screen width */
    double aspect;             /* aspect ratio (for non-square pixels) */
    double halfx, halfy;       /* half of screen resolutions */
    double xcenter, ycenter;   /* center of image */
} ViewParms;

typedef struct ViewDataStruct {
    int numpts;                /* number of data points */
    Point3 *pts;               /* array of three D data points */
    Point2 *scrpts;            /* array of screen data points */
} ViewData;

/* If you cannot handle arbitrary aspect ratios, change the following define
 * to an undef. The iteration will happen with the aspect ratio given in
 * the initial set of ViewParms.
 */
#define ITERATE_ASPECT_RATIO

#ifdef ITERATE_ASPECT_RATIO
#define NUM_VIEW_PARMS 10
#else
#define NUM_VIEW_PARMS 9
#endif
```

```
/*
 * viewfind.c - Simple program to read view and screen data, then run view
 * correlation on it, and dump the results suitable for a raytracing program.
 *
 * Author:      Rod G. Bogart
 * Date:       Oct 15 1990
 * Copyright (c) 1990, University of Michigan
 *
 */
#include <stdio.h>
#include <math.h>
#include "GraphicsGems.h"
#include "matrix.h"
#include "viewcorr.h"

main(argc, argv)
char **argv;
{
    ViewData datapts;
    ViewParms view_parms;
    int num_iterations = 0;

    if (argc >= 2)
        num_iterations = atoi(argv[1]);
    read_points_and_view(stdin, &datapts, &view_parms );
    iterate_view_parms( &datapts, &view_parms, num_iterations );
    dump_points_and_view(stdout, &datapts, &view_parms );
    dump_rayshade_parms(stdout, &view_parms);
}

read_points_and_view( infile, datapts, view_parms )
FILE * infile;
ViewData *datapts;
ViewParms *view_parms;
{
    int i;
    Matrix3 *vm;
    Point3 lookp, up;
    Point3 xvec, yvec, zvec;

    /* read viewparms first, then data points */

    fscanf(infile, "%lf %lf %lf", &(view_parms->eye.x), &(view_parms->eye.y),
           &(view_parms->eye.z));
    fscanf(infile, "%lf %lf %lf", &(lookp.x), &(lookp.y), &(lookp.z));
    fscanf(infile, "%lf %lf %lf", &(up.x), &(up.y), &(up.z));

    /* make coordinate frame unit vectors from eye, lookp, and up */
    V3Sub(&view_parms->eye, &lookp, &zvec);
    V3Normalize( &zvec );
    V3Normalize( &up );
    V3Cross(&up, &zvec, &xvec);
    V3Cross(&zvec, &xvec, &yvec);
    V3Normalize( &xvec );
    V3Normalize( &yvec );

    /* Store the coordinate frame unit vectors as columns to create
     * a rotation matrix
     */
    vm = &(view_parms->view);
    vm->element[0][0] = xvec.x;
```

```
vm->element[0][1] = yvec.x;
vm->element[0][2] = zvec.x;
vm->element[1][0] = xvec.y;
vm->element[1][1] = yvec.y;
vm->element[1][2] = zvec.y;
vm->element[2][0] = xvec.z;
vm->element[2][1] = yvec.z;
vm->element[2][2] = zvec.z;

propagate_rotate_change( view_parms );

fscanf(infile,"%lf %lf",&(view_parms->d_over_s),&(view_parms->aspect));
fscanf(infile,"%lf %lf %lf %lf",&(view_parms->halfx),&(view_parms->halfy),
        &(view_parms->xcenter),&(view_parms->ycenter));

fscanf(infile,"%d",&datapts->numpts);
datapts->pts = (Point3 *) malloc(datapts->numpts * sizeof(Point3));
datapts->scripts = (Point2 *) malloc(datapts->numpts * sizeof(Point2));
for(i=0; i < datapts->numpts; i++)
{
    fscanf(infile,"%lf %lf %lf",&datapts->pts[i].x,&datapts->pts[i].y,
            &datapts->pts[i].z);
    fscanf(infile,"%lf %lf",&datapts->scripts[i].x,&datapts->scripts[i].y);
}
}

dump_points_and_view( dumpfile, datapts, view_parms )
FILE * dumpfile;
ViewData *datapts;
ViewParms *view_parms;
{
    int i;
    Point3 dov, tmp, up;

    tmp.x = 0.0;
    tmp.y = 0.0;
    tmp.z = -1.0;
    V3MulPointByMatrix(&tmp, &view_parms->viewinv, &dov);
    tmp.x = 0.0;
    tmp.y = 1.0;
    tmp.z = 0.0;
    V3MulPointByMatrix(&tmp, &view_parms->viewinv, &up);

    fprintf(dumpfile,"%lf %lf %lf\n",view_parms->eye.x,view_parms->eye.y,
            view_parms->eye.z);

    fprintf(dumpfile,"%lf %lf %lf\n",
            view_parms->eye.x + dov.x,
            view_parms->eye.y + dov.y,
            view_parms->eye.z + dov.z);
    fprintf(dumpfile,"%lf %lf %lf\n", up.x, up.y, up.z);

    fprintf(dumpfile,"%lf %lf\n",view_parms->d_over_s,view_parms->aspect);
    fprintf(dumpfile,"%lf %lf %lf %lf\n",view_parms->halfx, view_parms->halfy,
            view_parms->xcenter, view_parms->ycenter);

    fprintf(dumpfile,"%d\n",datapts->numpts);
    for(i=0; i < datapts->numpts; i++)
    {
        fprintf(dumpfile,"%lf %lf %lf ",datapts->pts[i].x,datapts->pts[i].y,
                datapts->pts[i].z);
    }
}
```

```
    fprintf(dumpfile,"%lf %lf\n",datapts->scripts[i].x,  
           datapts->scripts[i].y);  
}
```

```
dump_rayshade_parms( dumpfile, view_parms )  
FILE * dumpfile;  
ViewParms *view_parms;  
{  
    double ds;  
    int halfx, halfy;  
    Point3 dov, tmp, up;  
  
    tmp.x = 0.0;  
    tmp.y = 0.0;  
    tmp.z = -1.0;  
    V3MulPointByMatrix(&tmp, &view_parms->viewinv, &dov);  
    tmp.x = 0.0;  
    tmp.y = 1.0;  
    tmp.z = 0.0;  
    V3MulPointByMatrix(&tmp, &view_parms->viewinv, &up);  
  
    if (view_parms->halfx > view_parms->xcenter) {  
        halfx = (int) (2.0*view_parms->halfx - view_parms->xcenter);  
    }  
    else  
        halfx = view_parms->xcenter;  
    if (view_parms->halfy > view_parms->ycenter) {  
        halfy = (int) (2.0*view_parms->halfy - view_parms->ycenter);  
    }  
    else  
        halfy = view_parms->ycenter;  
    ds = view_parms->d_over_s;  
    if (ds < 0.0) {  
        V3Negate(&up);  
        ds = -ds;  
    }  
    fprintf(dumpfile,"screen %d %d\n", halfx * 2, halfy * 2);  
    fprintf(dumpfile,"window %d %d %d %d\n",  
           (int) (halfx - view_parms->xcenter),  
           (int) (halfy - view_parms->ycenter),  
           (int) (halfx - view_parms->xcenter) +  
           (int) (view_parms->halfx*2 - 1),  
           (int) (halfy - view_parms->ycenter) +  
           (int) (view_parms->halfy*2 - 1));  
    fprintf(dumpfile,"eyep %lf %lf %lf\n", view_parms->eye.x,  
           view_parms->eye.y, view_parms->eye.z);  
    fprintf(dumpfile,"lookp %lf %lf %lf\n",  
           view_parms->eye.x + dov.x,  
           view_parms->eye.y + dov.y,  
           view_parms->eye.z + dov.z);  
    fprintf(dumpfile,"up %lf %lf %lf\n", up.x, up.y, up.z);  
    fprintf(dumpfile,"fov %lf %lf\n",  
           atan(((double) halfx / view_parms->halfx) *  
                (1.0/ds)) * RTOD * 2.0,  
           atan(((double) halfy / view_parms->halfx) *  
                (1.0/(ds * view_parms->aspect))) * RTOD * 2.0);  
}
```

```
/*
2d and 3d Vector C Library
by Andrew Glassner
from "Graphics Gems", Academic Press, 1990
*/

#include <math.h>
#include "GraphicsGems.h"

/*****
/*    2d Library    */
*****/

/* returns squared length of input vector */
double V2SquaredLength(a)
Vector2 *a;
{
    return((a->x * a->x)+(a->y * a->y));
}

/* returns length of input vector */
double V2Length(a)
Vector2 *a;
{
    return(sqrt(V2SquaredLength(a)));
}

/* negates the input vector and returns it */
Vector2 *V2Negate(v)
Vector2 *v;
{
    v->x = -v->x;  v->y = -v->y;
    return(v);
}

/* normalizes the input vector and returns it */
Vector2 *V2Normalize(v)
Vector2 *v;
{
    double len = V2Length(v);
    if (len != 0.0) { v->x /= len;  v->y /= len; }
    return(v);
}

/* scales the input vector to the new length and returns it */
Vector2 *V2Scale(v, newlen)
Vector2 *v;
double newlen;
{
    double len = V2Length(v);
    if (len != 0.0) { v->x *= newlen/len;  v->y *= newlen/len; }
    return(v);
}

/* return vector sum c = a+b */
Vector2 *V2Add(a, b, c)
Vector2 *a, *b, *c;
{
    c->x = a->x+b->x;  c->y = a->y+b->y;
    return(c);
}
```

```
/* return vector difference c = a-b */
Vector2 *V2Sub(a, b, c)
Vector2 *a, *b, *c;
{
    c->x = a->x-b->x;  c->y = a->y-b->y;
    return(c);
}

/* return the dot product of vectors a and b */
double V2Dot(a, b)
Vector2 *a, *b;
{
    return((a->x*b->x)+(a->y*b->y));
}

/* linearly interpolate between vectors by an amount alpha */
/* and return the resulting vector. */
/* When alpha=0, result=lo.  When alpha=1, result=hi. */
Vector2 *V2Lerp(lo, hi, alpha, result)
Vector2 *lo, *hi, *result;
double alpha;
{
    result->x = LERP(alpha, lo->x, hi->x);
    result->y = LERP(alpha, lo->y, hi->y);
    return(result);
}

/* make a linear combination of two vectors and return the result. */
/* result = (a * ascl) + (b * bscl) */
Vector2 *V2Combine(a, b, result, ascl, bscl)
Vector2 *a, *b, *result;
double ascl, bscl;
{
    result->x = (ascl * a->x) + (bscl * b->x);
    result->y = (ascl * a->y) + (bscl * b->y);
    return(result);
}

/* multiply two vectors together component-wise */
Vector2 *V2Mul(a, b, result)
Vector2 *a, *b, *result;
{
    result->x = a->x * b->x;
    result->y = a->y * b->y;
    return(result);
}

/* return the distance between two points */
double V2DistanceBetween2Points(a, b)
Point2 *a, *b;
{
    double dx = a->x - b->x;
    double dy = a->y - b->y;
    return(sqrt((dx*dx)+(dy*dy)));
}

/* return the vector perpendicular to the input vector a */
Vector2 *V2MakePerpendicular(a, ap)
Vector2 *a, *ap;
```

```
{
    ap->x = -a->y;
    ap->y = a->x;
    return(ap);
}

/* create, initialize, and return a new vector */
Vector2 *V2New(x, y)
double x, y;
{
    Vector2 *v = NEWTYPE(Vector2);
    v->x = x;  v->y = y;
    return(v);
}

/* create, initialize, and return a duplicate vector */
Vector2 *V2Duplicate(a)
Vector2 *a;
{
    Vector2 *v = NEWTYPE(Vector2);
    v->x = a->x;  v->y = a->y;
    return(v);
}

/* multiply a point by a projective matrix and return the transformed point */
Point2 *V2MulPointByProjMatrix(pin, m, pout)
Point2 *pin, *pout;
Matrix3 *m;
{
    double w;
    pout->x = (pin->x * m->element[0][0]) +
        (pin->y * m->element[1][0]) + m->element[2][0];
    pout->y = (pin->x * m->element[0][1]) +
        (pin->y * m->element[1][1]) + m->element[2][1];
    w = (pin->x * m->element[0][2]) +
        (pin->y * m->element[1][2]) + m->element[2][2];
    if (w != 0.0) { pout->x /= w;  pout->y /= w; }
    return(pout);
}

/* multiply together matrices c = ab */
/* note that c must not point to either of the input matrices */
Matrix3 *V2MatMul(a, b, c)
Matrix3 *a, *b, *c;
{
    int i, j, k;
    for (i=0; i<3; i++) {
        for (j=0; j<3; j++) {
            c->element[i][j] = 0;
            for (k=0; k<3; k++) c->element[i][j] +=
                a->element[i][k] * b->element[k][j];
        }
    }
    return(c);
}

/* transpose rotation portion of matrix a, return b */
Matrix3 *TransposeMatrix3(a, b)
Matrix3 *a, *b;
{

```

```
int i, j;
    for (i=0; i<3; i++) {
        for (j=0; j<3; j++)
            b->element[i][j] = a->element[j][i];
    }
    return(b);
}

/*****
/*    3d Library    */
*****/

/* returns squared length of input vector */
double V3SquaredLength(a)
Vector3 *a;
{
    return((a->x * a->x)+(a->y * a->y)+(a->z * a->z));
}

/* returns length of input vector */
double V3Length(a)
Vector3 *a;
{
    return(sqrt(V3SquaredLength(a)));
}

/* negates the input vector and returns it */
Vector3 *V3Negate(v)
Vector3 *v;
{
    v->x = -v->x;  v->y = -v->y;  v->z = -v->z;
    return(v);
}

/* normalizes the input vector and returns it */
Vector3 *V3Normalize(v)
Vector3 *v;
{
    double len = V3Length(v);
    if (len != 0.0) { v->x /= len;  v->y /= len; v->z /= len; }
    return(v);
}

/* scales the input vector to the new length and returns it */
Vector3 *V3Scale(v, newlen)
Vector3 *v;
double newlen;
{
    double len = V3Length(v);
    if (len != 0.0) {
        v->x *= newlen/len;  v->y *= newlen/len;  v->z *= newlen/len;
    }
    return(v);
}

/* return vector sum c = a+b */
Vector3 *V3Add(a, b, c)
```



```
Vector3 *a, *b, *c;
{
    c->x = a->x+b->x;  c->y = a->y+b->y;  c->z = a->z+b->z;
    return(c);
}

/* return vector difference c = a-b */
Vector3 *V3Sub(a, b, c)
Vector3 *a, *b, *c;
{
    c->x = a->x-b->x;  c->y = a->y-b->y;  c->z = a->z-b->z;
    return(c);
}

/* return the dot product of vectors a and b */
double V3Dot(a, b)
Vector3 *a, *b;
{
    return((a->x*b->x)+(a->y*b->y)+(a->z*b->z));
}

/* linearly interpolate between vectors by an amount alpha */
/* and return the resulting vector. */
/* When alpha=0, result=lo.  When alpha=1, result=hi. */
Vector3 *V3Lerp(lo, hi, alpha, result)
Vector3 *lo, *hi, *result;
double alpha;
{
    result->x = LERP(alpha, lo->x, hi->x);
    result->y = LERP(alpha, lo->y, hi->y);
    result->z = LERP(alpha, lo->z, hi->z);
    return(result);
}

/* make a linear combination of two vectors and return the result. */
/* result = (a * ascl) + (b * bscl) */
Vector3 *V3Combine (a, b, result, ascl, bscl)
Vector3 *a, *b, *result;
double ascl, bscl;
{
    result->x = (ascl * a->x) + (bscl * b->x);
    result->y = (ascl * a->y) + (bscl * b->y);
    result->z = (ascl * a->z) + (bscl * b->z);
    return(result);
}

/* multiply two vectors together component-wise and return the result */
Vector3 *V3Mul (a, b, result)
Vector3 *a, *b, *result;
{
    result->x = a->x * b->x;
    result->y = a->y * b->y;
    result->z = a->z * b->z;
    return(result);
}

/* return the distance between two points */
double V3DistanceBetween2Points(a, b)
Point3 *a, *b;
{

```

```
double dx = a->x - b->x;
double dy = a->y - b->y;
double dz = a->z - b->z;
    return(sqrt((dx*dx)+(dy*dy)+(dz*dz)));
}

/* return the cross product c = a cross b */
Vector3 *V3Cross(a, b, c)
Vector3 *a, *b, *c;
{
    c->x = (a->y*b->z) - (a->z*b->y);
    c->y = (a->z*b->x) - (a->x*b->z);
    c->z = (a->x*b->y) - (a->y*b->x);
    return(c);
}

/* create, initialize, and return a new vector */
Vector3 *V3New(x, y, z)
double x, y, z;
{
Vector3 *v = NEWTYPE(Vector3);
    v->x = x;  v->y = y;  v->z = z;
    return(v);
}

/* create, initialize, and return a duplicate vector */
Vector3 *V3Duplicate(a)
Vector3 *a;
{
Vector3 *v = NEWTYPE(Vector3);
    v->x = a->x;  v->y = a->y;  v->z = a->z;
    return(v);
}

/* multiply a point by a matrix and return the transformed point */
Point3 *V3MulPointByMatrix(pin, m, pout)
Point3 *pin, *pout;
Matrix3 *m;
{
    pout->x = (pin->x * m->element[0][0]) + (pin->y * m->element[1][0]) +
              (pin->z * m->element[2][0]);
    pout->y = (pin->x * m->element[0][1]) + (pin->y * m->element[1][1]) +
              (pin->z * m->element[2][1]);
    pout->z = (pin->x * m->element[0][2]) + (pin->y * m->element[1][2]) +
              (pin->z * m->element[2][2]);
    return(pout);
}

/* multiply a point by a projective matrix and return the transformed point */
Point3 *V3MulPointByProjMatrix(pin, m, pout)
Point3 *pin, *pout;
Matrix4 *m;
{
double w;
    pout->x = (pin->x * m->element[0][0]) + (pin->y * m->element[1][0]) +
              (pin->z * m->element[2][0]) + m->element[3][0];
    pout->y = (pin->x * m->element[0][1]) + (pin->y * m->element[1][1]) +
              (pin->z * m->element[2][1]) + m->element[3][1];
    pout->z = (pin->x * m->element[0][2]) + (pin->y * m->element[1][2]) +
              (pin->z * m->element[2][2]) + m->element[3][2];
}
```

```
w = (pin->x * m->element[0][3]) + (pin->y * m->element[1][3]) +
    (pin->z * m->element[2][3]) + m->element[3][3];
if (w != 0.0) { pout->x /= w; pout->y /= w; pout->z /= w; }
return(pout);
}
```

```
/* multiply together matrices c = ab */
/* note that c must not point to either of the input matrices */
Matrix4 *V3MatMul(a, b, c)
Matrix4 *a, *b, *c;
{
    int i, j, k;
    for (i=0; i<4; i++) {
        for (j=0; j<4; j++) {
            c->element[i][j] = 0;
            for (k=0; k<4; k++) c->element[i][j] +=
                a->element[i][k] * b->element[k][j];
        }
    }
    return(c);
}
```

```
/* binary greatest common divisor by Silver and Terzian. See Knuth */
/* both inputs must be >= 0 */
gcd(u, v)
int u, v;
{
    int t, f;
    if ((u<0) || (v<0)) return(1); /* error if u<0 or v<0 */
    f = 1;
    while ((0 == (u%2)) && (0 == (v%2))) {
        u>>=1; v>>=1, f*=2;
    }
    if (u&01) { t = -v; goto B4; } else { t = u; }
    B3: if (t > 0) { t >>= 1; } else { t = -((-t) >> 1); }
    B4: if (0 == (t%2)) goto B3;

    if (t > 0) u = t; else v = -t;
    if (0 != (t = u - v)) goto B3;
    return(u*f);
}
```

```
/******
/* Useful Routines */
/******
```

```
/* return roots of ax^2+bx+c */
/* stable algebra derived from Numerical Recipes by Press et al.*/
int quadraticRoots(a, b, c, roots)
double a, b, c, *roots;
{
    double d, q;
    int count = 0;
    d = (b*b)-(4*a*c);
    if (d < 0.0) { *roots = *(roots+1) = 0.0; return(0); }
    q = -0.5 * (b + (SGN(b)*sqrt(d)));
    if (a != 0.0) { *roots++ = q/a; count++; }
    if (q != 0.0) { *roots++ = c/q; count++; }
    return(count);
}
```

```
/* generic 1d regula-falsi step. f is function to evaluate */
/* interval known to contain root is given in left, right */
/* returns new estimate */
double RegulaFalsi(f, left, right)
double (*f)(), left, right;
{
double d = (*f)(right) - (*f)(left);
    if (d != 0.0) return (right - (*f)(right)*(right-left)/d);
    return((left+right)/2.0);
}

/* generic 1d Newton-Raphson step. f is function, df is derivative */
/* x is current best guess for root location. Returns new estimate */
double NewtonRaphson(f, df, x)
double (*f)(), (*df)(), x;
{
double d = (*df)(x);
    if (d != 0.0) return (x-((*f)(x)/d));
    return(x-1.0);
}

/* hybrid 1d Newton-Raphson/Regula Falsi root finder. */
/* input function f and its derivative df, an interval */
/* left, right known to contain the root, and an error tolerance */
/* Based on Blinn */
double findroot(left, right, tolerance, f, df)
double left, right, tolerance;
double (*f)(), (*df)();
{
double newx = left;
    while (ABS((*f)(newx)) > tolerance) {
        newx = NewtonRaphson(f, df, newx);
        if (newx < left || newx > right)
            newx = RegulaFalsi(f, left, right);
        if ((*f)(newx) * (*f)(left) <= 0.0) right = newx;
        else left = newx;
    }
    return(newx);
}
```

```
/*
 * GraphicsGems.h
 * Version 1.0 - Andrew Glassner
 * from "Graphics Gems", Academic Press, 1990
 */

#ifndef GG_H

#define GG_H 1

/*****
 * 2d geometry types */
*****/

typedef struct Point2Struct {    /* 2d point */
    double x, y;
} Point2;
typedef Point2 Vector2;

typedef struct IntPoint2Struct {    /* 2d integer point */
    int x, y;
} IntPoint2;

typedef struct Matrix3Struct {    /* 3-by-3 matrix */
    double element[3][3];
} Matrix3;

typedef struct Box2dStruct {    /* 2d box */
    Point2 min, max;
} Box2;

/*****
 * 3d geometry types */
*****/

typedef struct Point3Struct {    /* 3d point */
    double x, y, z;
} Point3;
typedef Point3 Vector3;

typedef struct IntPoint3Struct {    /* 3d integer point */
    int x, y, z;
} IntPoint3;

typedef struct Matrix4Struct {    /* 4-by-4 matrix */
    double element[4][4];
} Matrix4;

typedef struct Box3dStruct {    /* 3d box */
    Point3 min, max;
} Box3;

/*****
 * one-argument macros */
*****/

/* absolute value of a */
```

```
#define ABS(a)          (((a)<0) ? -(a) : (a))

/* round a to nearest int */
#define ROUND(a)        floor((a)+0.5)

/* take sign of a, either -1, 0, or 1 */
#define ZSGN(a)         (((a)<0) ? -1 : (a)>0 ? 1 : 0)

/* take binary sign of a, either -1, or 1 if >= 0 */
#define SGN(a)          (((a)<0) ? -1 : 1)

/* shout if something that should be true isn't */
#define ASSERT(x) \
if (!(x)) fprintf(stderr," Assert failed: x\n");

/* square a */
#define SQR(a)          ((a)*(a))

/*****/
/* two-argument macros */
/*****/

/* find minimum of a and b */
#define MIN(a,b)        (((a)<(b))?(a):(b))

/* find maximum of a and b */
#define MAX(a,b)        (((a)>(b))?(a):(b))

/* swap a and b (see Gem by Wyvill) */
#define SWAP(a,b)       { a^=b; b^=a; a^=b; }

/* linear interpolation from l (when a=0) to h (when a=1)*/
/* (equal to (a*h)+((1-a)*l) */
#define LERP(a,l,h)      ((l)+(((h)-(l))*(a)))

/* clamp the input to the specified range */
#define CLAMP(v,l,h)     ((v)<(l) ? (l) : (v) > (h) ? (h) : v)

/*****/
/* memory allocation macros */
/*****/

/* create a new instance of a structure (see Gem by Hultquist) */
#define NEWSTRUCT(x)     (struct x *) (malloc((unsigned)sizeof(struct x)))

/* create a new instance of a type */
#define NEWTYPE(x)        (x *) (malloc((unsigned)sizeof(x)))

/*****/
/* useful constants */
/*****/

#define PI                3.141592          /* the venerable pi */
#define PITIMES2          6.283185          /* 2 * pi */
#define PIOVER2           1.570796          /* pi / 2 */
#define E                 2.718282          /* the venerable e */
#define SQR2              1.414214          /* sqrt(2) */
#define SQR3              1.732051          /* sqrt(3) */
```

```
#define GOLDEN          1.618034          /* the golden ratio */
#define DTOR            0.017453          /* convert degrees to radians */
#define RTOD            57.29578          /* convert radians to degrees */

/*****/
/* booleans */
/*****/

#define TRUE            1
#define FALSE          0
#define ON              1
#define OFF            0
typedef int boolean;    /* boolean data type */
typedef boolean flag;   /* flag data type */

extern double V2SquaredLength(), V2Length();
extern double V2Dot(), V2DistanceBetween2Points();
extern Vector2 *V2Negate(), *V2Normalize(), *V2Scale(), *V2Add(), *V2Sub();
extern Vector2 *V2Lerp(), *V2Combine(), *V2Mul(), *V2MakePerpendicular();
extern Vector2 *V2New(), *V2Duplicate();
extern Point2 *V2MulPointByProjMatrix();
extern Matrix3 *V2MatMul(), *TransposeMatrix3();



extern double V3SquaredLength(), V3Length();
extern double V3Dot(), V3DistanceBetween2Points();
extern Vector3 *V3Normalize(), *V3Scale(), *V3Add(), *V3Sub();
extern Vector3 *V3Lerp(), *V3Combine(), *V3Mul(), *V3Cross();
extern Vector3 *V3New(), *V3Duplicate();
extern Point3 *V3MulPointByMatrix(), *V3MulPointByProjMatrix();
extern Matrix4 *V3MatMul();

extern double RegulaFalsi(), NewtonRaphson(), findroot();

#endif
```

# Index of

## /pubs/tog/GraphicsGems/gemsv/ch7-3/

Name	Last modified	Size	Description
 <a href="#">_Parent Directory</a>			
 <a href="#">_collide.cc</a>	29-Jun-00 08:24	4K	



```
// -*- C++ -*-
// by Bill Bouma and George Vaneczek Jr. Aug, 1994.
// Compile by: g++ -O2 -s -o cull cull.cc algebra3.o -lm
#include "algebra3.h" // See Graphics Gems IV, pg534-557
typedef vec3         Point; // Points are not Vectors
typedef vec3         Vector; // Vectors are not Points
typedef unsigned int Index; // Array Indices
typedef unsigned int Counter;

class Polygon {
public:
    Polygon          ( const char          pId,
                      const Vector&        nV,
                      const Counter        nPs,
                      const Point* const p )
        : id(pId), pts(p), nPts(nPs), normalVector(nV){ }
    const Vector& normal( ) const { return normalVector; }
    char          name( ) const { return id; }
    Counter        nPoints( ) const { return nPts; }
    const Point&   point( const Index i ) const { return pts[i]; }
private:
    const char          id; // Unique Id
    const Counter        nPts; // pts[0..nPts-1]
    const Point* const pts; // Points around Polygon
    const Vector        normalVector; // Unit Vector
};

class MovingPolyhedron {
public:
    MovingPolyhedron ( const char          pId,
                      const Vector&        rv,
                      const Vector&        vv,
                      const Vector&        wv,
                      const mat4&          m,
                      const Counter        nP,
                      const Polygon* const ps )
        : id(pId), r(rv), v(vv), w(wv), R(m), polys(ps), nPolys(nP) { }
    const Polygon&   polygon( const Index i ) const { return polys[i]; }
    void             cull( const MovingPolyhedron& ) const;
private:
    const char          id; // Unique Id
    const Polygon* const polys; // Points in local coordinates
    const Counter        nPolys; // polys[0..nPolys-1]
    Vector              r; // Center of Rotation (in world coords.)
    Vector              v; // Linear Velocity (in world coords.)
    Vector              w; // Angular Velocity (in world coords.)
    mat4                R; // Orientation Matrix
};

void MovingPolyhedron::cull( const MovingPolyhedron& j ) const
{
    const mat4  RIi = ((mat4&R).transpose());
    const Vector aij = RIi * (v - j.v - (w ^ r) + (j.w ^ j.r));
    const Vector wij = RIi * (j.w - w);
    for( Index gi = 0; gi < nPolys; ++gi ) {
        const Polygon& g = polygon(gi);
        for( Index pi = 0; pi < g.nPoints(); ++pi )
            if( ( aij + (g.point(pi) ^ wij) ) * g.normal() > 0.0 )
                break;
        cout << "Polygon " << g.name() << " of Polyhedron " << id
              << " is" << ( pi == g.nPoints() ? " " : " not " )
    }
}
```

```
<< "culled." << endl;
```

```
    }  
}  
  
const Counter NPolyPoints = 4;  
const Counter NFaces      = 6;  
static const Point leftPoints[NPolyPoints] = {  
    Point(-1,-1,-1), Point(-1,-1, 1), Point(-1, 1, 1), Point(-1, 1,-1) };  
static const Point rightPoints[NPolyPoints] = {  
    Point( 1,-1,-1), Point( 1, 1,-1), Point( 1, 1, 1), Point( 1,-1, 1) };  
static const Point topPoints[NPolyPoints]   = {  
    Point(-1, 1,-1), Point(-1, 1, 1), Point( 1, 1, 1), Point( 1, 1,-1) };  
static const Point bottomPoints[NPolyPoints]= {  
    Point(-1,-1,-1), Point( 1,-1,-1), Point( 1,-1, 1), Point(-1,-1, 1) };  
static const Point backPoints[NPolyPoints]  = {  
    Point(-1,-1,-1), Point(-1, 1,-1), Point( 1, 1,-1), Point( 1,-1,-1) };  
static const Point frontPoints[NPolyPoints] = {  
    Point(-1,-1, 1), Point( 1,-1, 1), Point( 1, 1, 1), Point(-1, 1, 1) };  
static const Polygon cube[NFaces]= {  
    Polygon( 'a', Vector(-1, 0, 0), NPolyPoints, leftPoints  ),  
    Polygon( 'b', Vector( 1, 0, 0), NPolyPoints, rightPoints ),  
    Polygon( 'c', Vector( 0, 1, 0), NPolyPoints, topPoints   ),  
    Polygon( 'd', Vector( 0,-1, 0), NPolyPoints, bottomPoints ),  
    Polygon( 'e', Vector( 0, 0,-1), NPolyPoints, backPoints  ),  
    Polygon( 'f', Vector( 0, 0, 1), NPolyPoints, frontPoints  )  
};
```

```
int main()  
{  
    MovingPolyhedron A( 'A',  
        Vector(10,10, 0 ), // Position  
        Vector( 0, 0, 0 ), // Velocity  
        Vector( 0, 0, 0 ), // Angular Velocity  
        identity3D(),  
        NFaces, cube );  
    MovingPolyhedron B( 'B',  
        Vector(10,10,10 ), // Position  
        Vector( 0, 0,-1 ), // Velocity  
        Vector( 0, 1, 0 ), // Angular Velocity  
        identity3D(),  
        NFaces, cube );  
  
    A.cull( B );  
    B.cull( A );  
}
```

\*\*\*\*\*/

```
    if (q >= 0 && q <= 255)
        rgb[i] = q;

    /* mask off lower 3 bits */
    rgb[i] &= 0xf8;
}
```

```
}
```

```
/*
 *
 * name          jitter_init - initialize jitter look-up tables
 *
 *              Adapted from Graphic Gems I (Cyshosz, page 64).
 *
 * notes         This function should be called once before any call to
 *              rgbvary()
 */
```

```
void jitter_init()
{
    int i;

    /* initialize look-up tables */
    for (i = 0; i < JITTER_TABLE_SIZE; ++i)
        uranx[i] = (double)(rand() % LARGE_NUMBER) / (double)LARGE_NUMBER;
    for (i = 0; i < JITTER_TABLE_SIZE; ++i)
        urany[i] = (double)(rand() % LARGE_NUMBER) / (double)LARGE_NUMBER;
    for (i = 0; i < JITTER_TABLE_SIZE; ++i)
        irand[i] = (int)((double)JITTER_TABLE_SIZE
            * ((double)(rand() % LARGE_NUMBER) / (double)LARGE_NUMBER));
}
```

```
/*
    rgbvaryW.c - an improved rgbvary.c, currently runs on Microsoft Windows

    The color reduction filter (rgbvary.c) uses floating point but none
    of the computations exceed a 16bit integer. I've rewritten the code to
    use 16 bit integers (with no loss of precision), updated jitter_init to
    precompute the scaled values used in the inner loop, and allowed the
    code to output a C include file with the table values filled in. The
    combined savings reduced the filter time on a 1024x768x24bit image from
    24 seconds to 3 seconds, reduced the table size by 12K (1/3 the
    original size), and removed 18K of runtime support for floating point
    emulation, the rand() library, etc.

    From: Ken Sykes <kensy@microsoft.com>
*/

/*****
    rgbvaryI.c - a color quantization pre-processor
*****/

/* remove the next line after the first run */
#define GENTABLE 1

#ifndef GENTABLE
#include "rgbvtab.h"
#endif

#define JITTER_TABLE_BITS    10
#define JITTER_TABLE_SIZE    (1<<JITTER_TABLE_BITS)
#define JITTER_MASK          (JITTER_TABLE_SIZE-1)

/* jitter macros */
#define jitterx(x,y,s) \
(ur anx[(x+(y<<2))+irand[(x+s)&JITTER_MASK]]&JITTER_MASK])
#define jittery(x,y,s) \
(ur any[(y+(x<<2))+irand[(y+s)&JITTER_MASK]]&JITTER_MASK])

#ifdef GENTABLE

/* global variables */
int irand[JITTER_TABLE_SIZE];    /* jitter look-up table */
int ur anx[JITTER_TABLE_SIZE];   /* jitter look-up table */
int ur any[JITTER_TABLE_SIZE];   /* jitter look-up table */

#include <stdio.h>

void print_tables(void)
{
    FILE *fp;
    int i;

    fp = fopen("rgbvtab.h", "w");

    fprintf(fp, "int irand[%d] = {", JITTER_TABLE_SIZE);
    for (i = 0; i < JITTER_TABLE_SIZE; ++i)
    {
        if (!(i % 8)) fprintf(fp, "\n\t");
        fprintf(fp, "%d,", irand[i]);
    }
}
```

```
fprintf(fp, "\n};\n");

fprintf(fp, "int uranx[%d] = {", JITTER_TABLE_SIZE);
for (i = 0; i < JITTER_TABLE_SIZE; ++i)
{
    if (!(i % 8)) fprintf(fp, "\n\t");
    fprintf(fp, "%d,", uranx[i]);
}
fprintf(fp, "\n};\n");

fprintf(fp, "int urany[%d] = {", JITTER_TABLE_SIZE);
for (i = 0; i < JITTER_TABLE_SIZE; ++i)
{
    if (!(i % 8)) fprintf(fp, "\n\t");
    fprintf(fp, "%d,", urany[i]);
}
fprintf(fp, "\n};\n");

fclose(fp);
}

/* function declarations */
void jitter_init();

/*
 *
 * name          jitter_init - initialize jitter look-up tables
 *
 *              Adapted from Graphic Gems I (Cyshosz, page 64).
 *
 * notes         This function should be called once before any call to
 *              rgbvary()
 */

void jitter_init()
{
    int i, urand;

/* magnitude of noise generated (0,1,2 allowed values) */
#define NOISE_LEVEL 2

/* determine order to fill table in */
for (i = 0; i < JITTER_TABLE_SIZE; ++i)
    irand[i] = rand() % JITTER_TABLE_SIZE;

/* fill in the X table */
for (i = 0; i < JITTER_TABLE_SIZE; ++i)
{
    uranx[i] = urand = rand() % JITTER_TABLE_SIZE;
    uranx[i] = ((urand << 3) + urand) >> JITTER_TABLE_BITS;
}

/* fill in the Y table */
for (i = 0; i < JITTER_TABLE_SIZE; ++i)
{
    urany[i] = urand = rand() % JITTER_TABLE_SIZE;
    urany[i] = (((urand * NOISE_LEVEL * 2) + JITTER_TABLE_SIZE/2)
                >> JITTER_TABLE_BITS) - NOISE_LEVEL) << 3;
}
```

```
    print_tables();
}
#endif

BOOL VaryDIB24(HANDLE hdib)
{
    LPBITMAPINFOHEADER lpbi;
    HPBYTE hpbyBits;
    WORD wNextScan;
    int x,y;

    /* Get pointer to bits */
    if (!hdib) return FALSE;
    lpbi = (LPBITMAPINFOHEADER)GlobalLock(hdib);
    GlobalUnlock(hdib);

    if (lpbi->biBitCount != 24)
    {
        ErrMsg("Jitter only applies to 24bpp images.");
        return FALSE;
    }

    /* Make sure it is not compressed */
    if (lpbi->biCompression != BI_RGB)
    {
        ErrMsg("Jitter only works on uncompressed images.");
        return FALSE;
    }
}

#ifdef GENTABLE
    jitter_init();
#endif

    hpbyBits = (HPBYTE)DibXY(lpbi,0,0);

    /* DIBs are DWORD-aligned. Determine amount of padding for each
scanline */
    wNextScan = (WORD)((((lpbi->biWidth << 1)+lpbi->biWidth) & 3);
    if (wNextScan) wNextScan = (WORD)(4-wNextScan);

    /* Loop through the image & apply the jitter */
    for (y = 0; y < lpbi->biHeight; ++y)
    {
        for (x = 0; x < lpbi->biWidth; ++x)
        {
            int i, p, q;

            for (i = 0; i < 3; ++i)
            {
                int rgb;

                rgb = (int)*hpbyBits;

                if (rgb < 248)
                {
                    /* bump up to next intensity if there is enough jitter */
                    p = rgb & 7;
                    q = jitterx(x, y, i);
                    if (p <= q)
                        rgb += 8;
                }
            }
        }
    }
}
```

```
        /* add some noise */
        q = rgb + jittery(x, y, i);

        /* make sure resulting intensity is within allowable
range */
        if (q >= 0 && q <= 255)
            rgb = q;

        /* mask off lower 3 bits & store */
        *hpbyBits = rgb & 0xf8;
    }

    /* update pointer */
    ++hpbyBits;
}

/* move pointer to next scan */
hpbyBits += wNextScan;
}

return TRUE;
}
```



```
/*
Matrix Inversion
by Richard Carling
from "Graphics Gems", Academic Press, 1990
*/

#define SMALL_NUMBER    1.e-8

/*
 *   inverse( original_matrix, inverse_matrix )
 *
 *   calculate the inverse of a 4x4 matrix
 *
 *   
$$A^{-1} = \frac{1}{\det A} \text{adjoint } A$$

 */

#include "GraphicsGems.h"
#include <math.h>
inverse( in, out ) Matrix4 *in, *out;
{
    int i, j;
    double det, det4x4();

    /* calculate the adjoint matrix */

    adjoint( in, out );

    /* calculate the 4x4 determinant
     * if the determinant is zero,
     * then the inverse matrix is not unique.
     */

    det = det4x4( in );

    if ( fabs( det ) < SMALL_NUMBER ) {
        printf("Non-singular matrix, no inverse!\n");
        exit(1);
    }

    /* scale the adjoint matrix to get the inverse */

    for (i=0; i<4; i++)
        for(j=0; j<4; j++)
            out->element[i][j] = out->element[i][j] / det;
}

/*
 *   adjoint( original_matrix, inverse_matrix )
 *
 *   calculate the adjoint of a 4x4 matrix
 *
 *   Let  $a_{ij}$  denote the minor determinant of matrix A obtained by
 *   deleting the  $i$ th row and  $j$ th column from A.
 *
 *   Let  $b_{ij} = (-1)^{i+j} a_{ji}$ 
 */
```

```
*
*   The matrix B = (bij) is the adjoint of A
*
*/
```

```
adjoint( in, out ) Matrix4 *in; Matrix4 *out;
{
    double a1, a2, a3, a4, b1, b2, b3, b4;
    double c1, c2, c3, c4, d1, d2, d3, d4;
    double det3x3();

    /* assign to individual variable names to aid */
    /* selecting correct values */

    a1 = in->element[0][0]; b1 = in->element[0][1];
    c1 = in->element[0][2]; d1 = in->element[0][3];

    a2 = in->element[1][0]; b2 = in->element[1][1];
    c2 = in->element[1][2]; d2 = in->element[1][3];

    a3 = in->element[2][0]; b3 = in->element[2][1];
    c3 = in->element[2][2]; d3 = in->element[2][3];

    a4 = in->element[3][0]; b4 = in->element[3][1];
    c4 = in->element[3][2]; d4 = in->element[3][3];

    /* row column labeling reversed since we transpose rows & columns */

    out->element[0][0] = det3x3( b2, b3, b4, c2, c3, c4, d2, d3, d4);
    out->element[1][0] = - det3x3( a2, a3, a4, c2, c3, c4, d2, d3, d4);
    out->element[2][0] = det3x3( a2, a3, a4, b2, b3, b4, d2, d3, d4);
    out->element[3][0] = - det3x3( a2, a3, a4, b2, b3, b4, c2, c3, c4);

    out->element[0][1] = - det3x3( b1, b3, b4, c1, c3, c4, d1, d3, d4);
    out->element[1][1] = det3x3( a1, a3, a4, c1, c3, c4, d1, d3, d4);
    out->element[2][1] = - det3x3( a1, a3, a4, b1, b3, b4, d1, d3, d4);
    out->element[3][1] = det3x3( a1, a3, a4, b1, b3, b4, c1, c3, c4);

    out->element[0][2] = det3x3( b1, b2, b4, c1, c2, c4, d1, d2, d4);
    out->element[1][2] = - det3x3( a1, a2, a4, c1, c2, c4, d1, d2, d4);
    out->element[2][2] = det3x3( a1, a2, a4, b1, b2, b4, d1, d2, d4);
    out->element[3][2] = - det3x3( a1, a2, a4, b1, b2, b4, c1, c2, c4);

    out->element[0][3] = - det3x3( b1, b2, b3, c1, c2, c3, d1, d2, d3);
    out->element[1][3] = det3x3( a1, a2, a3, c1, c2, c3, d1, d2, d3);
    out->element[2][3] = - det3x3( a1, a2, a3, b1, b2, b3, d1, d2, d3);
    out->element[3][3] = det3x3( a1, a2, a3, b1, b2, b3, c1, c2, c3);
}
/*
* double = det4x4( matrix )
*
* calculate the determinant of a 4x4 matrix.
*/
double det4x4( m ) Matrix4 *m;
{
    double ans;
    double a1, a2, a3, a4, b1, b2, b3, b4, c1, c2, c3, c4, d1, d2, d3,
d4;
    double det3x3();
```

```
/* assign to individual variable names to aid selecting */
/* correct elements */

a1 = m->element[0][0]; b1 = m->element[0][1];
c1 = m->element[0][2]; d1 = m->element[0][3];

a2 = m->element[1][0]; b2 = m->element[1][1];
c2 = m->element[1][2]; d2 = m->element[1][3];

a3 = m->element[2][0]; b3 = m->element[2][1];
c3 = m->element[2][2]; d3 = m->element[2][3];

a4 = m->element[3][0]; b4 = m->element[3][1];
c4 = m->element[3][2]; d4 = m->element[3][3];

ans = a1 * det3x3( b2, b3, b4, c2, c3, c4, d2, d3, d4)
      - b1 * det3x3( a2, a3, a4, c2, c3, c4, d2, d3, d4)
      + c1 * det3x3( a2, a3, a4, b2, b3, b4, d2, d3, d4)
      - d1 * det3x3( a2, a3, a4, b2, b3, b4, c2, c3, c4);
return ans;
}

/*
 * double = det3x3( a1, a2, a3, b1, b2, b3, c1, c2, c3 )
 *
 * calculate the determinant of a 3x3 matrix
 * in the form
 *
 *      | a1,  b1,  c1 |
 *      | a2,  b2,  c2 |
 *      | a3,  b3,  c3 |
 */

double det3x3( a1, a2, a3, b1, b2, b3, c1, c2, c3 )
double a1, a2, a3, b1, b2, b3, c1, c2, c3;
{
    double ans;
    double det2x2();

    ans = a1 * det2x2( b2, b3, c2, c3 )
          - b1 * det2x2( a2, a3, c2, c3 )
          + c1 * det2x2( a2, a3, b2, b3 );
    return ans;
}




/*
 * double = det2x2( double a, double b, double c, double d )
 *
 * calculate the determinant of a 2x2 matrix.
 */

double det2x2( a, b, c, d)
double a, b, c, d;
{
    double ans;
    ans = a * d - b * c;
    return ans;
}
```



# Index of

## /pubs/tog/GraphicsGems/gemsv/ch2-2/

Name	Last modified	Size	Description
 <a href="#">_Parent Directory</a>			
 <a href="#">_ptinply3.c</a>	29-Jun-00 08:22	4K	
 <a href="#">_rev.c</a>	29-Jun-00 08:22	1K	

```
#include <math.h>
#include <stdlib.h>
#include <stdio.h>

#ifndef max
#define max(a,b) ((a)>(b)?(a):(b))
#define min(a,b) ((a)<(b)?(a):(b))
#endif
#define PI 3.141592653589793324
#define GeoZeroVec(v) ((v).x = (v).y = (v).z = 0.0)
#define GeoMultVec(a,b,c) do {(c).x = a*(b).x; (c).y = a*(b).y; (c).z = a*(b).z; } while (0)
#define Geo_Vet(a,b,c) do {(c).x = (b).x-(a).x; (c).y = (b).y-(a).y; (c).z = (b).z-(a).z; } while (0)

typedef double Rdouble;
typedef float Rfloat;
typedef struct _GeoPoint { Rfloat x, y, z; } GeoPoint;

/*===== Geometrical Procedures ===== */

Rdouble GeoDotProd ( GeoPoint *vec0, GeoPoint *vec1 )
{
    return ( vec0->x * vec1->x + vec0->y * vec1->y + vec0->z * vec1->z );
}

void GeoCrossProd ( GeoPoint *in0, GeoPoint *in1, GeoPoint *out )
{
    out->x = (in0->y * in1->z) - (in0->z * in1->y);
    out->y = (in0->z * in1->x) - (in0->x * in1->z);
    out->z = (in0->x * in1->y) - (in0->y * in1->x);
}

Rdouble GeoTripleProd ( GeoPoint *vec0, GeoPoint *vec1, GeoPoint *vec2 )
{
    GeoPoint tmp;

    GeoCrossProd ( vec0, vec1, &tmp );
    return ( GeoDotProd( &tmp, vec2 ) );
}

Rdouble GeoVecLen ( GeoPoint *vec )
{
    return sqrt ( GeoDotProd ( vec, vec ) );
}

int GeoPolyNormal ( int n_verts, GeoPoint *verts, GeoPoint *n )
{
    int i;
    Rfloat n_size;
    GeoPoint v0, v1, p;

    GeoZeroVec ( *n );
    Geo_Vet ( verts[0], verts[1], v0 );
    for ( i = 2; i < n_verts; i++ )
    {
        Geo_Vet ( verts[0], verts[i], v1 );
        GeoCrossProd ( &v0, &v1, &p );
        n->x += p.x; n->y += p.y; n->z += p.z;
        v0 = v1;
    }
}
```

```
n_size = GeoVecLen ( n );
if ( n_size > 0.0 )
{
    GeoMultVec ( 1/n_size, *n, *n );
    return 1;
}
else
    return 0;
}

/*===== geo_solid_angle =====*/
/*
    Calculates the solid angle given by the spherical projection of
    a 3D plane polygon
*/

Rdouble geo_solid_angle (
    int      n_vert,    /* number of vertices */
    GeoPoint *verts,    /* vertex coordinates list */
    GeoPoint *p )       /* point to be tested */
{
    int      i;
    Rdouble  area = 0.0, ang, s, l1, l2;
    GeoPoint p1, p2, r1, a, b, n1, n2;
    GeoPoint plane;

    if ( n_vert < 3 ) return 0.0;

    GeoPolyNormal ( n_vert, verts, &plane );

    /*
        WARNING: at this point, a practical implementation should check
        whether p is too close to the polygon plane. If it is, then
        there are two possibilities:
        a) if the projection of p onto the plane is outside the
           polygon, then area zero should be returned;
        b) otherwise, p is on the polyhedron boundary.
    */

    p2 = verts[n_vert-1]; /* last vertex */
    p1 = verts[0];        /* first vertex */
    Geo_Vet ( p1, p2, a ); /* a = p2 - p1 */

    for ( i = 0; i < n_vert; i++ )
    {
        Geo_Vet(*p, p1, r1);
        p2 = verts[(i+1)%n_vert];
        Geo_Vet ( p1, p2, b );
        GeoCrossProd ( &a, &r1, &n1 );
        GeoCrossProd ( &r1, &b, &n2 );

        l1 = GeoVecLen ( &n1 );
        l2 = GeoVecLen ( &n2 );
        s = GeoDotProd ( &n1, &n2 ) / ( l1 * l2 );
        ang = acos ( max(-1.0,min(1.0,s)) );
        s = GeoTripleProd( &b, &a, &plane );
        area += s > 0.0 ? PI - ang : PI + ang;

        GeoMultVec ( -1.0, b, a );
        p1 = p2;
    }
}
```

```
    }

    area -= PI*(n_vert-2);

    return ( GeoDotProd ( &plane, &r1 ) > 0.0 ) ? -area : area;
}

/* ===== main ===== */

int main ( void )
{
    FILE      *f;
    char      s[32];
    int       nv, j;
    GeoPoint  verts[100], p;
    Rdouble   Area =0.0;

    fprintf ( stdout, "\nFile Name: " );
    gets ( s );
    if ( (f = fopen ( s, "r" )) == NULL )
    {
        fprintf ( stdout, "Can not open the Polyhedron file \n" );
        exit ( 1 );
    }
    fprintf ( stdout, "\nPoint to be tested: " );
    fscanf( stdin, "%f %f %f", &p.x, &p.y, &p.z );

    while ( fscanf ( f, "%d", &nv ) == 1 )
    {
        for ( j = 0; j < nv; j++ )
            if ( fscanf ( f, "%f %f %f",
                &verts[j].x, &verts[j].y, &verts[j].z ) != 3 )
            {
                fprintf ( stdout, "Invalid Polyhedron file \n" );
                exit ( 2 );
            }

        Area += geo_solid_angle ( nv, verts, &p );
    }

    fprintf ( stdout, "\n Area = %12.4lf spherical radians.\n", Area);
    fprintf ( stdout, "\n The point is %s",
        ( (Area > 2*PI) || (Area < -2*PI) )? "inside" : "outside" );
    fprintf ( stdout, "the given polyhedron \n" );
    return 1;
}
```










```
/*
 * Revisions to Spherical Polygon Area (GG vol.IV entry II.4 [page 136])
 * as described in GG vol.V entry II.2 (page 45, bottom; 46, top).
 */

/* OLD */
/*      if (Lam2 < Lam1) Excess = - Excess; */

/* NEW */
double Lam;
Lam = (Lam2 - Lam1 > 0) ? Lam2 - Lam1 : Lam2 - Lam1 + 4*HalfPi;
if (Lam > 2*HalfPi) Excess= -Excess;
```

# Index of

## /pubs/tog/GraphicsGems/gemsii/radiosity/

Name	Last modified	Size	Description
 <a href="#">_ Parent Directory</a>			
 <a href="#">_ Makefile</a>	29-Jun-00 08:14	1K	
 <a href="#">_ README</a>	29-Jun-00 08:15	1K	
 <a href="#">_ draw.c</a>	29-Jun-00 08:14	1K	
 <a href="#">_ rad.c</a>	29-Jun-00 08:14	12K	
 <a href="#">_ rad.h</a>	29-Jun-00 08:15	3K	
 <a href="#">_ room.c</a>	29-Jun-00 08:15	7K	

```
# -Aa is HPUX's way of invoking ANSI C
CFLAGS = -g -Aa

all:    draw.o rad.o room.o

draw.o: draw.o rad.h
        cc $(CFLAGS) -c draw.c -o draw.o

rad.o:  rad.o rad.h
        cc $(CFLAGS) -c rad.c -o rad.o

room.o: room.o rad.h
        cc $(CFLAGS) -c room.c -o room.o

clean:
        /bin/rm -f draw.o rad.o room.o
```

The source is Copyright (c) 1990-1991 by Apple Computer, Inc. You may not use it in anything for which you charge money, beyond reasonable fees to cover duplication and handling. If you give your software away, feel free to use this source.

The source is the implementation of a gem, "Implementing Progressive Radiosity with User-Provided Polygon Display Routines," that I wrote for Graphics Gems II. A hard copy of the source can be found in Appendix II of Graphics Gems II. The source included here is a newer version than the one in the book and should replace it. For more details on the source and radiosity methods, please refer to the book.

No warranties expressed, implied, or even hinted at.

The source is written in ANSI C and is C++ friendly (you can include the .h files in C++ programs).

Please send bug reports and/or enhancements to the author "chense@apple.com". I would also like to hear any system that you have ported the source to. However, please don't ask me for technical support on the source.

```

/*****
*
*   draw.c
*
*   This is a skeleton polygon drawing program
*
*   Copyright (C) 1990-1991 Apple Computer, Inc.
*   All rights reserved.
*
*   8/27/1991 S. Eric Chen
*****/
#include "rad.h"

void BeginDraw(TView *view, unsigned long color)
{
    /* first time this view is drawn */
    if (view->wid==0)
    {
        /* open a new window if you want to display it on the screen */
        /* assign the window id or pointer to view->wid */
        /* the window id cannot be zero */
        /* do all the necessary initialization here too */

    }

    /* make view->wid the current window */

    /* set up view transformation from the parameters in view */

    /* clear the frame buffer with color */
}

void DrawPolygon(int nPts, TPoint3f *pts, TVector3f* n, unsigned long color)
{
    /* draw a polygon with the given color */
    /* buffer the polygon if you are drawing in a display list mode */
}

void EndDraw()
{
    /* finish the drawing of all polygons */
    /* display the contents of view->buffer to the current window if necessary */
}

```

```
/* *****
*   rad.c
*
*   This program contains three functions that should be called in sequence to
*   perform radiosity rendering:
*   InitRad(): Initialize radiosity.
*   DoRad(): Perform the main radiosity iteration loop.
*   CleanUpRad(): Clean up.
*
*   The following routines are assumed to be provided by the user:
*   BeginDraw()
*   DrawPolygon()
*   EndDraw()
*   Refer to rad.h for details
*
*   Copyright (C) 1990-1991 Apple Computer, Inc.
*   All rights reserved.
*
*   12/1990 S. Eric Chen
* ***** */

#include "rad.h"
#include <math.h>
#include <stdlib.h>

#define kMaxPolyPoints 255
#define PI 3.1415926
#define AddVector(c,a,b) (c).x=(a).x+(b).x, (c).y=(a).y+(b).y, (c).z=(a).z+(b).z
#define SubVector(c,a,b) (c).x=(a).x-(b).x, (c).y=(a).y-(b).y, (c).z=(a).z-(b).z
#define CrossVector(c,a,b) (c).x = (a).y*(b).z - (a).z*(b).y, \
                            (c).y = (a).z*(b).x - (a).x*(b).z, \
                            (c).z = (a).x*(b).y - (a).y*(b).x
#define DotVector(a,b) (a).x*(b).x + (a).y*(b).y + (a).z*(b).z
#define ScaleVector(c,s) (c).x*=(s), (c).y*=(s), (c).z*=(s)
#define NormalizeVector(n,a) ((n)=sqrt(DotVector(a,a)), \
                              (n)?((a).x/=n, (a).y/=n, (a).z/=n):0)

typedef struct {
    TView view; /* we only need to store one face of the hemi-cube */
    double* topFactors; /* delta form-factors(weight for each pixel) of the top
face */
    double* sideFactors; /* delta form-factors of the side faces */
} THemicube;

static TRadParams *params; /* input parameters */
static THemicube hemicube; /* one hemi-cube */
static double *formfactors; /* a form-factor array which has the same length as the
number of elements */
static double totalEnergy; /* total emitted energy; used for convergence checking */

static const TSpectra black = { 0, 0, 0 }; /* for initialization */
static int FindShootPatch(unsigned long *shootPatch);
static void SumFactors(double* formfs, int xRes, int yRes,
    unsigned long* buf, double* deltaFactors);
static void MakeTopFactors(int hres, double* deltaFactors);
static void MakeSideFactors(int hres, double* deltaFactors);
static void ComputeFormfactors(unsigned long shootPatch);
static void DistributeRad(unsigned long shootPatch);
static void DisplayResults(TView* view);
static void DrawElement(TElement* ep, unsigned long color);
static TColor32b SpectraToRGB(TSpectra* spectra);
```

```
/* Initialize radiosity based on the input parameters p */
void InitRad(TRadParams *p)
{
    int n;
    int hRes;
    unsigned long i;
    int j;
    TPatch* pp;
    TElement* ep;

    params = p;

    /* initialize hemi-cube */
    hemicube.view.fovx = 90;
    hemicube.view.fovy = 90;
    /* make sure hemicube resolution is an even number */
    hRes = ((int)(params->hemicubeRes/2.0+0.5))*2;
    hemicube.view.xRes = hemicube.view.yRes = hRes;
    n = hRes*hRes;
    hemicube.view.buffer = calloc(n, sizeof(unsigned long));
    hemicube.view.wid=0;
    hemicube.view.near = params->worldSize*0.001;
    hemicube.view.far = params->worldSize;

    /* take advantage of the symmetry in the delta form-factors */
    hemicube.topFactors= calloc(n/4, sizeof(double));
    hemicube.sideFactors= calloc(n/4, sizeof(double));
    MakeTopFactors(hRes/2, hemicube.topFactors);
    MakeSideFactors(hRes/2, hemicube.sideFactors);

    formfactors = calloc(params->nElements, sizeof(double));

    /* initialize radiosity */
    pp = params->patches;
    for (i=params->nPatches; i--; pp++)
        pp->unshotRad = *(pp->emission);
    ep = params->elements;
    for (i=params->nElements; i--; ep++)
        ep->rad = *(ep->patch->emission);

    /* compute total energy */
    totalEnergy = 0;
    pp = params->patches;
    for (i=params->nPatches; i--; pp++)
        for (j=0; j<kNumberOfRadSamples; j++)
            totalEnergy += pp->emission->samples[j] * pp->area;

    DisplayResults(&params->displayView);
}

/* Main iterative loop */
void DoRad()
{
    unsigned long shootPatch;

    while (FindShootPatch(&shootPatch))
    {
        ComputeFormfactors(shootPatch);
    }
}
```

```
        DistributeRad(shootPatch);
        DisplayResults(&params->displayView);
    }
}

/* Clean up */
void CleanUpRad()
{
    free(hemicube.topFactors);
    free(hemicube.sideFactors);
    free(hemicube.view.buffer);
    free(formfactors);
}

/* Find the next shooting patch based on the unshot energy of each patch */
/* Return 0 if convergence is reached; otherwise, return 1 */
static int FindShootPatch(unsigned long *shootPatch)
{
    int i, j;
    double energySum, error, maxEnergySum=0;
    TPatch* ep;

    ep = params->patches;
    for (i=0; i< params->nPatches; i++, ep++)
    {
        energySum = 0;
        for (j=0; j<kNumberOfRadSamples; j++)
            energySum += ep->unshotRad.samples[j] * ep->area;

        if (energySum > maxEnergySum)
        {
            *shootPatch = i;
            maxEnergySum = energySum;
        }
    }

    error = maxEnergySum / totalEnergy;
    /* check convergence */
    if (error < params->threshold)
        return (0);          /* converged */
    else
        return (1);
}

/* Find out the index to the delta form-factors array */
#define Index(i)          ((i)<hres? i: (hres-1- ((i)%hres)))

/* Use the largest 32bit unsigned long for background */
#define kBackgroundItem 0xffffffff

/* Convert a hemi-cube face to form-factors */
static void SumFactors(
double* formfs, /* output */
int xRes, int yRes, /* resolution of the hemi-cube face */
unsigned long* buf, /* we only need the storage of the top hemi-cube face */
double* deltaFactors /* delta form-factors for each hemi-cube pixel */
)
```



```
{
    int i, j;
    int ii, jj;
    unsigned long *ip=buf;
    int hres = xRes/2;
    for (i=0; i<yRes; i++)
    {
        ii= Index(i)*hres;
        for (j=0; j<xRes; j++, ip++)
            if ((*ip) != kBackgroundItem)
            {
                jj = Index(j);
                formfs[*ip] += deltaFactors[ii+jj];
            }
    }
}

/* Create the delta form-factors for the top face of hemi-cube */
/* Only need to compute 1/4 of the form-factors because of the 4-way symmetry */
static void MakeTopFactors(
int hres, /* half resolution of the face */
double* deltaFactors /* output */
)
{
    int j,k;
    double xSq , ySq, xylSq;
    double n= hres;
    double* wp;
    double dj, dk;

    wp = deltaFactors;
    for (j=0; j<hres; j++)
    {
        dj = (double)j;
        ySq = (n - (dj+0.5)) / n;
        ySq *= ySq;
        for ( k=0 ; k<hres ; k++ )
        {
            dk = (double)k;
            xSq = ( n - (dk + 0.5) ) / n;
            xSq *= xSq;
            xylSq = xSq + ySq + 1.0 ;
            xylSq *= xylSq;
            *wp++ = 1.0 / (xylSq * PI * n * n);
        }
    }
}

/* Create the delta form-factors for the side face of hemi-cube */
/* Only need to compute 1/4 of the form-factors because of the 4-way symmetry */
static void MakeSideFactors(
int hres, /* half resolution of the face */
double* deltaFactors /* output */
)
{
    int j,k;
    double x, xSq , y, ySq, xyl, xylSq;
    double n= hres;
    double* wp;
    double dj, dk;
```

```
wp = deltaFactors;
for (j=0; j<hres; j++)
{
    dj = (double)j;
    y = (n - (dj+0.5)) / n;
ySq = y*y;
for ( k=0 ; k<hres ; k++ )
{
    dk = (double)k;
    x = ( n - (dk + 0.5) ) / n;
    xSq = x*x;
    xyl = xSq + ySq + 1.0 ;
    xylSq = xyl*xyl;
    *wp++ = y / (xylSq * PI * n * n);
}
}
```

```
/* Use drand48 instead if it is supported */
```

```
#define RandomFloat ((float)(rand())/((float)RAND_MAX))
```

```
/* Compute form-factors from the shooting patch to every elements */
```

```
static void ComputeFormfactors(unsigned long shootPatch)
```

```
{
    unsigned long i;
    TVector3f      up[5];
    TPoint3f       lookout[5];
    TPoint3f       center;
    TVector3f      normal, tangentU, tangentV, vec;
    int face;
    double         norm;
    TPatch*        sp;
    double*        fp;
    TElement*       ep;

    /* get the center of shootPatch */
    sp = &(params->patches[shootPatch]);
    center = sp->center;
    normal = sp->normal;

    /* rotate the hemi-cube along the normal axis of the patch randomly */
    /* this will reduce the hemi-cube aliasing artifacts */
    do {
        vec.x = RandomFloat;
        vec.y = RandomFloat;
        vec.z = RandomFloat;
        /* get a tangent vector */
        CrossVector(tangentU, normal, vec);
        NormalizeVector(norm, tangentU);
    } while (norm==0); /* bad choice of the random vector */

    /* compute tangentV */
    CrossVector(tangentV, normal, tangentU);

    /* assign the lookats and ups for each hemicube face */
    AddVector(lookat[0], center, normal);
    up[0] = tangentU;
    AddVector(lookat[1], center, tangentU);
    up[1] = normal;
    AddVector(lookat[2], center, tangentV);
    up[2] = normal;
```

```
SubVector(lookat[3], center, tangentU);
up[3] = normal;
SubVector(lookat[4], center, tangentV);
up[4] = normal;

/* position the hemicube slightly above the center of the shooting patch */
ScaleVector(normal, params->worldSize*0.0001);
AddVector(hemicube.view.camera, center, normal);

/* clear the formfactors */
fp = formfactors;
for (i=params->nElements; i--; fp++)
    *fp = 0.0;

for (face=0; face < 5; face++)
{
    hemicube.view.lookat = lookat[face];
    hemicube.view.up = up[face];

    /* draw elements */
    BeginDraw(&(hemicube.view), kBackgroundItem);
    for (i=0; i< params->nElements; i++)
        DrawElement(&params->elements[i], i);
    /* color element i with its index */
    EndDraw();

    /* get formfactors */
    if (face==0)
        SumFactors(formfactors, hemicube.view.xRes, hemicube.view.yRes,
                    hemicube.view.buffer, hemicube.topFactors);
    else
        SumFactors(formfactors, hemicube.view.xRes, hemicube.view.yRes/2,
                    hemicube.view.buffer, hemicube.sideFactors);
}

/* compute reciprocal form-factors */
ep = params->elements;
fp = formfactors;
for (i=params->nElements; i--; ep++, fp++)
{
    *fp *= sp->area / ep->area;

    /* This is a potential source of hemi-cube aliasing */
    /* To do this right, we need to subdivide the shooting patch
    and reshoot. For now we just clip it to unity */
    if ((*fp) > 1.0)        *fp = 1.0;
}
}
```

```
/* Distribute radiosity form shootPatch to every element */
/* Reset the shooter's unshot radiosity to 0 */
static void DistributeRad(unsigned long shootPatch)
{
    unsigned long i;
    int j;
    TPatch* sp;
    TElement* ep;
    double* fp;
    TSpectra deltaRad;
    double w;
```

```
    sp = &(params->patches[shootPatch]);

    /* distribute unshotRad to every element */
    ep = params->elements;
    fp = formfactors;
    for (i=params->nElements; i--; ep++, fp++)
    {
        if ((*fp) != 0.0)
        {
            for (j=0; j<kNumberOfRadSamples; j++)
                deltaRad.samples[j] = sp->unshotRad.samples[j] * (*fp)
*
ep->patch->reflectance->samples[j];

            /* incremental element's radiosity and patch's unshot radiosity
*/
            w = ep->area/ep->patch->area;
            for (j=0; j<kNumberOfRadSamples; j++)
            {
                ep->rad.samples[j] += deltaRad.samples[j];
                ep->patch->unshotRad.samples[j] += deltaRad.samples[j] *
w;
            }
        }
    }

    /* reset shooting patch's unshot radiosity */
    sp->unshotRad = black;
}

/* Convert a TSpectra (radiosity) to a TColor32b (rgb color) */
/* Assume the first three samples of the spectra are the r, g, b colors */
/* More elaborated color space transformation could be performed here */
static TColor32b
SpectraToRGB(TSpectra* spectra)
{
    TColor32b      c;
    TSpectra       r;
    double  max=1.0;
    int k;

    for (k=kNumberOfRadSamples; k--;) {
        if (spectra->samples[k] > max)
            max = spectra->samples[k];
    }
    /* Clip the intensity*/
    r = *spectra;
    if (max>1.0) {
        for (k=kNumberOfRadSamples; k--; )
            r.samples[k] /= max;
    }

    /* Convert to a 32-bit color; Assume the first 3 samples in TSpectra
are the r, g, b colors we want. Otherwise, do color conversion here */
    c.a= 0;
    c.r= (unsigned char) (r.samples[0] * 255.0 + 0.5);
    c.g= (unsigned char) (r.samples[1] * 255.0 + 0.5);
    c.b= (unsigned char) (r.samples[2] * 255.0 + 0.5);
}
```

```
    return c;
}

static void
GetAmbient(TSpectra* ambient)
{
    TPatch* p;
    unsigned long i;
    int k;
    static int first = 1;
    static TSpectra baseSum;
    TSpectra uSum;

    uSum=black;
    if (first) {
        double areaSum;
        TSpectra rSum;
        areaSum=0;
        rSum=black;
        /* sum area and (area*reflectivity) */
        p= params->patches;
        for (i=params->nPatches; i--; p++) {
            areaSum += p->area;
            for (k=kNumberOfRadSamples; k--; )
                rSum.samples[k] += p->reflectance->samples[k]* p->area;
        }
        for (k=kNumberOfRadSamples; k--; )
            baseSum.samples[k] = areaSum - rSum.samples[k];
        first = 0;
    }

    /* sum (unshot radiosity * area) */
    p= params->patches;
    for (i=params->nPatches; i--; p++) {
        for (k=kNumberOfRadSamples; k--; )
            uSum.samples[k] += p->unshotRad.samples[k] * p->area;
    }

    /* compute ambient */
    for (k=kNumberOfRadSamples; k--; )
        ambient->samples[k] = uSum.samples[k] / baseSum.samples[k];
}

static void
DisplayResults(TView* view)
{
    unsigned long i;
    register TElement* ep;
    TSpectra ambient;
    GetAmbient(&ambient);

    BeginDraw(view, 0);
    ep = params->elements;
    for (i=0; i< params->nElements; i++, ep++) {
        TColor32b      c;
        TSpectra      s;
        int k;
        /* add ambient approximation */
        if (params->addAmbient) {
            for (k=kNumberOfRadSamples; k--; )
```

```

        s.samples[k] = (ep->rad.samples[k] + (ambient.samples[k]*
ep->patch->reflectance->samples[k]))*params->intensityScale;
    } else {
        for (k=kNumberOfRadSamples; k--; )
            s.samples[k] = ep->rad.samples[k]*params->intensityScale;
    }
    /* quantize color */
    c = SpectraToRGB(&s);
    DrawElement(ep, *(unsigned long*)&c);
}

EndDraw();

}

static void
DrawElement(TElement* ep, unsigned long color)
{
    static TPoint3f pts[kMaxPolyPoints];
    int nPts = ep->nVerts;
    int j;
    for (j=0; j<nPts; j++)
        pts[j] = params->points[ep->verts[j]];

    DrawPolygon(nPts, pts, &ep->normal, color);
}
}
```

```

/*****
*
*      rad.h
*
*      This is the headerfile which defines the data structures used in rad.c
*
*      Copyright (C) 1990-1991 Apple Computer, Inc.
*      All rights reserved.
*
*      12/1990 S. Eric Chen
*****/

#ifndef __RAD__
#define __RAD__

#define kNumberOfRadSamples      3

typedef struct { float x, y, z; } TPoint3f;
typedef TPoint3f TVector3f;
typedef struct { unsigned char a, r, g, b; } TColor32b;

typedef struct {
    double samples[kNumberOfRadSamples];
} TSpectra;

typedef struct {
    TSpectra* reflectance; /* diffuse reflectance of the patch */
    TSpectra* emission;    /* emission of the patch */
    TPoint3f center;       /* center of the patch where hemi-cubes will be placed */
    TVector3f normal;      /* normal of the patch; for orienting the hemi-cube */
    TSpectra unshotRad;    /* unshot radiosity of the patch */
    double area;          /* area of the patch */
} TPatch;

typedef struct {
    unsigned short nVerts; /* number of vertices of the element */
    unsigned long* verts;  /* vertices */
    TVector3f normal;      /* normal of the element; for backface removal */
    TSpectra rad;          /* total radiosity of the element */
    double area;          /* area of the patch */
    TPatch* patch;        /* pointer to the parent patch */
} TElement;

typedef struct {
    TPoint3f camera;       /* camera location */
    TPoint3f lookat;       /* point of interest */
    TVector3f up;          /* view up vector */
    float fovx, fovy;      /* field of view in x, y (in degree) */
    float near, far;       /* distance from the camera to the near and far planes */
    unsigned short xRes, yRes; /* resolution of the frame buffer */
    unsigned long* buffer; /* pointer to the frame buffer */
    long wid;             /* id or pointer to the window associated with the view */
} TView;

/* Radiosity input parameters */
typedef struct {
    double threshold; /* convergence threshold (fraction of the total emitted
energy) */
    unsigned long nPatches; /* number of patches */
    TPatch* patches; /* patches */
    unsigned long nElements; /* number of elements */
    TElement* elements; /* elements */
}

```

```
    unsigned long nPoints; /* number of element vertices */
    TPoint3f *points;      /* element vertices */
    TView displayView;     /* view to display the results */
    unsigned short hemiCubeRes; /* hemi-cube resolution */
    float worldSize;       /* approximate diameter of the bounding sphere of the
world.                    used for placing near and far planes in the hemi-cube
computation*/
    float intensityScale; /* used to scale intensity for display */
    int addAmbient;       /* whether or not to add the ambient
approximation in display */
} TRadParams;

/* make it C++ friendly */
#ifdef __cplusplus
extern "C" {
#endif /* __cplusplus */

/* initialization */
void InitRad(TRadParams *p);
/* main iterative loop */
void DoRad();
/* final clean up */
void CleanUpRad();

/* The following routines should be provided by the user */

/* Clear buffer. Set up view transformation.*/
/* Open a window if necessary (check if view->wid is zero) */
void BeginDraw(
TView *view, /* the viewing parameters and frame buffer to draw to*/
unsigned long color /* color used to clear the buffer */
);

/* Draw a 3-d polygon with a constant color */
void DrawPolygon(
int nPts, /* number of points in the polygon */
TPoint3f *pts, /* points of the polygon */
TVector3f* normal, /* normal of the polygon */
unsigned long color /* color to be drawn with */
);

/* Finish the drawing of polygons to the frame buffer*/
/* Display the buffer content in the window if necessary */
void EndDraw();

#ifdef __cplusplus
};
#endif /* __cplusplus */

#endif /* __RAD__ */
```



```

/*****
*
*      room.c
*
*      This is a test program which constrcuts the Cornell radiosity room with
*      a ceiling light and two boxes inside. The side faces of the boxes are not
*      directly illuminated by the light. Therefore, they are a good example of
*      the color bleeding effects.
*      This program calls IniRad(), DoRad() and CleanUpRad() in rad.c to perform
*      the radiosity rendering.
*
*      Copyright (C) 1990-1991 Apple Computer, Inc.
*      All rights reserved.
*
*      12/1990 S. Eric Chen
*****/

#include "rad.h"

/* a quadrilateral */
typedef struct {
    short verts[4]; /* vertices of the quadrilateral */
    short patchLevel; /* patch subdivision level (how fine to subdivide the
quadrilateral?) */
    short elementLevel; /* element subdivision level (how fine to subdivide a patch?)
*/
    float area; /* area of the quadrilateral */
    TVector3f normal; /* normal of the quadrilateral */
    TSpectra* reflectance; /* diffuse reflectance of the quadrilateral */
    TSpectra* emission; /* emission of the quadrilateral */
} TQuad;

/* input parameters */
TRadParams      params = {
    0.001,          /* convergence threshold */
    0, 0, 0, 0, 0, 0, /* patches, elements and points; initialize these in
InitParams */
    {{ 108, 120, 400 }, /* camera location */
     { 108, 100, 100 }, /* look at point */
     { 0, 1, 0 }, /* up vector */
    60, 60, /* field of view in x, y*/
    1, 550, /* near, far */
    200, 200, /* resolution x, y */
    0 }, /* buffer */
    100, /* hemi-cube resolution */
    250, /* approximate diameter of the room */
    50, /* intensity scale */
    1 /* add the ambient term */
};

TPoint3f roomPoints[] = {
    0, 0, 0,
    216, 0, 0,
    216, 0, 215,
    0, 0, 215,
    0, 221, 0,
    216, 221, 0,
    216, 221, 215,
    0, 221, 215,

    85.5, 220, 90,
    130.5, 220, 90,

```

```
130.5, 220, 130,
85.5, 220, 130,

53.104, 0, 64.104,
109.36, 0, 96.604,
76.896, 0, 152.896,
20.604, 0, 120.396,
53.104, 65, 64.104,
109.36, 65, 96.604,
76.896, 65, 152.896,
20.604, 65, 120.396,

134.104, 0, 67.104,
190.396, 0, 99.604,
157.896, 0, 155.896,
101.604, 0, 123.396,
134.104, 130, 67.104,
190.396, 130, 99.604,
157.896, 130, 155.896,
101.604, 130, 123.396
```

```
};
```

```
static TSpectra red = { 0.80, 0.10, 0.075 };
static TSpectra yellow = { 0.9, 0.8, 0.1 };
static TSpectra blue = { 0.075, 0.10, 0.35 };
static TSpectra white = { 1.0, 1.0, 1.0 };
static TSpectra lightGrey = { 0.9, 0.9, 0.9 };
static TSpectra black = { 0.0, 0.0, 0.0 };
```

```
/* Assume a right-handed coordinate system */
```

```
/* Polygon vertices follow counter-clockwise order when viewing from front */
```

```
#define numberOfPolys 18
```

```
TQuad roomPolys[numberOfPolys] = {
    {{4, 5, 6, 7}, 2, 8, 216*215, {0, -1, 0}, &lightGrey, &black}, /* ceiling */
    {{0, 3, 2, 1}, 3, 8, 216*215, {0, 1, 0}, &lightGrey, &black}, /* floor */
    {{0, 4, 7, 3}, 2, 8, 221*215, {1, 0, 0}, &red, &black}, /* wall */
    {{0, 1, 5, 4}, 2, 8, 221*216, {0, 0, 1}, &lightGrey, &black}, /* wall */
    {{2, 6, 5, 1}, 2, 8, 221*215, {-1, 0, 0}, &blue, &black}, /* wall */
    {{8, 9, 10, 11}, 2, 1, 40*45, {0, -1, 0}, &black, &white}, /* light */
    {{16, 19, 18, 17}, 1, 5, 65*65, {0, 1, 0}, &yellow, &black}, /* box 1 */
    {{12, 13, 14, 15}, 1, 1, 65*65, {0, -1, 0}, &yellow, &black},
    {{12, 15, 19, 16}, 1, 5, 65*65, {-0.866, 0, -0.5}, &yellow, &black},
    {{12, 16, 17, 13}, 1, 5, 65*65, {0.5, 0, -0.866}, &yellow, &black},
    {{14, 13, 17, 18}, 1, 5, 65*65, {0.866, 0, 0.5}, &yellow, &black},
    {{14, 18, 19, 15}, 1, 5, 65*65, {-0.5, 0, 0.866}, &yellow, &black},
    {{24, 27, 26, 25}, 1, 5, 65*65, {0, 1, 0}, &lightGrey, &black}, /* box 2 */
    {{20, 21, 22, 23}, 1, 1, 65*65, {0, -1, 0}, &lightGrey, &black},
    {{20, 23, 27, 24}, 1, 6, 65*130, {-0.866, 0, -0.5}, &lightGrey, &black},
    {{20, 24, 25, 21}, 1, 6, 65*130, {0.5, 0, -0.866}, &lightGrey, &black},
    {{22, 21, 25, 26}, 1, 6, 65*130, {0.866, 0, 0.5}, &lightGrey, &black},
    {{22, 26, 27, 23}, 1, 6, 65*130, {-0.5, 0, 0.866}, &lightGrey, &black},
};
```

```
/* Compute the xyz coordinates of a point on a quadrilateral given its u, v coordinates
using bi-linear mapping */
```

```
void UVToXYZ(const TPoint3f quad[4], float u, float v, TPoint3f* xyz)
```

```
{
    xyz->x = quad[0].x * (1-u)*(1-v) + quad[1].x * (1-u)*v + quad[2].x * u*v +
quad[3].x * u*(1-v);
    xyz->y = quad[0].y * (1-u)*(1-v) + quad[1].y * (1-u)*v + quad[2].y * u*v +
```

```
quad[3].y * u*(1-v);
    xyz->z = quad[0].z * (1-u)*(1-v) + quad[1].z * (1-u)*v + quad[2].z * u*v +
quad[3].z * u*(1-v);
}

#define Index(i, j) ((i)*(nv+1)+(j))

int iOffset;    /* index offset to the point array */
TPatch* pPatch;
TElement* pElement;
TPoint3f* pPoint;

/* Mesh a quadrilateral into patches and elements */
/* Output goes to pPatch, pElement, pPoint */
void MeshQuad(TQuad* quad)
{
    TPoint3f pts[4];
    int nu, nv;
    double du, dv;
    int i, j;
    double u, v;
    int nPts=0;
    float fi, fj;
    int pi, pj;

    /* Calculate element vertices */
    for (i=0; i<4; i++)
        pts[i] = roomPoints[quad->verts[i]];
    nu = nv = quad->patchLevel * quad->elementLevel+1;
    du = 1.0 / (nu-1); dv = 1.0 / (nv-1);
    for (i = 0, u = 0; i < nu; i++, u += du)
        for (j = 0, v = 0; j < nv; j++, v += dv, nPts++)
            UVToXYZ(pts, u, v, pPoint++);

    /* Calculate elements */
    nu = nv = quad->patchLevel*quad->elementLevel;
    du = 1.0 / nu; dv = 1.0 / nv;
    for (i = 0, u = du/2.0; i < nu; i++, u += du)
        for (j = 0, v = dv/2.0; j < nv; j++, v += dv, pElement++) {
            pElement->normal = quad->normal;
            pElement->nVerts = 4;
            pElement->verts = (unsigned long*)calloc(4, sizeof(unsigned
long));

            pElement->verts[0] = Index(i, j)+iOffset;
            pElement->verts[1] = Index(i+1, j)+iOffset;
            pElement->verts[2] = Index(i+1, j+1)+iOffset;
            pElement->verts[3] = Index(i, j+1)+iOffset;
            pElement->area = quad->area / (nu*nv);
            /* find out the parent patch */
            fi = (float)i/(float)nu;
            fj = (float)j/(float)nv;
            pi = (int)(fi*(float)(quad->patchLevel));
            pj = (int)(fj*(float)(quad->patchLevel));
            pElement->patch = pPatch+pi*quad->patchLevel+pj;
        }

    /* Calculate patches */
    nu = quad->patchLevel; nv=quad->patchLevel;
    du = 1.0 / nu; dv = 1.0 / nv;
    for (i = 0, u = du/2.0; i < nu; i++, u += du)
        for (j = 0, v = dv/2.0; j < nv; j++, v += dv, pPatch++) {
```

```
        UVToXYZ(pts, u, v, &pPatch->center);
        pPatch->normal = quad->normal;
        pPatch->reflectance = quad->reflectance;
        pPatch->emission = quad->emission;
        pPatch->area = quad->area / (nu*nv);
    }

    iOffset += nPts;
}

/* Initialize input parameters */
void InitParams()
{
    int i;

    /* compute the total number of patches */
    params.nPatches=0;
    for (i=numberOfPolys; i--; )
        params.nPatches += roomPolys[i].patchLevel*roomPolys[i].patchLevel;
    params.patches = (TPatch*)calloc(params.nPatches, sizeof(TPatch));

    /* compute the total number of elements */
    params.nElements=0;
    for (i=numberOfPolys; i--; )
        params.nElements += roomPolys[i].elementLevel*roomPolys[i].patchLevel*
roomPolys[i].elementLevel*roomPolys[i].patchLevel;
    params.elements = (TElement*)calloc(params.nElements, sizeof(TElement));

    /* compute the total number of element vertices */
    params.nPoints=0;
    for (i=numberOfPolys; i--; )
        params.nPoints += (roomPolys[i].elementLevel*roomPolys[i].patchLevel+1)*
(roomPolys[i].elementLevel*roomPolys[i].patchLevel+1);
    params.points = (TPoint3f*)calloc(params.nPoints, sizeof(TPoint3f));

    /* mesh the room to patches and elements */
    iOffset = 0;
    pPatch= params.patches;
    pElement= params.elements;
    pPoint= params.points;
    for (i=0; i<numberOfPolys; i++)
        MeshQuad(&roomPolys[i]);

    params.displayView.buffer= (unsigned long*)calloc(
        params.displayView.xRes*params.displayView.yRes, sizeof(unsigned long));
    params.displayView.wid=0;
}

void main()
{
    InitParams();
    InitRad(&params);
    DoRad();
    CleanUpRad();
}
```

/\*\*\*\*\*

FAST LINEAR COLOR RENDERER

Russell C.H. Cheng, University of Wales, Cardiff 21 Jan 1992

The renderer assumes true 24-bit color with a linear memory frame-buffer of 4-byte integers, lowest byte is r, then g, then b, with the top byte not used.

The renderer precalculates and stores the merged 24-bit rgb x-offsets. This avoids a separate interpolation for each of the r, g and b variables for every pixel.

Entry:

For clarity the required input variables are passed to the renderer as arguments, Details of each variable are given in the listing. In a general program it will be more compact to hold these variables in a structure and pass a pointer to this.

Exit:

The renderer outputs directly to the designated frame-buffer.

\*\*\*\*\*/

```
#define HRES 768 /* horizontal resolution, adjust this as necessary */
#define VRES 512 /* vertical resolution, adjust this as necessary */

void FastLinearRend( xmin,xmax,ymin,ymax, xleft,xright, r0,g0,b0,z0, x0,y0,
                    dr_by_dx, dr_by_dy, dg_by_dx, dg_by_dy, db_by_dx, db_by_dy,
                    dz_by_dx, dz_by_dy, screen_buffer_ptr, z_buffer_ptr)

int xmin,          /* The extent of the polygon being rendered:          */
    xmax,          /* xmin and xmax give the x-values of the leftmost and      */
    ymin,          /* rightmost pixels of the polygon, and ymin and ymax the    */
    ymax;          /* largest and smallest y-values encountered in the polygon */

int * xleft,       /* Pointers to arrays holding the x-value of the leftmost    */
    * xright;      /* and rightmost pixel occupied by the polygon on each       */
                    /* scanline. The values xleft[ymin],..., xleft[ymax], and    */
                    /* xright[ymin],..., xright[ymax] should all be set on entry */
                    /* The arrays themselves in the main program should be each */
                    /* of size VRES                                              */

float r0,g0,b0; /* Base rgb values of the polygon at the pixel position      */
                /* x0, y0; each in the range 0.0 - 255.0 */

float z0;        /* Base z-distance of the polygon at pixel position x0, y0   */

int x0,y0;       /* Position of base pixel, this need not be a pixel of the   */
                /* polygon, but is typically a vertex or the origin (0,0)    */

float  dr_by_dx, dr_by_dy, /* r, g, b color and */
    dg_by_dx, dg_by_dy, /* z-distance increments */
    db_by_dx, db_by_dy, /* in the x and y */
    dz_by_dx, dz_by_dy; /* directions */

int * screen_buffer_ptr; /* Pointer to the frame buffer base address */
                    /* The frame buffer is assumed to be linear */
                    /* memory, comprising VRES horizontal lines */
                    /* with HRES pixels per line */

float * z_buffer_ptr;    /* Pointer to the z-buffer base address */
                    /* The buffer-size is HRES*VRES */
```

```
{
    float r,g,b,z,          /* current r g b z values */
          r1,g1,b1,z1;      /* r g b z values at position x0, y */

    float dx,dy;            /* offsets of current pixel position */
                              /* from that of base pixel */

    int x,y;                /* current pixel position */

    int *  screen_buffer;    /* pointer to current frame-buffer position */
    float *  z_buffer;       /* pointer to current z-buffer position */

    int col;                /* This holds the 24-bit rgb color of the */
                              /* leftmost pixel of the current scanline */

    int rgboffset[HRES];    /* This array holds the merged */
                              /* 24-bit rgb x-direction offsets */

    /* find r1, g1, b1, z1, the rgbz value at (x0,ymin) */
    dy = (ymin - y0);
    r1 = r0 + dr_by_dy * dy;
    g1 = g0 + dg_by_dy * dy;
    b1 = b0 + db_by_dy * dy;
    z1 = z0 + dz_by_dy * dy;

    /* find the merged 24-bit rgb x-offset values along a scanline */
    for (x = xmin; x <= xmax; x++) {
        dx = (x - x0);
        r = dr_by_dx * dx;
        g = dg_by_dx * dx;
        b = db_by_dx * dx;
        rgboffset[x] = ((int)r) + (((int)g)<<8) + (((int)b)<<16);
    }

    /* now go through each scanline */
    for (y = ymin; y<= ymax; y++) {
        /* for each scanline, find the 24-bit color value at (x0,y) */
        col = ((int)r1) + (((int)g1)<<8) + (((int)b1)<<16);

        /* and find the z value at (xleft[y],y) */
        dx = xleft[y] - x0;
        z = z1 + dz_by_dx * dx;

        /* then render the scanline */
        screen_buffer = screen_buffer_ptr + y*HRES;
        z_buffer = z_buffer_ptr + y*HRES;
        for (x = xleft[y]; x<=xright[y]; x++) {
            if ( z < z_buffer[x] ) {
                screen_buffer[x] = col + rgboffset[x];
                z_buffer[x] = z;
            }
            z += dz_by_dx;
        }

        /* and increment the r1 g1 b1 z1 value ready for the next y */
        r1 += dr_by_dy;
        g1 += dg_by_dy;
        b1 += db_by_dy;
        z1 += dz_by_dy;
    }
}
```

}

```

/*****
EDGE AND BIT-MASK CALCULATIONS FOR ANTI-ALIASING
Russell C.H. Cheng, University of Wales, Cardiff, 21 Jan 1992.

```

This routine calculates the geometry of the overlap of the edges of a polygon with square pixels, using a fast alternating Bresenham error-update technique.

Entry:

Pointer to a structure (defined below) giving details of the polygon.  
Pointers to precalculated bitmasks.

Exit:

The routine outputs the overlap information pixel by pixel via resultproc().  
The arguments are: the pixel position, position of the endpoints of the overlapping edge, the pixeltype, and the edgetype.

In this implementation resultproc() uses this information to calculate the positions of the leftmost and rightmost pixels on each scanline, and the bitmasks of overlap of the polygon with each pixel.

Implementation Details:

- (i) The procedure resultproc() accesses external pointers: int \* xl, int \* xr, aabufftype \* aabuffptr, short int \* fragptr. These are described in resultproc(). These should be globally defined.
- (ii) edge\_calculate() accesses polygonal information via a pointer to the structure:

```

typedef struct SCRNPOLYGON
{
    int numvert;           number of vertices
    float xcoord[10];      coordinates of each vertex
    float ycoord[10];      in clockwise order
    int imax;              subscript of vertex with largest y-coord
    int imin;              subscript of vertex with smallest y-coord
} SCRNPOLYGON;

```

```

*****/

```

```

#define LARGE 10000000.0
#define TRUE 1
#define FALSE 0

```

```

/*== The following two structures are included here to enable compilation,
they should normally be defined earlier for access by the main routine ==*/

```

```

typedef struct {
    unsigned short mask[4][5][5][5][5];
} aabufftype;

```

```

typedef struct {
    int numvert;           /* number of vertices */
    float xcoord[10];      /* coordinates of each vertex */
    float ycoord[10];      /* in clockwise order */
    int imax;              /* subscript of vertex with largest y-coord */
    int imin;              /* subscript of vertex with smallest y-coord */
} SCRNPOLYGON;

```

```

#define NN 0 /* The different types of overlap possible in a pixel. */
#define LX 1 /* See Fig.2 in the main text. */
#define LY 2
#define RX 3
#define RY 4

```



```
#define MX 5
#define MY 6
#define SX 7
#define SY 8
#define TX 9
#define TY 10

#define HRES 768 /* horizontal resolution */

int rl[11][5] = { /* Array used to select pixeltype/edgetype combinations */
  0,  0, 0, 1, -1, /* at which an xl or xr value should be set. */
-1,  0, 1, 1, -1, /* 1 indicates where xr is set */
  0,  0, 0, 0, -1, /* -1 indicates where xl is set */
  1, -1, 0, 1, -1,
  0,  0, 0, 1,  0,
  0, -1, 1, 1, -1,
  0,  0, 0, 0,  0,
  0,  0, 0, 1,  0,
  0, -1, 0, 0,  0,
  0,  0, 0, 0, -1,
  0,  0, 1, 0,  0,
};

void edge_calculator(scrnfaceptr)
SCRNPOLYGON * scrnfaceptr;

{
void resultproc();

float delx,dely,delymp,dx_by_dy,dy_by_dx;
int edgetype;
float ex; /* Bresenham x-error variables */
float exend, exstt, ex0; /* '' */
float ey; /* Bresenham y-error variables */
float eyend, eystt, ey0; /* '' */
int firstleftedgetype; /* edgetype of first left edge encountered*/
int firstrightedge; /* flag showing when 1st right-edge is reached */
float hpix = 1.0; /* Pixel edgelength */
int iv, ivml, ivpl; /* work variables holding array subscripts */
float largemp; /* work variable used in slope calculation */
int lastleftedgetype; /* edgetype of the last left-edge */
int toggle; /* switch-flag, 1=odd, 2=even edgetypes */
float xstt,xend; /* x-positions of start and end points of edge */
int xpix, xpend, xpstt; /* current, edge-start & edge-end pixel x-coords */
float ystt,yend; /* y-positions of start and end points of edge */
int ypix, ypend, ypstt; /* current, edge-start & edge-end pixel y-coords */

/*==== Process each edge of the left-hand side of polygon =====*/
iv = scrnfaceptr->imin; /* Find subscripts of first */
ivpl = (iv+1) % scrnfaceptr->numvert; /* left edge and the */
xend=scrnfaceptr->xcoord[iv]; yend=scrnfaceptr->ycoord[iv]; /* coords */
xpend = (int)xend; ypend = (int)yend; /* of its starting point. */
firstleftedgetype = 0;

while(iv!=scrnfaceptr->imax) {
  xpix = xpend; ypix = ypend; /* Get the coords of the */
  xstt= xend; ystt= yend; /* starting pixel of edge */
  xpstt = xpend; ypstt = ypend; /* and the coords of the */
  xpend=(int)(xend= scrnfaceptr->xcoord[ivpl]); /* end */
  ypend=(int)(yend= scrnfaceptr->ycoord[ivpl]); /* pixel. */
}
```

```
    extend= xend-xpend; eyend= yend-ypend; /* Bresenham errors at end pixel.*/

    if(xstt<xend) { /*-- Left: Edgetype 1 --*/
        if(firstleftedgetype==0) { /* If at first edge, */
            firstleftedgetype = 1; /* make a note of its type. */
            resultproc(xpix,ypix, 0.0,0.0,0.0,0.0, LX,0);
            /* Special call equivalent to xl[ypix] = xpix */
        }
        edgetype=1; toggle = 1; /* Set edgetype and */
        goto gamma; /* go to main edge-processor.*/
    }
    else { /*-- Left: Edgetype 4 --*/
        if(firstleftedgetype==0) /* If at first edge, make */
            firstleftedgetype = 4; /* a note of its type. */
        edgetype=4; toggle = 2; /* Set edgetype and */
        goto gamma; /* go to main edge-processor.*/
    }
}

edge_end_l: /* The main edge-processor returns here when the edge is done*/
    iv = ivp1; /* Set subscripts for the next left edge. */
    ivp1= (iv+1) % scrnfaceptr->numvert; /* ' ' */
}

/*=== Make a note of the edgetype of the last edge on the left. ===*/
lastleftedgetype = edgetype;

/*==== Process each edge of the right-hand side of polygon. ====*/
iv = scrnfaceptr->imin; /* Find subscripts of first */
ivm1= (iv+scrnfaceptr->numvert-1) % scrnfaceptr->numvert; /* right edge & */
xend=scrnfaceptr->xcoord[iv]; yend=scrnfaceptr->ycoord[iv]; /* the coords */
xpend = (int)xend; ypend = (int)yend; /* of its starting point.*/

firstrightedge = TRUE;

while(iv!=scrnfaceptr->imax) {
    xpix = xpend; ypix = ypend; /* get coords of the */
    xstt = xend; ystt = yend; /* starting pixel of edge */
    xpstt = xpend; ypstt = ypend; /* and */
    xpend=(int)(xend= scrnfaceptr->xcoord[ivm1]); /* the coords of */
    ypend=(int)(yend= scrnfaceptr->ycoord[ivm1]); /* the end pixel. */
    extend= xend-xpend; eyend= yend-ypend; /* Bresenham errors at end pixel. */

    if(xstt<xend) { /*--- Right: Edgetype 3 ---*/
        /*-- Correct for bottom-most pixel bitmask, if necessary. --*/
        if(firstrightedge == TRUE) {
            firstrightedge = FALSE;
            if(firstleftedgetype==1) {
                exstt=(xstt-xpstt); eystt=(ystt-ypstt);
                resultproc( xpix,ypix, exstt, eystt, exstt, eystt, MY, 4);
            }
        }
        edgetype=3; toggle = 1; /* Set the current edgetype and */
        goto gamma; /* go to the main edge-processor. */
    }
    else { /*--- Right: Edgetype 2 ---*/
        /*-- Correct for bottom-most pixel bitmask, if necessary. --*/
        if(firstrightedge==TRUE) {
            firstrightedge = FALSE;
            if(firstleftedgetype==4) {
                exstt=(xstt-xpstt); eystt=(ystt-ypstt);
                resultproc(xpix,ypix, exstt, eystt, exstt, eystt, MY, 3);
            }
        }
    }
}
```

```
    }
    resultproc(xpix,ypix, 0.0,0.0,0.0,0.0, RX,0);
    /* Special call equivalent to xr[ypix] = xpix. */
}
edgetype=2; toggle = 2; /* Set the current edgetype. */
goto gamma;             /* Go to main edge-processor.*/
}

edge_end_r: /* The main edge-processor returns here once edge is done */
iv = ivml;      /* set subscripts for next right-edge */
ivml= (iv+scrnfaceptr->numvert-1) % scrnfaceptr->numvert;
}
```

```
/*=== do the final vertex bitmask correction, if necessary ===*/
if(lastleftedgetype==4 && edgetype==2)
    resultproc(xpend,ypend, exend, eyend, exend, eyend, MY, 1);
else if(lastleftedgetype==1 && edgetype==3)
    resultproc(xpend,ypend, exend, eyend, exend, eyend, MY, 2);
```

```
return;
```

```
/*=== Routine ends here, extracted code follows =====
```

The following segment is the main edge-processor. It identifies the coords xpix,ypix of each pixel intersected by the edge, and the type of intersection (see Fig. 2 in the main text). All four edgetypes are handled using the switches 'edgetype' and 'toggle'

Two Bresenham 'error' quantities are used to identify the pixels:

ex: the x-distance from the left boundary of the pixel to the intercept of the edge with the upper boundary of the pixel.

ey: the y-distance from the right boundary of the pixel to the intercept of the edge with the right boundary of the pixel.

Each pixel is found from the previous one by updating either ex or ey and then testing its value to identify the type of intersection.

ex0, ey0 are initially the values of ex, ey at the starting pixel of the edge, but subsequently hold the previous values as ex,ey are updated.

```
=====*/
```

```
gamma:

/*=== If the edge lies entirely in one pixel, record this and finish. ===*/
if( (xpix==xpend) && (ypix==ypend) ) {
    resultproc(xpix,ypix, xstt-xpstt,ystt-ypstt, exend,eyend, NN, edgetype);
    switch(edgetype) {
        case 3: case 2: goto edge_end_r;
        case 4: case 1: goto edge_end_l;
    }
}
```

```
/*=== If not, calculate the slope of the edge. ===*/
delx = xend-xstt; dely = yend-ystt;
if(toggle==1) { /* edgetype 1 or 3 */
    delymp = dely; largemp = LARGE;
}
```

```
else { /* edgetype 2 or 4 */
    delymp = -dely; largemp = -LARGE;
}

if(delx!=0)    dy_by_dx=delymp/delx;
else {
    dx_by_dy = 0; dy_by_dx = LARGE;
}
if(dely!=0)    dx_by_dy=delx/dely;
else {
    dy_by_dx = 0; dx_by_dy = largemp;
}

/*== Look at first pixel. ==*/
ex0= xstt-xpstt; ey0= ystt-ypstt; /* Set Bresenham error variables */
ex = ex0 + dx_by_dy*(hpix-ey0);

if(toggle==1) { /* edgetype 1 or 3 */
    ey = ey0 + dy_by_dx*(hpix-ex0);
    if(ex < hpix) { /* Depending on the size of ex, the pixel is of type LX */
        resultproc(xpix, ypix, ex0, ey0, ex, hpix, LX, edgetype); goto alpha;
    }
    else { /* or of type LY. */
        resultproc(xpix, ypix, ex0, ey0, hpix, ey, LY, edgetype); goto beta;
    }
}
else { /* edgetype 2 or 4 */
    ey = ey0 + dy_by_dx*ex0;
    if(ex > 0) { /* Depending on the sign of ex, the pixel is of type RX */
        resultproc(xpix, ypix, ex0, ey0, ex, hpix, RX, edgetype); goto alpha;
    }
    else { /* or of type RY. */
        resultproc(xpix, ypix, ex0, ey0, 0.0, ey, RY, edgetype); goto beta;
    }
}

alpha: /*== The upper boundary just crossed; now at new y-level ==*/
ypix++;
ey -= hpix;

if(toggle==1) { /* edgetype 1 or 3 */
    /* If at last pixel of edge, finish off edge. */
    if( (ypix >= ypend) && (xpix >= xpend) ) {
        resultproc(xpend,ypend, ex, 0.0, xend, eyend, RX, edgetype);
        if(edgetype==1) goto edge_end_l;
        else goto edge_end_r;
    }
}
else { /* edgetype 2 or 4 */
    /* If at last pixel of edge, finish off edge. */
    if( (ypix >= ypend) && (xpix <= xpend) ) {
        resultproc(xpend, ypend, ex, 0.0, xend, eyend, LX, edgetype);
        if(edgetype==4) goto edge_end_l;
        else goto edge_end_r;
    }
}

/*== Not yet at end of edge. Update ex and carry on. ==*/
ex0 = ex; ex += dx_by_dy;
if(toggle==1) { /* edgetype 1 or 3 */
    if(ex < hpix) { /* Depending on the size of ex, pixel is of type MX */
```

```
    resultproc(xpix, ypix, ex0, 0.0, ex, hpix, MX, edgetype); goto alpha;
  }
  else {
    /* or of type SY. */
    resultproc(xpix, ypix, ex0, 0.0, hpix, ey, SY, edgetype); goto beta;
  }
}
else {
  /* edgetype 2 or 4 */
  if(ex > 0) { /* Depending on the sign of ex, pixel is of type MX */
    resultproc(xpix, ypix, ex0, 0.0, ex, hpix, MX, edgetype); goto alpha;
  }
  else {
    /* or of type TY. */
    resultproc(xpix, ypix, ex0, 0.0, 0.0, ey, TY, edgetype); goto beta;
  }
}

beta: /*== Just crossed a vertical pixel boundary; now at new x-level ==*/
if(toggle==1) { /* edgetype 1 or 3 */
  xpix++;
  ex -= hpix;
  /* If at last pixel of edge, finish off edge. */
  if( (ypix>=ypend) && (xpix>=xpend) ) {
    resultproc(xpend, ypend, 0.0, ey, xend, eyend, RY, edgetype);
    if(edgetype==3) goto edge_end_r;
    else goto edge_end_l;
  }
}
else {
  /* edgetype 2 or 4 */
  xpix--;
  ex += hpix;
  /* If at last pixel of edge, finish off edge. */
  if( (xpix <= xpend) && (ypix >= ypend) ) {
    resultproc(xpend, ypend, hpix, ey, xend, eyend, LY, edgetype);
    if(edgetype==4) goto edge_end_l;
    else goto edge_end_r;
  }
}

/*== Not yet at end of edge. Update ey, and carry on. ==*/
ey0 = ey; ey += dy_by_dx;
if(toggle==1) { /* edgetype 1 or 3 */
  if(ey < hpix) { /* Depending on size of ey, pixel is of type MY */
    resultproc(xpix, ypix, 0.0, ey0, hpix, ey, MY, edgetype); goto beta;
  }
  else {
    /* or of type SX. */
    resultproc(xpix, ypix, 0.0, ey0, ex, hpix, SX, edgetype); goto alpha;
  }
}
else {
  /* edgetype 2 or 4 */
  if(ey < hpix) { /* Depending on size of ey, pixel is of type MY */
    resultproc(xpix, ypix, hpix, ey0, 0.0, ey, MY, edgetype); goto beta;
  }
  else {
    /* or of type TX. */
    resultproc(xpix, ypix, hpix, ey0, ex, hpix, TX, edgetype); goto alpha;
  }
}
}
```

/\*=====

This illustrates how the output from the edge calculator can be used  
The routine modifies externally defined arrays fragptr[], xl[] and xr[] (which  
are accessed via global pointers).

It uses a global pointer to recognize precalculated bitmasks held in a structure.

```
===== */

void resultproc(xpix,ypix,x1,y1,x2,y2,pixtype,edgetype)
int xpix,ypix;          /* position of pixel to which information applies */
float x1,y1,x2,y2;      /* position of the two endpoints of edge intersecting
                        the polygon */
int pixtype;            /* type of overlap in this pixel (NN,LX,LY,...) */
int edgetype;           /* type of edge (1,2,3 or 4) */

{
extern aabufftype * aabuffptr; /* pointer to a structure in which the
    precomputed bitmasks are held. Their format is given in the text.*/
extern unsigned short * fragptr; /* pointer to linear array holding bitmasks
    of the overlaps of the polygon with pixels, assuming HRES
    pixels per scanline. On entry each entry should be set to 0xffff */
extern int * xl, * xr; /* pointers to arrays storing positions of the
    left and right edges of the polygon: xl[y] and xr[y] give the
    x-position of the leftmost and rightmost pixels of the polygon on
    scanline y */
static float scale = 3.999; /* this should be set to just less than the ratio
    of pixel to subpixel widths. The value 3.999 is for the case where
    each pixel is 4x4 subpixels */
int ij,ix1,ix2,iy1,iy2;

ij = xpix + HRES*ypix;
ix1 = x1*scale;  iy1 = y1*scale;
ix2 = x2*scale;  iy2 = y2*scale;






/*-- adjust current bitmask by bitwise ANDing it with computed bitmask --*/
if(edgetype>0)
    fragptr[ij] &= aabuffptr->mask[edgetype] [ix1] [iy1] [ix2] [iy2];

/*-- record the leftmost and rightmost pixel positions, xl[y] and xr[y]
    of the polygon on each scanline y --*/
if (rl[pixtype][edgetype]== 1)  xr[ypix]=xpix;
else if (rl[pixtype][edgetype]==-1)  xl[ypix]=xpix;

return ;
}
```




















# Index of

## /pubs/tog/GraphicsGems/gemsiii/partition3d/



























Name	Last modified	Size	Description
 <a href="#">_Parent Directory</a>			
 <a href="#">_Makefile</a>	29-Jun-00 08:16	1K	
 <a href="#">_main.c</a>	29-Jun-00 08:16	2K	
 <a href="#">_partition.c</a>	29-Jun-00 08:16	12K	
 <a href="#">_partition.h</a>	29-Jun-00 08:16	1K	

# Index of

## /pubs/tog/GraphicsGems/gemsiii/

Name	Last modified	Size	Description
 <a href="#">Parent Directory</a>			
 <a href="#">3d.c</a>	29-Jun-00 08:16	3K	
 <a href="#">AllGems.TOC</a>	29-Jun-00 08:22	26K	
 <a href="#">Errata.GraphicsGemsII+</a>	29-Jun-00 08:16	9K	
 <a href="#">GraphicsGems.c</a>	29-Jun-00 08:16	12K	
 <a href="#">GraphicsGems.h</a>	29-Jun-00 08:16	4K	
 <a href="#">Makefile</a>	29-Jun-00 08:16	2K	
 <a href="#">PIR.c</a>	29-Jun-00 08:16	1K	
 <a href="#">Polyintr.c</a>	29-Jun-00 08:16	10K	
 <a href="#">README</a>	29-Jun-00 08:17	5K	
 <a href="#">accForm.c</a>	29-Jun-00 08:16	7K	
 <a href="#">accurate_scan/</a>	29-Jun-00 08:16	1K	
 <a href="#">alloc/</a>	29-Jun-00 08:16	1K	
 <a href="#">bezierTri.C</a>	29-Jun-00 08:16	3K	
 <a href="#">bitmap.c</a>	29-Jun-00 08:16	7K	
 <a href="#">bounding_volumes.c</a>	29-Jun-00 08:16	12K	
 <a href="#">bsp.c</a>	29-Jun-00 08:16	14K	
 <a href="#">bzrinter.c</a>	29-Jun-00 08:16	7K	
 <a href="#">circlexc.c</a>	29-Jun-00 08:16	4K	
 <a href="#">con2d.c</a>	29-Jun-00 08:16	8K	
 <a href="#">contour.c</a>	29-Jun-00 08:16	10K	



 <a href="#">_cyclic.c</a>	29-Jun-00 08:16	3K
 <a href="#">_defs.h</a>	29-Jun-00 08:16	1K
 <a href="#">_edgeCalc.c</a>	29-Jun-00 08:16	16K
 <a href="#">_exttest/</a>	29-Jun-00 08:16	1K
 <a href="#">_fastBitmap.c</a>	29-Jun-00 08:16	4K
 <a href="#">_fastLinear.c</a>	29-Jun-00 08:16	5K
 <a href="#">_fastSpan.c</a>	29-Jun-00 08:16	4K
 <a href="#">_fillet.c</a>	29-Jun-00 08:16	4K
 <a href="#">_filter.c</a>	29-Jun-00 08:16	12K
 <a href="#">_filter_rcg.c</a>	29-Jun-00 08:16	21K
 <a href="#">_forfac.c</a>	29-Jun-00 08:16	1K
 <a href="#">_hemis.c</a>	29-Jun-00 08:16	6K
 <a href="#">_insectc.c</a>	29-Jun-00 08:16	2K
 <a href="#">_intell.c</a>	29-Jun-00 08:16	3K
 <a href="#">_interval.C</a>	29-Jun-00 08:16	4K
 <a href="#">_intqdr.c</a>	29-Jun-00 08:16	3K
 <a href="#">_luminaire/</a>	29-Jun-00 08:16	1K
 <a href="#">_motblur.c</a>	29-Jun-00 08:16	7K
 <a href="#">_ndline.c</a>	29-Jun-00 08:16	2K
 <a href="#">_newell.c</a>	29-Jun-00 08:16	1K
 <a href="#">_panorama.c</a>	29-Jun-00 08:16	5K
 <a href="#">_parelarc.c</a>	29-Jun-00 08:16	2K
 <a href="#">_partition3d/</a>	29-Jun-00 08:16	1K
 <a href="#">_pl2plane.c</a>	29-Jun-00 08:16	2K
 <a href="#">_planeSets.c</a>	29-Jun-00 08:16	7K
 <a href="#">_pt2plane.c</a>	29-Jun-00 08:16	1K

 <a href="#">__quatspin.c</a>	29-Jun-00 08:17	1K
 <a href="#">__rand_rotation.c</a>	29-Jun-00 08:17	2K
 <a href="#">__rgbvary.c</a>	29-Jun-00 08:17	2K
 <a href="#">__rgbvaryW.c</a>	29-Jun-00 08:17	5K
 <a href="#">__scallops8.c</a>	29-Jun-00 08:17	8K
 <a href="#">__simplex/</a>	29-Jun-00 08:16	1K
 <a href="#">__sqfinal.c</a>	29-Jun-00 08:17	9K
 <a href="#">__sqrt.c</a>	29-Jun-00 08:17	2K
 <a href="#">__triangleCube.c</a>	29-Jun-00 08:17	11K
 <a href="#">__urot.c</a>	29-Jun-00 08:17	3K
 <a href="#">__vector.h</a>	29-Jun-00 08:17	3K
 <a href="#">__zdepth.c</a>	29-Jun-00 08:17	3K

CFLAGS = -I..

main: main.c partition.o partition.h  
 cc \$(CFLAGS) -o main main.c partition.o -lm

partition.o: partition.c partition.h  
 cc \$(CFLAGS) -c partition.c -o partition.o

clean:  
 rm -rf main partition.o

```
/* main.c: sample driver for partition module.
 * Copyright (c) Norman Chin
 */
#include <stdio.h>
#include <assert.h>
#include <malloc.h>
#include "GraphicsGems.h"
#include "partition.h"

static FACE *getFace(/* void */);
static void dumpFace(/* FACE *face, char *string */);

int main()
{
    FACE *inputFace;
    FACE *faceOn, *faceNeg, *facePos;
    FACE *faceOn2, *faceNeg2, *facePos2;

    inputFace= getFace(); dumpFace(inputFace,"input");

    /* partition unit square on XY plane about origin */
    partitionFaceWithPlane(1.0,0.0,0.0,0.0,&inputFace,
                          &faceOn,&faceNeg,&facePos);
    assert(inputFace == NULL_FACE);
    dumpFace(faceNeg,"negative side"); dumpFace(facePos,"positive side");

    /* now partition positive piece's upper left corner */
    inputFace= facePos;
    partitionFaceWithPlane(0.707,-0.707,0.0,0.0,&inputFace,
                          &faceOn2,&faceNeg2,&facePos2);
    assert(inputFace == NULL_FACE);
    dumpFace(faceNeg2,"negative side"); dumpFace(facePos2,"positive side");

    /* code to free all faces & their vertices goes here */

    return(0);
} /* main() */

static void dumpFace(face,string)
FACE *face;
char *string;
{
    VERTEX *vtrav;

    if (face == NULL_FACE) {
        (void) printf("%s is empty\n",string);
        return;
    }

    (void) printf("begin dump of %s at 0x%x\n",string,face);
    for (vtrav= face->vhead; vtrav != NULL_VERTEX; vtrav= vtrav->vnext) {
        (void) printf("    0x%x: (%lf %lf %lf)\n",vtrav,
                    vtrav->xx,vtrav->yy,vtrav->zz);
    }
    (void) printf("end %s\n",string);
} /* dumpFace() */

static FACE *getFace()
{
    /* list of vertices for unit square on XY plane about origin */
    static VERTEX square[]= {
```

```
    {1.0,-1.0,0.0, NULL_VERTEX},
    {1.0, 1.0,0.0, NULL_VERTEX},
    {-1.0, 1.0,0.0, NULL_VERTEX},
    {-1.0,-1.0,0.0, NULL_VERTEX},
    {1.0,-1.0,0.0, NULL_VERTEX}
};
VERTEX *v1, *v2, *v3, *v4, *v5;
FACE *newFace;

v1= NEWTYPE(VERTEX); assert(v1 != NULL_VERTEX); *v1= square[0];
v2= NEWTYPE(VERTEX); assert(v2 != NULL_VERTEX); *v2= square[1];
v3= NEWTYPE(VERTEX); assert(v3 != NULL_VERTEX); *v3= square[2];
v4= NEWTYPE(VERTEX); assert(v4 != NULL_VERTEX); *v4= square[3];
v5= NEWTYPE(VERTEX); assert(v5 != NULL_VERTEX); *v5= square[4];
/* chain vertices */
v1->vnex= v2; v2->vnex= v3; v3->vnex= v4; v4->vnex= v5;

/* attach vertex list to a face */
newFace= NEWTYPE(FACE); assert(newFace != NULL_FACE); newFace->vhead= v1;

return(newFace);
} /* getFace() */
```

```
/* partition.c: module to partition 3D convex face with an arbitrary plane.
 * Copyright (c) Norman Chin
 */
#include <stdio.h>          /* NULL */
#include <assert.h>         /* assert() */
#include <malloc.h>         /* malloc() */
#include <math.h>           /* fabs() */
#include "GraphicsGems.h"
#include "partition.h"

/* local functions */
static VERTEX *findNextIntersection(/* VERTEX *vstart, double aa,
                                   double bb, double cc, double dd,
                                   double *ixx, double *iyy, double *izz,
                                   SIGN *sign */);
static SIGN anyEdgeIntersectWithPlane(/* double x1, double y1, double z1,
                                       double x2, double y2, double z2,
                                       double aa, double bb, double cc,
                                       double dd,
                                       double *ixx, double *iyy, double *izz
                                       */);
static FACE *createOtherFace(/* FACE *face, VERTEX *v1, double ixx1,
                             double iyy1, double izz1, VERTEX *v2,
                             double ixx2, double iyy2, double izz2 */);
static SIGN whichSideIsFaceWRTplane(/* FACE *face, double aa, double bb,
                                    double cc, double dd */);
static VERTEX *allocVertex(/* double xx, double yy, double zz */);
static FACE *allocFace(/* FACE *face, VERTEX *vhead */);

/* Partitions a 3D convex polygon (face) with an arbitrary plane into its
 * negative and positive fragments, if any, w.r.t. the partitioning plane.
 * Note that inputFace is unusable afterwards since its vertex list has been
 * parceled out to the other faces. It's set to null to avoid dangling
 * pointer problem.
 */
void partitionFaceWithPlane(aa,bb,cc,dd,inputFace,faceOn,faceNeg,facePos)
double aa,bb,cc,dd;          /* partitioning plane's coefficients */
FACE **inputFace;           /* face to be partitioned */
FACE **faceOn;              /* returns face embedded in plane, if any */
FACE **faceNeg;             /* returns face on negative side, if any */
FACE **facePos;             /* returns face on positive side, if any */
{
    VERTEX *v1, *v2;
    double ixx1,iyy1,izz1, ixx2,iyy2,izz2;
    SIGN signV1, signV2;

    *faceOn= *faceNeg= *facePos= NULL_FACE;

    /* find first intersection */
    v1= findNextIntersection((*inputFace)->vhead,aa,bb,cc,dd,
                            &ixx1,&iyy1,&izz1,&signV1);
    if (v1 != NULL_VERTEX) {
        assert(signV1 != ZERO);

        /* first one found, find the 2nd one, if any */
        v2= findNextIntersection(v1->vnext,aa,bb,cc,dd,
                                &ixx2,&iyy2,&izz2,&signV2);

        /* Due to numerical instabilities, both intersection points may
         * have the same sign such as in the case when splitting very close
         * to a vertex. This should not count as a split.
         */
    }
}
```

```
    */
    if (v2 != NULL_VERTEX && signV1 == signV2) v2= NULL_VERTEX;

}
else v2= NULL_VERTEX;          /* No first intersection found,
                                * therefore no second intersection.
                                */

/* an intersection? */
if (v1 != NULL_VERTEX && v2 != NULL_VERTEX) { /* yes, intersection */
    FACE *newOtherFace;

    /* inputFace's vertex list will be modified */
    newOtherFace= createOtherFace(*inputFace,v1,ixx1,iyy1,izz1,
                                   v2,ixx2,iyy2,izz2);

    /* return split faces on appropriate lists */
    if (signV1 == NEGATIVE) {
        *faceNeg= *inputFace;
        *facePos= newOtherFace;
    }
    else {
        assert(signV1 == POSITIVE);
        *faceNeg= newOtherFace;
        *facePos= *inputFace;
    }
}
else {                          /* no intersection */
    SIGN side;

    /* Face is embedded or wholly to one side of partitioning plane. */
    side= whichSideIsFaceWRTplane(*inputFace,aa,bb,cc,dd);
    if (side == NEGATIVE)
        *faceNeg= *inputFace;
    else if (side == POSITIVE)
        *facePos= *inputFace;
    else {
        assert(side == ZERO);
        *faceOn= *inputFace;
    }
}
/* inputFace's vertex list has been parceled out to other lists so
 * set this to null.
 */
*inputFace= NULL_FACE;

/* If a face is embedded in plane then there must not be any faces on
 * either side.
 * Otherwise the face isn't embedded in plane so there must be at least
 * one face on either side or both.
 */
assert( (*faceOn != NULL_FACE) ?
        (*faceNeg == NULL_FACE &&
         *facePos == NULL_FACE) :
        (*faceNeg != NULL_FACE ||
         *facePos != NULL_FACE) );
} /* partitionFaceWithPlane() */

/* Finds next intersection on or after vstart.
 *
 * If an intersection is found,
```

```
*      a pointer to first vertex of the edge is returned,  
*      the intersection point (ixx,iyy,izz) and its sign is updated.  
* Otherwise a null pointer is returned.  
*/
```

```
static VERTEX *findNextIntersection(vstart,aa,bb,cc,dd,ixx,iyy,izz,sign)  
VERTEX *vstart;           /* where to start searching on vertex list */  
double aa,bb,cc,dd;       /* plane's coefficients */  
double *ixx,*iyy,*izz;    /* returned intersection point */  
SIGN *sign;               /* returned classification of intersection point */  
{  
    VERTEX *vtrav;  
  
    /* for all edges starting from vstart ... */  
    for (vtrav= vstart; vtrav->vnnext != NULL_VERTEX; vtrav= vtrav->vnnext) {  
        if ((*sign= anyEdgeIntersectWithPlane(vtrav->xx,vtrav->yy,vtrav->zz,  
                                                vtrav->vnnext->xx,vtrav->vnnext->yy,  
                                                vtrav->vnnext->zz,aa,bb,cc,dd,  
                                                ixx,iyy,izz))) {  
            return(vtrav);  
        }  
    }  
  
    return(NULL_VERTEX);  
} /* findNextIntersection() */
```

```
/* Memory allocated for split face's vertices and pointers tediously updated.  
*/
```

```
static FACE *createOtherFace(face,v1,ixx1,iyy1,izz1,v2,ixx2,iyy2,izz2)  
FACE *face;               /* face to be split */  
VERTEX *v1;               /* 1st vertex of edge of where 1st intersection was found */  
double ixx1,iyy1,izz1;    /* 1st intersection */  
VERTEX *v2;               /* 1st vertex of edge of where 2nd intersection was found */  
double ixx2,iyy2,izz2;    /* 2nd intersection */  
{  
    VERTEX *ilp1, *i2p1;   /* new vertices for original face */  
    VERTEX *ilp2, *i2p2;   /* new vertices for new face */  
    VERTEX *p2end;         /* new vertex for end of new face */  
    VERTEX *vtemp;         /* place holders */  
    register VERTEX *beforeV2; /* place holders */  
    FACE *newFace;         /* newly allocated face */  
  
    /* new intersection vertices */  
    ilp1= allocVertex(ixx1,iyy1,izz1);  
    i2p1= allocVertex(ixx2,iyy2,izz2);  
    ilp2= allocVertex(ixx1,iyy1,izz1);  
    i2p2= allocVertex(ixx2,iyy2,izz2);  
  
    /* duplicate 1st vertex of 2nd list to close it up */  
    p2end= allocVertex(v2->xx,v2->yy,v2->zz);  
  
    vtemp= v1->vnnext;  
  
    /* merge both intersection vertices ilp1 & i2p1 into 1st list */  
    if (ISVERTEX_EQ(i2p1,v2->vnnext)) { /* intersection vertex coincident? */  
        FREEVERTEX(i2p1);             /* yes, so free it */  
        ilp1->vnnext= v2->vnnext;  
    }  
    else {  
        i2p1->vnnext= v2->vnnext;     /* attach intersection list onto 1st list */  
        ilp1->vnnext= i2p1;           /* attach both intersection vertices */  
    }  
}
```



```
    v1->vnext= ilp1; /* attach front of 1st list to intersection vertices */

/* merge intersection vertices ilp2, i2p2 & p2end into second list */
i2p2->vnext= ilp2;      /* attach both intersection vertices */
v2->vnext= i2p2;        /* attach 2nd list to intersection vertices */
if (vtemp == v2) {
    ilp2->vnext= p2end;  /* close up 2nd list */
}
else {
    if (ISVERTEX_EQ(ilp2,vtemp)) { /* intersection vertex coincident? */
        FREEVERTEX(ilp2);      /* yes, so free it */
        i2p2->vnext= vtemp;    /* attach intersection vertex to 2nd list */
    }
    else {
        ilp2->vnext= vtemp;    /* attach intersection list to 2nd list */
    }
    /* find previous vertex to v2 */
    for (beforeV2= vtemp; beforeV2->vnext != v2; beforeV2= beforeV2->vnext)
        ; /* lone semi-colon */
    beforeV2->vnext= p2end;    /* and attach it to complete the 2nd list */
}

/* copy original face info but with new vertex list */
newFace= allocFace(face,v2);

return(newFace);
} /* createOtherFace() */

/* Determines which side a face is with respect to a plane with coefficient
 * (aa,bb,cc,dd).
 *
 * However, due to numerical problems, when a face is very close to the plane,
 * some vertices may be misclassified.
 * There are several solutions, two of which are mentioned here:
 * 1- classify the one vertex furthest away from the plane, (note that
 *    one need not compute the actual distance) and use that side.
 * 2- count how many vertices lie on either side and pick the side
 *    with the maximum. (this is the one implemented).
 */
static SIGN whichSideIsFaceWRTplane(face,aa,bb,cc,dd)
FACE *face;
double aa,bb,cc,dd;
{
    register VERTEX *vtrav;
    double value;
    boolean isNeg, isPos;

    isNeg= isPos= FALSE;

    for (vtrav= face->vhead; vtrav->vnext != NULL_VERTEX; vtrav= vtrav->vnext){
        value= (aa*vtrav->xx) + (bb*vtrav->yy) + (cc*vtrav->zz) + dd;
        if (value < -TOLER)
            isNeg= TRUE;
        else if (value > TOLER)
            isPos= TRUE;
        else assert(ISDOUBLE_EQ(value,0.0));
    } /* for vtrav */

    /* in the very rare case that some vertices slipped thru to other side of
     * plane due to round-off errors, execute the above again but count the
     * vertices on each side instead and pick the maximum.

```

```
*/
if (isNeg && isPos) {          /* yes so handle this numerical problem */
    int countNeg, countPos;

    /* count how many vertices are on either side */
    countNeg= countPos= 0;
    for (vtrav= face->vhead; vtrav->vnext != NULL_VERTEX;
        vtrav= vtrav->vnext) {
        value= (aa*vtrav->xx) + (bb*vtrav->yy) + (cc*vtrav->zz) + dd;
        if (value < -TOLER)
            countNeg++;
        else if (value > TOLER)
            countPos++;
        else assert(ISDOUBLE_EQ(value,0.0));
    } /* for */

    /* return the side corresponding to the maximum */
    if (countNeg > countPos) return(NEGATIVE);
    else if (countPos > countNeg) return(POSITIVE);
    else return(ZERO);
}
else {                          /* this is the usual case */
    if (isNeg) return(NEGATIVE);
    else if (isPos) return(POSITIVE);
    else return(ZERO);
}
} /* whichSideIsFaceWRTplane() */

/* Determines if an edge bounded by (x1,y1,z1)->(x2,y2,z2) intersects
 * the plane with the coefficients (aa,bb,cc,dd).
 *
 * If there's an intersection,
 * the sign of (x1,y1,z1), NEGATIVE or POSITIVE, w.r.t. the plane is
 * returned with the intersection (ixx,iyy,izz) updated.
 * Otherwise ZERO is returned.
 */
static SIGN anyEdgeIntersectWithPlane(x1,y1,z1,x2,y2,z2,aa,bb,cc,dd,
                                      ixx,iyy,izz)
double x1,y1,z1, x2,y2,z2;          /* both vertices of edge */
double aa,bb,cc,dd;                 /* partitioning plane's coefficients */
double *ixx,*iyy,*izz;               /* returned intersection point, if any */
{
    double temp1, temp2;
    int sign1, sign2;                /* must be int since gonna do a bitwise ^ */

    /* get signs */
    temp1= (aa*x1) + (bb*y1) + (cc*z1) + dd;
    if (temp1 < -TOLER)
        sign1= -1;
    else if (temp1 > TOLER)
        sign1= 1;
    else {
        /* edges beginning with a 0 sign are not considered valid intersections
         * case 1 & 6 & 7, see text
         */
        assert(ISDOUBLE_EQ(temp1,0.0));
        return(ZERO);
    }

    temp2= (aa*x2) + (bb*y2) + (cc*z2) + dd;
    if (temp2 < -TOLER)
```

```
    sign2= -1;
else if (temp2 > TOLER)
    sign2= 1;
else {
    /* case 8 & 9, see text */
    assert(ISDOUBLE_EQ(temp2,0.0));
    *ixx= x2;
    *iyy= y2;
    *izz= z2;

    return( (sign1 == -1) ? NEGATIVE : POSITIVE);
}

/* signs different?
 * recall: -1^1 == 1^-1 ==> 1    case 4 & 5, see text
 *        -1^-1 == 1^1 ==> 0    case 2 & 3, see text
 */
if (sign1 ^ sign2) {
    double dx,dy,dz;
    double denom, tt;

    /* compute intersection point */
    dx= x2-x1;
    dy= y2-y1;
    dz= z2-z1;

    denom= (aa*dx) + (bb*dy) + (cc*dz);
    tt= - ((aa*x1) + (bb*y1) + (cc*z1) + dd) / denom;

    *ixx= x1 + (tt * dx);
    *iyy= y1 + (tt * dy);
    *izz= z1 + (tt * dz);

    assert(sign1 == 1 || sign1 == -1);

    return( (sign1 == -1) ? NEGATIVE : POSITIVE );
}
else return(ZERO);
} /* anyEdgeIntersectWithPlane() */

/* allocates vertex with coordinates (xx,yy,zz) */
static VERTEX *allocVertex(xx,yy,zz)
double xx,yy,zz;
{
    VERTEX *newVertex;

    newVertex= NEWTYPE(VERTEX); assert(newVertex != NULL_VERTEX);
    newVertex->xx= xx; newVertex->yy= yy; newVertex->zz= zz;
    newVertex->vnext= NULL_VERTEX;

    return(newVertex);
} /* allocVertex() */

/* allocates face with list of vertices, vhead */
static FACE *allocFace(face,vhead)
FACE *face;
VERTEX *vhead;
{
    FACE *newFace;

    newFace= NEWTYPE(FACE); assert(newFace != NULL_FACE);
```

```
    *newFace= *face;  
    newFace->vhead= vhead;  
  
    return(newFace);  
} /* allocFace() */
```

```
/* partition.h: header file for partition module.
 * Copyright (c) Norman Chin
 */
#ifndef _PARTITION_INCLUDED
#define _PARTITION_INCLUDED

typedef enum { ZERO, NEGATIVE, POSITIVE } SIGN;

typedef struct vertexTag {
    double xx,yy,zz;          /* 3D coordinates of vertex */

    struct vertexTag *vnext;   /* ptr to next vertex */
} VERTEX;
#define NULL_VERTEX ((VERTEX *) NULL)

typedef struct {
    int someInfo; /* some face info goes here, ie. material properties */

    VERTEX *vhead;      /* ptr to first vertex of face */
} FACE;
#define NULL_FACE ((FACE *) NULL)

#define TOLER 0.000001
#define ISDOUBLE_EQ(a,b) ((fabs((a)-(b)) >= (double) TOLER) ? 0 : 1)

#define ISVERTEX_EQ(v1,v2) \
    (ISDOUBLE_EQ((v1)->xx,(v2)->xx) && \
     ISDOUBLE_EQ((v1)->yy,(v2)->yy) && \
     ISDOUBLE_EQ((v1)->zz,(v2)->zz))













#define FREEVERTEX(v) (free((char *) (v)))

/* external functions */
void partitionFaceWithPlane(/* double aa, double bb, double cc, double dd,
    FACE *face, FACE **faceOn,
    FACE **faceNeg, FACE **facePos
    */
);

#endif _PARTITION_INCLUDED
```

# Index of

## /pubs/tog/GraphicsGems/gemsv/ch3-5/

Name	Last modified	Size	Description
 <a href="#">Parent Directory</a>			
 <a href="#">README</a>	29-Jun-00 08:22	1K	
 <a href="#">bsp.h</a>	29-Jun-00 08:22	2K	
 <a href="#">bsp.make</a>	29-Jun-00 08:22	1K	
 <a href="#">bspAlloc.c</a>	29-Jun-00 08:22	2K	
 <a href="#">bspCollide.c</a>	29-Jun-00 08:22	5K	
 <a href="#">bspMemory.c</a>	29-Jun-00 08:22	1K	
 <a href="#">bspPartition.c</a>	29-Jun-00 08:22	11K	
 <a href="#">bspTree.c</a>	29-Jun-00 08:22	8K	
 <a href="#">bspUtility.c</a>	29-Jun-00 08:22	2K	
 <a href="#">foo.dat</a>	29-Jun-00 08:22	1K	
 <a href="#">mainBsp.c</a>	29-Jun-00 08:22	3K	

```
# Instructions for making/compiling the BSP-tree code  
# as appearing throughout "A Walk Through BSP Trees",  
# in _Graphics Gems V_, pages 121-138.
```

```
$ rm *.o  
$ make -f bsp.make  
$ bsp foo.dat
```

```
/* bsp.h: header file for BSP tree algorithm
 * Copyright (c) Norman Chin
 */
#ifndef _BSP_INCLUDED
#define _BSP_INCLUDED

#include <stdio.h>
#include <stdlib.h>          /* exit() */
#include <assert.h>          /* assert() */
#include <math.h>             /* fabs() */
#include <values.h>           /* MAXINT */
#include "../ch7-7/GG4D/GGems.h"

typedef struct { float rr,gg,bb; } COLOR;
typedef struct { float xx,yy,zz; } POINT;
typedef struct { float aa,bb,cc,dd; } PLANE;

typedef struct vertexTag {
    float xx,yy,zz;          /* vertex position */
    struct vertexTag *vnext;  /* pointer to next vertex in CCW order */
} VERTEX;
#define NULL_VERTEX ((VERTEX *) NULL)

typedef struct faceTag {
    COLOR color;              /* color of face */
    VERTEX *vhead;            /* head of list of vertices */
    PLANE plane;              /* plane equation of face */
    struct faceTag *fnext;    /* pointer to next face */
} FACE;
#define NULL_FACE ((FACE *) NULL)

typedef enum {PARTITION_NODE= 'p', IN_NODE= 'i', OUT_NODE= 'o'} NODE_TYPE;

typedef struct partitionnodeTag {
    FACE *sameDir, *oppDir;   /* pointers to faces embedded in node */

    struct bspnodeTag *negativeSide, *positiveSide; /* "-" & "+" branches */
} PARTITIONNODE;
#define NULL_PARTITIONNODE ((PARTITIONNODE *) NULL)

typedef struct bspnodeTag {
    NODE_TYPE kind;           /* kind of BSP node */

    PARTITIONNODE *node; /* if kind == (IN_NODE || OUT_NODE) then NULL */
} BSPNODE;
#define NULL_BSPNODE ((BSPNODE *) NULL)

#define TOLER 0.0000076
#define IS_EQ(a,b) ((fabs((double)(a)-(b)) >= (double) TOLER) ? 0 : 1)
typedef enum {NEGATIVE= -1, ZERO= 0, POSITIVE= 1} SIGN;
#define FSIGN(f) (((f) < -TOLER) ? NEGATIVE : ((f) > TOLER ? POSITIVE : ZERO))

/* external functions */
BSPNODE *BSPconstructTree(FACE **faceList);
boolean BSPisViewerInPositiveSideOfPlane(const PLANE *plane,const POINT *position);
void BSPtraverseTreeAndRender(const BSPNODE *bspNode,const POINT *position);
void BSPfreeTree(BSPNODE **bspNode);

boolean BSPdidViewerCollideWithScene(const POINT *from, const POINT *to,
                                     const BSPNODE *bspTree);
```



```
VERTEX *allocVertex(float xx,float yy,float zz);
FACE *allocFace(const COLOR *color, VERTEX *vlist,const PLANE *plane);
void appendVertex(VERTEX **vhead,VERTEX **vtail,VERTEX *vertex);
void appendFace(FACE **fhead,FACE **ftail,FACE *face);
void freeVertexList(VERTEX **vlist);
void freeFaceList(FACE **flist);

void computePlane(float xx0,float yy0,float zz0,float xx1,float yy1,float zz1,
                  float xx2,float yy2,float zz2, PLANE *plane);

SIGN anyEdgeIntersectWithPlane(float x1, float y1, float z1,
                                float x2, float y2, float z2,
                                const PLANE *plane,
                                float *ixx, float *iyy, float *izz);
void BSPpartitionFaceListWithPlane(const PLANE *plane,FACE **faceList,
                                   FACE **faceNeg, FACE **facePos,
                                   FACE **faceSameDir, FACE **faceOppDir);
void drawFaceList(FILE *, const FACE *faceList);

char *MYMALLOC(unsigned num);
void MYFREE(char *ptr);
long MYMEMORYCOUNT(void);
#endif /* _BSP_INCLUDED */
```

```
# bsp.make
#
HEADERS = bsp.h GraphicsGems.h
OBJS     = bspAlloc.o bspCollide.o bspPartition.o \
bspTree.o bspUtility.o mainBsp.o bspMemory.o

OPT      = -g
LIBS     = -lm
BSP      = bsp
# ANSI-C: Use CC on Suns. Use cc -Aa on HPs.
CC       = CC

$(BSP)   : $(OBJS)
          $(CC) $(OPT) $(OBJS) $(LIBS) -o $(BSP)

bspAlloc.o      : $(HEADERS) bspAlloc.c
                  $(CC) $(OPT) -c bspAlloc.c
bspCollide.o    : $(HEADERS) bspCollide.c
                  $(CC) $(OPT) -c bspCollide.c
bspPartition.o  : $(HEADERS) bspPartition.c
                  $(CC) $(OPT) -c bspPartition.c
bspTree.o       : $(HEADERS) bspTree.c
                  $(CC) $(OPT) -c bspTree.c
bspUtility.o    : $(HEADERS) bspUtility.c
                  $(CC) $(OPT) -c bspUtility.c
bspMemory.o     : $(HEADERS) bspMemory.c
                  $(CC) $(OPT) -c bspMemory.c
mainBsp.o       : $(HEADERS) mainBsp.c
                  $(CC) $(OPT) -c mainBsp.c
# bsp.make
```

```
/* bspAlloc.c: module to allocate, free and append vertices and faces.
 * Copyright (c) Norman Chin
 */
#include "bsp.h"

/* Allocates a vertex with position (xx,yy,zz) */
VERTEX *allocVertex(float xx,float yy,float zz)
{
    VERTEX *newVertex;

    if ((newVertex= (VERTEX *) MYMALLOC(sizeof(VERTEX))) == NULL_VERTEX) {
        fprintf(stderr,"?Unable to malloc vertex.\n");
        exit(1);
    }
    newVertex->xx= xx; newVertex->yy= yy; newVertex->zz= zz;
    newVertex->vnext= NULL_VERTEX;

    return(newVertex);
} /* allocVertex() */

/* Allocates a face with color, a list of vertices, a plane equation.
 */
FACE *allocFace(const COLOR *color,VERTEX *vlist,const PLANE *plane)
{
    FACE *newFace;

    if ((newFace= (FACE *) MYMALLOC(sizeof(FACE))) == NULL_FACE) {
        fprintf(stderr,"?Unable to alloc face.\n");
        exit(1);
    }
    newFace->color= *color;
    newFace->vhead= vlist;
    newFace->plane= *plane;
    newFace->fnext= NULL_FACE;

    return(newFace);
} /* allocFace() */

/* Append a vertex to a list. */
void appendVertex(VERTEX **vhead,VERTEX **vtail,VERTEX *vertex)
{
    assert( (*vhead == NULL_VERTEX) ? (*vtail == NULL_VERTEX) : 1 );
    assert(vertex != NULL_VERTEX);

    if (*vhead == NULL_VERTEX)
        *vhead= vertex;
    else (*vtail)->vnext= vertex;

    *vtail= vertex;          /* update tail */
} /* appendVertex() */

/* Append a face to a list. */
void appendFace(FACE **fhead,FACE **ftail,FACE *face)
{
    assert( (*fhead == NULL_FACE) ? (*ftail == NULL_FACE) : 1 );
    assert(face != NULL_FACE);

    if (*fhead == NULL_FACE)
        *fhead= face;
    else (*ftail)->fnext= face;
}
```

```
    *ftail= face;                                /* update tail */
} /* appendFace() */

/* Frees list of vertices. */
void freeVertexList(VERTEX **vlist)
{
    VERTEX *vtrav= *vlist, *vdel;
    while (vtrav != NULL_VERTEX) {
        vdel= vtrav; vtrav= vtrav->vnnext;

        MYFREE((char *) vdel);
    }
    *vlist= NULL_VERTEX;
} /* freeVertexList() */

/* Frees list of faces. */
void freeFaceList(FACE **flist)
{
    FACE *ftrav= *flist, *fdel;
    while (ftrav != NULL_FACE) {
        fdel= ftrav; ftrav= ftrav->fnnext; freeVertexList(&fdel->vhead);

        MYFREE((char *)fdel);
    }
    *flist= NULL_FACE;
} /* freeFaceList() */
/**** bspAlloc.c ****/
```

```
/* bspCollide.c: module to detect collisions between the viewer and static
 * objects in an environment represented as a BSP tree.
 * Copyright (c) Norman Chin
 */
#include "bsp.h"

/* flags to indicate if any piece of a line segment is inside any polyhedron
 * or outside all polyhedra
 */
static boolean anyPieceOfLineIn, anyPieceOfLineOut;

/* local functions - see function definition */
static int BSPclassifyPoint(const POINT *point, const BSPNODE *bspNode);
static void BSPclassifyLineInterior(const POINT *from, const POINT *to,
                                   const BSPNODE *bspNode);

/* Returns a boolean to indicate whether or not a collision had occurred
 * between the viewer and any static objects in an environment represented as
 * a BSP tree.
 *
 * from      - start position of viewer
 * to        - end position of viewer
 * bspTree   - BSP tree of scene
 */
boolean BSPdidViewerCollideWithScene(const POINT *from, const POINT *to,
                                   const BSPNODE *bspTree)
{
    /* first classify the endpoints */
    int sign1= BSPclassifyPoint(from,bspTree);
    int sign2= BSPclassifyPoint(to,bspTree);

    /* collision occurs iff there's a state change between endpoints or
     * either endpoint is on an object
     */
    if (sign1 == 0 || sign2 == 0 || sign1 != sign2) return(1);
    else {
        anyPieceOfLineIn= anyPieceOfLineOut= 0; /* clear flags */
        /* since we already classified the endpoints, try interior of line */
        /* this routine will set the flags to appropriate values */
        BSPclassifyLineInterior(from,to,bspTree);

        /* if line interior is inside and outside an object, collision detected*/
        /* else no collision detected */
        return( (anyPieceOfLineIn && anyPieceOfLineOut) ? 1 : 0 );
    }
} /* BSPdidViewerCollideWithScene() */

/* Classifies point as to whether or not it is inside, outside or on an object
 * represented as a BSP tree, where inside is -1, outside is 1, on is 0.
 *
 * point      - position of point
 * bspNode    - a node in BSP tree
 */
static int BSPclassifyPoint(const POINT *point,const BSPNODE *bspNode)
{
    if (bspNode == NULL_BSPNODE) return(1); /* point is out since no tree */

    if (bspNode->kind == PARTITION_NODE) { /* compare point with plane */
        const PLANE *plane= &bspNode->node->sameDir->plane;
        float dp= plane->aa*point->xx + plane->bb*point->yy +
                  plane->cc*point->zz + plane->dd;
```

```
    if (dp < -TOLER)          /* point on "-" side, filter down "-" branch */
        return(BSPclassifyPoint(point,bspNode->node->negativeSide));
    else if (dp > TOLER)      /* point on "+" side, filter down "+" branch */
        return(BSPclassifyPoint(point,bspNode->node->positiveSide));
    else {
        /* point is on plane, so classify the neighborhood of point by
         * filtering the same point down both branches.
         */
        int sign1= BSPclassifyPoint(point,bspNode->node->negativeSide);
        int sign2= BSPclassifyPoint(point,bspNode->node->positiveSide);
        /* if classification is same then return it otherwise it's on */
        return( (sign1 == sign2) ? sign1 : 0 );
    }
}
else if (bspNode->kind == OUT_NODE) return(1); /* point is outside */
else { assert(bspNode->kind == IN_NODE); return(-1); } /* point is inside */
} /* BSPclassifyPoint() */

/* Classifies interior of line segment (not including endpoints) as to whether
 * or not any piece is inside or outside an object represented as a BSP tree.
 * If it's on, it's recursively called on both half-spaces to set the flags.
 * There is no explicit on condition like we have with BSPclassifyPoint().
 *
 * from      - endpoint of line segment
 * to        - other endpoint of line segment
 * bspNode - a node in BSP tree
 */
static void BSPclassifyLineInterior(const POINT *from,const POINT *to,
                                   const BSPNODE *bspNode)
{
    if (bspNode->kind == PARTITION_NODE) { /* compare line segment with plane */
        float ixx,iyy,izz;
        const PLANE *plane= &bspNode->node->sameDir->plane;
        float dp1= plane->aa*from->xx + plane->bb*from->yy +
                   plane->cc*from->zz + plane->dd;
        float dp2= plane->aa*to->xx + plane->bb*to->yy +
                   plane->cc*to->zz + plane->dd;
        SIGN sign1= FSIGN(dp1); SIGN sign2= FSIGN(dp2);

        if ( (sign1 == NEGATIVE && sign2 == POSITIVE) ||
             (sign1 == POSITIVE && sign2 == NEGATIVE) ) { /* split! */
            SIGN check= anyEdgeIntersectWithPlane(from->xx,from->yy,from->zz,
                                                    to->xx,to->yy,to->zz,
                                                    plane,&ixx,&iyy,&izz);

            POINT iPoint;
            assert(check != ZERO);

            /* filter split line segments down appropriate branches */
            iPoint.xx= ixx; iPoint.yy= iyy; iPoint.zz= izz;
            if (sign1 == NEGATIVE) { assert(sign2 == POSITIVE);
                BSPclassifyLineInterior(from,&iPoint,bspNode->node->negativeSide);
                BSPclassifyLineInterior(to,&iPoint,bspNode->node->positiveSide);
            }
            else { assert(sign1 == POSITIVE && sign2 == NEGATIVE);
                BSPclassifyLineInterior(from,&iPoint,bspNode->node->positiveSide);
                BSPclassifyLineInterior(to,&iPoint,bspNode->node->negativeSide);
            }
        }
    }
    else {
        /* no split,so on same side */
        if (sign1 == ZERO && sign2 == ZERO) {
```

```
        BSPclassifyLineInterior(from,to,bspNode->node->negativeSide);
        BSPclassifyLineInterior(from,to,bspNode->node->positiveSide);
    }
    else if (sign1 == NEGATIVE || sign2 == NEGATIVE) {
        BSPclassifyLineInterior(from,to,bspNode->node->negativeSide);
    }
    else { assert(sign1 == POSITIVE || sign2 == POSITIVE);
        BSPclassifyLineInterior(from,to,bspNode->node->positiveSide);
    }
}
}
else if (bspNode->kind == IN_NODE) anyPieceOfLineIn= 1; /* line inside */
else { assert(bspNode->kind == OUT_NODE); anyPieceOfLineOut= 1; }
} /* BSPclassifyLineInterior() */
/**** bspCollide.c ****/
```

```
/* bspMemory.c: module to allocate and free memory and also to count them.
 * Copyright (c) Norman Chin
 */
#include "bsp.h"
#include <malloc.h>

static long memoryCount= 0L;

/* Allocates memory of num bytes */
char *MYMALLOC(unsigned num)
{
    char *memory= malloc(num);    /* checked for null by caller */

    ++memoryCount;                /* increment memory counter for debugging */
    return(memory);
} /* myMalloc() */

/* Frees memory pointed to by ptr */
void MYFREE(char *ptr)
{
    --memoryCount;                /* decrement memory counter for debugging */
    free(ptr);
} /* myFree() */

/* Returns how many memory blocks are still allocated up to this point */
long MYMEMORYCOUNT(void)
{
    return(memoryCount);
} /* myMemoryCount() */

/**** bspMemory.c ****/
```



```
/* partition.c: module to partition 3D convex face with an arbitrary plane.
 * Copyright (c) Norman Chin
 */
#include "bsp.h"

#define PREPEND_FACE(f,fl) (f->fnext= fl, fl= f)
#define ISVERTEX_EQ(v1,v2) \
    (IS_EQ((v1)->xx,(v2)->xx) && \
     IS_EQ((v1)->yy,(v2)->yy) && \
     IS_EQ((v1)->zz,(v2)->zz))

/* local functions */
static VERTEX *findNextIntersection(VERTEX *vstart, const PLANE *plane,
                                     float *ixx, float *iyy, float *izz,
                                     SIGN *sign);
static FACE *createOtherFace(FACE *face, VERTEX *v1, float ixx1,
                             float iyy1, float izz1, VERTEX *v2,
                             float ixx2, float iyy2, float izz2
                             );
static SIGN whichSideIsFaceWRTplane(FACE *face, const PLANE *plane);

/* Partitions a 3D convex polygon (face) with an arbitrary plane into its
 * negative and positive fragments, if any, w.r.t. the partitioning plane.
 * Note that faceList is unusable afterwards since its vertex list has been
 * parceled out to the other faces. It's set to null to avoid dangling
 * pointer problem. Faces embedded in the plane are separated into two lists,
 * one facing the same direction as the partitioning plane, faceSameDir, and
 * the other facing the opposite direction, faceOppDir.
 */
void BSPpartitionFaceListWithPlane(const PLANE *plane,FACE **faceList,
                                   FACE **faceNeg, FACE **facePos,
                                   FACE **faceSameDir, FACE **faceOppDir)
{
    FACE *ftrav= *faceList;

    *faceSameDir= *faceOppDir = *faceNeg= *facePos= NULL_FACE;

    while (ftrav != NULL_FACE) {
        VERTEX *v1, *v2; FACE *nextFtrav;
        float ixx1,iyy1,izz1, ixx2,iyy2,izz2;
        SIGN signV1, signV2;

        nextFtrav= ftrav->fnext;      /* unchain current face from list */
        ftrav->fnext= NULL_FACE;

        /* find first intersection */
        v1= findNextIntersection(ftrav->vhead,plane,&ixx1,&iyy1,&izz1,&signV1);
        if (v1 != NULL_VERTEX) {
            assert(signV1 != ZERO);

            /* first one found, find the 2nd one, if any */
            v2= findNextIntersection(v1->vnext,plane,&ixx2,&iyy2,&izz2,&signV2);

            /* Due to numerical instabilities, both intersection points may
             * have the same sign such as in the case when splitting very close
             * to a vertex. This should not count as a split.
             */
            if (v2 != NULL_VERTEX && signV1 == signV2) v2= NULL_VERTEX;

        }
        else v2= NULL_VERTEX;          /* No first intersection found,
```

```

                                * therefore no second intersection.
                                */

/* an intersection? */
if (v1 != NULL_VERTEX && v2 != NULL_VERTEX) { /* yes, intersection */
    FACE *newOtherFace;

    /* ftrav's vertex list will be modified */
    newOtherFace= createOtherFace(ftrav,v1,ixx1,iyy1,izz1,
                                v2,ixx2,iyy2,izz2
                                );

    /* return split faces on appropriate lists */
    if (signV1 == NEGATIVE) {
        PREPEND_FACE(ftrav,*faceNeg);
        PREPEND_FACE(newOtherFace,*facePos);
    }
    else {
        assert(signV1 == POSITIVE);
        PREPEND_FACE(newOtherFace,*faceNeg);
        PREPEND_FACE(ftrav,*facePos);
    }
}
else { /* no intersection */
    SIGN side;

    /* Face is embedded or wholly to one side of partitioning plane. */
    side= whichSideIsFaceWRTplane(ftrav,plane);
    if (side == NEGATIVE)
        PREPEND_FACE(ftrav,*faceNeg);
    else if (side == POSITIVE)
        PREPEND_FACE(ftrav,*facePos);
    else {
        assert(side == ZERO);
        if (IS_EQ(ftrav->plane.aa,plane->aa) &&
            IS_EQ(ftrav->plane.bb,plane->bb) &&
            IS_EQ(ftrav->plane.cc,plane->cc))
            PREPEND_FACE(ftrav,*faceSameDir);
        else PREPEND_FACE(ftrav,*faceOppDir);
    }
}
ftrav= nextFtrav; /* get next */
} /* while ftrav != NULL_FACE */

/* faceList's vertex list has been parceled out to other lists so
 * set this to null.
 */
*faceList= NULL_FACE;
#if 0 /* only true for BSPconstructTree() */
    /* there's at least one face embedded in the partitioning plane */
    assert(*faceSameDir != NULL_FACE);
#endif
} /* BSPpartitionFaceListWithPlane() */

/* Finds next intersection on or after vstart.
 *
 * If an intersection is found,
 * a pointer to first vertex of the edge is returned,
 * the intersection point (ixx,iyy,izz) and its sign is updated.
 * Otherwise a null pointer is returned.
```

```
*/
static VERTEX *findNextIntersection(VERTEX *vstart,const PLANE *plane,
                                   float *ixx,float *iyy,float *izz,
                                   SIGN *sign)
{
    VERTEX *vtrav;

    /* for all edges starting from vstart ... */
    for (vtrav= vstart; vtrav->vnnext != NULL_VERTEX; vtrav= vtrav->vnnext) {
        if ((*sign= anyEdgeIntersectWithPlane(vtrav->xx,vtrav->yy,vtrav->zz,
                                              vtrav->vnnext->xx,vtrav->vnnext->yy,
                                              vtrav->vnnext->zz,plane,
                                              ixx,iyy,izz))) {
            return(vtrav);
        }
    }

    return(NULL_VERTEX);
} /* findNextIntersection() */

/* Memory allocated for split face's vertices and pointers tediously updated.
*
* face - face to be split
* v1    - 1st vertex of edge of where 1st intersection was found
* (ixx1,iyy1,izz1) - 1st intersection
* v2    - 1st vertex of edge of where 2nd intersection was found
* (ixx2,iyy2,izz2) - 2nd intersection
*/
static FACE *createOtherFace(FACE *face,
                             VERTEX *v1, float ixx1, float iyy1, float izz1,
                             VERTEX *v2, float ixx2, float iyy2, float izz2
                             )
{
    VERTEX *ilp1, *i2p1;          /* new vertices for original face */
    VERTEX *ilp2, *i2p2;          /* new vertices for new face */
    VERTEX *p2end;                 /* new vertex for end of new face */
    VERTEX *vtemp;                 /* place holders */
    register VERTEX *beforeV2;     /* place holders */
    FACE *newFace;                 /* newly allocated face */

    /* new intersection vertices */
    ilp1= allocVertex(ixx1,iyy1,izz1);
    i2p1= allocVertex(ixx2,iyy2,izz2);
    ilp2= allocVertex(ixx1,iyy1,izz1);
    i2p2= allocVertex(ixx2,iyy2,izz2);

    /* duplicate 1st vertex of 2nd list to close it up */
    p2end= allocVertex(v2->xx,v2->yy,v2->zz);

    vtemp= v1->vnnext;

    /* merge both intersection vertices ilp1 & i2p1 into 1st list */
    if (ISVERTEX_EQ(i2p1,v2->vnnext)) { /* intersection vertex coincident? */
        assert(i2p1->vnnext == NULL_VERTEX);
        freeVertexList(&i2p1);        /* yes, so free it */
        ilp1->vnnext= v2->vnnext;
    }
    else {
        i2p1->vnnext= v2->vnnext;      /* attach intersection list onto 1st list */
        ilp1->vnnext= i2p1;            /* attach both intersection vertices */
    }
}
```

```
    v1->vnext= ilp1; /* attach front of 1st list to intersection vertices */

/* merge intersection vertices ilp2, i2p2 & p2end into second list */
i2p2->vnext= ilp2;          /* attach both intersection vertices */
v2->vnext= i2p2;            /* attach 2nd list to intersection vertices */
if (vtemp == v2) {
    ilp2->vnext= p2end;      /* close up 2nd list */
}
else {
    if (ISVERTEX_EQ(ilp2,vtemp)) { /* intersection vertex coincident? */
        assert(ilp2->vnext == NULL_VERTEX);
        freeVertexList(&ilp2); /* yes, so free it */
        i2p2->vnext= vtemp;    /* attach intersection vertex to 2nd list */
    }
    else {
        ilp2->vnext= vtemp;    /* attach intersection list to 2nd list */
    }
    /* find previous vertex to v2 */
    for (beforeV2= vtemp; beforeV2->vnext != v2; beforeV2= beforeV2->vnext)
        ; /* lone semi-colon */
    beforeV2->vnext= p2end;    /* and attach it to complete the 2nd list */
}

/* copy original face info but with new vertex list */
newFace= allocFace(&face->color,v2,&face->plane);

return(newFace);
} /* createOtherFace() */

/* Determines which side a face is with respect to a plane.
 *
 * However, due to numerical problems, when a face is very close to the plane,
 * some vertices may be misclassified.
 * There are several solutions, two of which are mentioned here:
 * 1- classify the one vertex furthest away from the plane, (note that
 *    one need not compute the actual distance) and use that side.
 * 2- count how many vertices lie on either side and pick the side
 *    with the maximum. (this is the one implemented).
 */
static SIGN whichSideIsFaceWRTplane(FACE *face, const PLANE *plane)
{
    register VERTEX *vtrav;
    float value;
    boolean isNeg, isPos;

    isNeg= isPos= FALSE;

    for (vtrav= face->vhead; vtrav->vnext != NULL_VERTEX; vtrav= vtrav->vnext){
        value= (plane->aa*vtrav->xx) + (plane->bb*vtrav->yy) +
            (plane->cc*vtrav->zz) + plane->dd;
        if (value < -TOLER)
            isNeg= TRUE;
        else if (value > TOLER)
            isPos= TRUE;
        else assert(IS_EQ(value,0.0));
    } /* for vtrav */

    /* in the very rare case that some vertices slipped thru to other side of
     * plane due to round-off errors, execute the above again but count the
     * vertices on each side instead and pick the maximum.
     */
}
```

```
if (isNeg && isPos) {          /* yes so handle this numerical problem */
    int countNeg, countPos;

    /* count how many vertices are on either side */
    countNeg= countPos= 0;
    for (vtrav= face->vhead; vtrav->vnext != NULL_VERTEX;
        vtrav= vtrav->vnext) {
        value= (plane->aa*vtrav->xx) + (plane->bb*vtrav->yy) +
            (plane->cc*vtrav->zz) + plane->dd;
        if (value < -TOLER)
            countNeg++;
        else if (value > TOLER)
            countPos++;
        else assert(IS_EQ(value,0.0));
    } /* for */

    /* return the side corresponding to the maximum */
    if (countNeg > countPos) return(NEGATIVE);
    else if (countPos > countNeg) return(POSITIVE);
    else return(ZERO);
}
else {                          /* this is the usual case */
    if (isNeg) return(NEGATIVE);
    else if (isPos) return(POSITIVE);
    else return(ZERO);
}
} /* whichSideIsFaceWRTplane() */

/* Determines if an edge bounded by (x1,y1,z1)->(x2,y2,z2) intersects
 * the plane.
 *
 * If there's an intersection,
 * the sign of (x1,y1,z1), NEGATIVE or POSITIVE, w.r.t. the plane is
 * returned with the intersection (ixx,iyy,izz) updated.
 * Otherwise ZERO is returned.
 */
SIGN anyEdgeIntersectWithPlane(float x1,float y1,float z1,
                                float x2,float y2,float z2,
                                const PLANE *plane,
                                float *ixx, float *iyy, float *izz)
{
    float temp1, temp2;
    int sign1, sign2;          /* must be int since gonna do a bitwise ^ */
    float aa= plane->aa; float bb= plane->bb; float cc= plane->cc;
    float dd= plane->dd;

    /* get signs */
    temp1= (aa*x1) + (bb*y1) + (cc*z1) + dd;
    if (temp1 < -TOLER)
        sign1= -1;
    else if (temp1 > TOLER)
        sign1= 1;
    else {
        /* edges beginning with a 0 sign are not considered valid intersections
         * case 1 & 6 & 7, see Gems III.
         */
        assert(IS_EQ(temp1,0.0));
        return(ZERO);
    }

    temp2= (aa*x2) + (bb*y2) + (cc*z2) + dd;
```

```
if (temp2 < -TOLER)
    sign2= -1;
else if (temp2 > TOLER)
    sign2= 1;
else {
    /* case 8 & 9, see Gems III */
    assert(IS_EQ(temp2,0.0));
    *ixx= x2;
    *iyy= y2;
    *izz= z2;

    return( (sign1 == -1) ? NEGATIVE : POSITIVE);
}

/* signs different?
 * recall: -1^1 == 1^-1 ==> 1    case 4 & 5, see Gems III
 *         -1^-1 == 1^1 ==> 0    case 2 & 3, see Gems III
 */
if (sign1 ^ sign2) {
    float dx,dy,dz;
    float denom, tt;

    /* compute intersection point */
    dx= x2-x1;
    dy= y2-y1;
    dz= z2-z1;

    denom= (aa*dx) + (bb*dy) + (cc*dz);
    tt= - ((aa*x1) + (bb*y1) + (cc*z1) + dd) / denom;

    *ixx= x1 + (tt * dx);
    *iyy= y1 + (tt * dy);
    *izz= z1 + (tt * dz);

    assert(sign1 == 1 || sign1 == -1);

    return( (sign1 == -1) ? NEGATIVE : POSITIVE );
}
else return(ZERO);
} /* anyEdgeIntersectWithPlane() */
/**** bspPartition.c ****/
```

```
/* bspTree.c: module to construct and traverse a BSP tree.
 * Copyright (c) Norman Chin
 */
#include "bsp.h"

/* local functions */
static void BSPchoosePlane(FACE *faceList, PLANE *plane);
static boolean doesFaceStraddlePlane(const FACE *face, const PLANE *plane);
static BSPNODE *allocBspNode(NODE_TYPE kind, FACE *sameDir, FACE *oppDir);
static PARTITIONNODE *allocPartitionNode(FACE *sameDir, FACE *oppDir);
static void freePartitionNode(PARTITIONNODE **partitionNode);

/* Returns a BSP tree of scene from a list of convex faces.
 * These faces' vertices are oriented in counterclockwise order where the last
 * vertex is a duplicate of the first, i.e., a square has five vertices.
 *
 * faceList - list of faces
 */
BSPNODE *BSPconstructTree(FACE **faceList)
{
    BSPNODE *newBspNode; PLANE plane;
    FACE *sameDirList, *oppDirList, *faceNegList, *facePosList;

    /* choose plane to split scene with */
    BSPchoosePlane(*faceList, &plane);
    BSPpartitionFaceListWithPlane(&plane, faceList, &faceNegList, &facePosList,
                                &sameDirList, &oppDirList);
    assert(*faceList == NULL_FACE); assert(sameDirList != NULL_FACE);

    /* construct the tree */
    newBspNode = allocBspNode(PARTITION_NODE, sameDirList, oppDirList);

    /* construct tree's "-" branch */
    if (faceNegList == NULL_FACE)
        newBspNode->node->negativeSide = allocBspNode(IN_NODE, NULL_FACE, NULL_FACE);
    else newBspNode->node->negativeSide = BSPconstructTree(&faceNegList);

    /* construct tree's "+" branch */
    if (facePosList == NULL_FACE)
        newBspNode->node->positiveSide = allocBspNode(OUT_NODE, NULL_FACE, NULL_FACE);
    else newBspNode->node->positiveSide = BSPconstructTree(&facePosList);

    return(newBspNode);
} /* BSPconstructTree() */

/* Traverses BSP tree to render scene back-to-front based on viewer position.
 *
 * bspNode - a node in BSP tree
 * position - position of viewer
 */
void BSPtraverseTreeAndRender(const BSPNODE *bspNode, const POINT *position)
{
    if (bspNode == NULL_BSPNODE) return;

    if (bspNode->kind == PARTITION_NODE) {
        if (BSPisViewerInPositiveSideOfPlane(&bspNode->node->sameDir->plane, position)) {
            BSPtraverseTreeAndRender(bspNode->node->positiveSide, position);
            drawFaceList(stdout, bspNode->node->sameDir);
            drawFaceList(stdout, bspNode->node->oppDir); /* back-face cull */
            BSPtraverseTreeAndRender(bspNode->node->negativeSide, position);
        }
        else {
            BSPtraverseTreeAndRender(bspNode->node->negativeSide, position);
            drawFaceList(stdout, bspNode->node->sameDir);
            drawFaceList(stdout, bspNode->node->oppDir); /* back-face cull */
            BSPtraverseTreeAndRender(bspNode->node->positiveSide, position);
        }
    }
}
```

```
    }
    else {

        BSPtraverseTreeAndRender(bspNode->node->positiveSide,position);
        drawFaceList(stdout,bspNode->node->oppDir);
        drawFaceList(stdout,bspNode->node->sameDir); /* back-face cull */
        BSPtraverseTreeAndRender(bspNode->node->negativeSide,position);

    }
}
else assert(bspNode->kind == IN_NODE || bspNode->kind == OUT_NODE);
} /* BSPtraverseTreeAndRender() */

/* Frees a BSP tree and sets pointer to it to null
 *
 * bspNode - a pointer to a node in BSP tree, set to null upon exit
 */
void BSPfreeTree(BSPNODE **bspNode)
{
    if (*bspNode == NULL_BSPNODE) return;

    if ((*bspNode)->kind == PARTITION_NODE)
        freePartitionNode(&(*bspNode)->node);

    MYFREE((char *) *bspNode); *bspNode= NULL_BSPNODE;
} /* BSPfreeTree() */

/* Chooses plane with which to partition.
 * The algorithm is to examine the first MAX_CANDIDATES on face list. For
 * each candidate, count how many splits it would make against the scene.
 * Then return the one with the minimum amount of splits as the
 * partitioning plane.
 *
 * faceList - list of faces
 * plane    - plane equation returned
 */
static void BSPchoosePlane(FACE *faceList,PLANE *plane)
{
    FACE *rootrav; int ii;
    int minCount= MAXINT;
    FACE *chosenRoot= faceList; /* pick first face for now */

    assert(faceList != NULL_FACE);
    /* for all candidates... */
#define MAX_CANDIDATES 100
    for (rootrav= faceList, ii= 0; rootrav != NULL_FACE && ii< MAX_CANDIDATES;
        rootrav= rootrav->fnext, ii++) {
        FACE *ftrav; int count= 0;
        /* for all faces in scene other than itself... */
        for (ftrav= faceList; ftrav != NULL_FACE; ftrav= ftrav->fnext) {
            if (ftrav != rootrav)
                if (doesFaceStraddlePlane(ftrav,&rootrav->plane)) count++;
        }
        /* remember minimum count and its corresponding face */
        if (count < minCount) { minCount= count; chosenRoot= rootrav; }
        if (count == 0) break; /* can't do better than 0 so return this plane */
    }
    *plane= chosenRoot->plane; /* return partitioning plane */
} /* BSPchoosePlane() */
```



```
/* Returns a boolean to indicate whether the face straddles the plane
 *
 * face - face to check
 * plane - plane
 */
static boolean doesFaceStraddlePlane(const FACE *face, const PLANE *plane)
{
    boolean anyNegative= 0, anyPositive= 0;
    VERTEX *vtrav;

    assert(face->vhead != NULL_VERTEX);
    /* for all vertices... */
    for (vtrav= face->vhead; vtrav->vnnext !=NULL_VERTEX; vtrav= vtrav->vnnext) {
        float value= plane->aa*vtrav->xx + plane->bb*vtrav->yy +
                    plane->cc*vtrav->zz + plane->dd;
        /* check which side vertex is on relative to plane */
        SIGN sign= FSIGN(value);
        if (sign == NEGATIVE) anyNegative= 1;
        else if (sign == POSITIVE) anyPositive= 1;

        /* if vertices on both sides of plane then face straddles else it no */
        if (anyNegative && anyPositive) return(1);
    }
    return(0);
} /* doesFaceStraddlePlane() */

/* Returns a boolean to indicate whether or not point is in + side of plane.
 *
 * plane - plane
 * position - position of point
 */
boolean BSPisViewerInPositiveSideOfPlane(const PLANE *plane,const POINT *position)
{
    float dp= plane->aa*position->xx + plane->bb*position->yy +
            plane->cc*position->zz + plane->dd;
    return( (dp > 0.0) ? 1 : 0 );
} /* BSPisViewerInPositiveSideOfPlane() */

/* Allocates a BSP node.
 *
 * kind - type of BSP node
 * sameDir, oppDir - list of faces to be embedded in this node
 */
static BSPNODE *allocBspNode(NODE_TYPE kind,FACE *sameDir,FACE *oppDir)
{
    BSPNODE *newBspNode;
    if ((newBspNode= (BSPNODE *) MYMALLOC(sizeof(BSPNODE))) == NULL_BSPNODE) {
        fprintf(stderr,"?Unable to malloc bspnode.\n");
        exit(1);
    }
    newBspNode->kind= kind;

    if (newBspNode->kind == PARTITION_NODE)
        newBspNode->node= allocPartitionNode(sameDir,oppDir);
    else { assert(kind == IN_NODE || kind == OUT_NODE);
        assert(sameDir == NULL_FACE && oppDir == NULL_FACE);
        newBspNode->node= NULL_PARTITIONNODE;
    }
    return(newBspNode);
} /* allocBspNode() */
```

```
/* Allocates a partition node.
 *
 * sameDir, oppDir - list of faces embedded in partition node
 */
static PARTITIONNODE *allocPartitionNode(FACE *sameDir,FACE *oppDir)
{
    PARTITIONNODE *newPartitionNode;
    if ((newPartitionNode= (PARTITIONNODE *) MYMALLOC(sizeof(PARTITIONNODE)))==
        NULL_PARTITIONNODE) {
        fprintf(stderr,"?Unable to malloc partitionnode.\n");
        exit(1);
    }
    newPartitionNode->sameDir= sameDir; newPartitionNode->oppDir= oppDir;
    newPartitionNode->negativeSide= NULL_BSPNODE;
    newPartitionNode->positiveSide= NULL_BSPNODE;

    return(newPartitionNode);
} /* allocPartitionNode() */

/* Frees partition node and sets pointer to it to null.
 *
 * partitonNode - partition node to be freed, pointer is set to null
 */
static void freePartitionNode(PARTITIONNODE **partitionNode)
{
    freeFaceList(&(*partitionNode)->sameDir);
    freeFaceList(&(*partitionNode)->oppDir);
    BSPfreeTree(&(*partitionNode)->negativeSide);
    BSPfreeTree(&(*partitionNode)->positiveSide);

    MYFREE((char *) *partitionNode); *partitionNode= NULL_PARTITIONNODE;
} /* freePartitionNode() */

/* Dumps information on faces. This should be replaced with user-supplied
 * polygon scan converter.
 */
void drawFaceList(FILE *fp,const FACE *faceList)
{
    const FACE *ftrav;
    for (ftrav= faceList; ftrav != NULL_FACE; ftrav= ftrav->fnext) {
        VERTEX *vtrav;

        fprintf(fp,"Face: RGBi:%.2f/%.2f/%.2f a: %.3f b: %.3f c: %.3f d: %.3f ",
            ftrav->color.rr,ftrav->color.gg,ftrav->color.bb,
            ftrav->plane.aa,ftrav->plane.bb,ftrav->plane.cc,ftrav->plane.dd);
        fprintf(fp,"\n");
        for (vtrav= ftrav->vhead; vtrav->vnnext != NULL_VERTEX;
            vtrav= vtrav->vnnext) {
            fprintf(fp,"\t(%.3f,%.3f,%.3f) ",vtrav->xx,vtrav->yy,vtrav->zz);
            fprintf(fp,"\n");
        }
    }
} /* drawFaceList() */
/**** bspTree.c ****/
```

```
/* bspUtility.c: module to compute plane equation, normalize a vector and
 * perform cross products.
 * Copyright (c) Norman Chin
 */
#include "bsp.h"

/* local functions */
static void normalizeVector(float ii,float jj,float kk,
                           float *ii2,float *jj2,float *kk2);
static void crossProduct(float ii,float jj,float kk,
                        float ii2,float jj2,float kk2,
                        float *iicp,float *jjcp,float *kkcp);

/* Computes plane equation.
 *
 * xx0,yy0,zz0, xx1,yy1,zz1, xx2,yy2,zz2 - 3 non-collinear vertices
 * plane - plane equation returned
 */
void computePlane(float xx0,float yy0,float zz0,float xx1,float yy1,float zz1,
                 float xx2,float yy2,float zz2, PLANE *plane)
{
    float ii1= xx1 - xx0; float jj1= yy1 - yy0; float kk1= zz1 - zz0;
    float ii2= xx2 - xx0; float jj2= yy2 - yy0; float kk2= zz2 - zz0;
    float iicp, jjcp, kkcp;

    crossProduct(ii1,jj1,kk1,ii2,jj2,kk2,&iicp,&jjcp,&kkcp);
    assert(!(IS_EQ(iicp,0.0) && IS_EQ(jjcp,0.0) && IS_EQ(kkcp,0.0)));

    /* normalize plane equation */
    normalizeVector(iicp,jjcp,kkcp,&plane->aa,&plane->bb,&plane->cc);

    /* compute D of plane equation */
    plane->dd= - (plane->aa * xx0) - (plane->bb * yy0) - (plane->cc * zz0);
} /* computePlane() */

/* Performs cross product.
 *
 * ii1,jj1,kk1, ii2,jj2,kk2 - two vectors
 * iicp,jjcp,kkcp - cross product
 */
static void crossProduct(float ii1,float jj1,float kk1,
                        float ii2,float jj2,float kk2,
                        float *iicp,float *jjcp,float *kkcp)
{
    *iicp= jj1*kk2 - jj2*kk1; *jjcp= ii2*kk1 - ii1*kk2; *kkcp= ii1*jj2 - ii2*jj1;
} /* crossProduct() */

/* Normalize a vector.
 *
 * ii,jj,kk - vector to be normalized
 * ii2,jj2,kk2 - vector normalized
 */
static void normalizeVector(float ii,float jj,float kk,
                           float *ii2,float *jj2,float *kk2)
{
    double magnitude= sqrt((double)ii*ii + (double)jj*jj + (double)kk*kk);
    double dfactor= 1.0 / magnitude;

    *ii2= (float) (ii * dfactor);
    *jj2= (float) (jj * dfactor);
    *kk2= (float) (kk * dfactor);
}
```

```
} /* normalizeVector() */  
/*** bspUtility.c ***/  

```

```
;this is a comment
f 0.0 1.0 0.0
v 2.250000 0.000000 -2.000000
v 4.250000 0.000000 -2.000000
v 4.250000 7.000000 -2.000000
v 2.250000 7.000000 -2.000000
;
f 0.0 1.0 0.0
v 4.250000 0.000000 -2.000000
v 4.250000 0.000000 -4.000000
v 4.250000 7.000000 -4.000000
v 4.250000 7.000000 -2.000000
;
f 0.0 1.0 0.0
v 4.250000 0.000000 -4.000000
v 2.250000 0.000000 -4.000000
v 2.250000 7.000000 -4.000000
v 4.250000 7.000000 -4.000000
;
f 0.0 1.0 0.0
v 2.250000 0.000000 -4.000000
v 2.250000 0.000000 -2.000000
v 2.250000 7.000000 -2.000000
v 2.250000 7.000000 -4.000000
;
f 0.0 1.0 0.0
v 2.250000 7.000000 -2.000000
v 4.250000 7.000000 -2.000000
v 4.250000 7.000000 -4.000000
v 2.250000 7.000000 -4.000000
;white floor
f 1.0 1.0 1.0
v -12.000000 0.000000 9.000000
v 9.000000 0.000000 9.000000
v 9.000000 0.000000 -9.000000
v -12.000000 0.000000 -9.000000
```

```
/* mainBsp.c: main driver of BSP application
 * Copyright (c) Norman Chin
 */
#include "bsp.h"

#define MAXBUFFER 80
#define NOBLOCK 'n'
#define COMMENT ';'

/* local functions */
int main(int argc, char *argv[]);
void getScene(const char *fileName, POINT *position, FACE **faceList);
#define dumpPosition(p) (printf("Position: (%f,%f,%f)\n", p.xx, p.yy, p.zz))

/* Main driver */
int main(int argc, char *argv[])
{
    FACE *faceList; POINT oldPosition;
    if (argc != 2) {fprintf(stderr, "Usage: %s <datefile>\n", argv[0]); exit(1);}

    getScene(argv[1], &oldPosition, &faceList); /* get list of faces from file */
    drawFaceList(stdout, faceList); /* dump faces */
    dumpPosition(oldPosition); /* dump viewer position */

    {
        BSPNODE *root= BSPconstructTree(&faceList); /* construct BSP tree */

        BSPtraverseTreeAndRender(root, &oldPosition); /* traverse and render it */

        BSPfreeTree(&root); /* free it */
    }
    freeFaceList(&faceList); /* free list of faces read in */

    return(0);
} /* main() */

/* Reads from fileName a list of convex planar faces oriented counter-clockwise
 * of at least 3 vertices each where the first three vertices are not
 * collinear. The last vertex of each face will automatically be duplicated.
 * This list of faces is returned.
 */
void getScene(const char *fileName, POINT *position, FACE **fList)
{
    FACE *fListTail= NULL_FACE;
    VERTEX *vList= NULL_VERTEX, *vListTail= NULL_VERTEX;
    static char buffer[MAXBUFFER];
    COLOR color; PLANE plane;

    FILE *fp= fopen(fileName, "r");
    if (fp == NULL) {fprintf(stderr, "Unable to open %s\n", fileName); exit(1);}
    printf("File: %s\n", fileName);

    position->xx= 0.0; position->yy= 5.0; position->zz= 10.0;
    *fList= NULL_FACE;

    while (fgets(buffer, MAXBUFFER, fp)) {
        if (buffer[0] == 'f') { /* start of face */
            if (vList != NULL_VERTEX) { /* previous face? */
                appendVertex(&vList, &vListTail, /* append duplicate 1st vertex */
                    allocVertex(vList->xx, vList->yy, vList->zz));
            }
        }
    }
}
```






```
        computePlane(vList->xx,vList->yy,vList->zz,
                    vList->vnext->xx,vList->vnext->yy,vList->vnext->zz,
                    vList->vnext->vnext->xx,
                    vList->vnext->vnext->yy,
                    vList->vnext->vnext->zz,&plane);
    appendFace(fList,&fListTail,allocFace(&color,vList,&plane));
}

    sscanf(buffer,"%*c %f %f %f",&color.rr,&color.gg,&color.bb);
    /* save vars for this face and start new vertex list */
    /* printf("f %f/%f/%f",color.rr,color.gg,color.bb); */
    vList= vListTail= NULL_VERTEX;
}
else if (buffer[0] == 'v') { /* read in vertex */
    float xx,yy,zz; sscanf(buffer,"%*c %f %f %f",&xx,&yy,&zz);
    /* printf("v (%f,%f,%f)\n",xx,yy,zz); */
    appendVertex(&vList,&vListTail,allocVertex(xx,yy,zz));
}
else if (buffer[0] == COMMENT) { } /* comment */
else fprintf(stderr,"?Illegal command: '%c'\n",buffer[0]);
} /* while */
if (vList != NULL_VERTEX) { /* process last face (or only) */
    appendVertex(&vList,&vListTail, /* append duplicate 1st vertex */
                allocVertex(vList->xx,vList->yy,vList->zz));
    computePlane(vList->xx,vList->yy,vList->zz,vList->vnext->xx,
                vList->vnext->yy,vList->vnext->zz,vList->vnext->vnext->xx,
                vList->vnext->vnext->yy,vList->vnext->vnext->zz,&plane);
    appendFace(fList,&fListTail,allocFace(&color,vList,&plane));
}

    fclose(fp);
} /* getScene() */
/**** mainBsp.c ****/
```

# Index of

## /pubs/tog/GraphicsGems/gemsiv/multi\_jitter/




















Name	Last modified	Size	Description
 <a href="#">Parent Directory</a>			
 <a href="#">Makefile</a>	29-Jun-00 08:20	1K	
 <a href="#">README</a>	29-Jun-00 08:20	1K	
 <a href="#">multi.c</a>	29-Jun-00 08:20	1K	
 <a href="#">test.c</a>	29-Jun-00 08:20	1K	



# Index of

## /pubs/tog/GraphicsGems/gemsiv/

Name	Last modified	Size	Description
 <a href="#">Parent Directory</a>			
 <a href="#">AllGems.TOC</a>	29-Jun-00 08:21	26K	
 <a href="#">Errata.GraphicsGemsIV</a>	29-Jun-00 08:21	1K	
 <a href="#">GraphicsGems.c</a>	29-Jun-00 08:21	10K	
 <a href="#">GraphicsGems.h</a>	29-Jun-00 08:21	3K	
 <a href="#">README</a>	29-Jun-00 08:21	4K	
 <a href="#">arcball/</a>	29-Jun-00 08:19	1K	
 <a href="#">centroid.c</a>	29-Jun-00 08:21	1K	
 <a href="#">clahe.c</a>	29-Jun-00 08:21	13K	
 <a href="#">collide.c</a>	29-Jun-00 08:21	28K	
 <a href="#">convex_test/</a>	29-Jun-00 08:19	1K	
 <a href="#">convolve.c</a>	29-Jun-00 08:21	10K	
 <a href="#">coons_warp.c</a>	29-Jun-00 08:21	7K	
 <a href="#">curve_isect/</a>	29-Jun-00 08:19	1K	
 <a href="#">data_smooth/</a>	29-Jun-00 08:19	1K	
 <a href="#">delaunay/</a>	29-Jun-00 08:19	1K	
 <a href="#">dist_fast.c</a>	29-Jun-00 08:21	1K	
 <a href="#">dyn_range/</a>	29-Jun-00 08:19	1K	
 <a href="#">emboss.c</a>	29-Jun-00 08:21	4K	
 <a href="#">euler_angle/</a>	29-Jun-00 08:19	1K	
 <a href="#">gemsiv.bib</a>	29-Jun-00 08:21	28K	
 <a href="#">graph_layout/</a>	29-Jun-00 08:20	1K	

 <a href="#">_implicit.c</a>	29-Jun-00 08:21	23K
 <a href="#">_interp_fast.c</a>	29-Jun-00 08:21	1K
 <a href="#">_inv_fast.c</a>	29-Jun-00 08:21	5K
 <a href="#">_minray/</a>	29-Jun-00 08:20	1K
 <a href="#">_mrsfoley.im</a>	29-Jun-00 08:21	122K
 <a href="#">_multi_jitter/</a>	29-Jun-00 08:20	1K
 <a href="#">_nurb_polyg/</a>	29-Jun-00 08:20	1K
 <a href="#">_outcode/</a>	29-Jun-00 08:20	1K
 <a href="#">_patch_conv.C</a>	29-Jun-00 08:21	8K
 <a href="#">_polar_decomp/</a>	29-Jun-00 08:20	1K
 <a href="#">_ptpoly_haines/</a>	29-Jun-00 08:20	1K
 <a href="#">_ptpoly_weiler/</a>	29-Jun-00 08:20	1K
 <a href="#">_ray_cyl.c</a>	29-Jun-00 08:21	6K
 <a href="#">_sph_poly.c</a>	29-Jun-00 08:21	2K
 <a href="#">_thin_image.c</a>	29-Jun-00 08:21	4K
 <a href="#">_trilerp.c</a>	29-Jun-00 08:21	2K
 <a href="#">_vec_mat/</a>	29-Jun-00 08:21	1K
 <a href="#">_vert_norm/</a>	29-Jun-00 08:21	1K
 <a href="#">_vox_traverse.c</a>	29-Jun-00 08:21	1K

multi:

```
gcc -o multi test.c multi.c -lm
```

C code from the article

"Multi-Jittered Sampling"

by Kenneth Chiu, Peter Shirley, and Changyaw Wang,

(chiuk@cs.indiana.edu, shirley@iuvax.cs.indiana.edu,  
and wangc@iuvax.cs.indiana.edu)

in "Graphics Gems IV", Academic Press, 1994

files:

multi.c - source file

Makefile - make file; creates executable called "multi"

test.c - test program that performs consistency checks on  
the generated sampling pattern

The makefile uses 'gcc' as the compiler, but any ANSI compatible compiler should do.

To run the test program, give it three arguments. The first two are 'm' and 'n' (as described in the gem), and the last one is a seed to the random number generator.

Kenneth Chiu

```
#include <math.h>

#define RAN_DOUBLE(l, h)      (((double) random())/0x80000000U)*((h) - (l)) + (l))
#define RAN_INT(l, h)        ((int) (RAN_DOUBLE(0, (h)-(l)+1) + (l)))

typedef struct {
    double x, y;
} Point2;

unsigned long random();      /* expected to return a random int in [0, 2^31-1] */

/*
 * MultiJitter() takes an array of Point2's and the dimension, and fills the
 * the array with the generated sample points.
 *
 * p[] must have length m*n.
 * m is the number of columns of cells.
 * n is the number of rows of cells.
 */
void
MultiJitter(Point2 p[], int m, int n) {

    double subcell_width;
    int i, j;

    subcell_width = 1.0/(m*n);

    /* Initialize points to the "canonical" multi-jittered pattern. */
    for (i = 0; i < m; i++) {
        for (j = 0; j < n; j++) {
            p[i*n + j].x = i*n*subcell_width + j*subcell_width
                + RAN_DOUBLE(0, subcell_width);
            p[i*n + j].y = j*m*subcell_width + i*subcell_width
                + RAN_DOUBLE(0, subcell_width);
        }
    }

    /* Shuffle x coordinates within each column of cells. */
    for (i = 0; i < m; i++) {
        for (j = 0; j < n; j++) {

            double t;
            int k;

            k = RAN_INT(j, n - 1);
            t = p[i*n + j].x;
            p[i*n + j].x = p[i*n + k].x;
            p[i*n + k].x = t;
        }
    }

    /* Shuffle y coordinates within each row of cells. */
    for (i = 0; i < n; i++) {
        for (j = 0; j < m; j++) {

            double t;
            int k;
```

```
k = RAN_INT(j, m - 1);  
t = p[j*n + i].y;  
p[j*n + i].y = p[k*n + i].y;  
p[k*n + i].y = t;
```

```
}
```

```
}
```

```
}
```

```
/* TEST FILE FOR kc.multi.c, NOT FOR BOOK */
```

```
#include <stdio.h>
```

```
typedef struct {  
    double x, y;  
} Point2;
```

```
int srandom(int seed);
```

```
void MultiJitter(Point2 points[], int m, int n);
```

```
main(int argc, char *argv[]) {
```

```
    Point2 *points;  
    int m, n, i, x, y;  
    int **counts, *x_bins, *y_bins;
```

```
    if (argc != 4) {  
        fprintf(stderr, "Usage: multi-jitter <m> <n> <seed>.\n");  
        exit(1);  
    }
```

```
    m = atoi(argv[1]);  
    n = atoi(argv[2]);  
    srandom(atoi(argv[3]));
```

```
    points = (Point2 *) malloc(m*n*sizeof(Point2));  
    counts = (int **) malloc(m*sizeof(int *));  
    for (i = 0; i < m; i++)  
        counts[i] = (int *) malloc(n*sizeof(int));  
    x_bins = (int *) malloc(m*n*sizeof(int));  
    y_bins = (int *) malloc(m*n*sizeof(int));
```

```
    MultiJitter(points, m, n);
```

```
    /*  
     * Test jittered condition  
     */
```

```
    for (x = 0; x < m; x++) {  
        for (y = 0; y < n; y++) {  
            counts[x][y] = 0;  
        }  
    }
```

```
    for (i = 0; i < m*n; i++) {  
        x = points[i].x / (1.0/m);  
        y = points[i].y / (1.0/n);  
        counts[x][y]++;  
    }
```

```
    for (x = 0; x < m; x++) {
```

```
    for (y = 0; y < n; y++) {
        if (counts[x][y] != 1) {
            fprintf(stderr, "Jittered condition not satisfied.\n");
            goto done1;
        }
    }
done1:;

/*
 * Test n-rooks condition
 */

for (x = 0; x < m*n; x++)
    x_bins[x] = 0;
for (y = 0; y < m*n; y++)
    y_bins[y] = 0;

for (i = 0; i < m*n; i++) {
    x = points[i].x / (1.0/(m*n));
    y = points[i].y / (1.0/(m*n));
    x_bins[x]++;
    y_bins[y]++;
}

for (x = 0; x < m*n; x++) {
    if (x_bins[x] != 1) {
        fprintf(stderr, "n-rooks condition not satisfied in x.\n");
        goto done2;
    }
}
done2:;

for (y = 0; y < m*n; y++) {
    if (y_bins[y] != 1) {
        fprintf(stderr, "n-rooks condition not satisfied in y.\n");
        goto done3;
    }
}
done3:;

for (i = 0; i < m*n; i++)
    fprintf(stderr, "(%f, %f)\n", points[i].x, points[i].y);
}
```



```
/*
 * C code from the article
 * "Voxel Traversal along a 3D Line"
 * by Daniel Cohen, danny@bengus.bgu.ac.il
 * in "Graphics Gems IV", Academic Press, 1994
 */



/* The following C subroutine visits all voxels along the line
segment from (x, y, z) and (x + dx, y + dy, z + dz) */

Line ( x, y, z, dx, dy, dz )
int x, y, z, dx, dy, dz;
{
    int n, sx, sy, sz, exy, exz, ezy, ax, ay, az, bx, by, bz;

    sx = sgn(dx);  sy = sgn(dy);  sz = sgn(dz);
    ax = abs(dx);  ay = abs(dy);  az = abs(dz);
    bx = 2*ax;     by = 2*ay;     bz = 2*az;
    exy = ay-ax;   exz = az-ax;   ezy = ay-az;
    n = ax+ay+az;
    while ( n-- ) {
        VisitVoxel ( x, y, z );
        if ( exy < 0 ) {
            if ( exz < 0 ) {
                x += sx;
                exy += by; exz += bz;
            }
            else {
                z += sz;
                exz -= bx; ezy += by;
            }
        }
        else {
            if ( ezy < 0 ) {
                z += sz;
                exz -= bx; ezy += by;
            }
            else {
                y += sy;
                exy -= bx; ezy -= bz;
            }
        }
    }
}
```

# Index of

## /pubs/tog/GraphicsGems/gemsv/ch6-7/

Name	Last modified	Size	Description
 <a href="#">_Parent Directory</a>			
 <a href="#">_sampat.c</a>	29-Jun-00 08:24	1K	

```
/* Example sample patterns appearing in Graphics Gems V
   ``Sampling Patterns Optimized for Uniform Distribution of Edges''
   Figures 3 and 4. */
```

```
typedef float sample[2];
```

```
sample foursamples[3][4]={
    {{0.274942,  0.884325},    {0.797099,  0.207128},
     {0.765063,  0.715779},    {0.122774,  0.282759}},

    {{0.152302,  0.657716},    {0.649413,  0.907929},
     {0.305133,  0.221223},    {0.784722,  0.280605}},

    {{0.775219,  0.152203},    {0.846312,  0.737633},
     {0.247618,  0.777035},    {0.228821,  0.197385}}};
```

```
sample sixteensamples[3][16]={
    {{0.755279,  0.0497319},    {0.384479,  0.688268},
     {0.666094,  0.868388},    {0.317172,  0.0331764},
     {0.729309,  0.43103},     {0.0867931, 0.368519},
     {0.322668,  1.0},         {0.442302,  0.572752},
     {0.889074,  0.606985},    {0.0343768, 0.191404},
     {0.910321,  0.872547},    {0.92479,   0.345332},
     {0.289126,  0.389783},    {0.896551,  0.141167},
     {0.23357,   0.678942},    {0.11281,   0.526939}},

    {{0.740161,  0.0942363},    {0.384479,  0.688268},
     {0.642662,  0.884825},    {0.324146,  0.0213393},
     {0.729309,  0.43103},     {0.0867931, 0.368519},
     {0.306925,  0.995787},    {0.442302,  0.572752},
     {0.889074,  0.606985},    {0.0343768, 0.191404},
     {0.910321,  0.872547},    {0.92479,   0.345332},
     {0.299325,  0.371848},    {0.896551,  0.141167},
     {0.226811,  0.658172},    {0.27796,   0.873217}},

    {{0.73534,   0.316016},    {0.755279,  0.0497319},
     {0.152649,  0.442638},    {0.917626,  0.771549},
     {0.0492709, 0.836601},    {0.0642901, 0.155284},
     {0.94238,   0.458705},    {0.392657,  0.644079},
     {0.626425,  0.534164},    {0.0918845, 0.468493},
     {0.372743,  0.0552449},   {0.217678,  0.319869},
     {0.460074,  0.759592},    {0.827202,  0.875453},
     {0.596844,  0.352386},    {0.387125,  0.96096}}};
```

```
/*
3D Viewing and Rotation Using Orthonormal Bases
by Steve Cunningham
from "Graphics Gems", Academic Press, 1990
*/

/*
 * Transformations are presented as 4 by 3 matrices, omitting the
 * fourth column to save memory.
 *
 * Functions are used from the Graphics Gems vector C library
 */

#include "GraphicsGems.h"

typedef float Transform[4][3];

void BuildViewTransform( VRP, EP, UP, T )
    Point3 VRP, EP, UP;
    Transform T;
{
    Vector3 U, V, N;
    float dot;

    /*
     * Compute vector N = EP - VRP and normalize N
     */
    N.x = EP.x - VRP.x; N.y = EP.y - VRP.y; N.z = EP.z - VRP.z;
    V3Normalize(&N);

    /*
     * Compute vector V = UP - VRP
     * Make vector V orthogonal to N and normalize V
     */
    V.x = UP.x - VRP.x; V.y = UP.y - VRP.y; V.z = UP.z - VRP.z;
    dot = V3Dot(&V,&N);
    V.x -= dot * N.x; V.y -= dot * N.y; V.z -= dot * N.z;
    V3Normalize(&V);

    /*
     * Compute vector U = V x N (cross product)
     */
    V3Cross(&V,&N,&U);

    /*
     * Write the vectors U, V, and N as the first three rows of the
     * first, second, and third columns of T, respectively
     */
    T[0][0] = U.x;          /* column 1 , vector U */
    T[1][0] = U.y;
    T[2][0] = U.z;
    T[0][1] = V.x;          /* column 2 , vector V */
    T[1][1] = V.y;
    T[2][1] = V.z;
    T[0][2] = N.x;          /* column 3 , vector N */
    T[1][2] = N.y;
    T[2][2] = N.z;

    /*
```

```
    * Compute the fourth row of T to include the translation of
    *      VRP to the origin
    */
T[3][0] = - U.x * VRP.x - U.y * VRP.y - U.z * VRP.z;
T[3][1] = - V.x * VRP.x - V.y * VRP.y - V.z * VRP.z;
T[3][2] = - N.x * VRP.x - N.y * VRP.y - N.z * VRP.z;

return;
```

```
}
```

```
/*
 * Efficient Generation of Sampling Jitter Using Look-up Tables
 * by Joseph M. Cychosz
 * from "Graphics Gems", Academic Press, 1990
 */

/*
    Jitter.c - Sampling jitter generation routines.
*/
/*
    Description:
    Jitter.c contains the routines for generation of sampling
    jitter using look-up tables.
*/
/*
    Contents:
    Jitter1      Generate random jitter function 1.
    Jitter2      Generate random jitter function 2.
    JitterInit   Initialize look-up tables.
*/ */

#define          NRAN      1024      /* Random number table length */

static double   URANX[NRAN],          /* Random number tables */
                URANY[NRAN];
static int      IRAND[NRAN];          /* Shuffle table */
static int      MASK = NRAN-1;        /* Mask for jitter mod function */
extern double   xranf();              /* Random number generator pro-
                                     /* ducing uniform numbers 0 to 1 */

/*
    Jitter1 - Generate random jitter.
*/

void Jitter1 (x,y,s,xj,yj)
    int      x, y;          /* Pixel location */
    int      s;             /* Sample number for the pixel */
    double   *xj, *yj;      /* Jitter (x,y) */
{
    *xj = URANX[ (x + (y<<2) + IRAND[(x+s)&MASK]) & MASK ];
    *yj = URANY[ (y + (x<<2) + IRAND[(y+s)&MASK]) & MASK ];
}

/*
    Jitter2 - Generate random jitter.
*/

void Jitter2 (x,y,s,xj,yj)
    int      x, y;          /* Pixel location */
    int      s;             /* Sample number for the pixel */
    double   *xj, *yj;      /* Jitter (x,y) */
{
    *xj = URANX[ ((x | (y<<2)) + IRAND[(x+s)&MASK]) & MASK ];
    *yj = URANY[ ((y | (x<<2)) + IRAND[(y+s)&MASK]) & MASK ];
}

/*
    JitterInit - Initialize look-up tables.
*/

void JitterInit      ()
{
    int      i;

    for ( i = 0 ; i < NRAN ; i++ ) URANX[i] = xranf();
    for ( i = 0 ; i < NRAN ; i++ ) URANY[i] = xranf();
    for ( i = 0 ; i < NRAN ; i++ ) IRAND[i] = (int) (NRAN *

```

```
xrandf ( ) ;
```

```
}
```

```
/*
Efficient Post-Concatenation of Transformation Matrices
by Joseph M. Cychosz
from "Graphics Gems", Academic Press, 1990
*/

#include      <math.h>
#include      "GraphicsGems.h"

/*
M4xform.c - Basic 4x4 transformation package.
*
*
* Description:
*
*   M4xform.c contains a collection of routines used to perform
*   direct post-concatenated transformation operations.
*
*
* Contents:
*
*   M4RotateX      Post-concatenate a x-axis rotation.
*   M4RotateY      Post-concatenate a y-axis rotation.
*   M4RotateZ      Post-concatenate a z-axis rotation.
*   M4Scale        Post-concatenate a scaling.
*   M4Translate    Post-concatenate a translation.
*   M4ZPerspective Post-concatenate a z-axis perspective
*                   transformation.
*
*
* Externals:
*
*   cos, sin.
*/

/*
M4RotateX - Post-concatenate a x-axis rotation matrix.  */

Matrix4 *M4RotateX      (m,a)
    Matrix4 *m;           /* Current transformation matrix*/
    double  a;            /* Rotation angle          */
{
    double  c, s;
    double  t;
    int     i;

    c = cos (a);    s = sin (a);

    for (i = 0 ; i < 4 ; i++) {
        t = m->element[i][1];
        m->element[i][1] = t*c - m->element[i][2]*s;
        m->element[i][2] = t*s + m->element[i][2]*c;
    }
    return (m);
}

/*
M4RotateY - Post-concatenate a y-axis rotation matrix.  */

Matrix4 *M4RotateY      (m,a)
    Matrix4 *m;           /* Current transformation matrix*/
    double  a;            /* Rotation angle          */
{
    double  c, s;
```



```
double t;
int i;

c = cos (a);    s = sin (a);

for (i = 0 ; i < 4 ; i++) {
    t = m->element[i][0];
    m->element[i][0] = t*c + m->element[i][2]*s;
    m->element[i][2] = m->element[i][2]*c - t*s;
}
return (m);
}
```

/\* M4RotateZ - Post-concatenate a z-axis rotation matrix. \*/

```
Matrix4 *M4RotateZ      (m,a)
    Matrix4 *m;           /* Current transformation matrix*/
    double a;             /* Rotation angle              */
{
    double c, s;
    double t;
    int i;

    c = cos (a);    s = sin (a);

    for (i = 0 ; i < 4 ; i++) {
        t = m->element[i][0];
        m->element[i][0] = t*c - m->element[i][1]*s;
        m->element[i][1] = t*s + m->element[i][1]*c;
    }
    return (m);
}
```

/\* M4Scale - Post-concatenate a scaling. \*/

```
Matrix4 *M4Scale      (m,sx,sy,sz)
    Matrix4 *m;           /* Current transformation matrix */
    double sx, sy, sz;    /* Scale factors about x,y,z */
{
    int i;

    for (i = 0 ; i < 4 ; i++) {
        m->element[i][0] *= sx;
        m->element[i][1] *= sy;
        m->element[i][2] *= sz;
    }
    return (m);
}
```

/\* M4Translate - Post-concatenate a translation. \*/

```
Matrix4 *M4Translate   (m,tx,ty,tz)
    Matrix4 *m;           /* Current transformation matrix */
    double tx, ty, tz;    /* Translation distance */
{
    int i;
```

```
    for (i = 0 ; i < 4 ; i++) {
        m->element[i][0] += m->element[i][3]*tx;
        m->element[i][1] += m->element[i][3]*ty;
        m->element[i][2] += m->element[i][3]*tz;
    }
    return (m);
}
```

```
/* M4ZPerspective Post-concatenate a perspective */
```

```
/*transformation.
```

```
*/
```

```
/*
```

```
*/
```

```
/* Perspective is along the z-axis with the eye at +z. */
```

```
Matrix4 *M4ZPerspective (m,d)
```

```
Matrix4 *m;
```

```
/* Current transformation matrix */
```

```
double d;
```

```
/* Perspective distance */
```

```
{
```

```
int i;
```

```
double f = 1. / d;
```

```
for (i = 0 ; i < 4 ; i++) {
```

```
    m->element[i][3] += m->element[i][2]*f;
```

```
    m->element[i][2] = 0.;
```

```
}
```

```
return (m);
```

```
}
```

```

#include      <math.h>
#include      "GraphicsGems.h"

/* ---- inttor - Intersect a ray with a torus. ----- */
/*
/*
/*
/* Description:
/*   Inttor determines the intersection of a ray with a torus.
/*
/*
/* On entry:
/*   raybase = The coordinate defining the base of the
/*             intersecting ray.
/*   raycos  = The direction cosines of the above ray.
/*   center  = The center location of the torus.
/*   radius  = The major radius of the torus.
/*   rplane  = The minor radius in the plane of the torus.
/*   rnorm   = The minor radius normal to the plane of the torus.
/*   tran    = A 4x4 transformation matrix that will position
/*             the torus at the origin and orient it such that
/*             the plane of the torus lyes in the x-z plane.
/*
/* On return:
/*   nhits   = The number of intersections the ray makes with
/*             the torus.
/*   rhits   = The entering/leaving distances of the
/*             intersections.
/*
/* Returns:  True if the ray intersects the torus.
/*
/* ----- */

int      inttor  (raybase,raycos,center,radius,rplane,rnorm,tran,nhits,rhits)

    Point3  raybase;          /* Base of the intersection ray */
    Vector3 raycos;           /* Direction cosines of the ray */
    Point3  center;           /* Center of the torus          */
    double  radius;           /* Major radius of the torus    */
    double  rplane;           /* Minor planer radius          */
    double  rnorm;            /* Minor normal radius          */
    Matrix4 tran;             /* Transformation matrix        */
    int *   nhits;            /* Number of intersections      */
    double  rhits[4];         /* Intersection distances       */

{
    int      hit;              /* True if ray intersects torus */
    double   rsphere;          /* Bounding sphere radius       */
    Vector3  Base, Dcos;       /* Transformed intersection ray */
    double   rmin, rmax;       /* Root bounds                   */
    double   yin, yout;
    double   rho, a0, b0;      /* Related constants            */
    double   f, l, t, g, q, m, u; /* Ray dependent terms         */
    double   C[5];             /* Quartic coefficients          */

extern int  intsph ();         /* Intersect ray with sphere    */
extern int  SolveQuartic ();  /* Solve quartic equation       */

    *nhits = 0;

```

```
/*      Compute the intersection of the ray with a bounding sphere.      */

rsphere = radius + MAX (rplane,rnorm);
hit      = intsph (raybase,raycos,center,rsphere,&rmin,&rmax);

if (!hit) return (hit);          /* If ray misses bounding sphere*/

/*      Transform the intersection ray      */

Base = raybase;
Dcos = raycos;
V3MulPointByMatrix (&Base,&tran);
V3MulVectorByMatrix (&Dcos,&tran);

/*      Bound the torus by two parallel planes rnorm from the x-z plane.*/

yin  = Base.y + rmin * Dcos.y;
yout = Base.y + rmax * Dcos.y;
hit  = !((yin > rnorm && yout > rnorm) ||
         (yin < -rnorm && yout < -rnorm) );

if (!hit) return (hit);          /* If ray is above/below torus. */

/*      Compute constants related to the torus.      */

rho = rplane*rplane / (rnorm*rnorm);
a0  = 4. * radius*radius;
b0  = radius*radius - rplane*rplane;

/*      Compute ray dependent terms.      */

f = 1. - Dcos.y*Dcos.y;
l = 2. * (Base.x*Dcos.x + Base.z*Dcos.z);
t = Base.x*Base.x + Base.z*Base.z;
g = f + rho * Dcos.y*Dcos.y;
q = a0 / (g*g);
m = (l + 2.*rho*Dcos.y*Base.y) / g;
u = (t + rho*Base.y*Base.y + b0) / g;

/*      Compute the coefficients of the quartic.      */

C[4] = 1.0;
C[3] = 2. * m;
C[2] = m*m + 2.*u - q*f;
C[1] = 2.*m*u - q*l;
C[0] = u*u - q*t;

/*      Use quartic root solver found in "Graphics Gems" by Jochen      */
/*      Schwarze.      */

*nhits = SolveQuartic (C,rhits);

/*      SolveQuartic returns root pairs in reversed order.      */

m = rhits[0]; u = rhits[1]; rhits[0] = u; rhits[1] = m;
m = rhits[2]; u = rhits[3]; rhits[2] = u; rhits[3] = m;

return (*nhits != 0);
}
```

```
#include      <math.h>
#include      "GraphicsGems.h"

/* ---- intqdr - Intersect a ray with a quadric surface. ----- */
/*
/*
/*
/* Description:
/*   Intqdr determines the intersection of a ray with a quadric
/*   surface in matrix form.
/*
/*
/* On entry:
/*   raybase = The coordinate defining the base of the
/*             intersecting ray.
/*   raycos  = The direction cosines of the above ray.
/*   base    = The location of the base of the quadric.
/*   axis    = The axis of symmetry for the quadric.
/*   radius  = The bounding radius from the axis of symmetry.
/*   Q       = A 4x4 coefficient matrix representing the quadric
/*             surface.
/*
/*
/* On return:
/*   rin     = The entering distance of the intersection.
/*   rout    = The leaving distance of the intersection.
/*
/*
/* Returns:  True if the ray intersects the quadric surface.
/*
/* ----- */
```

```
boolean intqdr (raybase,raycos,base,axis,radius,Q,rin,rout)

    Point3  raybase;          /* Base of the intersection ray */
    Vector3 raycos;           /* Direction cosines of the ray */
    Vector3 base;             /* Base point for the quadric */
    Vector3 axis;             /* Axis of symmetry */
    double  radius;           /* Bounding radius */
    Matrix4 Q;                /* Quadric coefficient matrix */
    double  *rin;             /* Entering distance */
    double  *rout;            /* Leaving distance */

{
    boolean hit;              /* True if ray intersects quad. */
    double  qa, qb, qc, qd, qe, /* Coefficient matrix terms */
           qf, qg, qh, qi, qj;
    double  k2, k1, k0;       /* Quadric coefficients */
    double  disc;
    Vector3 D;                /* Ray base to quadric base */
    double  d;                /* Shortest distance between
                               /* ray and axis of symmetry */

    Vector3 N;

    /* Compute the perpendicular distance between the symmetry axis
    /* and the ray.

    V3Sub   (&base,&raybase,&D); /* compute shortest distance */
    V3Cross (&raycos,&axis,&N);
    if ((d=V3Length(&N)) == 0) { /* if parallel */
        V3Cross (&raycos,&D,&N);
        d = fabs (V3Length(&N));
```

```
    } else {
        V3Normalize (&N);
        d = fabs (V3Dot(&D,&N)/d);
    }

    hit = (d <= radius);
    if (!hit) return (hit);          /* If outside of cylinder      */

    qa = Q.element[0][0];
    qb = Q.element[0][1] + Q.element[1][0];
    qc = Q.element[0][2] + Q.element[2][0];
    qd = Q.element[0][3] + Q.element[3][0];
    qe = Q.element[1][1];
    qf = Q.element[1][2] + Q.element[2][1];
    qg = Q.element[1][3] + Q.element[3][1];
    qh = Q.element[2][2];
    qi = Q.element[2][3] + Q.element[3][2];
    qj = Q.element[3][3];

    k2 = raycos.x * (raycos.x*qa + raycos.y*qb) +
        raycos.y * (raycos.y*qe + raycos.z*qf) +
        raycos.z * (raycos.z*qh + raycos.z*qc);
    k1 = raycos.x * ((raybase.x*2.*qa + raybase.y*qb + raybase.z*qc) + qd) +
        raycos.y * ((raybase.x*qb + raybase.y*2.*qe + raybase.z*qf) + qg) +
        raycos.z * ((raybase.x*qc + raybase.y*qg + raybase.z*2.*qh) + qi);
    k0 = raybase.x * (raybase.x*qa + raybase.y*qb + raybase.z*qc + qd) +
        raybase.y * (
            raybase.y*qe + raybase.z*qf + qg) +
        raybase.z * (
            raybase.z*qh + qi) +
        qj;

    disc = k1*k1 - 4.*k2*k0;

    hit = (disc >= 0.0);

    if (hit) {
        disc = sqrt(disc);
        *rin = (-k1 - disc) / (2.*k2);    /* entering distance */
        *rout = (-k1 + disc) / (2.*k2);  /* leaving distance  */
    }

    return (hit);
}
```

```
#include      <math.h>
#include      "GraphicsGems.h"

/* ---- intell - Intersect a ray with an ellipsoid. ----- */
/*
/*
/*
/* Description:
/*   Intell determines the intersection of a ray with an
/*   ellipsoid.
/*
/*
/* On entry:
/*   raybase = The coordinate defining the base of the
/*             intersecting ray.
/*   raycos  = The direction cosines of the above ray.
/*   center  = The location of the center of the ellipsoid.
/*   radius  = The radius of a bounding sphere.
/*   Q       = A 4x4 coefficient matrix representing the quadric
/*             surface.
/*
/* On return:
/*   rin     = The entering distance of the intersection.
/*   rout    = The leaving distance of the intersection.
/*
/* Returns:  True if the ray intersects the ellipsoid.
/*
/* ----- */
```

```
boolean intell (raybase,raycos,center,radius,Q,rin,rout)
```

```
    Point3 raybase;          /* Base of the intersection ray */
    Vector3 raycos;          /* Direction cosines of the ray */
    Vector3 center;          /* Base point for the quadric */
    double radius;           /* Bounding radius */
    Matrix4 Q;               /* Quadric coefficient matrix */
    double *rin;             /* Entering distance */
    double *rout;            /* Leaving distance */
```

```
{
    boolean hit;              /* True if ray intersects quad. */
    double qa, qb, qc, qd, qe, /* Coefficient matrix terms */
           qf, qg, qh, qi, qj;
    double k2, k1, k0;        /* Quadric coefficients */
    double disc;
    Vector3 D;                 /* Ray base to ellipsoid base */
    double d;                  /* Shortest distance between
                               /* ray and ellipsoid */
```

```
/* Compute the minimal distance between the ray and the center
/* of the ellipsoid.
```

```
V3Sub (&center,&raybase,&D);
d = V3Length (&D);
V3Normalize (&D);
d = d * sqrt (1. - V3Dot(&D,&raycos));
hit = (d <= radius);
```

```
if (!hit) return (hit);      /* If outside of sphere */
```

```
qa = Q.element[0][0];
qb = Q.element[0][1] + Q.element[1][0];
qc = Q.element[0][2] + Q.element[2][0];
qd = Q.element[0][3] + Q.element[3][0];
qe = Q.element[1][1];
qf = Q.element[1][2] + Q.element[2][1];
qg = Q.element[1][3] + Q.element[3][1];
qh = Q.element[2][2];
qi = Q.element[2][3] + Q.element[3][2];
qj = Q.element[3][3];

k2 = raycos.x * (raycos.x*qa + raycos.y*qb) +
     raycos.y * (raycos.y*qe + raycos.z*qf) +
     raycos.z * (raycos.z*qh + raycos.z*qc);
k1 = raycos.x * ((raybase.x*2.*qa + raybase.y*qb + raybase.z*qc) + qd) +
     raycos.y * ((raybase.x*qb + raybase.y*2.*qe + raybase.z*qf) + qg) +
     raycos.z * ((raybase.x*qc + raybase.y*qg + raybase.z*2.*qh) + qi);
k0 = raybase.x * (raybase.x*qa + raybase.y*qb + raybase.z*qc + qd) +
     raybase.y * (
         raybase.y*qe + raybase.z*qf + qg) +
     raybase.z * (
         raybase.z*qh + qi) +
     qj;

disc = k1*k1 - 4.*k2*k0;

hit = (disc >= 0.0);

if (hit) {
    disc = sqrt(disc);
    *rin = (-k1 - disc) / (2.*k2);
    *rout = (-k1 + disc) / (2.*k2);
}

return (hit);
}
```



```
/*
 * ANSI C code from the article
 * "Intersecting a Ray with a Cylinder"
 * by Joseph M. Cychosz and Warren N. Waggenspack, Jr.,
 * (3ksnn64@ecn.purdue.edu, mewagg@mewnw.dnet.lsu.edu)
 * in "Graphics Gems IV", Academic Press, 1994
 */

#include      "GraphicsGems.h"
#include      <math.h>

/* ---- intcyl - Intersect a ray with a cylinder. ----- */
/*
/*
/*
/* Description:
/*   Intcyl determines the intersection of a ray with a
/*   cylinder.
/*
/*
/* On entry:
/*   raybase = The base point of the intersecting ray.
/*   raycos  = The direction cosines of the above ray. (unit)
/*   base    = The base location of the cylinder.
/*   axis    = The axis of symmetry for the cylinder. (unit)
/*   radius  = The radius of the cylinder.
/*
/*
/* On return:
/*   in      = The entering distance of the intersection.
/*   out     = The leaving distance of the intersection.
/*
/*
/* Returns:  True if the ray intersects the cylinder.
/*
/*
/* Note:    In and/or out may be negative indicating the
/*          cylinder is located behind the origin of the ray.
/*
/* ----- */

#define HUGE          1.0e21          /* Huge value */

boolean intcyl (raybase,raycos,base,axis,radius,in,out)

    Point3      *raybase;          /* Base of the intersection ray */
    Vector3     *raycos;           /* Direction cosines of the ray */
    Point3      *base;             /* Base of the cylinder
    Vector3     *axis;             /* Axis of the cylinder
    double      radius;            /* Radius of the cylinder
    double      *in;               /* Entering distance
    double      *out;              /* Leaving distance

{
    boolean      hit;              /* True if ray intersects cyl
    Vector3      RC;               /* Ray base to cylinder base
    double       d;                /* Shortest distance between
                                /* the ray and the cylinder
    double       t, s;             /* Distances along the ray
    Vector3      n, D, O;
    double       ln;
const double    pinf = HUGE;      /* Positive infinity

    RC.x = raybase->x - base->x;
```

```
RC.y = raybase->y - base->y;
RC.z = raybase->z - base->z;
V3Cross (raycos,axis,&n);

if ( (ln = V3Length (&n)) == 0. ) {      /* ray parallel to cyl */
    d = V3Dot (&RC,axis);
    D.x = RC.x - d*axis->x;
    D.y = RC.y - d*axis->y;
    D.z = RC.z - d*axis->z;
    d = V3Length (&D);
    *in = -pinf;
    *out = pinf;
    return (d <= radius);                /* true if ray is in cyl*/
}

V3Normalize (&n);
d = fabs (V3Dot (&RC,&n));                /* shortest distance */
hit = (d <= radius);

if (hit) {                                /* if ray hits cylinder */
    V3Cross (&RC,axis,&O);
    t = - V3Dot (&O,&n) / ln;
    V3Cross (&n,axis,&O);
    V3Normalize (&O);
    s = fabs (sqrt(radius*radius - d*d) / V3Dot (raycos,&O));
    *in = t - s;                          /* entering distance */
    *out = t + s;                          /* exiting distance */
}

return (hit);
}
```

```
/* ---- clipobj - Clip object with plane pair. ----- */
/*
/*
/*
/* Description:
/* Clipobj clips the supplied infinite object with two
/* (a top and a bottom) bounding planes.
/*
/*
/* On entry:
/* raybase = The base point of the intersecting ray.
/* raycos = The direction cosines of the above ray. (unit)
/* bot = The normal and perpendicular distance of the
/* bottom plane.
/* top = The normal and perpendicular distance of the
/* top plane.
/* objin = The entering distance of the intersection with
/* the object.
/* objout = The exiting distance of the intersection with
/* the object.
/*
/*
/* On return:
/* objin = The entering distance of the intersection.
/* objout = The exiting distance of the intersection.
/* surfin = The identifier for the entering surface.
/* surfout = The identifier for the leaving surface.
/*
/* Returns: True if the ray intersects the bounded object.
/*
/* ----- */
```

```
#define SIDE 0 /* Object surface */
#define BOT 1 /* Bottom end-cap surface */
#define TOP 2 /* Top end-cap surface */

typedef struct { /* Plane: ax + by + cz + d = 0 */
    double a ,b ,c, d;
} Plane;

boolean clipobj (raybase,raycos,bot,top,objin,objout,surfin,surfout)

    Point3 *raybase; /* Base of the intersection ray */
    Vector3 *raycos; /* Direction cosines of the ray */
    Plane *bot; /* Bottom end-cap plane */
    Plane *top; /* Top end-cap plane */
    double *objin; /* Entering distance */
    double *objout; /* Exiting distance */
    int *surfin; /* Entering surface identifier */
    int *surfout; /* Exiting surface identifier */

{
    double dc, dw, t;
    double in, out; /* Object intersection dists. */

    *surfin = *surfout = SIDE;
    in = *objin;
    out = *objout;

/* Intersect the ray with the bottom end-cap plane. */

    dc = bot->a*raycos->x + bot->b*raycos->y + bot->c*raycos->z;
    dw = bot->a*raybase->x + bot->b*raybase->y + bot->c*raybase->z + bot->d;

    if ( dc == 0.0 ) { /* If parallel to bottom plane */
        if ( dw >= 0. ) return (FALSE);
    } else {
        t = - dw / dc;
        if ( dc >= 0.0 ) { /* If far plane */
            if ( t > in && t < out ) { out = t; *surfout = BOT; }
            if ( t < in ) return (FALSE);
        } else { /* If near plane */
            if ( t > in && t < out ) { in = t; *surfin = BOT; }
            if ( t > out ) return (FALSE);
        }
    }
}

/* Intersect the ray with the top end-cap plane. */

    dc = top->a*raycos->x + top->b*raycos->y + top->c*raycos->z;
    dw = top->a*raybase->x + top->b*raybase->y + top->c*raybase->z + top->d;

    if ( dc == 0.0 ) { /* If parallel to top plane */
        if ( dw >= 0. ) return (FALSE);
    } else {
        t = - dw / dc;
        if ( dc >= 0.0 ) { /* If far plane */
            if ( t > in && t < out ) { out = t; *surfout = TOP; }
            if ( t < in ) return (FALSE);
        } else { /* If near plane */
            if ( t > in && t < out ) { in = t; *surfin = TOP; }
        }
    }
}
```

```
        if ( t > out ) return (FALSE);
    }

    *objin  = in;
    *objout = out;
    return (in < out);
}
```

```
/*
 * C code from the article
 * "Efficient Binary Image Thinning using Neighborhood Maps"
 * by Joseph M. Cychosz, 3ksnn64@ecn.purdue.edu
 * in "Graphics Gems IV", Academic Press, 1994
 */

#include <stdio.h>

typedef unsigned char    Pixel;          /* Pixel data type          */

typedef struct {                      /* Image control structure */
    short    Hres;                  /* Horizontal resolution (x) */
    short    Vres;                  /* Vertical resolution (y) */
    int      Size;                  /* Image size (bytes) */
    Pixel    *i;                   /* Image array */
    Pixel    *p[1];                /* Scanline pointer array */
                                /* Pixel (x,y) is given by */
                                /* image->p[y][x] */
} Image;

/* ---- ThinImage - Thin binary image. ----- */
/*
/* Description:
/*     Thins the supplied binary image using Rosenfeld's parallel
/*     thinning algorithm.
/*
/* On Entry:
/*     image = Image to thin.
/*
/* ----- */

                                /* Direction masks: */
                                /*   N     S     W     E */
static int    masks[]        = { 0200, 0002, 0040, 0010 };

/* True if pixel neighbor map indicates the pixel is 8-simple and
/* not an end point and thus can be deleted. The neighborhood
/* map is defined as an integer of bits abcdefghi with a non-zero
/* bit representing a non-zero pixel. The bit assignment for the
/* neighborhood is:
/*
/*           a b c
/*           d e f
/*           g h i
/*
static unsigned char    delete[512] = {
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

```

```
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1};
```

```
void ThinImage (image)

Image          *image;          /* Image control structure */

{

    int         xsize, ysize;    /* Image resolution */
    int         x, y;           /* Pixel location */
    int         i;              /* Pass index */
    int         pc = 0;         /* Pass count */
    int         count = 1;      /* Deleted pixel count */
    int         p, q;          /* Neighborhood maps of adjacent
                                /* cells
    Pixel        *qb;           /* Neighborhood maps of previous
                                /* scanline
    int         m;             /* Deletion direction mask */

    xsize = image->Hres;
    ysize = image->Vres;
    qb = (Pixel *) malloc (xsize*sizeof(Pixel));
    qb[xsize-1] = 0;           /* Used for lower-right pixel */

    while ( count ) {          /* Scan image while deletions */
        pc++;
        count = 0;

        for ( i = 0 ; i < 4 ; i++ ) {

            m = masks[i];

            /* Build initial previous scan buffer. */

            p = image->p[0][0] != 0;
            for ( x = 0 ; x < xsize-1 ; x++ )
                qb[x] = p = ((p<<1)&0006) | (image->p[0][x+1] != 0);

            /* Scan image for pixel deletion candidates. */

            for ( y = 0 ; y < ysize-1 ; y++ ) {

                q = qb[0];
```

```
p = ((q<<3)&0110) | (image->p[y+1][0] != 0);

for ( x = 0 ; x < xsize-1 ; x++ ) {
    q = qb[x];
    p = ((p<<1)&0666) | ((q<<3)&0110) |
        (image->p[y+1][x+1] != 0);
    qb[x] = p;
    if ( (p&m) == 0 && delete[p] ) {
        count++;
        image->p[y][x] = 0;
    }
}

/* Process right edge pixel.                                */

p = (p<<1)&0666;
if ( (p&m) == 0 && delete[p] ) {
    count++;
    image->p[y][xsize-1] = 0;
}

/* Process bottom scan line.                                */

for ( x = 0 ; x < xsize ; x++ ) {
    q = qb[x];
    p = ((p<<1)&0666) | ((q<<3)&0110);
    if ( (p&m) == 0 && delete[p] ) {
        count++;
        image->p[ysize-1][x] = 0;
    }
}

printf ("ThinImage: pass %d, %d pixels deleted\n", pc, count);
}

free (qb);
}
```

```
/*
 * A High Speed, Low Precision Square Root
 * by Paul Lalonde and Robert Dawson
 * from "Graphics Gems", Academic Press, 1990
 */

/*
 * SPARC implementation of a fast square root by table
 * lookup.
 * SPARC floating point format is as follows:
 *
 * BIT 31      30      23      22      0
 *    sign      exponent      mantissa
 */

#include <math.h>

static short sqrttab[0x100];    /* declare table of square roots */

void
build_table()
{
    unsigned short i;
    float f;
    unsigned int *fi = (unsigned int *)&f;
                        /* to access the bits of a float in */
                        /* C quickly we must misuse pointers */

    for(i=0; i<= 0x7f; i++) {
        *fi = 0;

        /*
         * Build a float with the bit pattern i as mantissa
         * and an exponent of 0, stored as 127
         */

        *fi = (i << 16) | (127 << 23);
        f = sqrt(f);

        /*
         * Take the square root then strip the first 7 bits of
         * the mantissa into the table
         */

        sqrttab[i] = (*fi & 0x7fffff) >> 16;

        /*
         * Repeat the process, this time with an exponent of
         * 1, stored as 128
         */

        *fi = 0;
        *fi = (i << 16) | (128 << 23);
        f = sqrt(f);
        sqrttab[i+0x80] = (*fi & 0x7fffff) >> 16;
    }
}

/*
 * fsqrt - fast square root by table lookup
 */
```





```
*/
float
fsqrt(float n)
{
    unsigned int *num = (unsigned int *)&n;
                        /* to access the bits of a float in C
                        * we must misuse pointers */

    short e;           /* the exponent */
    if (n == 0) return (0); /* check for square root of 0 */
    e = (*num >> 23) - 127; /* get the exponent - on a SPARC the */
                        /* exponent is stored with 127 added */
    *num &= 0x7fffff;    /* leave only the mantissa */
    if (e & 0x01) *num |= 0x800000;
                        /* the exponent is odd so we have to */
                        /* look it up in the second half of */
                        /* the lookup table, so we set the */
                        /* high bit */
    e >>= 1;           /* divide the exponent by two */
                        /* note that in C the shift */
                        /* operators are sign preserving */
                        /* for signed operands */
    /* Do the table lookup, based on the quaternary mantissa,
     * then reconstruct the result back into a float
     */
    *num = ((sqrctab[*num >> 16]) << 16) | ((e + 127) << 23);
    return(n);
}
```

# Index of

## /pubs/tog/GraphicsGems/gemsv/ch4-4/

Name	Last modified	Size	Description
 <a href="#">_Parent Directory</a>			
 <a href="#">_aspc.c</a>	29-Jun-00 08:23	1K	

```
/* aspc.c -- generic adaptive sampling of parametric curves */
#include <stdlib.h>
#include <time.h>

typedef struct point { double t,x,y,z; } Point;

#define T(p)      ((p)->t)
#define X(p)      ((p)->x)
#define Y(p)      ((p)->y)
#define Z(p)      ((p)->z)

extern void gamma(Point* p);          /* user supplied */
extern void line(Point* p, Point* q); /* user supplied */

static int flat(Point* p, Point* q, Point* r);











static void sample(Point* p, Point* q)
{
    Point rr, *r=&rr;
    double t = 0.45 + 0.1 * (rand()/(double) RAND_MAX);
    T(r) = T(p) + t*(T(q)-T(p));
    gamma(r);
    if (flat(p,q,r)) line(p,q); else { sample(p,r); sample(r,q); }
}

static int flat(Point* p, Point* q, Point* r)
{
    extern double tol;                /* user supplied */
    double xp = X(p)-X(r); double yp = Y(p)-Y(r); double zp = Z(p)-Z(r);
    double xq = X(q)-X(r); double yq = Y(q)-Y(r); double zq = Z(q)-Z(r);
    double x =  yp*zq-yq*zp;
    double y =  xp*zq-xq*zp;
    double z =  xp*yq-xq*yp;
    return (x*x+y*y+z*z) < tol;      /* |pr x qr|^2 < tol */
}

void aspc(double a, double b)        /* entry point */
{
    Point pp, *p = &pp;
    Point qq, *q = &qq;
    srand(time(0));                  /* randomize */
    T(p)= a; gamma(p); T(q)=b; gamma(q); /* set up */
    sample(p,q);                      /* sample */
}
```

# Index of

## /pubs/tog/GraphicsGems/gemsiv/outcode/

Name	Last modified	Size	Description
 <a href="#">_ Parent Directory</a>			
 <a href="#">_ README</a>	29-Jun-00 08:20	1K	
 <a href="#">_ outcode2.c</a>	29-Jun-00 08:20	1K	
 <a href="#">_ outcode4.c</a>	29-Jun-00 08:20	1K	
 <a href="#">_ xcc2d.c</a>	29-Jun-00 08:20	2K	
 <a href="#">_ xcc4d.c</a>	29-Jun-00 08:20	2K	
 <a href="#">_ xf1.c</a>	29-Jun-00 08:20	3K	
 <a href="#">_ xf2.c</a>	29-Jun-00 08:20	2K	
 <a href="#">_ xf3.c</a>	29-Jun-00 08:20	1K	
 <a href="#">_ xf4.c</a>	29-Jun-00 08:20	1K	

C code from the article  
"Direct Outcode Calculation for Faster Clip Testing"  
by Walt Donovan and Tim Van Hook,  
(donovan@eng.sun.com, tvh@sgi.com)  
in "Graphics Gems IV", Academic Press, 1994

The files outcode2.c and outcode4.c are C code fragments for 2D and 4D outcode calculation that were extracted from the paper, which you can insert in your own code.

The following test programs demonstrate the performance gains from using the direct outcode method.

For a SuperSparc (SS10), we get the following figures:

xcc2d1: ~27 clocks	(would be faster with overlap)
xcc2d2: ~40 clocks	
xcc4d1: ~54 clocks	(the paper says 50, but that's with a more tightly written version than here)
xcc4d2: ~94 clocks	(ditto)

All of the routines were run several times, and the lowest figure reported.

-- walt donovan  
donovan@eng.sun.com

```
/* Direct calculation of 2D outcodes, C code fragment */

/* number of bits in a word */
#define NUMBITS      sizeof(long)*8

/* get the integral form of a floating point number */
#define v(x)         *((long *) &(x))

/* get the sign bit of an integer as a value 0 or 1 */
#define s(x)         (((unsigned long)(x)) >> (NUMBITS-1))

/* get the absolute value of a floating point number in integral form */
#define a(x)         ((x) & ~(1 << (NUMBITS-1)))

/* these values typically would be calculated once and cached */
ixmin = v(xmin);
ixmax = v(xmax);
iymin = v(ymin);
ymax = v(ymax);

/* get the bits and absolute value */
ix = v(x); ax = a(ix);
/* put the sign bit in bit 0 */
outcode = s(ix | (ax - ixmin));

/* put the sign bit in bit 1 */
outcode |= s(~ix & (ixmax - ax)) << 1;

/* do the same for y */
iy = v(y);
ay = a(iy);
outcode |= s(iy | (ay - iymin)) << 2;
outcode |= s(~iy & (ymax - ay)) << 3;
```

```
/* Direct calculation of 4D outcodes, C code fragment */

/* number of bits in a word */
#define NUMBITS      sizeof(long)*8

/* get the integral form of a floating point number */
#define v(x)         *((long *) &(x))

/* get the sign bit of an integer as a value 0 or 1 */
#define s(x)         (((unsigned long)(x)) >> (NUMBITS-1))

/* get the absolute value of a floating point number in integral form */
#define a(x)         ((x) & ~(1 << (NUMBITS-1)))

    iw = v(w);
    abs_w = a(iw);
    outcode = s(iw);          /* 0 or 1 per w's sign */

    ix = v(x);
    diff = s(abs_w - a(ix));
    t = s(ix) + 1;
    outcode |= diff << t;     /* 0, 2, or 4 or'd with outcode */

    iy = v(y);
    diff = s(abs_w - a(iy));
    t = s(iy) + 3;
    outcode |= diff << t;     /* 0, 8, or 16 or'd with outcode */

    iz = v(z);
    diff = s(abs_w - a(iz));
    t = s(iz) + 5;
    outcode |= diff << t;     /* 0, 32, or 64 or'd with outcode */
```

```
/* Time xform, clip checking, lighting only */

#include <stdio.h>
#include <math.h>
#include <fcntl.h>
#include <sys/time.h>

/* BSD timer macros -- replace with your own */
struct timeval start, stop;

#define START    gettimeofday(&start, NULL)
#define STOP     gettimeofday(&stop, NULL)
#define DELTAT   ((float)(stop.tv_sec - start.tv_sec)+1.0e-6* \
                  (float)(stop.tv_usec-start.tv_usec))

#define LOOPCOUNT 10000

#define VSIZ 2 /* vertex size */

#define XSIZ 1280
#define YSIZ 1024
#define ZSIZ 32768

/* clip test bit flags */
#define CXMAX 1
#define CXMIN 2
#define CYMAX 4
#define CYMIN 8
#define CZMAX 16
#define CZMIN 32

#define VERTS 12 /* number of points in primitive */

typedef float verttype[VSIZ];

/* 100 pixel triangles in a triangle strip */
#define X 14.0
verttype vtxm[] = { /* model triangle strip points */
    { 0.0, 0.0 + X*0},
    { X,   X/2 + X*0},
    { 0.0, 0.0 + X*1},
    { X,   X/2 + X*1},
    { 0.0, 0.0 + X*2},
    { X,   X/2 + X*2},
    { 0.0, 0.0 + X*3},
    { X,   X/2 + X*3},
    { 0.0, 0.0 + X*4},
    { X,   X/2 + X*4},
    { 0.0, 0.0 + X*5},
    { X,   X/2 + X*5}, };

/*
 * Transform/clip check test program
 *
 * takes the following command line options:
 *
 * -m MHz    : set MHz for clock calculation
 * -c count  : loop counter (defaults to 10000)
 */
```



```
main( argc, argv)
int argc; char *argv[];
{
int MHz = 40;
int i;
float mtx[16], proj[6];
float vtxw[VERTS+1][VSIZ-1];    /* output xyz device coords */
int vflg[VERTS];
int tflg[2];
int loopcount = LOOPCOUNT;

/* read command line options */
while (--argc){
    if ((*++argv)[0] == '-'){
        switch((*argv)[1]){
            case 'm': MHz = atoi((*++argv));
                --argc;
                break;
            case 'c': loopcount = atoi((*++argv));
                --argc;
                break;
        }
    }
}

/* set up transform and projection matrices */

/* identity matrix */
for (i=0; i<16; i++) mtx[i] = 0;
mtx[0] = mtx[5] = mtx[10] = mtx[15] = 1.0;

/* projection values */
proj[0] = 100.0;
proj[1] = 0.0;
proj[2] = 100.0;
proj[3] = 0.0;
proj[4] = 100.0;
proj[5] = 0.0;

/* start the operation */
START;

for (i=0; i < loopcount; i++) {
    xform_ctp(VERTS,vtxm,vtxw,VSIZ,VSIZ-1,mtx,proj,vflg,tflg);
}

STOP;

printf("%d points in %g seconds\n", i*VERTS, DELTAT);
printf("%g clocks/point at %d MHz\n", DELTAT/(i*VERTS)*MHz*1.0e6, MHz);
}
```

```
/* Time xform, clip checking, lighting only */

#include <stdio.h>
#include <math.h>
#include <fcntl.h>
#include <sys/time.h>

/* BSD timer macros -- replace with your own */
struct timeval start, stop;

#define START    gettimeofday(&start, NULL)
#define STOP     gettimeofday(&stop, NULL)
#define DELTAT   ((float)(stop.tv_sec - start.tv_sec)+1.0e-6* \
                  (float)(stop.tv_usec-start.tv_usec))

#define LOOPCOUNT 10000

#define VSIZ 4 /* vertex size */

#define XSIZ 1280
#define YSIZ 1024
#define ZSIZ 32768

/* clip test bit flags */
#define CXMAX 1
#define CXMIN 2
#define CYMAX 4
#define CYMIN 8
#define CZMAX 16
#define CZMIN 32

#define VERTS 12          /* number of points in primitive */

typedef float verttype[VSIZ];

/* 100 pixel triangles in a triangle strip */
#define X      14.0
verttype vtxm[] = { /* model triangle strip points */
    { 0.0, 0.0 + X*0, 1.0, 100.0 },
    { X,   X/2 + X*0, 1.0, 100.0 },
    { 0.0, 0.0 + X*1, 1.0, 100.0 },
    { X,   X/2 + X*1, 1.0, 100.0 },
    { 0.0, 0.0 + X*2, 1.0, 100.0 },
    { X,   X/2 + X*2, 1.0, 100.0 },
    { 0.0, 0.0 + X*3, 1.0, 100.0 },
    { X,   X/2 + X*3, 1.0, 100.0 },
    { 0.0, 0.0 + X*4, 1.0, 100.0 },
    { X,   X/2 + X*4, 1.0, 100.0 },
    { 0.0, 0.0 + X*5, 1.0, 100.0 },
    { X,   X/2 + X*5, 1.0, 100.0 }, };

/*
 * Transform/clip check test program
 *
 * takes the following command line options:
 *
 * -m MHz      : set MHz for clock calculation
 * -c count    : loop counter (defaults to 10000)
 */
```

```
main( argc, argv)
int argc; char *argv[];
{
int MHz = 40;
int i;
float mtx[16], proj[6];
float vtxw[VERTS+1][VSIZ-1];    /* output xyz device coords */
int vflg[VERTS];
int tflg[2];
int loopcount = LOOPCOUNT;

/* read command line options */
while (--argc){
    if ((*++argv)[0] == '-'){
        switch((*argv)[1]){
            case 'm': MHz = atoi((*++argv));
                --argc;
                break;
            case 'c': loopcount = atoi((*++argv));
                --argc;
                break;
        }
    }
}

/* set up transform and projection matrices */

/* identity matrix */
for (i=0; i<16; i++) mtx[i] = 0;
mtx[0] = mtx[5] = mtx[10] = mtx[15] = 1.0;

/* projection values */
proj[0] = 100.0;
proj[1] = 0.0;
proj[2] = 100.0;
proj[3] = 0.0;
proj[4] = 100.0;
proj[5] = 0.0;

/* start the operation */
START;

for (i=0; i < loopcount; i++) {
    xform_ctp(VERTS,vtxm,vtxw,VSIZ,VSIZ-1,mtx,proj,vflg,tflg);
}

STOP;

printf("%d points in %g seconds\n", i*VERTS, DELTAT);
printf("%g clocks/point at %d MHz\n", DELTAT/(i*VERTS)*MHz*1.0e6, MHz);
}
```

```
/* transform, clip test, and project a vertex list */

/* oflo and division traps should be disabled, by the way */

#define FI(x)    (*(int *)&x)

/*****
xform_ctp(vc, vpi, vpo, visiz, vosiz, mtx, prj, pf, pa)
int vc ;          /* vertex count */
int visiz;        /* input vertex array stride */
int vosiz;        /* output vertex array stride */
int *pf;          /* flag array */
int *pa;          /* 2 element - global or_flag, and_flag */
float *vpi, *vpo, *mtx, *prj;
{
register int flag, flag_or, flag_and;
register int ti0, s, t, u;
register int sign = 0x80000000;

/* flpt registers, total 32 */
register float xx, xy, xz, xw;          /* transformation matrix */
register float yx, yy, yz, yw;
register float zx, zy, zz, zw;
register float wx, wy, wz, ww;
register float xo, yo, zo, wo;          /* output vertex */
register float xs, xt, ys, yt, zs /*zt*/; /* projection scale and translate */
register float t0, t1, t2, t3, t4, t5; /* temps */
register float one = 1.0;

/* load up local projection matrix */
xs = *(prj+0); xt = *(prj+1);
ys = *(prj+2); yt = *(prj+3);
zs = *(prj+4); /* zt = *(prj+5); later */

/* load up local transform matrix */
xx = *(mtx+0 ); xy = *(mtx+1 ); xz = *(mtx+2 ); xw = *(mtx+3 );
yx = *(mtx+4 ); yy = *(mtx+5 ); yz = *(mtx+6 ); yw = *(mtx+7 );
zx = *(mtx+8 ); zy = *(mtx+9 ); zz = *(mtx+10); zw = *(mtx+11);
wx = *(mtx+12); wy = *(mtx+13); wz = *(mtx+14); ww = *(mtx+15);

/* initialize accumulated flags */
flag_or = 0;
flag_and = -1;

do {
#define wi          t4
        wi = vpi[3];      /* wi */
#define xi          t5
        xi = vpi[0];      /* xi */

        /* calculate 4x4 transform, use as few regs as possible */
        wo = wi*ww; xo = wi*wx; yo = wi*wy; zo = wi*wz;
#define yi          t4
        yi = vpi[1];      /* yi */

        t0 = xi*xw; t1 = xi*xx; t2 = xi*xy; t3 = xi*xz;
#define zi          t5
        zi = vpi[2];      /* zi */
        wo += t0; xo += t1; yo += t2; zo += t3;

        t0 = yi*yw; t1 = yi*yx; t2 = yi*yy; t3 = yi*yz;
```

```
wo += t0; xo += t1; yo += t2; zo += t3;
```

```
t0 = zi*zw; t1 = zi*zx; t2 = zi*zy; t3 = zi*zz;  
wo += t0; xo += t1; yo += t2; zo += t3;
```

```
#define winv    t0  
winv = one / wo;
```

```
{  
/* put values on stack so we can get them as integers */  
float sxo, syo, szo, swo;          /* stacked output values */
```

```
swo = wo;  
sxo = xo;  
syo = yo;  
szo = zo;
```

```
#define zt      t1  
/* ready for this now */  
zt = prj[5];
```

```
/* interleave clip checking with projection to DC */  
xo *= xs;  
yo *= ys;  
zo *= zs;
```

```
/* branch-free clip checker*/  
ti0 = FI(swo);  
s = ti0 & ~sign;          /* abs(w) */
```

```
ti0 = FI(sxo);  
t = ti0 & ~sign;          /* abs(x) */  
u = s - t;                /* compare to abs(w) */  
u = (unsigned int)u >> 31; /* extract sign, 1 is abs(x) > abs(w) */  
t = (unsigned int)ti0 >> 31; /* x sign */  
flag = u << t;            /* bit 0: x pos bit 1: x neg */
```

```
xo *= winv;  
yo *= winv;  
zo *= winv;
```

```
ti0 = FI(syo);  
t = ti0 & ~sign;          /* abs(y) */  
u = s - t;                /* compare to w */  
u = (unsigned int)u >> 31; /* extract sign, 1 is abs(y) > w */  
t = (unsigned int)ti0 >> 31; /* y sign */  
t += 2;  
t = u << t;               /* bit 2: y pos bit 3: y neg */  
flag |= t;
```

```
xo += xt;  
yo += yt;  
zo += zt;
```

```
ti0 = FI(szo);  
t = ti0 & ~sign;          /* abs(z) */  
u = s - t;                /* compare to w */  
u = (unsigned int)u >> 31; /* extract sign, 1 is abs(z) > w */  
t = (unsigned int)ti0 >> 31; /* z sign */  
t += 4;  
t = u << t;               /* bit 4: z pos bit 5: z neg */
```

```
    flag |= t;
```

```
}
```

```
flag_or |= flag;
```

```
flag_and &= flag;
```

```
*(vpo+0) = xo;
```

```
*(vpo+1) = yo;
```

```
*(vpo+2) = zo;
```

```
*pf = flag;
```

```
vpi += visiz;
```

```
vpo += vosiz;
```

```
pf++;
```

```
} while (--vc > 0);
```

```
*(pa+0) = flag_or;
```

```
*(pa+1) = flag_and;
```

```
return;
```

```
}
```

```
/* transform, clip test, and project a vertex list */

/* oflo and division traps should be disabled, by the way */

#define FI(x)    (*(int *)&x)

/*****/
xform_ctp(vc, vpi, vpo, visiz, vosiz, mtx, prj, pf, pa)
int vc ;          /* vertex count */
int visiz;        /* input vertex array stride */
int vosiz;        /* output vertex array stride */
int *pf;          /* flag array */
int *pa;          /* 2 element - global or_flag, and_flag */
float *vpi, *vpo, *mtx, *prj;
{
    register int flag, flag_or, flag_and;

    /* flpt registers, total 32 */
    register float xx, xy, xz, xw;          /* transformation matrix */
    register float yx, yy, yz, yw;
    register float zx, zy, zz, zw;
    register float wx, wy, wz, ww;
    register float xo, yo, zo, wo;          /* output vertex */
    register float xs, xt, ys, yt, zs /*zt*/; /* projection scale and translate */
    register float t0, t1, t2, t3, t4, t5; /* temps */
    register float one = 1.0;

    /* load up local projection matrix */
    xs = *(prj+0); xt = *(prj+1);
    ys = *(prj+2); yt = *(prj+3);
    zs = *(prj+4); /* zt = *(prj+5); later */

    /* load up local transform matrix */
    xx = *(mtx+0 ); xy = *(mtx+1 ); xz = *(mtx+2 ); xw = *(mtx+3 );
    yx = *(mtx+4 ); yy = *(mtx+5 ); yz = *(mtx+6 ); yw = *(mtx+7 );
    zx = *(mtx+8 ); zy = *(mtx+9 ); zz = *(mtx+10); zw = *(mtx+11);
    wx = *(mtx+12); wy = *(mtx+13); wz = *(mtx+14); ww = *(mtx+15);

    /* initialize accumulated flags */
    flag_or = 0;
    flag_and = -1;

    do {
#define wi          t4
        wi = vpi[3];    /* wi */
#define xi          t5
        xi = vpi[0];    /* xi */

        /* calculate 4x4 transform, use as few regs as possible */
        wo = wi*ww; xo = wi*wx; yo = wi*wy; zo = wi*wz;
#define yi          t4
        yi = vpi[1];    /* yi */

        t0 = xi*xw; t1 = xi*xx; t2 = xi*xy; t3 = xi*xz;
#define zi          t5
        zi = vpi[2];    /* zi */
        wo += t0; xo += t1; yo += t2; zo += t3;

        t0 = yi*yw; t1 = yi*yx; t2 = yi*yy; t3 = yi*yz;
        wo += t0; xo += t1; yo += t2; zo += t3;
    } while (FI(flag_or) < 0 || FI(flag_and) > 0);
}
```

```
t0 = zi*zw; t1 = zi*zx; t2 = zi*zy; t3 = zi*zz;
wo += t0; xo += t1; yo += t2; zo += t3;
```

```
#define winv    t0
winv = one / wo;
```

```
#define zt      t1
/* ready for this now */
zt =    prj[5];

/* let's try branches for comparison purposes */
flag = 0;
```

```
#define CXMAX    1
#define CXMIN    2
#define CYMAX    4
#define CYMIN    8
#define CZMAX   16
#define CZMIN   32
```

```
if (xo >  wo) flag |= CXMAX;
if (xo < -wo) flag |= CXMIN;
if (yo >  wo) flag |= CYMAX;
if (yo < -wo) flag |= CYMIN;
if (zo >  wo) flag |= CZMAX;
if (zo < -wo) flag |= CZMIN;
```

```
flag_or |= flag;
flag_and &= flag;
```

```
xo *= xs;
yo *= ys;
zo *= zs;
```

```
xo *= winv;
yo *= winv;
zo *= winv;
```

```
xo += xt;
yo += yt;
zo += zt;
```

```
*(vpo+0) = xo;
*(vpo+1) = yo;
*(vpo+2) = zo;
*pf = flag;
```

```
vpi += visiz;
vpo += vosiz;
pf++;
```

```
} while (--vc > 0);
```

```
*(pa+0) = flag_or;
*(pa+1) = flag_and;
```

```
return;
}
```



```
/* 2D transform and clip test a vertex list */

/* oflo and division traps should be disabled, by the way */

#define FI(x)    (*(int *)&x)

/*****
xform_ctp(vc, vpi, vpo, visiz, vosiz, mtx, prj, pf, pa)
int vc ;          /* vertex count */
int visiz;        /* input vertex array stride */
int vosiz;        /* output vertex array stride */
int *pf;          /* flag array */
int *pa;          /* 2 element - global or_flag, and_flag */
float *vpi, *vpo, *mtx, *prj;
{
register int flag, flag_or, flag_and;
register int sign = 0x80000000;

/* flpt registers, total 32 */
register float xx, xy, xt;      /* transformation matrix */
register float yx, yy, yt;
float xo, yo;                  /* output vertex */
register float xi, yi;          /* temps */

register int ixmin, ixmax, iymmin, iymax;
register int ix, ax;

#define XMIN      0
#define XMAX      1279
#define YMIN      0
#define YMAX      1023

float xmin = XMIN, ymin = YMIN, xmax = XMAX, ymax = YMAX;

/* load up local transform matrix */
xx = *(mtx+0 ); xy = *(mtx+1 ); xt = *(mtx+2 );
yx = *(mtx+4 ); yy = *(mtx+5 ); yt = *(mtx+6 );

/* initialize accumulated flags */
flag_or = 0;
flag_and = -1;

ixmin = FI(xmin); iymmin = FI(ymin);
ixmax = FI(xmax); iymax = FI(ymax);

do {
    xi = vpi[0];      /* xi */
    yi = vpi[1];      /* yi */

    xo = xi * xx + yi * xy + xt;
    yo = xi * yx + yi * yy + yt;

    ix = FI(xo); ax = ix & ~sign;
    flag = (ix | (ax - ixmin)) >> 31;
    flag |= ((~ix & (ixmax - ax)) >> 31 ) << 1;

    ix = FI(yo); ax = ix & ~sign;
    flag = ((ix | (ax - iymmin)) >> 31) << 2;
    flag |= ((~ix & (iymax - ax)) >> 31 ) << 3;

    flag_or |= flag;
}
```

```
    flag_and &= flag;

    *(vpo+0) = xo;
    *(vpo+1) = yo;
    *pf = flag;

    vpi += visiz;
    vpo += vosiz;
    pf++;
} while (--vc > 0);

*(pa+0) = flag_or;
*(pa+1) = flag_and;

return;
}
```

```
/* 2D transform and clip test a vertex list */

/* oflo and division traps should be disabled, by the way */

#define FI(x)    (*(int *)&x)

/*****
xform_ctp(vc, vpi, vpo, visiz, vosiz, mtx, prj, pf, pa)
int vc ;          /* vertex count */
int visiz;        /* input vertex array stride */
int vosiz;        /* output vertex array stride */
int *pf;          /* flag array */
int *pa;          /* 2 element - global or_flag, and_flag */
float *vpi, *vpo, *mtx, *prj;
{
register int flag, flag_or, flag_and;

/* flpt registers, total 32 */
register float xx, xy, xt;      /* transformation matrix */
register float yx, yy, yt;
register float xo, yo;         /* output vertex */
register float xi, yi;         /* temps */

/* load up local transform matrix */
xx = *(mtx+0 ); xy = *(mtx+1 ); xt = *(mtx+2 );
yx = *(mtx+4 ); yy = *(mtx+5 ); yt = *(mtx+6 );

/* initialize accumulated flags */
flag_or = 0;
flag_and = -1;

do {
    xi = vpi[0];      /* xi */
    yi = vpi[1];      /* yi */

    xo = xi * xx + yi * xy + xt;
    yo = xi * yx + yi * yy + yt;

    /* let's try branches for comparison purposes */
    flag = 0;

#define CXMIN    1
#define CXMAX    2
#define CYMIN    4
#define CYMAX    8

#define XMIN     0
#define XMAX     1279
#define YMIN     0
#define YMAX     1023

    if (xo < XMIN) flag |= CXMIN;
    if (xo > XMAX) flag |= CXMAX;
    if (yo < YMIN) flag |= CYMIN;
    if (yo > YMAX) flag |= CYMAX;

    flag_or |= flag;
    flag_and &= flag;

    *(vpo+0) = xo;
    *(vpo+1) = yo;
}
```






```
    *pf = flag;

    vpi += visiz;
    vpo += vosiz;
    pf++;
} while (--vc > 0);

*(pa+0) = flag_or;
*(pa+1) = flag_and;

return;
}
```

# Index of /pubs/tog/GraphicsGems/gemsiv/vec\_mat/

Name	Last modified	Size	Description
 <a href="#">_ Parent Directory</a>			
 <a href="#">_ README</a>	29-Jun-00 08:21	1K	
 <a href="#">_ algebra3.c</a>	29-Jun-00 08:21	22K	
 <a href="#">_ algebra3.h</a>	29-Jun-00 08:21	12K	
 <a href="#">_ ray/</a>	29-Jun-00 08:21	1K	

C++ code from the article

"C++ Vector and Matrix Algebra Routines"

by Jean-Francois Doue, h058@frhec1.hec.fr

in "Graphics Gems IV", Academic Press, 1994

algebra3.[CH]     - the subroutine library

ray/             - subdirectory containing a ray caster built on this library

```
#include "algebra3.h"
#include <ctype.h>

/*****
*
*          vec2 Member functions
*
*****/

// CONSTRUCTORS

vec2::vec2() {}

vec2::vec2(const double x, const double y)
{ n[VX] = x; n[VY] = y; }

vec2::vec2(const double d)
{ n[VX] = n[VY] = d; }

vec2::vec2(const vec2& v)
{ n[VX] = v.n[VX]; n[VY] = v.n[VY]; }

vec2::vec2(const vec3& v) // it is up to caller to avoid divide-by-zero
{ n[VX] = v.n[VX]/v.n[VZ]; n[VY] = v.n[VY]/v.n[VZ]; };

vec2::vec2(const vec3& v, int dropAxis) {
    switch (dropAxis) {
        case VX: n[VX] = v.n[VY]; n[VY] = v.n[VZ]; break;
        case VY: n[VX] = v.n[VX]; n[VY] = v.n[VZ]; break;
        default: n[VX] = v.n[VX]; n[VY] = v.n[VY]; break;
    }
}

// ASSIGNMENT OPERATORS

vec2& vec2::operator = (const vec2& v)
{ n[VX] = v.n[VX]; n[VY] = v.n[VY]; return *this; }

vec2& vec2::operator += ( const vec2& v )
{ n[VX] += v.n[VX]; n[VY] += v.n[VY]; return *this; }

vec2& vec2::operator -= ( const vec2& v )
{ n[VX] -= v.n[VX]; n[VY] -= v.n[VY]; return *this; }

vec2& vec2::operator *= ( const double d )
{ n[VX] *= d; n[VY] *= d; return *this; }

vec2& vec2::operator /= ( const double d )
{ double d_inv = 1./d; n[VX] *= d_inv; n[VY] *= d_inv; return *this; }

double& vec2::operator [] ( int i) {
    if (i < VX || i > VY)
        V_ERROR("vec2 [] operator: illegal access; index = " << i << '\n')
    return n[i];
}

// SPECIAL FUNCTIONS

double vec2::length()
```

```
{ return sqrt(length2()); }

double vec2::length2()
{ return n[VX]*n[VX] + n[VY]*n[VY]; }

vec2& vec2::normalize() // it is up to caller to avoid divide-by-zero
{ *this /= length(); return *this; }

vec2& vec2::apply(V_FCT_PTR fct)
{ n[VX] = (*fct)(n[VX]); n[VY] = (*fct)(n[VY]); return *this; }

// FRIENDS

vec2 operator - (const vec2& a)
{ return vec2(-a.n[VX], -a.n[VY]); }

vec2 operator + (const vec2& a, const vec2& b)
{ return vec2(a.n[VX]+ b.n[VX], a.n[VY] + b.n[VY]); }

vec2 operator - (const vec2& a, const vec2& b)
{ return vec2(a.n[VX]-b.n[VX], a.n[VY]-b.n[VY]); }

vec2 operator * (const vec2& a, const double d)
{ return vec2(d*a.n[VX], d*a.n[VY]); }

vec2 operator * (const double d, const vec2& a)
{ return a*d; }

vec2 operator * (const mat3& a, const vec2& v) {
    vec3 av;

    av.n[VX] = a.v[0].n[VX]*v.n[VX] + a.v[0].n[VY]*v.n[VY] + a.v[0].n[VZ];
    av.n[VY] = a.v[1].n[VX]*v.n[VX] + a.v[1].n[VY]*v.n[VY] + a.v[1].n[VZ];
    av.n[VZ] = a.v[2].n[VX]*v.n[VX] + a.v[2].n[VY]*v.n[VY] + a.v[2].n[VZ];
    return av;
}

vec2 operator * (const vec2& v, mat3& a)
{ return a.transpose() * v; }

double operator * (const vec2& a, const vec2& b)
{ return (a.n[VX]*b.n[VX] + a.n[VY]*b.n[VY]); }

vec2 operator / (const vec2& a, const double d)
{ double d_inv = 1./d; return vec2(a.n[VX]*d_inv, a.n[VY]*d_inv); }

vec3 operator ^ (const vec2& a, const vec2& b)
{ return vec3(0.0, 0.0, a.n[VX] * b.n[VY] - b.n[VX] * a.n[VY]); }

int operator == (const vec2& a, const vec2& b)
{ return (a.n[VX] == b.n[VX]) && (a.n[VY] == b.n[VY]); }

int operator != (const vec2& a, const vec2& b)
{ return !(a == b); }

ostream& operator << (ostream& s, vec2& v)
{ return s << "|" << v.n[VX] << " " << v.n[VY] << "|"; }

istream& operator >> (istream& s, vec2& v) {
    vec2      v_tmp;
```



```
char      c = ' ';

while (isspace(c))
    s >> c;
// The vectors can be formatted either as x y or | x y |
if (c == '|') {
    s >> v_tmp[VX] >> v_tmp[VY];
    while (s >> c && isspace(c)) ;
    if (c != '|')
        s.set(_bad);
}
else {
    s.putback(c);
    s >> v_tmp[VX] >> v_tmp[VY];
}
if (s)
    v = v_tmp;
return s;
}

void swap(vec2& a, vec2& b)
{ vec2 tmp(a); a = b; b = tmp; }

vec2 min(const vec2& a, const vec2& b)
{ return vec2(MIN(a.n[VX], b.n[VX]), MIN(a.n[VY], b.n[VY])); }

vec2 max(const vec2& a, const vec2& b)
{ return vec2(MAX(a.n[VX], b.n[VX]), MAX(a.n[VY], b.n[VY])); }

vec2 prod(const vec2& a, const vec2& b)
{ return vec2(a.n[VX] * b.n[VX], a.n[VY] * b.n[VY]); }

/*****
*
*          vec3 Member functions
*
*****/

// CONSTRUCTORS

vec3::vec3() {}

vec3::vec3(const double x, const double y, const double z)
{ n[VX] = x; n[VY] = y; n[VZ] = z; }

vec3::vec3(const double d)
{ n[VX] = n[VY] = n[VZ] = d; }

vec3::vec3(const vec3& v)
{ n[VX] = v.n[VX]; n[VY] = v.n[VY]; n[VZ] = v.n[VZ]; }

vec3::vec3(const vec2& v)
{ n[VX] = v.n[VX]; n[VY] = v.n[VY]; n[VZ] = 1.0; }

vec3::vec3(const vec2& v, double d)
{ n[VX] = v.n[VX]; n[VY] = v.n[VY]; n[VZ] = d; }

vec3::vec3(const vec4& v) // it is up to caller to avoid divide-by-zero
{ n[VX] = v.n[VX] / v.n[VW]; n[VY] = v.n[VY] / v.n[VW];
  n[VZ] = v.n[VZ] / v.n[VW]; }
```

```
vec3::vec3(const vec4& v, int dropAxis) {
    switch (dropAxis) {
        case VX: n[VX] = v.n[VY]; n[VY] = v.n[VZ]; n[VZ] = v.n[VW]; break;
        case VY: n[VX] = v.n[VX]; n[VY] = v.n[VZ]; n[VZ] = v.n[VW]; break;
        case VZ: n[VX] = v.n[VX]; n[VY] = v.n[VY]; n[VZ] = v.n[VW]; break;
        default: n[VX] = v.n[VX]; n[VY] = v.n[VY]; n[VZ] = v.n[VZ]; break;
    }
}
```

// ASSIGNMENT OPERATORS

```
vec3& vec3::operator = (const vec3& v)
{ n[VX] = v.n[VX]; n[VY] = v.n[VY]; n[VZ] = v.n[VZ]; return *this; }

vec3& vec3::operator += ( const vec3& v )
{ n[VX] += v.n[VX]; n[VY] += v.n[VY]; n[VZ] += v.n[VZ]; return *this; }

vec3& vec3::operator -= ( const vec3& v )
{ n[VX] -= v.n[VX]; n[VY] -= v.n[VY]; n[VZ] -= v.n[VZ]; return *this; }

vec3& vec3::operator *= ( const double d )
{ n[VX] *= d; n[VY] *= d; n[VZ] *= d; return *this; }

vec3& vec3::operator /= ( const double d )
{ double d_inv = 1./d; n[VX] *= d_inv; n[VY] *= d_inv; n[VZ] *= d_inv;
  return *this; }

double& vec3::operator [] ( int i) {
    if (i < VX || i > VZ)
        V_ERROR("vec3 [] operator: illegal access; index = " << i << '\n')
    return n[i];
}
```

// SPECIAL FUNCTIONS

```
double vec3::length()
{ return sqrt(length2()); }

double vec3::length2()
{ return n[VX]*n[VX] + n[VY]*n[VY] + n[VZ]*n[VZ]; }

vec3& vec3::normalize() // it is up to caller to avoid divide-by-zero
{ *this /= length(); return *this; }

vec3& vec3::apply(V_FCT_PTR fct)
{ n[VX] = (*fct)(n[VX]); n[VY] = (*fct)(n[VY]); n[VZ] = (*fct)(n[VZ]);
  return *this; }
```

// FRIENDS

```
vec3 operator - (const vec3& a)
{ return vec3(-a.n[VX],-a.n[VY],-a.n[VZ]); }

vec3 operator + (const vec3& a, const vec3& b)
{ return vec3(a.n[VX]+ b.n[VX], a.n[VY] + b.n[VY], a.n[VZ] + b.n[VZ]); }

vec3 operator - (const vec3& a, const vec3& b)
{ return vec3(a.n[VX]-b.n[VX], a.n[VY]-b.n[VY], a.n[VZ]-b.n[VZ]); }
```

```
vec3 operator * (const vec3& a, const double d)
{ return vec3(d*a.n[VX], d*a.n[VY], d*a.n[VZ]); }

vec3 operator * (const double d, const vec3& a)
{ return a*d; }

vec3 operator * (const mat4& a, const vec3& v)
{ return a * vec4(v); }

vec3 operator * (const vec3& v, mat4& a)
{ return a.transpose() * v; }

double operator * (const vec3& a, const vec3& b)
{ return (a.n[VX]*b.n[VX] + a.n[VY]*b.n[VY] + a.n[VZ]*b.n[VZ]); }

vec3 operator / (const vec3& a, const double d)
{ double d_inv = 1./d; return vec3(a.n[VX]*d_inv, a.n[VY]*d_inv,
  a.n[VZ]*d_inv); }

vec3 operator ^ (const vec3& a, const vec3& b) {
    return vec3(a.n[VY]*b.n[VZ] - a.n[VZ]*b.n[VY],
                a.n[VZ]*b.n[VX] - a.n[VX]*b.n[VZ],
                a.n[VX]*b.n[VY] - a.n[VY]*b.n[VX]);
}

int operator == (const vec3& a, const vec3& b)
{ return (a.n[VX] == b.n[VX]) && (a.n[VY] == b.n[VY]) && (a.n[VZ] == b.n[VZ]); }

int operator != (const vec3& a, const vec3& b)
{ return !(a == b); }

ostream& operator << (ostream& s, vec3& v)
{ return s << "|" << v.n[VX] << " " << v.n[VY] << " " << v.n[VZ] << " |"; }

istream& operator >> (istream& s, vec3& v) {
    vec3      v_tmp;
    char      c = ' ';

    while (isspace(c))
        s >> c;
    // The vectors can be formatted either as x y z or | x y z |
    if (c == '|') {
        s >> v_tmp[VX] >> v_tmp[VY] >> v_tmp[VZ];
        while (s >> c && isspace(c)) ;
        if (c != '|')
            s.set(_bad);
    }
    else {
        s.putback(c);
        s >> v_tmp[VX] >> v_tmp[VY] >> v_tmp[VZ];
    }
    if (s)
        v = v_tmp;
    return s;
}

void swap(vec3& a, vec3& b)
{ vec3 tmp(a); a = b; b = tmp; }
```

```
vec3 min(const vec3& a, const vec3& b)
{ return vec3(MIN(a.n[VX], b.n[VX]), MIN(a.n[VY], b.n[VY]), MIN(a.n[VZ],
  b.n[VZ])); }

vec3 max(const vec3& a, const vec3& b)
{ return vec3(MAX(a.n[VX], b.n[VX]), MAX(a.n[VY], b.n[VY]), MAX(a.n[VZ],
  b.n[VZ])); }

vec3 prod(const vec3& a, const vec3& b)
{ return vec3(a.n[VX] * b.n[VX], a.n[VY] * b.n[VY], a.n[VZ] * b.n[VZ]); }

/*****
*
*                               *
*               vec4 Member functions               *
*
*                               *
*****/

// CONSTRUCTORS

vec4::vec4() {}

vec4::vec4(const double x, const double y, const double z, const double w)
{ n[VX] = x; n[VY] = y; n[VZ] = z; n[VW] = w; }

vec4::vec4(const double d)
{ n[VX] = n[VY] = n[VZ] = n[VW] = d; }

vec4::vec4(const vec4& v)
{ n[VX] = v.n[VX]; n[VY] = v.n[VY]; n[VZ] = v.n[VZ]; n[VW] = v.n[VW]; }

vec4::vec4(const vec3& v)
{ n[VX] = v.n[VX]; n[VY] = v.n[VY]; n[VZ] = v.n[VZ]; n[VW] = 1.0; }

vec4::vec4(const vec3& v, const double d)
{ n[VX] = v.n[VX]; n[VY] = v.n[VY]; n[VZ] = v.n[VZ]; n[VW] = d; }

// ASSIGNMENT OPERATORS

vec4& vec4::operator = (const vec4& v)
{ n[VX] = v.n[VX]; n[VY] = v.n[VY]; n[VZ] = v.n[VZ]; n[VW] = v.n[VW];
return *this; }

vec4& vec4::operator += ( const vec4& v )
{ n[VX] += v.n[VX]; n[VY] += v.n[VY]; n[VZ] += v.n[VZ]; n[VW] += v.n[VW];
return *this; }

vec4& vec4::operator -= ( const vec4& v )
{ n[VX] -= v.n[VX]; n[VY] -= v.n[VY]; n[VZ] -= v.n[VZ]; n[VW] -= v.n[VW];
return *this; }

vec4& vec4::operator *= ( const double d )
{ n[VX] *= d; n[VY] *= d; n[VZ] *= d; n[VW] *= d; return *this; }

vec4& vec4::operator /= ( const double d )
{ double d_inv = 1./d; n[VX] *= d_inv; n[VY] *= d_inv; n[VZ] *= d_inv;
  n[VW] *= d_inv; return *this; }

double& vec4::operator [] ( int i ) {
  if (i < VX || i > VW)
```

```
        V_ERROR("vec4 [] operator: illegal access; index = " << i << '\n')
    return n[i];
}

// SPECIAL FUNCTIONS

double vec4::length()
{ return sqrt(length2()); }

double vec4::length2()
{ return n[VX]*n[VX] + n[VY]*n[VY] + n[VZ]*n[VZ] + n[VW]*n[VW]; }

vec4& vec4::normalize() // it is up to caller to avoid divide-by-zero
{ *this /= length(); return *this; }

vec4& vec4::apply(V_FCT_PTR fct)
{ n[VX] = (*fct)(n[VX]); n[VY] = (*fct)(n[VY]); n[VZ] = (*fct)(n[VZ]);
  n[VW] = (*fct)(n[VW]); return *this; }

// FRIENDS

vec4 operator - (const vec4& a)
{ return vec4(-a.n[VX],-a.n[VY],-a.n[VZ],-a.n[VW]); }

vec4 operator + (const vec4& a, const vec4& b)
{ return vec4(a.n[VX] + b.n[VX], a.n[VY] + b.n[VY], a.n[VZ] + b.n[VZ],
  a.n[VW] + b.n[VW]); }

vec4 operator - (const vec4& a, const vec4& b)
{ return vec4(a.n[VX] - b.n[VX], a.n[VY] - b.n[VY], a.n[VZ] - b.n[VZ],
  a.n[VW] - b.n[VW]); }

vec4 operator * (const vec4& a, const double d)
{ return vec4(d*a.n[VX], d*a.n[VY], d*a.n[VZ], d*a.n[VW] ); }

vec4 operator * (const double d, const vec4& a)
{ return a*d; }

vec4 operator * (const mat4& a, const vec4& v) {
    #define ROWCOL(i) a.v[i].n[0]*v.n[VX] + a.v[i].n[1]*v.n[VY] \
    + a.v[i].n[2]*v.n[VZ] + a.v[i].n[3]*v.n[VW]
    return vec4(ROWCOL(0), ROWCOL(1), ROWCOL(2), ROWCOL(3));
    #undef ROWCOL(i)
}

vec4 operator * (const vec4& v, mat4& a)
{ return a.transpose() * v; }

double operator * (const vec4& a, const vec4& b)
{ return (a.n[VX]*b.n[VX] + a.n[VY]*b.n[VY] + a.n[VZ]*b.n[VZ] +
  a.n[VW]*b.n[VW]); }

vec4 operator / (const vec4& a, const double d)
{ double d_inv = 1./d; return vec4(a.n[VX]*d_inv, a.n[VY]*d_inv, a.n[VZ]*d_inv,
  a.n[VW]*d_inv); }

int operator == (const vec4& a, const vec4& b)
{ return (a.n[VX] == b.n[VX]) && (a.n[VY] == b.n[VY]) && (a.n[VZ] == b.n[VZ])
  && (a.n[VW] == b.n[VW]); }
```

```
int operator != (const vec4& a, const vec4& b)
{ return !(a == b); }

ostream& operator << (ostream& s, vec4& v)
{ return s << "|" << v.n[VX] << " " << v.n[VY] << " " << v.n[VZ] << " "
  << v.n[VW] << " |"; }

istream& operator >> (istream& s, vec4& v) {
    vec4      v_tmp;
    char      c = ' ';

    while (isspace(c))
        s >> c;
    // The vectors can be formatted either as x y z w or | x y z w |
    if (c == '|') {
        s >> v_tmp[VX] >> v_tmp[VY] >> v_tmp[VZ] >> v_tmp[VW];
        while (s >> c && isspace(c)) ;
        if (c != '|')
            s.set(_bad);
    }
    else {
        s.putback(c);
        s >> v_tmp[VX] >> v_tmp[VY] >> v_tmp[VZ] >> v_tmp[VW];
    }
    if (s)
        v = v_tmp;
    return s;
}

void swap(vec4& a, vec4& b)
{ vec4 tmp(a); a = b; b = tmp; }

vec4 min(const vec4& a, const vec4& b)
{ return vec4(MIN(a.n[VX], b.n[VX]), MIN(a.n[VY], b.n[VY]), MIN(a.n[VZ],
  b.n[VZ]), MIN(a.n[VW], b.n[VW])); }

vec4 max(const vec4& a, const vec4& b)
{ return vec4(MAX(a.n[VX], b.n[VX]), MAX(a.n[VY], b.n[VY]), MAX(a.n[VZ],
  b.n[VZ]), MAX(a.n[VW], b.n[VW])); }

vec4 prod(const vec4& a, const vec4& b)
{ return vec4(a.n[VX] * b.n[VX], a.n[VY] * b.n[VY], a.n[VZ] * b.n[VZ],
  a.n[VW] * b.n[VW]); }

/*****
*
*                               *
*               mat3 member functions                               *
*                               *
*****/

// CONSTRUCTORS

mat3::mat3() {}

mat3::mat3(const vec3& v0, const vec3& v1, const vec3& v2)
{ v[0] = v0; v[1] = v1; v[2] = v2; }

mat3::mat3(const double d)
{ v[0] = v[1] = v[2] = vec3(d); }
```

```
mat3::mat3(const mat3& m)
{ v[0] = m.v[0]; v[1] = m.v[1]; v[2] = m.v[2]; }

// ASSIGNMENT OPERATORS

mat3& mat3::operator = ( const mat3& m )
{ v[0] = m.v[0]; v[1] = m.v[1]; v[2] = m.v[2]; return *this; }

mat3& mat3::operator += ( const mat3& m )
{ v[0] += m.v[0]; v[1] += m.v[1]; v[2] += m.v[2]; return *this; }

mat3& mat3::operator -= ( const mat3& m )
{ v[0] -= m.v[0]; v[1] -= m.v[1]; v[2] -= m.v[2]; return *this; }

mat3& mat3::operator *= ( const double d )
{ v[0] *= d; v[1] *= d; v[2] *= d; return *this; }

mat3& mat3::operator /= ( const double d )
{ v[0] /= d; v[1] /= d; v[2] /= d; return *this; }

vec3& mat3::operator [] ( int i ) {
    if (i < VX || i > VZ)
        V_ERROR("mat3 [] operator: illegal access; index = " << i << '\n')
    return v[i];
}

// SPECIAL FUNCTIONS

mat3 mat3::transpose() {
    return mat3(vec3(v[0][0], v[1][0], v[2][0]),
                vec3(v[0][1], v[1][1], v[2][1]),
                vec3(v[0][2], v[1][2], v[2][2]));
}

mat3 mat3::inverse()          // Gauss-Jordan elimination with partial pivoting
{
    mat3 a(*this),            // As a evolves from original mat into identity
        b(identity2D());      // b evolves from identity into inverse(a)
    int i, j, il;

    // Loop over cols of a from left to right, eliminating above and below diag
    for (j=0; j<3; j++) {      // Find largest pivot in column j among rows j..2
        il = j;                // Row with largest pivot candidate
        for (i=j+1; i<3; i++)
            if (fabs(a.v[i].n[j]) > fabs(a.v[il].n[j]))
                il = i;

        // Swap rows il and j in a and b to put pivot on diagonal
        swap(a.v[il], a.v[j]);
        swap(b.v[il], b.v[j]);

        // Scale row j to have a unit diagonal
        if (a.v[j].n[j]==0.)
            V_ERROR("mat3::inverse: singular matrix; can't invert\n")
        b.v[j] /= a.v[j].n[j];
        a.v[j] /= a.v[j].n[j];

        // Eliminate off-diagonal elems in col j of a, doing identical ops to b
```

```
    for (i=0; i<3; i++)
        if (i!=j) {
            b.v[i] -= a.v[i].n[j]*b.v[j];
            a.v[i] -= a.v[i].n[j]*a.v[j];
        }
    }
    return b;
}

mat3& mat3::apply(V_FCT_PTR fct) {
    v[VX].apply(fct);
    v[VY].apply(fct);
    v[VZ].apply(fct);
    return *this;
}

// FRIENDS

mat3 operator - (const mat3& a)
{ return mat3(-a.v[0], -a.v[1], -a.v[2]); }

mat3 operator + (const mat3& a, const mat3& b)
{ return mat3(a.v[0] + b.v[0], a.v[1] + b.v[1], a.v[2] + b.v[2]); }

mat3 operator - (const mat3& a, const mat3& b)
{ return mat3(a.v[0] - b.v[0], a.v[1] - b.v[1], a.v[2] - b.v[2]); }

mat3 operator * (mat3& a, mat3& b) {
    #define ROWCOL(i, j) \
        a.v[i].n[0]*b.v[0][j] + a.v[i].n[1]*b.v[1][j] + a.v[i].n[2]*b.v[2][j]
    return mat3(vec3(ROWCOL(0,0), ROWCOL(0,1), ROWCOL(0,2)),
                vec3(ROWCOL(1,0), ROWCOL(1,1), ROWCOL(1,2)),
                vec3(ROWCOL(2,0), ROWCOL(2,1), ROWCOL(2,2)));
    #undef ROWCOL(i, j)
}

mat3 operator * (const mat3& a, const double d)
{ return mat3(a.v[0] * d, a.v[1] * d, a.v[2] * d); }

mat3 operator * (const double d, const mat3& a)
{ return a*d; }

mat3 operator / (const mat3& a, const double d)
{ return mat3(a.v[0] / d, a.v[1] / d, a.v[2] / d); }

int operator == (const mat3& a, const mat3& b)
{ return (a.v[0] == b.v[0]) && (a.v[1] == b.v[1]) && (a.v[2] == b.v[2]); }

int operator != (const mat3& a, const mat3& b)
{ return !(a == b); }

ostream& operator << (ostream& s, mat3& m)
{ return s << m.v[VX] << '\n' << m.v[VY] << '\n' << m.v[VZ]; }

istream& operator >> (istream& s, mat3& m) {
    mat3    m_tmp;

    s >> m_tmp[VX] >> m_tmp[VY] >> m_tmp[VZ];
    if (s)
        m = m_tmp;
}
```



```
        return s;
    }

void swap(mat3& a, mat3& b)
{ mat3 tmp(a); a = b; b = tmp; }

/*****
*
*          mat4 member functions
*
*****/

// CONSTRUCTORS

mat4::mat4() {}

mat4::mat4(const vec4& v0, const vec4& v1, const vec4& v2, const vec4& v3)
{ v[0] = v0; v[1] = v1; v[2] = v2; v[3] = v3; }

mat4::mat4(const double d)
{ v[0] = v[1] = v[2] = v[3] = vec4(d); }

mat4::mat4(const mat4& m)
{ v[0] = m.v[0]; v[1] = m.v[1]; v[2] = m.v[2]; v[3] = m.v[3]; }

// ASSIGNMENT OPERATORS

mat4& mat4::operator = ( const mat4& m )
{ v[0] = m.v[0]; v[1] = m.v[1]; v[2] = m.v[2]; v[3] = m.v[3];
return *this; }

mat4& mat4::operator += ( const mat4& m )
{ v[0] += m.v[0]; v[1] += m.v[1]; v[2] += m.v[2]; v[3] += m.v[3];
return *this; }

mat4& mat4::operator -= ( const mat4& m )
{ v[0] -= m.v[0]; v[1] -= m.v[1]; v[2] -= m.v[2]; v[3] -= m.v[3];
return *this; }

mat4& mat4::operator *= ( const double d )
{ v[0] *= d; v[1] *= d; v[2] *= d; v[3] *= d; return *this; }

mat4& mat4::operator /= ( const double d )
{ v[0] /= d; v[1] /= d; v[2] /= d; v[3] /= d; return *this; }

vec4& mat4::operator [] ( int i ) {
    if (i < VX || i > VW)
        V_ERROR("mat4 [] operator: illegal access; index = " << i << '\n')
    return v[i];
}

// SPECIAL FUNCTIONS;

mat4 mat4::transpose() {
    return mat4(vec4(v[0][0], v[1][0], v[2][0], v[3][0]),
                vec4(v[0][1], v[1][1], v[2][1], v[3][1]),
                vec4(v[0][2], v[1][2], v[2][2], v[3][2]),
                vec4(v[0][3], v[1][3], v[2][3], v[3][3]));
}
```

```
mat4 mat4::inverse()          // Gauss-Jordan elimination with partial pivoting
{
    mat4 a(*this),            // As a evolves from original mat into identity
        b(identity3D());      // b evolves from identity into inverse(a)
    int i, j, il;

    // Loop over cols of a from left to right, eliminating above and below diag
    for (j=0; j<4; j++) {      // Find largest pivot in column j among rows j..3
        il = j;                // Row with largest pivot candidate
        for (i=j+1; i<4; i++)
            if (fabs(a.v[i].n[j]) > fabs(a.v[il].n[j]))
                il = i;

        // Swap rows il and j in a and b to put pivot on diagonal
        swap(a.v[il], a.v[j]);
        swap(b.v[il], b.v[j]);

        // Scale row j to have a unit diagonal
        if (a.v[j].n[j]==0.)
            V_ERROR("mat4::inverse: singular matrix; can't invert\n");
        b.v[j] /= a.v[j].n[j];
        a.v[j] /= a.v[j].n[j];

        // Eliminate off-diagonal elems in col j of a, doing identical ops to b
        for (i=0; i<4; i++)
            if (i!=j) {
                b.v[i] -= a.v[i].n[j]*b.v[j];
                a.v[i] -= a.v[i].n[j]*a.v[j];
            }
    }
    return b;
}

mat4& mat4::apply(V_FCT_PTR fct)
{ v[VX].apply(fct); v[VY].apply(fct); v[VZ].apply(fct); v[VW].apply(fct);
  return *this; }

// FRIENDS

mat4 operator - (const mat4& a)
{ return mat4(-a.v[0], -a.v[1], -a.v[2], -a.v[3]); }

mat4 operator + (const mat4& a, const mat4& b)
{ return mat4(a.v[0] + b.v[0], a.v[1] + b.v[1], a.v[2] + b.v[2],
              a.v[3] + b.v[3]); }

mat4 operator - (const mat4& a, const mat4& b)
{ return mat4(a.v[0] - b.v[0], a.v[1] - b.v[1], a.v[2] - b.v[2], a.v[3] - b.v[3]); }

mat4 operator * (mat4& a, mat4& b) {
    #define ROWCOL(i, j) a.v[i].n[0]*b.v[0][j] + a.v[i].n[1]*b.v[1][j] + \
        a.v[i].n[2]*b.v[2][j] + a.v[i].n[3]*b.v[3][j]
    return mat4(
        vec4(ROWCOL(0,0), ROWCOL(0,1), ROWCOL(0,2), ROWCOL(0,3)),
        vec4(ROWCOL(1,0), ROWCOL(1,1), ROWCOL(1,2), ROWCOL(1,3)),
        vec4(ROWCOL(2,0), ROWCOL(2,1), ROWCOL(2,2), ROWCOL(2,3)),
        vec4(ROWCOL(3,0), ROWCOL(3,1), ROWCOL(3,2), ROWCOL(3,3))
    );
}
```

```
}

mat4 operator * (const mat4& a, const double d)
{ return mat4(a.v[0] * d, a.v[1] * d, a.v[2] * d, a.v[3] * d); }

mat4 operator * (const double d, const mat4& a)
{ return a*d; }

mat4 operator / (const mat4& a, const double d)
{ return mat4(a.v[0] / d, a.v[1] / d, a.v[2] / d, a.v[3] / d); }

int operator == (const mat4& a, const mat4& b)
{ return ((a.v[0] == b.v[0]) && (a.v[1] == b.v[1]) && (a.v[2] == b.v[2]) &&
(a.v[3] == b.v[3])); }

int operator != (const mat4& a, const mat4& b)
{ return !(a == b); }

ostream& operator << (ostream& s, mat4& m)
{ return s << m.v[VX] << '\n' << m.v[VY] << '\n' << m.v[VZ] << '\n' << m.v[VW]; }

istream& operator >> (istream& s, mat4& m)
{
    mat4    m_tmp;

    s >> m_tmp[VX] >> m_tmp[VY] >> m_tmp[VZ] >> m_tmp[VW];
    if (s)
        m = m_tmp;
    return s;
}

void swap(mat4& a, mat4& b)
{ mat4 tmp(a); a = b; b = tmp; }

/*****
*
*          2D functions and 3D functions
*
*****/

mat3 identity2D()
{ return mat3(vec3(1.0, 0.0, 0.0),
               vec3(0.0, 1.0, 0.0),
               vec3(0.0, 0.0, 1.0)); }

mat3 translation2D(vec2& v)
{ return mat3(vec3(1.0, 0.0, v[VX]),
               vec3(0.0, 1.0, v[VY]),
               vec3(0.0, 0.0, 1.0)); }

mat3 rotation2D(vec2& Center, const double angleDeg) {
    double  angleRad = angleDeg * M_PI / 180.0,
           c = cos(angleRad),
           s = sin(angleRad);

    return mat3(vec3(c, -s, Center[VX] * (1.0-c) + Center[VY] * s),
                 vec3(s, c, Center[VY] * (1.0-c) - Center[VX] * s),
                 vec3(0.0, 0.0, 1.0));
}
```

```
mat3 scaling2D(vec2& scaleVector)
{
    return mat3(vec3(scaleVector[VX], 0.0, 0.0),
                 vec3(0.0, scaleVector[VY], 0.0),
                 vec3(0.0, 0.0, 1.0)); }

mat4 identity3D()
{
    return mat4(vec4(1.0, 0.0, 0.0, 0.0),
                 vec4(0.0, 1.0, 0.0, 0.0),
                 vec4(0.0, 0.0, 1.0, 0.0),
                 vec4(0.0, 0.0, 0.0, 1.0)); }

mat4 translation3D(vec3& v)
{
    return mat4(vec4(1.0, 0.0, 0.0, v[VX]),
                 vec4(0.0, 1.0, 0.0, v[VY]),
                 vec4(0.0, 0.0, 1.0, v[VZ]),
                 vec4(0.0, 0.0, 0.0, 1.0)); }

mat4 rotation3D(vec3& Axis, const double angleDeg) {
    double angleRad = angleDeg * M_PI / 180.0,
           c = cos(angleRad),
           s = sin(angleRad),
           t = 1.0 - c;

    Axis.normalize();
    return mat4(vec4(t * Axis[VX] * Axis[VX] + c,
                     t * Axis[VX] * Axis[VY] - s * Axis[VZ],
                     t * Axis[VX] * Axis[VZ] + s * Axis[VY],
                     0.0),
                vec4(t * Axis[VX] * Axis[VY] + s * Axis[VZ],
                     t * Axis[VY] * Axis[VY] + c,
                     t * Axis[VY] * Axis[VZ] - s * Axis[VX],
                     0.0),
                vec4(t * Axis[VX] * Axis[VZ] - s * Axis[VY],
                     t * Axis[VY] * Axis[VZ] + s * Axis[VX],
                     t * Axis[VZ] * Axis[VZ] + c,
                     0.0),
                vec4(0.0, 0.0, 0.0, 1.0));
}

mat4 scaling3D(vec3& scaleVector)
{
    return mat4(vec4(scaleVector[VX], 0.0, 0.0, 0.0),
                 vec4(0.0, scaleVector[VY], 0.0, 0.0),
                 vec4(0.0, 0.0, scaleVector[VZ], 0.0),
                 vec4(0.0, 0.0, 0.0, 1.0)); }

mat4 perspective3D(const double d)
{
    return mat4(vec4(1.0, 0.0, 0.0, 0.0),
                 vec4(0.0, 1.0, 0.0, 0.0),
                 vec4(0.0, 0.0, 1.0, 0.0),
                 vec4(0.0, 0.0, 1.0/d, 0.0)); }
```

```

/*****
*
* C++ Vector and Matrix Algebra routines
* Author: Jean-Francois DOUE
* Version 3.1 --- October 1993
*
*****/

#include <stream.h>
#include <stdlib.h>

// this line defines a new type: pointer to a function which returns a
// double and takes as argument a double
typedef double (*V_FCT_PTR)(double);

// min-max macros
#define MIN(A,B) ((A) < (B) ? (A) : (B))
#define MAX(A,B) ((A) > (B) ? (A) : (B))

// error handling macro
#define V_ERROR(E) { cerr << E; exit(1); }

class vec2;
class vec3;
class vec4;
class mat3;
class mat4;

enum {VX, VY, VZ, VW};           // axes
enum {PA, PB, PC, PD};           // planes
enum {RED, GREEN, BLUE};         // colors
enum {KA, KD, KS, ES};           // phong coefficients

/*****
*
*                               2D Vector
*
*****/

class vec2
{
protected:

    double n[2];

public:

// Constructors

vec2();
vec2(const double x, const double y);
vec2(const double d);
vec2(const vec2& v);               // copy constructor
vec2(const vec3& v);               // cast v3 to v2
vec2(const vec3& v, int dropAxis); // cast v3 to v2

// Assignment operators

vec2& operator = ( const vec2& v ); // assignment of a vec2
vec2& operator += ( const vec2& v ); // incrementation by a vec2
vec2& operator -= ( const vec2& v ); // decrementation by a vec2

```

```
vec2& operator *= ( const double d );    // multiplication by a constant
vec2& operator /= ( const double d );    // division by a constant
double& operator [] ( int i );          // indexing

// special functions

double length();                        // length of a vec2
double length2();                       // squared length of a vec2
vec2& normalize();                      // normalize a vec2
vec2& apply(V_FCT_PTR fct);             // apply a func. to each component

// friends

friend vec2 operator - (const vec2& v);    // -v1
friend vec2 operator + (const vec2& a, const vec2& b);    // v1 + v2
friend vec2 operator - (const vec2& a, const vec2& b);    // v1 - v2
friend vec2 operator * (const vec2& a, const double d);    // v1 * 3.0
friend vec2 operator * (const double d, const vec2& a);    // 3.0 * v1
friend vec2 operator * (const mat3& a, const vec2& v);    // M . v
friend vec2 operator * (const vec2& v, mat3& a);          // v . M
friend double operator * (const vec2& a, const vec2& b);    // dot product
friend vec2 operator / (const vec2& a, const double d);    // v1 / 3.0
friend vec3 operator ^ (const vec2& a, const vec2& b);    // cross product
friend int operator == (const vec2& a, const vec2& b);    // v1 == v2 ?
friend int operator != (const vec2& a, const vec2& b);    // v1 != v2 ?
friend ostream& operator << (ostream& s, vec2& v);        // output to stream
friend istream& operator >> (istream& s, vec2& v);        // input from strm.
friend void swap(vec2& a, vec2& b);                        // swap v1 & v2
friend vec2 min(const vec2& a, const vec2& b);             // min(v1, v2)
friend vec2 max(const vec2& a, const vec2& b);             // max(v1, v2)
friend vec2 prod(const vec2& a, const vec2& b);            // term by term *

// necessary friend declarations

friend class vec3;
};

/*****
*
*                               3D Vector
*
*****/

class vec3
{
protected:

    double n[3];

public:

// Constructors

vec3();
vec3(const double x, const double y, const double z);
vec3(const double d);
vec3(const vec3& v);                // copy constructor
vec3(const vec2& v);                // cast v2 to v3
vec3(const vec2& v, double d);      // cast v2 to v3
vec3(const vec4& v);                // cast v4 to v3
vec3(const vec4& v, int dropAxis);  // cast v4 to v3
```

```
// Assignment operators

vec3& operator = ( const vec3& v );           // assignment of a vec3
vec3& operator += ( const vec3& v );          // incrementation by a vec3
vec3& operator -= ( const vec3& v );          // decrementation by a vec3
vec3& operator *= ( const double d );         // multiplication by a constant
vec3& operator /= ( const double d );         // division by a constant
double& operator [] ( int i );               // indexing

// special functions

double length();                             // length of a vec3
double length2();                             // squared length of a vec3
vec3& normalize();                             // normalize a vec3
vec3& apply(V_FCT_PTR fct);                   // apply a func. to each component

// friends

friend vec3 operator - (const vec3& v);       // -v1
friend vec3 operator + (const vec3& a, const vec3& b); // v1 + v2
friend vec3 operator - (const vec3& a, const vec3& b); // v1 - v2
friend vec3 operator * (const vec3& a, const double d); // v1 * 3.0
friend vec3 operator * (const double d, const vec3& a); // 3.0 * v1
friend vec3 operator * (const mat4& a, const vec3& v); // M . v
friend vec3 operator * (const vec3& v, mat4& a);      // v . M
friend double operator * (const vec3& a, const vec3& b); // dot product
friend vec3 operator / (const vec3& a, const double d); // v1 / 3.0
friend vec3 operator ^ (const vec3& a, const vec3& b); // cross product
friend int operator == (const vec3& a, const vec3& b); // v1 == v2 ?
friend int operator != (const vec3& a, const vec3& b); // v1 != v2 ?
friend ostream& operator << (ostream& s, vec3& v);    // output to stream
friend istream& operator >> (istream& s, vec3& v);    // input from strm.
friend void swap(vec3& a, vec3& b);                  // swap v1 & v2
friend vec3 min(const vec3& a, const vec3& b);        // min(v1, v2)
friend vec3 max(const vec3& a, const vec3& b);        // max(v1, v2)
friend vec3 prod(const vec3& a, const vec3& b);       // term by term *

// necessary friend declarations

friend class vec2;
friend class vec4;
friend class mat3;
friend vec2 operator * (const mat3& a, const vec2& v); // linear transform
friend mat3 operator * (mat3& a, mat3& b);             // matrix 3 product
};

/*****
*
*                               4D Vector
*
*****/

class vec4
{
protected:

    double n[4];

public:
```

```
// Constructors

vec4();
vec4(const double x, const double y, const double z, const double w);
vec4(const double d);
vec4(const vec4& v);           // copy constructor
vec4(const vec3& v);           // cast vec3 to vec4
vec4(const vec3& v, const double d); // cast vec3 to vec4

// Assignment operators

vec4& operator = ( const vec4& v ); // assignment of a vec4
vec4& operator += ( const vec4& v ); // incrementation by a vec4
vec4& operator -= ( const vec4& v ); // decrementation by a vec4
vec4& operator *= ( const double d ); // multiplication by a constant
vec4& operator /= ( const double d ); // division by a constant
double& operator [] ( int i ); // indexing

// special functions

double length(); // length of a vec4
double length2(); // squared length of a vec4
vec4& normalize(); // normalize a vec4
vec4& apply(V_FCT_PTR fct); // apply a func. to each component

// friends

friend vec4 operator - (const vec4& v); // -v1
friend vec4 operator + (const vec4& a, const vec4& b); // v1 + v2
friend vec4 operator - (const vec4& a, const vec4& b); // v1 - v2
friend vec4 operator * (const vec4& a, const double d); // v1 * 3.0
friend vec4 operator * (const double d, const vec4& a); // 3.0 * v1
friend vec4 operator * (const mat4& a, const vec4& v); // M . v
friend vec4 operator * (const vec4& v, mat4& a); // v . M
friend double operator * (const vec4& a, const vec4& b); // dot product
friend vec4 operator / (const vec4& a, const double d); // v1 / 3.0
friend int operator == (const vec4& a, const vec4& b); // v1 == v2 ?
friend int operator != (const vec4& a, const vec4& b); // v1 != v2 ?
friend ostream& operator << (ostream& s, vec4& v); // output to stream
friend istream& operator >> (istream& s, vec4& v); // input from strm.
friend void swap(vec4& a, vec4& b); // swap v1 & v2
friend vec4 min(const vec4& a, const vec4& b); // min(v1, v2)
friend vec4 max(const vec4& a, const vec4& b); // max(v1, v2)
friend vec4 prod(const vec4& a, const vec4& b); // term by term *

// necessary friend declarations

friend class vec3;
friend class mat4;
friend vec3 operator * (const mat4& a, const vec3& v); // linear transform
friend mat4 operator * (mat4& a, mat4& b); // matrix 4 product
};

/*****
*
*           3x3 Matrix
*
*****/

class mat3
{
```



protected:

```
    vec3 v[3];
```

public:

// Constructors

```
mat3();
mat3(const vec3& v0, const vec3& v1, const vec3& v2);
mat3(const double d);
mat3(const mat3& m);
```

// Assignment operators

```
mat3& operator = ( const mat3& m );           // assignment of a mat3
mat3& operator += ( const mat3& m );          // incrementation by a mat3
mat3& operator -= ( const mat3& m );          // decrementation by a mat3
mat3& operator *= ( const double d );         // multiplication by a constant
mat3& operator /= ( const double d );         // division by a constant
vec3& operator [] ( int i );                  // indexing
```

// special functions

```
mat3 transpose();                            // transpose
mat3 inverse();                              // inverse
mat3& apply(V_FCT_PTR fct);                  // apply a func. to each element
```

// friends

```
friend mat3 operator - (const mat3& a);       // -m1
friend mat3 operator + (const mat3& a, const mat3& b); // m1 + m2
friend mat3 operator - (const mat3& a, const mat3& b); // m1 - m2
friend mat3 operator * (mat3& a, mat3& b);    // m1 * m2
friend mat3 operator * (const mat3& a, const double d); // m1 * 3.0
friend mat3 operator * (const double d, const mat3& a); // 3.0 * m1
friend mat3 operator / (const mat3& a, const double d); // m1 / 3.0
friend int operator == (const mat3& a, const mat3& b); // m1 == m2 ?
friend int operator != (const mat3& a, const mat3& b); // m1 != m2 ?
friend ostream& operator << (ostream& s, mat3& m); // output to stream
friend istream& operator >> (istream& s, mat3& m); // input from strm.
friend void swap(mat3& a, mat3& b);           // swap m1 & m2
```

// necessary friend declarations

```
friend vec3 operator * (const mat3& a, const vec3& v); // linear transform
friend vec2 operator * (const mat3& a, const vec2& v); // linear transform
};
```

```
/*
 *
 *                      4x4 Matrix
 *
 */
```

class mat4

```
{
protected:
```

```
    vec4 v[4];
```

```
public:

// Constructors

mat4();
mat4(const vec4& v0, const vec4& v1, const vec4& v2, const vec4& v3);
mat4(const double d);
mat4(const mat4& m);

// Assignment operators

mat4& operator = ( const mat4& m );           // assignment of a mat4
mat4& operator += ( const mat4& m );          // incrementation by a mat4
mat4& operator -= ( const mat4& m );          // decrementation by a mat4
mat4& operator *= ( const double d );         // multiplication by a constant
mat4& operator /= ( const double d );         // division by a constant
vec4& operator [] ( int i );                 // indexing

// special functions

mat4 transpose();                           // transpose
mat4 inverse();                             // inverse
mat4& apply(V_FCT_PTR fct);                 // apply a func. to each element

// friends

friend mat4 operator - (const mat4& a);           // -m1
friend mat4 operator + (const mat4& a, const mat4& b); // m1 + m2
friend mat4 operator - (const mat4& a, const mat4& b); // m1 - m2
friend mat4 operator * (mat4& a, mat4& b);       // m1 * m2
friend mat4 operator * (const mat4& a, const double d); // m1 * 4.0
friend mat4 operator * (const double d, const mat4& a); // 4.0 * m1
friend mat4 operator / (const mat4& a, const double d); // m1 / 3.0
friend int operator == (const mat4& a, const mat4& b); // m1 == m2 ?
friend int operator != (const mat4& a, const mat4& b); // m1 != m2 ?
friend ostream& operator << (ostream& s, mat4& m); // output to stream
friend istream& operator >> (istream& s, mat4& m); // input from strm.
friend void swap(mat4& a, mat4& b);             // swap m1 & m2

// necessary friend declarations

























friend vec4 operator * (const mat4& a, const vec4& v); // linear transform
friend vec3 operator * (const mat4& a, const vec3& v); // linear transform
};

/*****
*
*           2D functions and 3D functions
*
*****/

mat3 identity2D();                           // identity 2D
mat3 translation2D(vec2& v);                 // translation 2D
mat3 rotation2D(vec2& Center, const double angleDeg); // rotation 2D
mat3 scaling2D(vec2& scaleVector);           // scaling 2D
mat4 identity3D();                           // identity 3D
mat4 translation3D(vec3& v);                 // translation 3D
mat4 rotation3D(vec3& Axis, const double angleDeg); // rotation 3D
mat4 scaling3D(vec3& scaleVector);           // scaling 3D
mat4 perspective3D(const double d);          // perspective 3D
```













# Index of

## /pubs/tog/GraphicsGems/gemsiv/vec\_mat/ray/

Name	Last modified	Size	Description
 <a href="#">Parent Directory</a>			
 <a href="#">Camera.c</a>	29-Jun-00 08:20	1K	
 <a href="#">Camera.h</a>	29-Jun-00 08:20	1K	
 <a href="#">Light.c</a>	29-Jun-00 08:21	1K	
 <a href="#">Light.h</a>	29-Jun-00 08:21	1K	
 <a href="#">Makefile</a>	29-Jun-00 08:21	1K	
 <a href="#">Object3D.c</a>	29-Jun-00 08:21	1K	
 <a href="#">Object3D.h</a>	29-Jun-00 08:21	1K	
 <a href="#">Polyhedron.c</a>	29-Jun-00 08:21	5K	
 <a href="#">Polyhedron.h</a>	29-Jun-00 08:21	1K	
 <a href="#">Primitive.c</a>	29-Jun-00 08:21	1K	
 <a href="#">Primitive.h</a>	29-Jun-00 08:21	1K	
 <a href="#">README</a>	29-Jun-00 08:21	7K	
 <a href="#">Scene3D.c</a>	29-Jun-00 08:21	5K	
 <a href="#">Scene3D.h</a>	29-Jun-00 08:21	1K	
 <a href="#">Sphere.c</a>	29-Jun-00 08:21	1K	
 <a href="#">Sphere.h</a>	29-Jun-00 08:21	1K	
 <a href="#">algebra3.c</a>	29-Jun-00 08:20	22K	
 <a href="#">algebra3.h</a>	29-Jun-00 08:20	12K	
 <a href="#">example.data</a>	29-Jun-00 08:20	1K	
 <a href="#">example.tiff</a>	29-Jun-00 08:21	117K	
 <a href="#">main.c</a>	29-Jun-00 08:21	1K	
 <a href="#">solver.c</a>	29-Jun-00 08:21	5K	
 <a href="#">solver.h</a>	29-Jun-00 08:21	1K	

# Index of

## /pubs/tog/GraphicsGems/gemsv/ch6-4/

Name	Last modified	Size	Description
 <a href="#">Parent Directory</a>			
 <a href="#">Makefile</a>	29-Jun-00 08:24	1K	
 <a href="#">README</a>	29-Jun-00 08:24	1K	
 <a href="#">bitmap_1</a>	29-Jun-00 08:24	1K	
 <a href="#">bitmap_2</a>	29-Jun-00 08:24	1K	
 <a href="#">bitmap_3</a>	29-Jun-00 08:24	1K	
 <a href="#">chainCode.C</a>	29-Jun-00 08:24	3K	
 <a href="#">chainCode.h</a>	29-Jun-00 08:24	1K	
 <a href="#">pt2.C</a>	29-Jun-00 08:24	1K	
 <a href="#">pt2.h</a>	29-Jun-00 08:24	1K	
 <a href="#">test.C</a>	29-Jun-00 08:24	3K	
 <a href="#">vectorize.C</a>	29-Jun-00 08:24	6K	

```
# TEST FILE FOR vectorize, NOT FOR BOOK
#
# UNIX makefile for 'vectorize'
# AUTHOR: Jean-Francois DOUE
# LAST MODIFICATION: July 27, 1993
#

CC = gcc

C_FLAGS = -g -O

H_FILES = pt2.h chainCode.h
C_FILES = test.C chainCode.C vectorize.C pt2.C
O_FILES = test.o chainCode.o vectorize.o pt2.o

.SUFFIXES : .o .c .C

.C.o:
    $(CC) $(C_FLAGS) -c $@ $*.C

vectorize: $(O_FILES)
    $(CC) $(C_FLAGS) -o $@ $(O_FILES)
```

This directory contains the files necessary to vectorize a 2D shape.

```
vectorize.C      /* main routine          */
test.C          /* interface for the prog */
pt2D.h          /* a header               */
pt2D.C          /* a 2D point package     */
chainCode.h     /* a header               */
chainCode.C     /* a chain code class     */
Makefile        /* the makefile           */
bitmap_1        /* a test file            */
bitmap_2        /* another test file      */
bitmap_3        /* a third test file      */
```

The program is written in C++. I used gcc2.1 to compile it.  
To run the demo, simply type:

```
>> make
>> vectorize some_file
```

You can view the bitmap files by typing:  
>> cat some\_file

since the files are encoded in text. The program will display the results and save them in '.vec' file. You can get more explanations by simply typing 'vectorize' without arguments. As you will notice, the file does not use GraphicsGems.m. This is because the programs needs 2D integer operations and GraphicsGems.m only provides 2D floating point operations through the Point2 type.

Only the 5 files:  
vectorize.C  
pt2D.h  
pt2D.C  
chainCode.h  
chainCode.C

are meant to make it for the book and the disk accompanying it.

JFD.

```
5 7
00100
01100
10100
00100
00100
00100
11111
% TEST FILE FOR vectorize, NOT FOR THE BOOK %
```

```
10 10
0001001000
0001001000
0001001000
1111111111
0001001000
0001001000
1111111111
0001001000
0001001000
0001001000
% TEST FILE FOR vectorize, NOT FOR THE BOOK %
```





```
#include <stdlib.h>
#include <stdio.h>
#include "chainCode.h"

/*****
/*
/* Class constructor.
/*
/*
*****/

chainCode::chainCode()
{
code = malloc(DEFAULT_CODE_LENGTH * sizeof(char));
code[0] = '\0';
length = DEFAULT_CODE_LENGTH;
}

/*****
/*
/* Class destructor.
/*
/*
*****/

chainCode::~chainCode()
{
free(code);
}

/*****
/*
/* This method appends a new code to the chain. If there
/* is not enough memory left, the function doubles the size
/* of the chain code.
/* It receives as a parameter the new code to be added (c).
/*
*****/

void chainCode::add(char c)
{
int l = strlen(code);

if (l >= length-1){
length *= 2;
code = realloc(code, length);
}
code[l] = c;
code[l+1] = '\0';
}

/*****
/*
/* This method post-processes a 4x chain code to generate a 1x
/* chain code. A pointer to the 1x code is returned. The method
/* uses the 4 following rules:
/* CCCC -> C : reduce to one copy
/* CCC -> {} : eliminate
/* CC -> CC : (ignored)
*****/
```

```
/*      C ->  C :  identity                                */
/*                                                    */
/*****/

chainCode* chainCode::postProcess()
{
    int      i = 0, j;
    chainCode *filtCode;
    int      trueLength = strlen(code);

    filtCode = new chainCode();
    while (i<trueLength){
        if (i+SCALE-1 < trueLength){
            for (j=0; j<SCALE-1; j++)
                if (code[i+j] != code[i+j+1])
                    break;
            if (j == SCALE-1){
                filtCode->add(code[i]);
                i += SCALE;
                continue;
            }
        }

        if (i+SCALE-2 < trueLength){
            for (j=0; j<SCALE-2; j++)
                if (code[i+j] != code[i+j+1])
                    break;
            if (j == SCALE-2){
                i += SCALE-1;
                continue;
            }
        }
        filtCode->add(code[i]);
        i++;
    }
    return filtCode;
}

/*****/
/*                                                    */
/* A utility method to display the chain code          */
/*                                                    */
/*****/

void chainCode::printSelf()
{
    printf("\n%s", code);
}
```

```
#define DEFAULT_CODE_LENGTH 512
#define SCALE 4

class chainCode{
public:
    char* code;
    int length;

    chainCode();
    ~chainCode();
    void add(char c);
    chainCode* postProcess();
    void printSelf();
};
```

```
#include "pt2.h"
```

```

/*****
/*
/* Two utility functions to add and subtract 2D integer
/* points.
/*
/*
*****/
```

```
pt2* addPt2(pt2 *a, pt2 *b, pt2 *c)
{
c->x = a->x + b->x;
c->y = a->y + b->y;
return c;
}
```

```
pt2* subPt2(pt2 *a, pt2 *b, pt2 *c)
{
c->x = a->x - b->x;
c->y = a->y - b->y;
return c;
}
```

```
typedef struct pt2Struct{
    int x,y;
} pt2;

extern pt2* addPt2(pt2 *a, pt2 *b, pt2 *c);
extern pt2* subPt2(pt2 *a, pt2 *b, pt2 *c);
```

```
/* TEST FILE vectorize NOT FOR BOOK */
```

```
#include <stdio.h>
#include <stdlib.h>
#include "chainCode.h"
#include "pt2.h"
```

```
#define MAXPATHLEN 1024
```

```
extern chainCode* encode(pt2 *size, char *bitmap);
```

```
char *mes1[]={ "VECTORIZE",
               "by Jean-Francois Doue",
               " ",
               "This program transforms a bitmap image into a chain code",
               "Bitmap images should be encoded in the following format:",
               " ",
               "x_size y_size",
               "00001111000",
               "00001111000",
               "00001001000",
               ".....",
               "00001111000",
               " ",
               "To convert the image, simply type",
               " ",
               "vectorized fileName",
               " ",
               "The encoded file will be saved under fileName.vec",
               "\n",
               0};
```

```
char *mes2[]={ "Error !",
               "Your file cannot be opened",
               0};
```

```

/*****
/*
/* A simple function to display messages on the screen
/*
/*
/*****/
```

```
void printMessage(char** mes)
{
    int i;
    if (!mes)
        return;
    while(mes[i]){
        printf("\n%s",mes[i]);
        i++;
    }
}
```

```

/*****
/*
*/
```

```
/* This function transforms a text-file containing a bitmap */
/* image into another text file containing the vectorized */
/* image. */
/* */
/*****/

main(int argc, char** argv)
{
    pt2      size;
    int      i, fileN = 0;
    unsigned char  c;
    char      output_name[MAXPATHLEN],
              *bitmap;
    FILE      *input,
              *output;
    chainCode *code;

    /* make sure the user uses the right syntax */
    if (argc == 1){
        printMessage(mes1);
        exit(0);
    }

    /* for all the specified files... */
    while (fileN < argc -1){
        /* open the data file */
        fileN++;
        if ((input = fopen(argv[fileN], "r")) == NULL){
            printMessage(mes2);
            exit(0);
        }

        /* create the .vec file */
        sprintf(output_name,"%s.vec", argv[fileN]);
        if ((output = fopen(output_name, "w")) == NULL){
            printMessage(mes2);
            exit(0);
        }

        /* read x_size and y_size of the bitmap image*/
        fscanf(input,"%d%d",&size.x, &size.y);
        printf("\nEncoding file:%s (x=%d, y=%d)", argv[fileN], size.x, size.y);

        /* read the data from the bitmap image*/
        i = 0;
        bitmap = malloc(size.x * size.y * sizeof(char));
        while (!feof(input)){
            fscanf(input,"%c", &c);
            if (c == '0' || c == '1'){
                bitmap[i] = c;
                i++;
            }
        }

        /* encode */
        code = encode(&size, bitmap);
        printf("\nThe encoded vector is:");
        code->printSelf();
        fprintf(output,"%s", code->code);
        fclose(output);
        fclose(input);
    }
}
```



```
}
```

```
printf( "\n" );  
return 0;  
}
```

```
#include <string.h>
#include <stdlib.h>
#include <limits.h>
#include "chainCode.h"
#include "pt2.h"

/* DEFINITION OF THE CONSTANTS */

#define CONTOUR 'c'
#define VISITED 'v'
#define BLACK '1'
#define WHITE '0'

/* DEFINITION OF THE MACROS */

#define PIX(a,b) ((b) * f_size.x + (a))
#define PIX2(a,b) ((b) * size->x + (a))
#define MIN(x,y) ((x)<(y) ? (x) : (y))
#define MAX(x,y) ((x)>(y) ? (x) : (y))

/*****
*/
/* This is the main function. It receives as a parameter a
/* bitmap image of size 'size' and outputs a chain code.
/* The following constraints are placed on the bitmap:
/* + Each pixel is encoded as a char.
/* + Only white (0) and black (1) pixels are taken into
/* account.
/* + The shape to encode should have no holes and should be
/* in a single piece.
*/
*****/

chainCode* encode(pt2 *size, char *bitmap)
{
static pt2    contour_dir[8] = {{ 1, 0},
    { 0,-1},
    {-1, 0},
    { 0, 1},
    { 1,-1},
    {-1,-1},
    {-1, 1},
    { 1, 1}};

chainCode *code1,
    code4;
char *fatmap,
    direction_code[8] = {'0','2','4','6','1','3','5','7'};
int i,j,u,v,
    flag,
    d, distance,
    last_dir;
pt2 pixel,
    test_pixel,
    start_pixel,
    f_size,
    bbox[2] = {{INT_MAX, INT_MAX},
    {-INT_MAX, -INT_MAX}};

/* CREATE AN EMPTY CHAIN CODE TO RETURN THE RESULT */
code1 = new chainCode();
```

```
/* RESCAN THE BITMAP AT A GREATER RESOLUTION (4x4 GREATER) */
/* ADD TWO BLANK LINES TO THE LEFT, RIGHT, TOP AND BOTTOM */
/* OF THE FATMAP. THESE COULD BE NECESSARY TO AVOID THE */
/* CONTOUR TO BE DRAWN OUTSIDE OF THE BOUNDS OF THE MATRIX */

f_size.x = 2 + SCALE*size->x + 2;
f_size.y = 2 + SCALE*size->y + 2;
fatmap = malloc(f_size.x * f_size.y * sizeof(char));
for (i=0; i<f_size.x * f_size.y; i++)
    fatmap[i] = WHITE;
for (j=0; j<size->y; j++)
    for (i=0; i<size->x; i++)
        if (bitmap[PIX2(i,j)] == BLACK)
            for(v=0; v<SCALE; v++)
                for(u=0; u<SCALE; u++)
                    fatmap[PIX(2+4*i+u, 2+4*j+v)] = BLACK;

/* GENERATE THE CONTOUR OF THE BITMAP USING 4 SUCCESSIVE */
/* PASSES: FOR EACH DIRECTION, WE SCAN EACH LINE UNTIL */
/* WE REACH A BLACK PIXEL: THE PIXEL JUST BEFORE IT IS A */
/* CONTOUR PIXEL */

/* PASS 1: LEFTWARDS */
for (j=0; j<f_size.y; j++)
    for(i=1; i<f_size.x; i++)
        if (fatmap[PIX(i,j)] == BLACK){
            if (flag == 0) {
                fatmap[PIX(i-1, j)] = CONTOUR;
                flag = 1;
            }
        }
    else
        flag = 0;

/* PASS 2: RIGHTWARDS */
for (j=0; j<f_size.y; j++)
    for (i=f_size.x - 1; i>=0; i--)
        if (fatmap[PIX(i,j)] == BLACK){
            if (flag == 0) {
                fatmap[PIX(i+1, j)] = CONTOUR;
                flag = 1;
            }
        }
    else
        flag = 0;

/* PASS 3: DOWNWARDS */
flag = 0;
for (i=0; i<f_size.x; i++)
    for (j=0; j<f_size.y; j++)
        if (fatmap[PIX(i,j)] == BLACK){
            if (flag == 0) {
                fatmap[PIX(i, j-1)] = CONTOUR;
                flag = 1;
            }
        }
    else
        flag = 0;

/* PASS 4: UPWARDS */
```

```
flag = 0;
for (i=0; i<f_size.x; i++)
    for (j=f_size.y - 1; j>=0; j--)
        if (fatmap[PIX(i,j)] == BLACK){
            if (flag == 0) {
                fatmap[PIX(i, j+1)] = CONTOUR;
                flag = 1;
            }
        }
    else
        flag = 0;

/* COMPUTE THE BOUNDING BOX OF THE CHARACTER (L,T,R,B) */
for (j=0; j<f_size.y; j++)
    for(i=1; i<f_size.x; i++)
        if (fatmap[PIX(i,j)]==CONTOUR){
            bbox[0].x = MIN(i, bbox[0].x);
            bbox[0].y = MIN(j, bbox[0].y);
            bbox[1].x = MAX(i, bbox[1].x);
            bbox[1].y = MAX(j, bbox[1].y);
        }

/* DETERMINE THE CONTOUR PIXEL CLOSEST TO THE UPPER LEFT CORNER */
/* OF THE BOUNDING BOX */

distance = INT_MAX;
for (j=0; j<f_size.y; j++)
    for(i=1; i<f_size.x; i++)
        if (fatmap[PIX(i,j)]==CONTOUR){
            d = (i-bbox[0].x) * (i-bbox[0].x) + (j-bbox[0].y) * (j-bbox[0].y);
            if (d < distance) {
                distance = d;
                start_pixel.x = i;
                start_pixel.y = j;
            }
        }

/* BEGIN THE ENCODING PROCEDURE */
pixel.x = start_pixel.x;
pixel.y = start_pixel.y;
fatmap[PIX(pixel.x, pixel.y)] = VISITED;
last_dir = 4;
while(0 < 1) {
    /* AT FIRST, CHECK THE PIXEL IN THE LAST KNOWN DIRECTION */
    addPt2(&pixel, &contour_dir[last_dir], &test_pixel);
    if (fatmap[PIX(test_pixel.x, test_pixel.y)] == CONTOUR){
        pixel.x = test_pixel.x;
        pixel.y = test_pixel.y;
        fatmap[PIX(pixel.x, pixel.y)] = VISITED;
        code4.add(direction_code[last_dir]);
    }
    /* CHECK ALL THE POSSIBLE DIRECTIONS, CLOCKWISE */
    for (i=0;i<8;i++) {
        addPt2(&pixel, &contour_dir[i], &test_pixel);
        if (fatmap[PIX(test_pixel.x, test_pixel.y)] == CONTOUR){
            pixel.x = test_pixel.x;
            pixel.y = test_pixel.y;
            fatmap[PIX(pixel.x, pixel.y)] = VISITED;
            code4.add(direction_code[i]);
            last_dir = i;
        }
    }
}
```






```
        break;
    }
}
if (i == 8)
    break;
}

/* WRITE THE LAST MOVE TO THE OUTPUT VECTOR */
for (i=0; i<8; i++) {
    subPt2(&start_pixel, &pixel, &test_pixel);
    if (test_pixel.x==contour_dir[i].x && test_pixel.y==contour_dir[i].y){
        code4.add(direction_code[i]);
        break;
    }
}

/* POST-PROCESSING LOOP: */
/* GO BACK TO A LOWER RESOLUTION BY FILTERING THE 4x CODE */

code1 = code4.postProcess();
return code1;
}
```

# Index of /pubs/tog/GraphicsGems/gemsiv/data\_smooth/

Name	Last modified	Size	Description
 <a href="#">_Parent Directory</a>			
 <a href="#">_README</a>	29-Jun-00 08:19	1K	
 <a href="#">_smooth1.c</a>	29-Jun-00 08:19	1K	
 <a href="#">_smooth2.c</a>	29-Jun-00 08:19	1K	
 <a href="#">_smooth3.c</a>	29-Jun-00 08:19	2K	

ANSI C code from the article  
"Smoothing and Interpolation with Finite Differences"  
by Paul H. C. Eilers, paul@dcmr.nl  
in "Graphics Gems IV", Academic Press, 1994

files:

- smooth1.c: Smoothing and interpolation with first differences.
- smooth2.c: Smoothing and interpolation with second differences.
- smooth3.c: Smoothing and interpolation with any difference equation.

```
/* Program 1. Smoothing and interpolation with first differences. */
/* Contribution to Graphic Gems IV */

/* Paul H. C. Eilers, DCMR Milieudienst Rijnmond, 's-Gravelandseweg 565,
   3119 XT Schiedam, The Netherlands, E-Mail: paul@dcmr.nl */

#define MMAX 100          /* choose the right length for your application */

typedef float vec[MMAX + 1];

void smooth1(vec w, vec y, vec z, float lambda, int m)
/* Smoothing and interpolation with first differences.
   Input:  weights (w), data (y): vector from 1 to m.
   Input:  smoothing parameter (lambda), length (m).
   Output: smoothed vector (z): vector from 1 to m. */
{
    int i, i1;
    vec c, d;
    d[1] = w[1] + lambda;
    c[1] = -lambda / d[1];
    z[1] = w[1] * y[1];
    for (i = 2; i < m; i++) {
        i1 = i - 1;
        d[i] = w[i] + 2 * lambda - c[i1] * c[i1] * d[i1];
        c[i] = -lambda / d[i];
        z[i] = w[i] * y[i] - c[i1] * z[i1];
    }
    d[m] = w[m] + lambda - c[m - 1] * c[m - 1] * d[m - 1];
    z[m] = (w[m] * y[m] - c[m - 1] * z[m - 1]) / d[m];
    for (i = m - 1; 1 <= i; i--) z[i] = z[i] / d[i] - c[i] * z[i + 1];
}
```



```
/* Program 2. Smoothing and interpolation with second differences. */
/* Contribution to Graphic Gems IV */

/* Paul H. C. Eilers, DCMR Milieudienst Rijnmond, 's-Gravelandseweg 565,
   3119 XT Schiedam, The Netherlands, E-Mail: paul@dcmr.nl */

#define MMAX 100 /* choose the right length for your application */

typedef float vec[MMAX + 1];

void smooth2(vec w, vec y, vec z, float lambda, int m)
/* Smoothing and interpolation with second differences.
   Input: weights (w), data (y): vector from 1 to m.
   Input: smoothing parameter (lambda), length (m).
   Output: smoothed vector (z): vector from 1 to m. */
{
    int i, i1, i2;
    vec c, d, e;
    d[1] = w[1] + lambda;
    c[1] = -2 * lambda / d[1];
    e[1] = lambda / d[1];
    z[1] = w[1] * y[1];
    d[2] = w[2] + 5 * lambda - d[1] * c[1] * c[1];
    c[2] = (-4 * lambda - d[1] * c[1] * e[1]) / d[2];
    e[2] = lambda / d[2];
    z[2] = w[2] * y[2] - c[1] * z[1];
    for (i = 3; i < m - 1; i++) {
        i1 = i - 1; i2 = i - 2;
        d[i] = w[i] + 6 * lambda - c[i1] * c[i1] * d[i1] - e[i2] * e[i2] * d[i2];
        c[i] = (-4 * lambda - d[i1] * c[i1] * e[i1]) / d[i];
        e[i] = lambda / d[i];
        z[i] = w[i] * y[i] - c[i1] * z[i1] - e[i2] * z[i2];
    }
    i1 = m - 2; i2 = m - 3;
    d[m - 1] = w[m - 1] + 5 * lambda - c[i1] * c[i1] * d[i1] - e[i2] * e[i2] * d[i2];
    c[m - 1] = (-2 * lambda - d[i1] * c[i1] * e[i1]) / d[m - 1];
    z[m - 1] = w[m - 1] * y[m - 1] - c[i1] * z[i1] - e[i2] * z[i2];
    i1 = m - 1; i2 = m - 2;
    d[m] = w[m] + lambda - c[i1] * c[i1] * d[i1] - e[i2] * e[i2] * d[i2];
    z[m] = (w[m] * y[m] - c[i1] * z[i1] - e[i2] * z[i2]) / d[m];
    z[m - 1] = z[m - 1] / d[m - 1] - c[m - 1] * z[m];
    for (i = m - 2; 1 <= i; i--)
        z[i] = z[i] / d[i] - c[i] * z[i + 1] - e[i] * z[i + 2];
}
```

```
/* Program 3. Smoothing and interpolation with any difference equation. */
/* Contribution to Graphic Gems IV */

/* Paul H. C. Eilers, DCMR Milieudienst Rijnmond, 's-Gravelandseweg 565,
   3119 XT Schiedam, The Netherlands, E-Mail: paul@dcmr.nl */

#define MMAX 100 /* choose the right length for your application */

typedef float vec[MMAX + 1];
typedef float vecn[6];

void asmooth(vec w, vec y, vec z, vecn a, float lambda, int m, int n)
/* Smoothing and interpolation with any difference equation of order <=5.
   Input: weights (w), data (y): vector from 1 to m.
   Input: smoothing parameter (lambda), length (m).
   Input: coefficients (a) and order of difference equation (n).
   Output: smoothed vector (z): vector from 1 to m. */
{
    static float b[MMAX + 1][6];
    static int v[MMAX + 1];
    int i, j, j1, j2, k, k1;
    float s;
    for (i = 1; i <= m + n; i++) {
        v[i] = 1; if ((i <= n) || (i > m)) v[i] = 0;
    }
    /* construct band matrix */
    for (i = 1; i <= m; i++) {
        j2 = m - i; if (j2 > n) j2 = n;
        for (j = 0; j <= j2; j++) {
            s = 0.0; if (j == 0) s = w[i] / lambda;
            for (k = j; k <= n; k++) s = s + v[i + k] * a[k] * a[k - j];
            b[i][j] = s;
        }
    }
    /* compute cholesky-decomposition */
    for (i = 1; i <= m; i++) {
        s = b[i][0];
        j1 = i - n; if (j1 < 1) j1 = 1;
        for (j = j1; j <= i - 1; j++) s = s - b[j][0] * b[j][i - j] * b[j][i - j];
        b[i][0] = (s);
        j2 = i + n; if (j2 > m) j2 = m;
        for (j = i + 1; j <= j2; j++) {
            s = b[i][j - i];
            k1 = j - n; if (k1 < 1) k1 = 1;
            for (k = k1; k <= i - 1; k++) s = s - b[k][0] * b[k][i - k] * b[k][j - k];
            b[i][j - i] = s / b[i][0];
        }
    }
    /* solve triangular systems */
    for (i = 1; i <= m; i++) {
        s = w[i] * y[i] / lambda;
        j1 = i - n; if (j1 < 1) j1 = 1;
        for (j = j1; j <= i - 1; j++) s = s - z[j] * b[j][i - j];
        z[i] = s;
    }
    for (i = m; i >= 1; i--) {
        s = z[i] / b[i][0];
        j2 = i + n; if (j2 > m) j2 = m;
        for (j = i + 1; j <= j2; j++) s = s - z[j] * b[i][j - i];
        z[i] = s;
    }
}
```

```
}

void pascalrow(vecn a, int n)
/* Construct row n of Pascal's triangle in a */
{
    int i, j;
    for (j = 0; j <= n; j++) a[j] = 0;
    a[0] = 1;
    for (j = 1; j <= n; j++) for (i = n; i >= 1; i--) a[i] = a[i] - a[i - 1];
}

void gensmooth(vec w, vec y, vec z, float lambda, int m, int n)
/* Smoothing and interpolation differences of order <=5.
   Input:  weights (w), data (y): vector from 1 to m.
   Input:  smoothing parameter (lambda), length (m).
   Input:  order of differences (n).
   Output: smoothed vector (z): vector from 1 to m. */
{
    vecn a;
    pascalrow(a, n);
    asmooth(w, y, z, a, lambda, m, n);
}
```

```
/*
 * C code from the article
 * "Faster Linear Interpolation"
 * by Steven Eker, steve@cs.city.ac.uk
 * in "Graphics Gems IV", Academic Press, 1994
 */

#include <stdio.h>

/*
 *      Test routine:
 *          Read in a, b, c
 *          Compute interpolations using old and new routines
 *          Check for differences
 *          Print interpolation
 */
main()
{
    short o[1000], o2[1000];
    int a, b, c, i;

    while(scanf("%d%d%d", &a, &b, &c) == 3){
        dec_var(a, b, c, o);
        linear(a, b, c, o2);
        printf("\n");
        for(i = 0; i <= a; i++){
            if(o[i] != o2[i]){
                printf("Error\n");
                printf("i = %d, o[i] = %d, o2[i] = %d\n", i, o[i], o2[i]);
                exit(1);
            }
            printf("%d ", o[i]);
        }
        printf("\n\n");
    }
}

/*
 *      Decision variable method
 */
dec_var(a, b, c, o)
int a, b, c;
short *o;
{
    int i1 = b / a, e1 = 2 * (b % a);
    int i2 = i1 + 1, e2 = e1 - 2 * a;
    int t = c, r = e1 - a;

    do{
        *o++ = t;
        if(r < 0){
            t += i1; r += e1;
        }
        else{
            t += i2; r += e2;
        }
    }while(--a >= 0);
}



/*
 *      Fast decision variable method
 */
```

```
linear(a, b, c, o)
int a, b, c;
short *o;
{
    int t, i1, i2;

    if(a > b){
        t = 0; i1 = b;
    }
    else{
        t = b / a; i1 = b % a;
    }
    i1 = (i1 << 16) + t;
    i2 = i1 - (a << 16) + 1;
    t = c - t - (((a + 1) >> 1) << 16);
    switch(a & 3){
    do{
        case 3:
            if((t += i1) >= 0) goto pos1;
neg1:
            *o++ = t;
        case 2:
            if((t += i1) >= 0) goto pos2;
neg2:
            *o++ = t;
        case 1:
            if((t += i1) >= 0) goto pos3;
neg3:
            *o++ = t;
        case 0:
            if((t += i1) >= 0) goto pos4;
neg4:
            *o++ = t;
    }while((a -= 4) >= 0);
    }
    return;
    do{
        if((t += i2) < 0) goto neg1;
pos1:
        *o++ = t;
        if((t += i2) < 0) goto neg2;
pos2:
        *o++ = t;
        if((t += i2) < 0) goto neg3;
pos3:
        *o++ = t;
        if((t += i2) < 0) goto neg4;
pos4:
        *o++ = t;
    }while((a -= 4) >= 0);
}
```

# Index of

## /pubs/tog/GraphicsGems/gemsv/ch6-3/

Name	Last modified	Size	Description
 <a href="#">_Parent Directory</a>			
 <a href="#">_pclipper.c</a>	29-Jun-00 08:24	5K	

```
#define LEFT          1
#define RIGHT         2
#define BOTTOM        4
#define TOP           8

#define SWAP(x, y)    { int _t = x; x = y; y = _t; }

#define OUTCODE(x, y, outcode, type)
{
    if (x < x1) outcode = LEFT, type = 1;
    else if (x > xr) outcode = RIGHT, type = 1;
    else outcode = type = 0;
    if (y < yb) outcode |= BOTTOM, type++;
    else if (y > yt) outcode |= TOP, type++;
}

#define CLIP(a1, a2, b1, da, da2, db2, as, bs, sa, sb,
            amin, AMIN, amax, AMAX, bmin, BMIN, bmax, BMAX)
{
    if (out1) {
        if (out1 & AMIN) { ca = db2 * (amin - a1); as = amin; }
        else if (out1 & AMAX) { ca = db2 * (a1 - amax); as = amax; }
        if (out1 & BMIN) { cb = da2 * (bmin - b1); bs = bmin; }
        else if (out1 & BMAX) { cb = da2 * (b1 - bmax); bs = bmax; }
        if (type1 == 2)
            out1 &= (ca + da < cb + !dir) ? ~(AMIN | AMAX) : ~(BMAX | BMIN);
        if (out1 & (AMIN | AMAX)) {
            cb = (ca + da - !dir) / da2;
            if (sb >= 0) { if ((bs = b1 + cb) > bmax) return; }
            else { if ((bs = b1 - cb) < bmin) return; }
            r += ca - da2 * cb;
        }
        else {
            ca = (cb - da + db2 - dir) / db2;
            if (sa >= 0) { if ((as = a1 + ca) > amax) return; }
            else { if ((as = a1 - ca) < amin) return; }
            r += db2 * ca - cb;
        }
    }
    else { as = a1; bs = b1; }
    alt = 0;
    if (out2) {
        if (type2 == 2) {
            ca = db2 * ((out2 & AMIN) ? a1 - amin : amax - a1);
            cb = da2 * ((out2 & BMIN) ? b1 - bmin : bmax - b1);
            out2 &= (cb + da < ca + dir) ? ~(AMIN | AMAX) : ~(BMIN | BMAX);
        }
        if (out2 & (AMIN | AMAX)) n = (out2 & AMIN) ? as - amin : amax - as;
        else { n = (out2 & BMIN) ? bs - bmin : bmax - bs; alt = 1; }
    }
    else n = (a2 >= as) ? a2 - as : as - a2;
}

void clip(int dir, int x1, int y1, int x2, int y2,
         int x1, int yb, int xr, int yt)
/*
 *   If dir = 0, round towards (x1, y1)
 *   If dir = 1, round towards (x2, y2)
 */
{
```

```
int adx, ady, adx2, ady2, sx, sy;
int out1, out2, type1, type2;
int ca, cb, r, diff, xs, ys, n, alt;

OUTCODE(x1, y1, out1, type1);
OUTCODE(x2, y2, out2, type2);
if (out1 & out2) return;
if ((type1 != 0 && type2 == 0) || (type1 == 2 && type2 == 1)){
    SWAP(out1, out2);
    SWAP(type1, type2);
    SWAP(x1, x2);
    SWAP(y1, y2);
    dir ^= 1;
}
xs = x1;
ys = y1;
sx = 1;
adx = x2 - x1;
if (adx < 0) { adx = -adx; sx = -1; }
sy = 1;
ady = y2 - y1;
if (ady < 0) { ady = -ady; sy = -1; }
adx2 = adx + adx;
ady2 = ady + ady;
if (adx >= ady) {
/*
 *      line is semi-horizontal
 */
    r = ady2 - adx - !dir;
    CLIP(x1, x2, y1, adx, adx2, ady2, xs, ys, sx, sy,
        x1, LEFT, xr, RIGHT, yb, BOTTOM, yt, TOP);
    diff = ady2 - adx2;
    if (alt) {
        for (;;) xs += sx) {          /* alternate Bresenham */
            plot(xs, ys);
            if (r >= 0 ) {
                if (--n < 0) break;
                r += diff;
                ys += sy;
            }
            else r += ady2;
        }
    }
    else{
        for (;;) xs += sx) {          /* standard Bresenham */
            plot(xs, ys);
            if (--n < 0) break;
            if (r >= 0 ) { r += diff; ys += sy; }
            else r += ady2;
        }
    }
}
else {
/*
 *      line is semi-vertical
 */
    r = adx2 - ady - !dir;
    CLIP(y1, y2, x1, ady, ady2, adx2, ys, xs, sy, sx,
        yb, BOTTOM, yt, TOP, x1, LEFT, xr, RIGHT);
    diff = adx2 - ady2;
    if (alt) {
```



```
    for (;;) ys += sy) {          /* alternate Bresenham */
        plot(xs, ys);
        if (r >= 0 ) {
            if (--n < 0) break;
            r += diff;
            xs += sx;
        }
        else r += adx2;
    }
}
else {
    for (;;) ys += sy) {          /* standard Bresenham */
        plot(xs, ys);
        if (--n < 0) break;
        if (r >= 0 ) { r += diff; xs += sx; }
        else r += adx2;
    }
}
}
```

```

/*****
 *
 *      contour.c
 *
 *      Author:  Tim Feldman
 *               Island Graphics Corporation
 *               modified 6/13/97 by Michael Beregov <michael@beregov.spb.ru>:
 *                   correction to Freeman chain termination (contours through
 *                   the same starting point were not producing the full contour)
 *
 *      Vectorizes the outline of an elevation contour in a set of sampled
 *      data.  Uses Freeman chain encoding.
 *****/

/*****
 *
 *      Include files
 *
 *****/

#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>

/*****
 *
 *      Constants
 *
 *****/

/****
 these are the coordinates of the edges of the
 rectangular array of sample points          ****/

#define X_MIN    0
#define X_MAX    7
#define Y_MIN    0
#define Y_MAX    7

/*****
 *
 *      Structure definitions
 *
 *****/

/****
 a Vector is one link in a simple chain that follows
 the edge of a contour from sample point to sample point ****/

struct vector
{
    short          dir;
    struct vector * next;
};

typedef struct vector  Vector;

/****
 these are the 'dir' values in a Vector:

    0      right
    1      right and up
    2      up

```

```
3      left and up
4      left
5      left and down
6      down
7      right and down      ***/
```

```
/*
 *
 *      Global data
 *
 */
```

```
/**      this points to the first member in the Freeman chain of vectors      ***/
```

```
Vector *      chain;
```

```
/**      these are the coordinates of the starting point
of the Freeman chain, in the array of sample points      ***/
```

```
int      start_x;
int      start_y;
```

```
/**      this is the elevation of the contour to be outlined
in the array of sample points      ***/
```

```
int      elev;
```

```
/**      this is the array of elevation samples for this example      ***/
```

```
int      sample[8][8] =
{
    100, 100, 100, 100, 100, 100, 100,  0,
    100, 100, 200, 200, 200, 200, 100, 100,
    100, 200, 200, 250, 255, 200, 100, 100,
    100, 200, 200, 250, 200, 200, 100, 100,
    100, 200, 200, 250, 200, 200, 100, 100,
    100, 200, 100, 200, 100, 200, 200, 100,
    100, 200, 100, 100, 100, 200, 200, 200,
    100, 100, 100, 100, 100, 100, 100, 200
};
```

```
/*
 *
 *      Procedures
 *
 */
```

```
/*
 *
 *      in_cont(x, y)
 *
 *      Determines whether the sample point at 'x, y' is within the contour
 *      being outlined.  Points outside of the array of samples are not
 *      in the contour.
 *
 *      Returns 0 if the point is not in the contour.
 *      Returns 1 if the point is      in the contour.
 *
 */
```

```
short
```

```
in_cont(x, y)

int      x;
int      y;

{
    if ( (x < X_MIN) || (x > X_MAX) || (y < Y_MIN) || (y > Y_MAX) )
        return(0);

    if (sample[x][y] == elev)
        return(1);
    else
        return(0);
}

/*****
 *
 *      probe(x, y, dir, new_x, new_y)
 *
 *      Checks a sample neighboring 'x, y' to see if it is in the contour
 *      being outlined.  'dir' specifies which neighboring sample to check.
 *      'new_x, new_y' always get the coordinates of the neighbor.
 *
 *      Returns 0 if the neighbor is not in the contour.
 *      Returns 1 if the neighbor is      in the contour.
 *
 *****/

short
probe(x, y, dir, new_x, new_y)

int      x;
int      y;
int      dir;
int *    new_x;
int *    new_y;

{
    /***      figure out coordinates of neighbor      ***/

    if ( (dir < 2) || (dir > 6) )
        ++x;

    if ( (dir > 2) && (dir < 6) )
        --x;

    if ( (dir > 0) && (dir < 4) )
        ++y;

    if (dir > 4)
        --y;

    /***      always return the new coordinates      ***/

    *new_x = x;
    *new_y = y;

    /***      determine if the new sample point is in the contour      ***/

    return(in_cont(x, y));
}
```

```

/*****
*
*   neighbor(x, y, last_dir, new_x, new_y)
*
*   Finds a neighbor of the sample at 'x, y' that is in the same
*   contour.  Always follows the contour in a clockwise direction.
*   'last_dir' is the direction that was used to get to 'x, y'
*   when it was found.  'new_x, new_y' always get the coordinates
*   of the neighbor.
*
*   This procedure should only be called for a sample that has at
*   least one neighbor in the same contour.
*
*   Returns the direction to the neighbor.
*
*****/

int
neighbor(x, y, last_dir, new_x, new_y)

int      x;
int      y;
int      last_dir;
int *    new_x;
int *    new_y;

{
int      n;
int      new_dir;

    /***   figure out where to start looking for a neighbor --
           always look ahead and to the left of the last direction

           if the last vector was 0
           then start looking at 1

           if the last vector was 1
           then start looking at 3

           if the last vector was 2
           then start looking at 3

           if the last vector was 3
           then start looking at 5

           if the last vector was 4
           then start looking at 5

           if the last vector was 5
           then start looking at 7

           if the last vector was 6
           then start looking at 7

           if the last vector was 7
           then start looking at 1           ***/

    if (last_dir & 0x01)
    {
        /***   last dir is odd --

```

```
        add 2 to it          ***/  
  
    new_dir = last_dir + 2;  
}  
else  
{  
    /***    last dir is even --  
        add 1 to it          ***/  
  
    new_dir = last_dir + 1;  
}  
  
/***    keep new_dir in the range 0 through 7    ***/  
  
if (new_dir > 7)  
    new_dir -= 8;  
  
/***    probe the neighbors, looking for one on the edge    ***/  
  
for (n = 0; n < 8; n++)  
{  
    if (probe(x, y, new_dir, new_x, new_y))  
    {  
        /***    found the next clockwise edge neighbor --  
            its coordinates have already been  
            stuffed into new_x, new_y    ***/  
  
        return(new_dir);  
    }  
    else  
    {  
        /***    check the next clockwise neighbor    ***/  
  
        if (--new_dir < 0)  
            new_dir += 8;  
    }  
}  
/***    should never exit routine by this way!    ***/  
}
```

```
/*  
*  
*    build()  
*  
*    Builds a Freeman chain of vectors describing the edge of the  
*    contour with elevation 'elev'.  Always follows the contour  
*    in a clockwise direction.  Uses 'start_x, start_y' as tentative  
*    starting point; modifies them to hold coordinates of first point  
*    in chain.  
*  
*    Returns 0 if unsuccessful.  
*    Returns 1 if    successful.  
*  
***/
```

```
short  
build()
```

```
{  
int    x;  
int    y;
```

```
int      new_x;
int      new_y;
int      dir;
int      last_dir;
Vector * new;
Vector * prev;

    /***    go left in the starting row until out of the contour    ***/

while (in_cont(start_x, start_y))
{
    --start_x;
}

    /***    move back right one point, to the leftmost edge
            in the contour, in that row    ***/

start_x++;

    /***    create the head of the chain    ***/

chain = (Vector *)NULL;
prev = (Vector *)NULL;

    /***    check if the starting point
            has no neighbors in the contour --
            the starting direction to check is arbitrary    ***/

x = start_x;
y = start_y;

dir = 0;

for ( ; ; )
{
    if (probe(x, y, dir, &new_x, &new_y))
    {
        /***    found a neighbor in that direction
                (its coordinates are in new_x, new_y
                but we don't use them here)    ***/

        break;
    }

    /***    try next direction    ***/

    if (++dir == 8)
    {
        /***    starting point has no neighbors --
                make the chain one vector long    ***/

        /***    allocate storage for the vector    ***/

        if ( (chain = (Vector *)malloc(sizeof(Vector))) == NULL)
        {
            printf("Insufficient memory available.\n");
            return(0);
        }

        /***    fill in the vector --
                the direction is arbitrary,
```

```

                                since the point is isolated    ***/

    chain->dir = 0;
    chain->next = (Vector *)NULL;

    return(1);
}

/**
 * get ready to follow the edge --
 * since we are at the left edge,
 * force initial probe to be to upper left
 * by initializing last_dir to 1    ***/

last_dir = 1;

/**
 * follow the edge clockwise, building a Freeman chain    ***/

for ( ; ; )
{
    /**
     * get the next point on the edge
     * and the vector to it    ***/

    dir = neighbor(x, y, last_dir, &new_x, &new_y);

    /**
     * maybe done with contour    ***/

    if ( (x == start_x) && (y == start_y) && (chain != NULL) )
        if (dir == chain->dir)
            return(1);

    /**
     * allocate storage for the new vector    ***/

    if ( (new = (Vector *)malloc(sizeof(Vector))) == NULL)
    {
        printf("Insufficient memory available.\n");
        return(0);
    }

    /**
     * fill in the new vector    ***/

    new->dir = dir;
    new->next = (Vector *)NULL;

    if (prev)
    {
        /**
         * add it to the existing chain    ***/

        prev->next = new;
    }
    else
    {
        /**
         * it is the first vector in the chain    ***/

        chain = new;
    }

    /**
     * else get ready to continue following the edge    ***/

    prev = new;
    x = new_x;
}
```



```
        y = new_y;
        last_dir = dir;
    }
}

/*****
 *
 *      report()
 *
 *      Follows the Freeman chain of vectors describing the edge of the
 *      contour with elevation 'elev'.  Reports the elevation, start point,
 *      direction vectors, and the number of vectors in the chain.
 *
 *****/

void
report()
{
    Vector *    p;
    int         n;

    printf("Elevation = %d\n", elev);
    printf("Start point (x, y) = %d, %d\n", start_x, start_y);

    p = chain;
    n = 0;

    while (p)
    {
        printf("%d\n", p->dir);
        p = p->next;
        ++n;
    }

    if (n > 1)
        printf("%d vectors in the chain.\n", n);
    else
        printf("1 vector in the chain.\n");
}

/*****
 *
 *      main()
 *
 *      Describes the outline of an elevation contour in the sampled data.
 *
 *      Returns 0 if successful.
 *      Returns -1 if unsuccessful.
 *
 *****/

int
main()
{
    /***      get the elevation of the contour to follow
               and get a starting point within the contour --

               they are given explicitly in this example, but
               in a real application the user would provide them,
```

or they would be found algorithmically

\*\*\* /

```
elev = 200;
start_x = 3;
start_y = 2;
```

```
/** follow the edge of the contour,
    building a Freeman chain of vectors
```

\*\*\* /

```
if (build())
{
```

```
    /** report the results    ***/
```

```
    report();
    return(0);
```

```
}
else
{
```

```
    /** failed    ***/
```

```
    return(-1);
```

```
}
```

```
}
```

```
#include "GraphicsGems.h"
#undef ON

/* Comparison macros */
#define LEFT -1 /* value to left of (less than) another */
#define ON 0 /* two values equal */
#define RIGHT 1 /* value to right of (greater than) another */

typedef struct endpoint
{ /* An a priori endpoint */
    short x, y;
} ENDPOINT;

typedef struct segment
{ /* A priori line segment with integer endpoints */
    ENDPOINT first, last; /* defined to be ordered from "first" to "last" */
} SEGMENT;

typedef struct param
{ /* Parameterized description of an intersection along a SEGMENT */
    long num, denom;
} PARAM;

typedef struct subsegment
{
    SEGMENT apriori; /* The a priori segment this subsegment falls on */
    PARAM ParOne, ParTwo; /* Parameterized description of intersection points */
} SUBSEGMENT;

typedef struct intpoint
{ /* Intersection point returned by SegIntersect */
    PARAM par[2]; /* par[0] is on the first segment, par[1] on the second */
    long a, b, c, d; /* storing these allows fast computation for direction of
                        crossing */
} INTPOINT;

/* All a priori endpoints must have coordinates falling in this range */
#define PointRange(X) (((X) > -16384) && ((X) < 16383))

#define SHORTMASK 0xffff /* Used by mult64 */

/* TRUE iff A and B have same signs. */
#define SAME_SIGNS(A, B) (((long)((unsigned long)A ^ (unsigned long)B)) >= 0)
```

```
/* Return the max value, storing the minimum value in min */
#define maxmin(x1, x2, min) (x1 >= x2 ? (min = x2, x1) : (min = x1, x2))

/* *****
   Below are two utility functions for implementing exact intersection
   calculation: SubsegIntersect and SideOfPoint. SubsegIntersect uses
   SegIntersect to do the bulk of its work.
   ***** */

/* *****
   Compute the intersection points between two a priori line segments.
   Return 0 if no intersection, 1 if intersects in a point,
   and 2 if intersecting lines are colinear. Parameter values for
   intersection point are returned in ipt.

   Entry: s1, s2: the line segments
   Exit:  ipt: the intersection point.
   ***** */
int SegIntersect(SEGMENT *s1, SEGMENT *s2, INTPOINT *ipt)
{
    long a, b, c, d, tdet, sdet, det; /* parameter calculation variables */
    short max1, max2, min1, min2; /* bounding box check variables */
    ENDPOINT p1, p2, q1, q2; /* dereference a priori endpoints */

    p1 = s1->first;    p2 = s1->last;
    q1 = s2->first;    q2 = s2->last;

    /* First make the bounding box test. */
    max1 = maxmin(p1.x, p2.x, min1);
    max2 = maxmin(q1.x, q2.x, min2);
    if((max1 < min2) || (min1 > max2)) return(0); /* no intersection */
    max1 = maxmin(p1.y, p2.y, min1);
    max2 = maxmin(q1.y, q2.y, min2);
    if((max1 < min2) || (min1 > max2)) return(0); /* no intersection */

    /* See if the endpoints of the second segment lie on the opposite
       sides of the first. If not, return 0. */
    a = (long)(q1.x - p1.x) * (long)(p2.y - p1.y) -
        (long)(q1.y - p1.y) * (long)(p2.x - p1.x);
    b = (long)(q2.x - p1.x) * (long)(p2.y - p1.y) -
        (long)(q2.y - p1.y) * (long)(p2.x - p1.x);
    if(a!=0 && b!=0 && SAME_SIGNS(a, b)) return(0);

    /* See if the endpoints of the first segment lie on the opposite
       sides of the second. If not, return 0. */
    c = (long)(p1.x - q1.x) * (long)(q2.y - q1.y) -
        (long)(p1.y - q1.y) * (long)(q2.x - q1.x);
    d = (long)(p2.x - q1.x) * (long)(q2.y - q1.y) -
        (long)(p2.y - q1.y) * (long)(q2.x - q1.x);
    if(c!=0 && d!=0 && SAME_SIGNS(c, d) ) return(0);

    /* At this point each segment meets the line of the other. */
    det = a - b;
    if(det == 0) return(2); /* The segments are colinear. Determining
        colinear intersection parameters would be tedious and not instructive. */

    /* The segments intersect since each segment crosses the other's line;
       however, since the lines are not parallel, either a or b is not
       zero. Similarly either c or d is not zero. */
    tdet = -c;    sdet = a;
```

```
    if(det < 0) /* The denominator of the parameter must be positive. */
        { det = -det; sdet = -sdet; tdet = -tdet; }
    ipt->a = a; ipt->b = b;
    ipt->c = c; ipt->d = d;
    ipt->par[0].num = tdet;
    ipt->par[0].denom = det;
    ipt->par[1].num = sdet;
    ipt->par[1].denom = det;
    return(1);
}

/* *****
Returns the number of intersection points (0, 1 or 2) between line
subsegments ss1 and ss2. Note that 2 intersection points would
represent the endpoints of overlap between two colinear line segments.
If there is a single intersection point, it is returned in ipt.

Entry: ss1, ss2: the line subsegments.
Exit:  ipt: the intersection point (if exactly one).
***** */
int SubsegIntersect(SUBSEGMENT *ss1, SUBSEGMENT *ss2, INTPOINT *ipt)
{
    int i; /* Number of intersection points */

    /* intersect a priori segments */
    i = SegIntersect(&ss1->apriori, &ss2->apriori, ipt);
    if(i != 1) return(i); /* either no intersection or colinear */
    /* one intersection point - check if it falls outside of subsegments */
    if(LessThan(&(ipt->par[0]), &(ss1->ParOne)) ||
        GreaterThan(&(ipt->par[0]), &(ss1->ParTwo)))
        return(0); /* A priori segments intersect, but subsegments do not */
    if(LessThan(&(ipt->par[1]), &(ss2->ParOne)) ||
        GreaterThan(&(ipt->par[1]), &(ss2->ParTwo)))
        return(0); /* A priori segments intersect, but subsegments do not */
    return(1);
}

/* *****
64 bit multiplication function.
Multiply two long integers in1 and in2, returning the result in out.
***** */
void mult64(long in1, long in2, unsigned long out[2])
{
    unsigned short *x, *y, *z;
    unsigned long temp;

    x = (unsigned short *) &in1; y = (unsigned short *) &in2;
    z = (unsigned short *) out;
    temp = x[0] * y[0];
    z[1] = temp >> 16; z[0] = temp & SHORTMASK;
    temp = x[0] * y[1];
    z[2] = temp >> 16; z[1] += temp & SHORTMASK;
    z[2] += z[1] >>16; z[1] = z[1] & SHORTMASK;
    temp = x[1] * y[0];
    z[2] += temp >> 16; z[1] += temp & SHORTMASK;
    z[2] += z[1] >>16; z[1] = z[1] & SHORTMASK;
    z[3] = z[2] >>16; z[2] = z[2] & SHORTMASK;
    temp = x[1] * y[1];
    z[3] += temp >> 16; z[2] += temp & SHORTMASK;
    z[3] += z[2] >>16; z[2] = z[2] & SHORTMASK;
```

}

/\* \*\*\*\*\*

Comparison primitive to test  $par1 \leq par2$ .Return LEFT if  $par1 < par2$ ; return ON if  $par1 = par2$ ;return RIGHT if  $par1 > par2$ .

\*\*\*\*\* \*/

int CompPrim(PARAM \*par1, PARAM \*par2)

{

unsigned long r1[2],r2[2];

mult64(par1-&gt;num, par2-&gt;denom, r1);

mult64(par2-&gt;num, par1-&gt;denom, r2);

if(r1[1] != r2[1]) { if(r1[1] &lt; r2[1]) return(LEFT); else return(RIGHT); }

if(r1[0] != r2[0]) { if(r1[0] &lt; r2[0]) return(LEFT); else return(RIGHT); }

return(ON);

}

/\* \*\*\*\*\*

Helper function for all parameter comparisons.

    Returns LEFT if  $par1 < par2$ , ON if  $par1 = par2$ , and RIGHT otherwise.

Denominators must be positive.

\*\*\*\*\* \*/

int CompHelp(PARAM \*par1, PARAM \*par2)

{

PARAM tpar1, tpar2;

tpar1 = \*par1;    tpar2 = \*par2;

if(tpar1.num &lt; 0)

{

if(tpar2.num &gt;= 0) return(LEFT);

tpar1.num = -tpar1.num;    tpar2.num = -tpar2.num;

return(CompPrim(&amp;tpar2, &amp;tpar1));

}

if(tpar2.num &lt; 0) return(RIGHT);

return(CompPrim(&amp;tpar1, &amp;tpar2));

}

/\* \*\*\*\*\*

    Returns TRUE if  $par1 < par2$  and FALSE otherwise;

\*\*\*\*\* \*/

boolean LessThan(PARAM \*par1, PARAM \*par2)

{

double x1, x2;

x1 = ((double)par1-&gt;num) \* par2-&gt;denom;

x2 = ((double)par2-&gt;num) \* par1-&gt;denom;

if(x1 != x2)

if(x1 &lt; x2) return(TRUE);

else return(FALSE);

return(CompHelp(par1, par2) == LEFT);

}

/\* \*\*\*\*\*

    Returns TRUE if  $par1 > par2$  and FALSE otherwise;

\*\*\*\*\* \*/

boolean GreaterThan(PARAM \*par1, PARAM \*par2)

```

{
    double x1, x2;

    x1 = ((double)par1->num) * par2->denom;
    x2 = ((double)par2->num) * par1->denom;
    if(x1 != x2)
        if(x1 > x2) return(TRUE);
        else return(FALSE);
    return(CompHelp(par1, par2) == RIGHT);
}

/* *****
   Returns TRUE if par1 = par2 and FALSE otherwise;
   ***** */
boolean Equal(PARAM *par1, PARAM *par2)
{
    double x1, x2;

    x1 = ((double)par1->num) * par2->denom;
    x2 = ((double)par2->num) * par1->denom;
    if(x1 != x2) return(FALSE);
    return(CompHelp(par1, par2) == ON);
}




/* *****
   Determine if a given point is to the left of, right of,
   or on segment s1. The point is defined as the position (inpt)
   along a line segment (in). Returns LEFT, RIGHT or ON respectively.

   Entry:
       in: line segment defining intersection point
       inpt: intersection point parameter along "in"
       s1: segment to check on which side of
   ***** */
int SideOfPoint(SEGMENT *in, PARAM *inpar, SEGMENT *s1)
{
    PARAM par; /* build a fraction for use in comparison against inpar */
    long delx, dely; /* multiplications must be done as longs */
    short *x1, *x2, *y1, *y2, *x3, *x4, *y3, *y4; /* alias names */

    x1 = &s1->first.x; y1 = &s1->first.y; x2 = &s1->last.x; y2 = &s1->last.y;
    x3 = &in->first.x; y3 = &in->first.y; x4 = &in->last.x; y4 = &in->last.y;
    delx = *x2 - *x1; dely = *y2 - *y1;
    par.denom = (*y4 - *y3) * delx - (*x4 - *x3) * dely;
    par.num = (*x3 - *x1) * dely - (*y3 - *y1) * delx;
    if(par.denom > 0) return(CompHelp(&par, inpar));
    else if(par.denom < 0) /* switch signs and compare */
        { par.denom = -par.denom; par.num = -par.num;
          return(CompHelp(inpar, &par)); }
    else if(par.num > 0) return(RIGHT);
    else if(par.num < 0) return(LEFT);
    else return(ON);
}

```

# Index of /pubs/tog/GraphicsGems/gemsiv/convex\_test/

Name	Last modified	Size	Description
 <a href="#">Parent Directory</a>			
 <a href="#">convex.c</a>	29-Jun-00 08:19	3K	
 <a href="#">convex_opt.c</a>	29-Jun-00 08:19	3K	



```
/*
 * C code from the article
 * "Testing the Convexity of a Polygon"
 * by Peter Schorn and Frederick Fisher,
 * (schorn@inf.ethz.ch, fred@kpc.com)
 * in "Graphics Gems IV", Academic Press, 1994
 */

/* Program to Classify a Polygon's Shape */

#include <stdio.h>

typedef enum { NotConvex, NotConvexDegenerate,
              ConvexDegenerate, ConvexCCW, ConvexCW } PolygonClass;

typedef struct { double x, y; } Point2d;

int WhichSide(p, q, r)          /* Given a directed line pq, determine */
Point2d      p, q, r;          /* whether qr turns CW or CCW.          */
{
    double result;
    result = (p.x - q.x) * (q.y - r.y) - (p.y - q.y) * (q.x - r.x);
    if (result < 0) return -1; /* q lies to the left  (qr turns CW).    */
    if (result > 0) return  1; /* q lies to the right (qr turns CCW).    */
    return 0;                 /* q lies on the line from p to r. */
}

int Compare(p, q)               /* Lexicographic comparison of p and q */
Point2d      p, q;
{
    if (p.x < q.x) return -1; /* p is less than q.              */
    if (p.x > q.x) return  1; /* p is greater than q.           */
    if (p.y < q.y) return -1; /* p is less than q.              */
    if (p.y > q.y) return  1; /* p is greater than q.           */
    return 0;                 /* p is equal to q.               */
}

int GetPoint(f, p)              /* Read p's x- and y-coordinate from f */
FILE      *f;                  /* and return true, iff successful.    */
Point2d *p;
{
    return !feof(f) && (2 == fscanf(f, "%lf%lf", &(p->x), &(p->y)));
}

int GetDifferentPoint(f, previous, next)
FILE      *f;                  /* Read next point into 'next' until it */
Point2d previous, *next;        /* is different from 'previous' and      */
{
    /* return true iff successful.      */
    int eof;
    while((eof = GetPoint(f, next)) && (Compare(previous, *next) == 0));
    return eof;
}

/* CheckTriple tests three consecutive points for change of direction
 * and for orientation.
 */
#define CheckTriple \
    if ( (thisDir = Compare(second, third)) == -curDir ) \
        ++dirChanges; \
    curDir = thisDir; \
    if ( thisSign = WhichSide(first, second, third) ) { \
```

```
        if ( angleSign == -thisSign )           \
            return NotConvex;                   \
        angleSign = thisSign;                   \
    }                                           \
    first = second; second = third;

/* Classify the polygon vertices on file 'f' according to: 'NotConvex' */
/* 'NotConvexDegenerate', 'ConvexDegenerate', 'ConvexCCW', 'ConvexCW'. */
PolygonClass ClassifyPolygon(f)
FILE          *f;
{
    int          curDir, thisDir, thisSign, angleSign = 0, dirChanges = 0;
    Point2d      first, second, third, saveFirst, saveSecond;

    if ( !GetPoint(f, &first) || !GetDifferentPoint(f, first, &second) )
        return ConvexDegenerate;
    saveFirst = first; saveSecond = second;
    curDir = Compare(first, second);
    while( GetDifferentPoint(f, second, &third) ) {
        CheckTriple;
    }
    /* Must check that end of list continues back to start properly */
    if ( Compare(second, saveFirst) ) {
        third = saveFirst; CheckTriple;
    }
    third = saveSecond; CheckTriple;

    if ( dirChanges > 2 ) return angleSign ? NotConvex : NotConvexDegenerate;
    if ( angleSign > 0 ) return ConvexCCW;
    if ( angleSign < 0 ) return ConvexCW;
    return ConvexDegenerate;
}

int main()
{
    switch ( ClassifyPolygon(stdin) ) {
        case NotConvex:          fprintf( stderr, "Not Convex\n" );
                                exit(-1); break;
        case NotConvexDegenerate: fprintf( stderr, "Not Convex Degenerate\n" );
                                exit(-1); break;
        case ConvexDegenerate:    fprintf( stderr, "Convex Degenerate\n" );
                                exit( 0 ); break;
        case ConvexCCW:          fprintf( stderr, "Convex Counter-Clockwise\n" );
                                exit( 0 ); break;
        case ConvexCW:           fprintf( stderr, "Convex Clockwise\n" );
                                exit( 0 ); break;
    }
}
```

```
/*
 * C code from the article
 * "Testing the Convexity of a Polygon"
 * by Peter Schorn and Frederick Fisher,
 * (schorn@inf.ethz.ch, fred@kpc.com)
 * in "Graphics Gems IV", Academic Press, 1994
 */

/* Reasonably Optimized Routine to Classify a Polygon's Shape */

/*
.. code omitted which reads polygon, stores in an array, and calls
   classifyPolygon2()
*/

typedef enum { NotConvex, NotConvexDegenerate,
               ConvexDegenerate, ConvexCCW, ConvexCW } PolygonClass;

typedef double Number;          /* float or double */

#define ConvexCompare(delta) \
    ( (delta[0] > 0) ? -1 :    /* x coord diff, second pt > first pt */ \
      (delta[0] < 0) ? 1 :    /* x coord diff, second pt < first pt */ \
      (delta[1] > 0) ? -1 :    /* x coord same, second pt > first pt */ \
      (delta[1] < 0) ? 1 :    /* x coord same, second pt < first pt */ \
      0 )                    /* second pt equals first point */

#define ConvexGetPointDelta(delta, pprev, pcur ) \
    /* Given a previous point 'pprev', read a new point into 'pcur' */ \
    /* and return delta in 'delta'. */ \
    pcur = pVert[iread++]; \
    delta[0] = pcur[0] - pprev[0]; \
    delta[1] = pcur[1] - pprev[1];

#define ConvexCross(p, q) p[0] * q[1] - p[1] * q[0];

#define ConvexCheckTriple \
    if ( (thisDir = ConvexCompare(dcur)) == -curDir ) { \
        ++dirChanges; \
        /* The following line will optimize for polygons that are */ \
        /* not convex because of classification condition 4, */ \
        /* otherwise, this will only slow down the classification. */ \
        /* if ( dirChanges > 2 ) return NotConvex; */ \
    } \
    curDir = thisDir; \
    cross = ConvexCross(dprev, dcur); \
    if ( cross > 0 ) { if ( angleSign == -1 ) return NotConvex; \
                     angleSign = 1; \
    } \
    else if (cross < 0) { if (angleSign == 1) return NotConvex; \
                       angleSign = -1; \
    } \
    pSecond = pThird;          /* Remember ptr to current point. */ \
    dprev[0] = dcur[0];        /* Remember current delta. */ \
    dprev[1] = dcur[1];

classifyPolygon2( nvert, pVert )
int nvert;
Number pVert[][2];
/* Determine polygon type. return one of:
```

```
*      NotConvex, NotConvexDegenerate,
*      ConvexCCW, ConvexCW, ConvexDegenerate
*/
{
    int      curDir, thisDir, dirChanges = 0,
            angleSign = 0, iread ;
    Number   *pSecond, *pThird, *pSaveSecond, dprev[2], dcur[2], cross;

    /* if ( nvert <= 0 ) return error;          if you care */

    /* Get different point, return if less than 3 diff points. */
    if ( nvert < 3 ) return ConvexDegenerate;
    iread = 1;
    while ( 1 ) {
        ConvexGetPointDelta( dprev, pVert[0], pSecond );
        if ( dprev[0] || dprev[1] ) break;
        /* Check if out of points. Check here to avoid slowing down cases
         * without repeated points.
         */
        if ( iread >= nvert ) return ConvexDegenerate;
    }

    pSaveSecond = pSecond;

    curDir = ConvexCompare(dprev);          /* Find initial direction */

    while ( iread < nvert ) {
        /* Get different point, break if no more points */
        ConvexGetPointDelta(dcur, pSecond, pThird );
        if ( dcur[0] == 0.0 && dcur[1] == 0.0 ) continue;

        ConvexCheckTriple;                  /* Check current three points */
    }

    /* Must check for direction changes from last vertex back to first */
    pThird = pVert[0];                      /* Prepare for 'ConvexCheckTriple' */
    dcur[0] = pThird[0] - pSecond[0];
    dcur[1] = pThird[1] - pSecond[1];
    if ( ConvexCompare(dcur) ) {
        ConvexCheckTriple;
    }

    /* and check for direction changes back to second vertex */
    dcur[0] = pSaveSecond[0] - pSecond[0];
    dcur[1] = pSaveSecond[1] - pSecond[1];
    ConvexCheckTriple;                      /* Don't care about 'pThird' now */

    /* Decide on polygon type given accumulated status */
    if ( dirChanges > 2 )
        return angleSign ? NotConvex : NotConvexDegenerate;

    if ( angleSign > 0 ) return ConvexCCW;
    if ( angleSign < 0 ) return ConvexCW;
    return ConvexDegenerate;
}
```

```
/*
A Fast HSL-to-RGB Transform
by Ken Fishkin
from "Graphics Gems", Academic Press, 1990
*/

#include <math.h>
#include <stdio.h>
#include "GraphicsGems.h"

/*
 * RGB-HSL transforms.
 * Ken Fishkin, Pixar Inc., January 1989.
 */

/*
 * given r,g,b on [0 ... 1],
 * return (h,s,l) on [0 ... 1]
 */
void
RGB_to_HSL      (r,g,b,h,s,l)
double  r,g,b;
double *h, *s, *l;
{
    double v;
    double m;
    double vm;
    double r2, g2, b2;

    v = MAX(r,g);
    v = MAX(v,b);
    m = MIN(r,g);
    m = MIN(m,b);

    if ((*l = (m + v) / 2.0) <= 0.0) return;
    if ((*s = vm = v - m) > 0.0) {
        *s /= (*l <= 0.5) ? (v + m) :
            (2.0 - v - m) ;
    } else
        return;

    r2 = (v - r) / vm;
    g2 = (v - g) / vm;
    b2 = (v - b) / vm;

    if (r == v)
        *h = (g == m ? 5.0 + b2 : 1.0 - g2);
    else if (g == v)
        *h = (b == m ? 1.0 + r2 : 3.0 - b2);
    else
        *h = (r == m ? 3.0 + g2 : 5.0 - r2);

    *h /= 6;
}

/*
 * given h,s,l on [0..1],
 * return r,g,b on [0..1]
 */
void
```










```
HSL_to_RGB(h,s,l,r,g,b)
double  h,s,l;
double  *r, *g, *b;
{
    double v;

    v = (l <= 0.5) ? (l * (1.0 + s)) : (l + s - l * s);
    if (v <= 0) {
        *r = *g = *b = 0.0;
    } else {
        double m;
        double sv;
        int sextant;
        double fract, vsf, mid1, mid2;

        m = l + l - v;
        sv = (v - m) / v;
        h *= 6.0;
        sextant = h;
        fract = h - sextant;
        vsf = v * sv * fract;
        mid1 = m + vsf;
        mid2 = v - vsf;
        switch (sextant) {
            case 0: *r = v; *g = mid1; *b = m; break;
            case 1: *r = mid2; *g = v; *b = m; break;
            case 2: *r = m; *g = v; *b = mid1; break;
            case 3: *r = m; *g = mid2; *b = v; break;
            case 4: *r = mid1; *g = m; *b = v; break;
            case 5: *r = v; *g = m; *b = mid2; break;
        }
    }
}
```

# Index of

## /pubs/tog/GraphicsGems/gemsiii/accurate\_scan/

Name	Last modified	Size	Description
 <a href="#">_ Parent Directory</a>			
 <a href="#">_ Makefile</a>	29-Jun-00 08:16	3K	
 <a href="#">_ dblfixpoint.c</a>	29-Jun-00 08:16	4K	
 <a href="#">_ exhaust.c</a>	29-Jun-00 08:16	3K	
 <a href="#">_ fixpoint.c</a>	29-Jun-00 08:16	5K	
 <a href="#">_ fixpoint.h</a>	29-Jun-00 08:16	1K	
 <a href="#">_ test.c</a>	29-Jun-00 08:16	4K	
 <a href="#">_ tri.c</a>	29-Jun-00 08:16	5K	
 <a href="#">_ tri.data</a>	29-Jun-00 08:16	1K	

```
# Accurate Polygon Scan Conversion Using Half-Open Intervals,
#   by Kurt Fleischer and David Salesin
#
# don't put any -g or -O here -- it's taken care of elsewhere
# -Ac
CFLAGS= -Aa

OBJ =    fixpoint.o dblfixpoint.o tri.o

GRAPHICSLIB = -L /usr/lib/X11R4 -lXwindow -lsb -lXhp11 -lX11 -ldld
# non-shared library, non-X version:
#GRAPHICSLIB = -ldd98721 -ldd98731 -lsb1 -lsb2

OBJO= $(OBJ:.o=.oo)
OBJG= $(OBJ:.o=.og)
OBJP= $(OBJ:.o=.op)

default: tri

depend: ;mkmf && ed - makefile < Make.mkmf.ed

clean:  ;rm -f core tri a.out *.o? *.o *~ *.og *.oo *.op

# debugging version
tri:  $(OBJG) test.og
      cc -g -o $@ $(OBJG) test.og $(GRAPHICSLIB) -lm

# optimized version
trio: $(OBJO) test.oo
      cc -O -o $@ $(OBJO) test.oo $(GRAPHICSLIB) -lm

# exhaustive test program (lores)
exhaust: $(OBJO) exhaust.oo
      cc -O -o $@ $(OBJO) exhaust.oo $(GRAPHICSLIB) -lm

# debugging version
dtri: gemfile.og test.og addendum.og
      cc -g -o $@ gemfile.og test.og addendum.og \
      -l

# exhaustive test program (lores)
dexhaust: gemfile.og exhaust.oo
      cc -O -o $@ gemfile.og exhaust.oo addendum.og \
      $(GRAPHICSLIB) -lm

#
# Compilation rules
#
.SUFFIXES:
.SUFFIXES: .op .og .oo .c

.c.og:
      cc -c -g $(CFLAGS) $< && mv -f $*.o $*.og

.c.oo:
      cc -c -O $(CFLAGS) $< && mv -f $*.o $*.oo

.c.op:
      cc -c -O -G $(CFLAGS) $< && mv -f $*.o $*.op
```



```
###
dblfixpoint.og: /usr/include/stdio.h /usr/include/sys/stdsyms.h fixpoint.h
exhaust.og: /usr/include/stdio.h /usr/include/sys/stdsyms.h \
            /usr/include/stdlib.h /usr/include/math.h /usr/include/starbase.c.h \
            fixpoint.h
fixpoint.og: /usr/include/stdio.h /usr/include/sys/stdsyms.h \
            /usr/include/stdlib.h /usr/include/math.h fixpoint.h
formatted.tri.og: /usr/include/stdio.h /usr/include/sys/stdsyms.h \
                /usr/include/math.h
gemfile.og: /usr/include/stdio.h /usr/include/sys/stdsyms.h \
            /usr/include/math.h fixpoint.h
test.og: /usr/include/stdio.h /usr/include/sys/stdsyms.h /usr/include/stdlib.h \
         /usr/include/math.h /usr/include/starbase.c.h
tri.og: /usr/include/stdio.h /usr/include/sys/stdsyms.h /usr/include/math.h \
        fixpoint.h
dblfixpoint.oo: /usr/include/stdio.h /usr/include/sys/stdsyms.h fixpoint.h
exhaust.oo: /usr/include/stdio.h /usr/include/sys/stdsyms.h \
            /usr/include/stdlib.h /usr/include/math.h /usr/include/starbase.c.h \
            fixpoint.h
fixpoint.oo: /usr/include/stdio.h /usr/include/sys/stdsyms.h \
            /usr/include/stdlib.h /usr/include/math.h fixpoint.h
formatted.tri.oo: /usr/include/stdio.h /usr/include/sys/stdsyms.h \
                /usr/include/math.h
gemfile.oo: /usr/include/stdio.h /usr/include/sys/stdsyms.h \
            /usr/include/math.h fixpoint.h
test.oo: /usr/include/stdio.h /usr/include/sys/stdsyms.h /usr/include/stdlib.h \
         /usr/include/math.h /usr/include/starbase.c.h
tri.oo: /usr/include/stdio.h /usr/include/sys/stdsyms.h /usr/include/math.h \
        fixpoint.h
dblfixpoint.op: /usr/include/stdio.h /usr/include/sys/stdsyms.h fixpoint.h
exhaust.op: /usr/include/stdio.h /usr/include/sys/stdsyms.h \
            /usr/include/stdlib.h /usr/include/math.h /usr/include/starbase.c.h \
            fixpoint.h
fixpoint.op: /usr/include/stdio.h /usr/include/sys/stdsyms.h \
            /usr/include/stdlib.h /usr/include/math.h fixpoint.h
formatted.tri.op: /usr/include/stdio.h /usr/include/sys/stdsyms.h \
                /usr/include/math.h
gemfile.op: /usr/include/stdio.h /usr/include/sys/stdsyms.h \
            /usr/include/math.h fixpoint.h
test.op: /usr/include/stdio.h /usr/include/sys/stdsyms.h /usr/include/stdlib.h \
         /usr/include/math.h /usr/include/starbase.c.h
tri.op: /usr/include/stdio.h /usr/include/sys/stdsyms.h /usr/include/math.h \
        fixpoint.h
```

```
#include <stdio.h>
#include "fixpoint.h"

static int verbose = 0;

/* The dblfixpoint representation is signed magnitude, with
   hi is 2n bits of integer (n <= 16)
   lo is 2m bits of fraction (m <=16)
   number = hi + lo * 2^(-2m)
*/

int fp_dblnegative(dblfixpoint x) { return (x.neg); }

dblfixpoint fp_dblnegate(dblfixpoint x)
{ x.neg = !x.neg; return x; }

dblfixpoint fp_dblmultiply(fixpoint x, fixpoint y)
{
    dblfixpoint answer;
    unsigned int a, b, c, d;
    unsigned int xhi, xlo, yhi, ylo;

    xhi = (x<0) ? -fp_integer(x) : fp_integer(x);
    yhi = (y<0) ? -fp_integer(y) : fp_integer(y);
    xlo = fp_fraction(x);
    ylo = fp_fraction(y);

    a = xhi * yhi;
    b = xhi * ylo;
    c = xlo * yhi;
    d = xlo * ylo;

    a += ((b & HIMASK) >> LOBITS);
    a += ((c & HIMASK) >> LOBITS);
    b = (b & LOMASK) + (c & LOMASK) + ((d & HIMASK) >> LOBITS);
    a += ((b & HIMASK) >> LOBITS);
    answer.hi = a;
    answer.lo = ((b & LOMASK) << LOBITS) | (d & LOMASK);
    answer.neg = 0;

    if (((x<0) && (y>0)) || ((x>0) && (y<0))) answer = fp_dblnegate(answer);
    return(answer);
}

int fp_dbllssththan(dblfixpoint x, dblfixpoint y)
{
    int bothneg, xneg, yneg;
    xneg = fp_dblnegative(x);
    yneg = fp_dblnegative(y);
    bothneg = xneg && yneg;
    if (xneg && !yneg) return 1;
    if (yneg && !xneg) return 0;
    if (bothneg) {
        x = fp_dblnegate(x);
        y = fp_dblnegate(y);
    }
    if (x.hi < y.hi) return !bothneg;
    if (y.hi < x.hi) return bothneg;
    if (x.lo < y.lo) return !bothneg;
    return bothneg;
}
```

```
}

fixpoint fp_trunc(dblfixpoint a)
{
    int hi = (a.hi << LOBITS);
    int lo = (a.lo >> LOBITS);

    if ((hi >> LOBITS) != a.hi) {
        printf("fp_trunc() Overflow converting hibits 0x%08x to 0x%08x fixpoint in
(%d,%d) bits\n", a.hi, hi, HIBITS, LOBITS);
    }

    if (a.neg)
        return -(hi + lo);
    else
        return (hi + lo);
}

unsigned int add_with_carry(unsigned int a, unsigned int b,
                           int carry_in,
int *carry_out)
{
    int ahi, bhi;
    unsigned int answer;

    *carry_out = 0;

    ahi = (0x80000000 & a) ? 1 : 0;    /* Is high bit on? */
    bhi = (0x80000000 & b) ? 1 : 0;

    a &= 0x7fffffff;
    b &= 0x7fffffff;

    answer = a + b + carry_in;          /* this can't overflow (I think) */
    if (answer & 0x80000000) {
        if (ahi && bhi) *carry_out = 1;    /* Hi bit is on, leave it on */
        else if (ahi || bhi) {             /* Turn hi bit off, turn on carry */
            *carry_out = 1;
            answer &= 0x7fffffff;
        }
    }
    else {
        /* else (if answer high bit is off) */
        if (ahi && bhi) *carry_out = 1;
        else if (ahi || bhi) answer |= 0x80000000;
    }
    return(answer);
}

dblfixpoint fp_dbladd(dblfixpoint a, dblfixpoint b)
{
    dblfixpoint answer;

    if (verbose) printf("\nfpdbladd\n");

    if ((a.neg && b.neg) || (!a.neg && !b.neg)) {
        int carry;
        answer.neg = a.neg;
        answer.lo = add_with_carry(a.lo, b.lo, 0, &carry);
    }
}
```

```
if (verbose && ((answer.lo >> 2*LOBITS) || carry))
    printf("  Lobits overflow (ok, put into hibits).\n");

/*This puts lobit overflow into .hi, and removes from .lo */
answer.hi = a.hi + b.hi + (answer.lo >> 2*LOBITS) + carry;
answer.lo &= (LOMASK | (LOMASK << LOBITS));
}
else {
    if (a.neg) {
        unsigned int tmp;
        if (verbose) printf("\tCase: a.neg, b.pos, a<b, swapping\n");
        a.neg = 0; b.neg = 1;
        tmp = a.hi; a.hi = b.hi; b.hi = tmp;
        tmp = a.lo; a.lo = b.lo; b.lo = tmp;
    }
    if (a.hi < b.hi) {
        if (a.lo <= b.lo) {
            if (verbose) printf("\tCase: a.pos, b.neg, a.hi<b.hi, a.lo<=b.lo \n");
            answer.neg = 1;
            answer.lo = b.lo - a.lo;
            answer.hi = b.hi - a.hi;
        }
        else {
            if (verbose) printf("\tCase: a.pos, b.neg, a.hi<b.hi, a.lo > b.lo \n");
            answer.neg = 1;
            answer.lo = (b.lo - a.lo) & (LOMASK | (LOMASK << LOBITS));
            answer.hi = b.hi - a.hi - 1;
        }
    }
    else if ((a.hi == b.hi) && (a.lo < b.lo)) {
        if (verbose) printf("\tCase: a.pos, b.neg, a.hi = b.hi, a.lo < b.lo \n");
        answer.neg = 1;
        answer.lo = b.lo - a.lo;
        answer.hi = 0;
    }
    else {
        if (a.lo >= b.lo) {
            if (verbose) printf("\tCase: a.pos, b.neg, a.hi > b.hi, a.lo >=
b.lo\n");

            answer.neg = 0;
            answer.lo = a.lo - b.lo;
            answer.hi = a.hi - b.hi;
        }
        else {
            if (verbose) printf("\tCase: a.pos, b.neg, a.hi > b.hi, a.lo <
b.lo\n");

            answer.neg = 0;
            answer.lo = (a.lo - b.lo) & (LOMASK | (LOMASK << LOBITS));
            answer.hi = a.hi - b.hi - 1;
        }
    }
}
return(answer);
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <starbase.c.h>

#include "fixpoint.h"

extern void printnbits();
extern void subpixel_triangle(fixpoint x0, fixpoint y0,
                             fixpoint x1, fixpoint y1,
                             fixpoint x2, fixpoint y2);
extern void triangle(int x0, int y0, int x1, int y1, int x2, int y2);

/*
   Exhaustively test all cases of (subpixel) triangle drawing.

   In fixpoint.h, Set HIBITS, LOBITS to small values (like 5, 2) to run this,
   or you'll wait forever and run out of memory.

*/

static int fildes;
static struct color {float r, g, b;};
int verbose = 0;

#define SIZE 200
#define NPIXELS (1 << (HIBITS - 2))
#define BUFSIZE NPIXELS
int buffer[BUFSIZE][BUFSIZE];

void clear_buffer()
{
    register int i, j;
    for(i=0; i<BUFSIZE; i++)
        for(j=0; j<BUFSIZE; j++)
            buffer[i][j] = 0;
}

/* Check if test succeeded (ie. buffer is full of ones,
   except for first line/column).
*/
void buffer_check()
{
    register int i, j;
    int broken = 0;

    for(i=1; i<BUFSIZE; i++)
        for(j=1; j<BUFSIZE; j++)
            if (buffer[i][j] != 1) {
                printf("ERROR: Buffer[%d][%d] = %d, not 1!!!\n",
                       i, j, buffer[i][j]);
                broken = 1;
                break;
            }

    if (broken) {
        for(j=1; j<BUFSIZE; j++) {
            for(i=1; i<BUFSIZE; i++) {
                printf("%c", buffer[i][j] ? 'o' : '.' );
            }
        }
    }
}
```

```
        printf("\n");
    }
}

int xoff = 100;
int yoff = 100;

void
draw_point (int ix, int iy)
{
    if ((ix >= BUFSIZE) || (iy >= BUFSIZE)) {
        printf("ERROR: ix (%d) or iy (%d) >= BUFSIZE (%d)\n", ix, iy, BUFSIZE);
        abort();
    }

    if (buffer[ix][iy]){
        printf("OOPS -- repainting pixel (%d, %d)\n", ix, iy);

        fill_color(fildes, 1.0, 0.0, 0.5);
        intrectangle(fildes, xoff + ix, yoff + iy, xoff + ix+1, yoff + iy+1);
        fill_color(fildes, 1.0, 1.0, 1.0);

        buffer[ix][iy] = 0;

        make_picture_current(fildes);
        abort();
    }
    else {
        buffer[ix][iy] = 1;
        intrectangle(fildes, xoff + ix, yoff + iy, xoff + ix+1, yoff + iy+1);
    }
}

int
InitScreen()
{
    char      *sb_dev, *driver;

    printnbits();

    sb_dev = getenv("SB_OUTDEV");
    if(!sb_dev) sb_dev = getenv("OUTDEV");
    if(!sb_dev) sb_dev = "/dev/crt1";

    driver = getenv("SB_OUTDRIVER");
    if(!driver) driver = getenv("OUTDRIVER");
    if (!driver ) driver = "hp98731";

    fildes = fopen(sb_dev,OUTDEV,driver,INIT|INT_XFORM);
    interior_style(fildes,INT_SOLID,FALSE);

    intvdc_extent(fildes,0,0,SIZE, SIZE);
    mapping_mode(fildes,FALSE);

    drawing_mode(fildes,6);    /* xor mode */

    clear_control(fildes, CLEAR_VIEWPORT);
}
```

```
    fill_color(fildes, 1.0, 1.0, 1.0);

    return(fildes);
}

void
main()
{
    int i, j;
    int nsubpixels = 1 << (HIBITS+LOBITS);
    fixpoint zero, maxpix;

    zero = 0;
    maxpix = ((NPIXELS-1) << LOBITS) | LOMASK;

    InitScreen();

    if (verbose) {
        printf("zero "); fp_print(zero); printf(", max "); fp_print(maxpix);
        printf("\n");
    }

    for (i=0; i != maxpix + 1; i++) {
        for (j=0; j != maxpix + 1; j++) {

            if (verbose) {
                printf("\n\n***** New Iteration *****\n\n");
                printf("%3d, %3d --> ", i, j);
                fp_print(i); printf(", "); fp_print(j); printf("\n");
            }

            clear_buffer();
            clear_view_surface(fildes);

            fill_color(fildes, 0.5, 0.5, 0.5);
            subpixel_triangle(zero, zero, zero, maxpix, i, j);

            fill_color(fildes, 0.0, 1.0, 0.0);
            subpixel_triangle(zero, zero, maxpix, zero, i, j);

            fill_color(fildes, 0.0, 0.0, 1.0);
            subpixel_triangle(zero, maxpix, maxpix, maxpix, i, j);

            fill_color(fildes, 1.0, 0.0, 0.0);
            subpixel_triangle(maxpix, zero, maxpix, maxpix, i, j);

            buffer_check();
            make_picture_current(fildes);

            if (verbose) {
                printf(" hit <return> for next iteration\n");
                getchar();
            }

        }
    }
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "fixpoint.h"

/* A non-optimized fixpoint pkg, with some overflow checking. */

#define TwosComplement(_x_)  (-(_x_))

int fp_verbose;
int fp_error = 0;          /* will contain error indicators for fp_multiply */
int fp_print_error = 1;    /* if 1, print overflow errors */

/* Return the largest (smallest) number representable in this format */
fixpoint fp_max() { return(~OVERFLOWMASK); }
fixpoint fp_min() { return(OVERFLOWMASK); }

/* return integer part */
int fp_integer(fixpoint x)
{
    fixpoint floor = x >> LOBITS;
    fixpoint bits = (x & LOMASK);

    if (x < 0) {
        if (!bits) return(floor);
        return(floor+1);
    }
    else return (floor);
}

/* return fraction part, in bits */
int fp_fraction(fixpoint x)
{
    if (x < 0) return(TwosComplement(x) & LOMASK);
    else return(x & LOMASK);
}

/* return fraction part, as a double */
double fp_fraction_double(fixpoint x)
{
    if (x < 0) return(-(TwosComplement(x) & LOMASK) / ((double) (1 << LOBITS)));
    else return((x & LOMASK) / ((double) (1 << LOBITS)));
}

/* in 2s complement, integers have all lower order bits = 0,
   and truncating the fraction = floor.
   */
fixpoint fp_floor(fixpoint x)
{
    fixpoint answer = x & HIMASK;
    return (answer);
}

/* The std integer truncating divide does a floor operation,
   but for negative numbers it does floor(abs(x/y)), so
   we need to case on sign of x to fix that.

   if x%y has no remainder, then the divide is on an integer
```



Overflow can't happen in `fp_floor_div`, since  $x/y \leq x$  for  $y \geq 1$ , which must be the case since  $y$  is an integer and  $y > 0$ .

\*/

```
fixpoint fp_floor_div(fixpoint x, fixpoint y)
{
    fixpoint answer;

    if (y == 0) {
        printf("Error: fp_floor_div -- can't divide by 0!\n");
        return(fp_fix(0));
    }
    else if (y < 0) {
        printf("Sorry, fp_floor_div(a,b) doesn't currently work for b<0.\n");
        return(fp_fix(0));
    }
    else if (x >= 0) {
        fixpoint tmp = x/y;
        answer = tmp << LOBITS;
    }
    else {
        fixpoint tmp = ((x/y) + (((x % y) == 0) ? 0 : -1));
        answer = tmp << LOBITS;
    }
    return (answer);
}

fixpoint fp_multiply(fixpoint x, fixpoint y)
{
    fixpoint answer;
    unsigned int xhi, xlo, yhi, ylo, tmp;

    fp_error = 0;

    xhi = (x<0) ? -fp_integer(x) : fp_integer(x);
    yhi = (y<0) ? -fp_integer(y) : fp_integer(y);
    xlo = fp_fraction(x);
    ylo = fp_fraction(y);

    /* If xhi and yhi are both < 16 bits, this can't have machine overflow
       (overflow of the 32bit int). But it CAN get larger than HIBITS, or
       crush the sign bit, causing overflow of our fixpoint representation.
    */
    tmp = (xhi * yhi);
    if (tmp & (((int) OVERFLOWMASK) >> LOBITS)) {
        if (fp_print_error)
            printf ("ERROR: fp_multiply() xhi*yhi = 0x%08x overflows by 0x%08x\n",
                    tmp,
                    (tmp & (((int) OVERFLOWMASK) >> LOBITS)));
        fp_error = -1;
        abort();
        return(0);
    }

    answer = (tmp << LOBITS) +
        (xhi * ylo) + (xlo * yhi) +
        (((xlo * ylo) >> LOBITS) & LOMASK);

    if (answer & ~(HIMASK | LOMASK)) {
```

```
    if (fp_print_error)
        printf ("ERROR: fp_multiply() answer = 0x%08x, overflow 0x%08x\n",
                answer, (answer & ~(HIMASK | LOMASK)));
    fp_error = -2;
    abort();
    return(0);
}
```

```
/* Also, it can fill up the high order (sign) bit, which is overflow. */
if (answer & SIGNBIT) {
    if (fp_print_error)
        printf ("ERROR: fp_multiply() SIGNBIT overflow for answer = 0x%08x\n",
                answer);
    fp_error = -3;
    return(0);
}
```

```
/* "this should never happen" */
if ((xlo >> 16) && (ylo >> 16)) {
    if (fp_print_error)
        printf("+++++ ERROR -- OVERFLOW OF LOW BITS*****\n");
    fp_error = -4;
    abort();
    return(0);
}
```

```
if (((x<0) && (y>0)) || ((x>0) && (y<0))) answer = -answer;
```

```
return(answer);
}
```

```
/* Turn a double into a fixpoint number int two's complement.
   Negative numbers become: ~a + 1
*/
```

```
fixpoint fp_fix(double x)
```

```
{
    int negative = (x < 0);
    double i;
    double fraction;
    fixpoint p;

    /* integer part */
    /* fraction part, positive only */
    /* fixpoint version of abs(x) */
}
```

```
if ((x < fp_integer(fp_min())) || (x >= fp_integer(fp_max())))
    printf("sorry, %g doesn't fit in (%d hi, %d lo) bit fix point.\n",
           x, HIBITS, LOBITS);
```

```
x = fabs(x);
i = floor(x);
fraction = x - i;
```

```
p = (((int) i) << LOBITS) | ((int) ((x-i)*(1<<LOBITS)));
```

```
if (negative) return TwosComplement(p);
else return (p);
}
```

```
/* Print as a binary number */
```

```
void fp_printb(fixpoint x)
```

```
{
    int i;
```

```
unsigned int mask;

if (x<0) x = -x;
mask = 1 << (HIBITS + LOBITS - 1); /* start printing hi bit */
for(i=0; i<HIBITS; i++) {
    if (mask & x) putchar('1');
    else          putchar('0');
    mask = mask >> 1;
}
putchar('.');
for(i=0; i<LOBITS; i++) {
    if (mask & x) putchar('1');
    else          putchar('0');
    mask = mask >> 1;
}
}

/* Print as a hex number, but gets things a bit screwed
   up unless HIBITS=LOBITS=16
*/
void fp_printx(fixpoint x)
{
    printf("%4x.%04x", fp_integer(x) & LOMASK, fp_fraction(x));
}

double fp_double(fixpoint x)
{
    return(((double) fp_integer(x)) + fp_fraction_double(x));
}

void fp_print(fixpoint x)
{
    printf("%.11g", fp_double(x));
}

void printnbits()
{
    printf("HIBITS = %d, LOBITS = %d, OVERFLOWMASK = 0x%08x, SIGNBIT = 0x%08x\n",
           HIBITS, LOBITS, OVERFLOWMASK, SIGNBIT);
    printf("HIMASK = 0x%08x, LOMASK = 0x%08x\n", HIMASK, LOMASK);
    printf("overflowmask >> lobits = 0x%08x\n", ((int) OVERFLOWMASK) >> LOBITS);
}
```

```
#ifndef FIXPOINT_H
#define FIXPOINT_H

/* Requires: LOBITS to be divisible by 2, HIBITS<=16, and LOBITS <=16. */

#define HIBITS 16
#define LOBITS 16

#define LOMASK      (~(0xffffffff << LOBITS))
#define HIMASK      (~(0xffffffff << HIBITS)) << LOBITS
#define SIGNBIT     (1 << (HIBITS+LOBITS-1))
#define OVERFLOWMASK (SIGNBIT | ~(HIMASK | LOMASK))

typedef int fixpoint;
typedef struct {unsigned int hi, lo, neg;} dblfixpoint;

extern int fp_error;
extern fixpoint fp_max();
extern fixpoint fp_min();
extern int fp_integer();
extern int fp_fraction();
extern double fp_fraction_double();

extern fixpoint fp_multiply();
extern void fp_print();
extern fixpoint fp_fix();
extern double fp_double(fixpoint x);

extern int fp_dblnegative();
extern dblfixpoint fp_dblnegate();
extern dblfixpoint fp_dblmultiply();
extern int fp_dbllessthan();
extern dblfixpoint fp_dbladd();
extern fixpoint fp_trunc();

#endif
```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <starbase.c.h>

#ifndef M_PI
# define M_PI          3.14159265358979323846
#endif

extern double drand48();

typedef int fixpoint;
extern void subpixel_triangle(fixpoint x0, fixpoint y0,
                             x1, fixpoint y1,
                             x2, fixpoint y2);
extern void triangle(int x0, int y0, int x1, int y1, int x2, int y2);
extern void printnbits();

static int      fildes;
static struct color {float r, g, b;};
int verbose = 0;

#define MAXVERTICES    1000
#define MAXTRIANGLES  1000

#define SIZE 200
#define BUFSIZE SIZE*2
int buffer[BUFSIZE][BUFSIZE];
int which_tri;

void clear_buffer()
{
    register int i, j;
    if (verbose) printf("\n\n***** New Iteration *****\n\n");

    for(i=0; i<BUFSIZE; i++)
        for(j=0; j<BUFSIZE; j++)
            buffer[i][j] = 0;
}

int
InitScreen()
{
    char      *sb_dev, *driver;

    printnbits();

    sb_dev = getenv("SB_OUTDEV");
    if(!sb_dev) sb_dev = getenv("OUTDEV");
    if(!sb_dev) sb_dev = "/dev/crt1";

    driver = getenv("SB_OUTDRIVER");
    if(!driver) driver = getenv("OUTDRIVER");
    if (!driver ) driver = "hp98731";

    fildes = fopen(sb_dev,OUTDEV,driver,INIT|INT_XFORM);
    interior_style(fildes,INT_SOLID,FALSE);
```

```
    intvdc_extent(fildes,0,0,SIZE, SIZE);
    mapping_mode(fildes,FALSE);

    drawing_mode(fildes,6);    /* xor mode */

    if (!verbose)
        double_buffer(fildes,TRUE,12);

    clear_control(fildes, CLEAR_VIEWPORT);

    return(fildes);
}
```

```
int buf = 0;
int t0[MAXTRIANGLES], t1[MAXTRIANGLES], t2[MAXTRIANGLES];
fixpoint x[MAXVERTICES], y[MAXVERTICES];
float xf[MAXVERTICES], yf[MAXVERTICES];
```

```
void print_triangle(int n)
{
    printf("Triangle %d contains points %d, %d, %d\n", n, t0[n], t1[n], t2[n]);
    printf("which is (%g, %g), (%g, %g), (%g, %g)\n",
           xf[t0[n]], yf[t0[n]],
           xf[t1[n]], yf[t1[n]],
           xf[t2[n]], yf[t2[n]]);

    printf("which is ");
    fp_print(x[t0[n]]);
    printf(", ");
    fp_print(y[t0[n]]);
    printf(", ");
    fp_print(x[t1[n]]);
    printf(", ");
    fp_print(y[t1[n]]);
    printf(", ");
    fp_print(x[t2[n]]);
    printf(", ");
    fp_print(y[t2[n]]);
    printf("\n");
}
```

```
void
draw_point (int ix, int iy)
{
    /*  if ((ix == 91) && (iy == 74)) {
        printf("Triangle painting (%d, %d) ... Triangle %d\n", ix, iy, which_tri);
        print_triangle(which_tri); }
    */

    if (buffer[ix][iy]){
        printf("OOPS -- repainted pixel (%d, %d) for Triangle %d\n",
               ix, iy, which_tri);
        print_triangle(which_tri);

        fill_color(fildes, 1.0, 0.0, 0.5);
        intrectangle(fildes,ix,iy,ix+1,iy+1);
        fill_color(fildes, 1.0, 1.0, 1.0);
    /*
        dbuffer_switch(fildes, !buf);
        make_picture_current(fildes);
    */
}
```

```
        abort();
```

```
    */
    }
    else {
        buffer[ix][iy] = 1;
        intrectangle(fildes, ix, iy, ix+1, iy+1);
    }
}
```

```
#define CENTER 0.5
```

```
main (int argc, char *argv[])
{
```

```
    int xi[MAXVERTICES], yi[MAXVERTICES];
    struct color tc[MAXTRIANGLES];
    int nv, nt, i, j, k, white;
    double c, s;
    float rotinc, xr, yr;
    float scale = 90.0;
```

```
    char *filename = NULL;
    FILE *fp;
```

```
    if (argc == 2) {
        filename = argv[1];
        printf("reading from file %s\n", filename);
    }
```

```
    if (filename) fp = fopen(filename, "r");
    else fp = stdin;
```

```
    fscanf(fp, "%f %d %d", &rotinc, &white, &nv);
    printf("Read header info.\n");
```

```
    for (i=0; i<nv; i++) fscanf(fp, "%f %f", &xf[i], &yf[i]);
    printf("Read %d verts\n", nv);
```

```
    for (nt=0; (fscanf(fp, "%d %d %d", &t0[nt], &t1[nt], &t2[nt])!=EOF) &&
               (nt < MAXTRIANGLES); nt++) {
        if (white)
            tc[nt].r = tc[nt].g = tc[nt].b = 1.0;
        else
            tc[nt].r = drand48(), tc[nt].g = drand48(), tc[nt].b = drand48();
    }
```

```
    printf("Read %d triangles\n", nt);
```

```
    InitScreen();
```

```
    for (k=0; k<1000; k++) {
        c = cos(k*rotinc*M_PI/180.0);
        s = sin(k*rotinc*M_PI/180.0);
```

```
        if (verbose) printf ("%4d: (%g, %g)\n", k, c, s);
```

```
        for (j=0; j<nv; j++) {
            xr = (( xf[j]-CENTER)*c + (yf[j]-CENTER)*s + 0.5);
            yr = ((-xf[j]+CENTER)*s + (yf[j]-CENTER)*c + 0.5);
            x[j] = fp_fix(xr * scale + 105.0);
            y[j] = fp_fix(yr * scale + 32.0);
```

```
        xi[j] = (int) (xr * scale + 5.5);
        yi[j] = (int) (yr * scale + 32.5);
    }

    clear_view_surface(fildes);

    clear_buffer();
    for (i=0; i<nt; i++) {
        if (verbose)
            printf("+++++++ Drawing triangle %d ++++++\n",
i);

        which_tri = i;
        if (i < nt/2) fill_color(fildes,tc[i].r,tc[i].g,tc[i].b);
        else          fill_color(fildes,1.0, 1.0, 0.5);

        triangle(xi[t0[i]], yi[t0[i]],
                xi[t1[i]], yi[t1[i]], xi[t2[i]],
yi[t2[i]]);

        if (verbose) printf(" Subpixel: ");
        subpixel_triangle(x[t0[i]], y[t0[i]],
                                x[t1[i]],
y[t1[i]], x[t2[i]], y[t2[i]]);

        if (verbose) make_picture_current(fildes);
    }
    if (!verbose) {
        dbuffer_switch(fildes, buf = !buf);
        make_picture_current(fildes);
    }
    if (verbose) {
        printf("hit <return> for next iteration\n");
        getchar();
    }
}
```



```
#include <stdio.h>
#include <math.h>

extern void draw_point(int x, int y);

#include "fixpoint.h"

#define SWAP(_a_, _b_, _c_) { _c_ = _a_; _a_ = _b_; _b_ = _c_; }
#define min(_a_, _b_) ((_a_ < _b_) ? _a_ : _b_)
#define max(_a_, _b_) ((_a_ > _b_) ? _a_ : _b_)

struct edge {
    int ymin, ymax, xi, si;
    int r, inc, dec;
};

/* floor(x/y). Assumes y>0. */
int floor_div(int x, int y)
{
    if (x >= 0) return(x/y);
    else return((x/y) + ((x % y) == 0) ? 0 : -1));
}

/*
 * Draws pixels in the half-open interval (x1, x2].
 */
void draw_span(int x1, int x2, int y)
{
    int x;
    for (x=x1+1; x<=x2; x++) draw_point(x, y);
}

/*
 * Initializes the Bresenham-like scan conversion for a single edge,
 * setting values in the structure containing the increment variables.
 */
struct edge *EdgeSetup(struct edge *e, int x0, int y0, int x1, int y1)
{
    int sf, dx = x1-x0, dy = y1-y0;

    e->ymin = y0;
    e->ymax = y1;

    if (dy != 0) {
        e->si = floor_div(dx, dy);
        e->xi = x0 + e->si;
        sf = dx - e->si * dy;
        e->r = 2*sf - dy;
        e->inc = sf;
        e->dec = sf - dy;
    }
    return(e);
}

/*
 * Returns the intersection of edge e with the next scanline.
 */
int EdgeScan(struct edge *e)
{
    int x = e->xi;
```

```
if (e->r >= 0) {
    e->xi += e->si + 1;
    e->r += e->dec;
}
else {
    e->xi += e->si;
    e->r += e->inc;
}
return x;
}

/*
 * Scan-converts a triangle with integer endpoints. Assumes the
 * triangle's vertices are ordered so that y0 <= y1 <= y2.
 */
void sorted_triangle(int x0, int y0, int x1, int y1, int x2, int y2)
{
    int det, yi, xmin, xmax;
    struct edge left, right;

    /* Compute handedness of triangle (points left or right) */
    /* (see Pavlidis '82, [Computer Science Press], Ch 14) */
    det = (y1-y0)*(x2-x0) - (x1-x0)*(y2-y0);

    /* Setup first pair of edges */
    if (det < 0)
        { EdgeSetup(&left, x0, y0, x2, y2);
          EdgeSetup(&right, x0, y0, x1, y1); }
    else
        { EdgeSetup(&left, x0, y0, x1, y1);
          EdgeSetup(&right, x0, y0, x2, y2); }

    /* Scan first pair of edges. */
    for (yi = left.ymin + 1; yi <= min(left.ymax, right.ymax); yi++) {
        xmin = EdgeScan(&left);
        xmax = EdgeScan(&right);
        draw_span(xmin, xmax, yi);
    }

    /* Setup third edge */
    if (det >= 0) EdgeSetup(&left, x1, y1, x2, y2);
    else         EdgeSetup(&right, x1, y1, x2, y2);

    /* Scan remainder of triangle. */
    for (yi = max(left.ymin, right.ymin) + 1; yi <= left.ymax; yi++) {
        xmin = EdgeScan(&left);
        xmax = EdgeScan(&right);
        draw_span(xmin, xmax, yi);
    }
}

/*
 * Scan-converts a triangle with integer endpoints.
 * Sorts the vertices, then calls a routine to do the scan conversion.
 */
void triangle(int x0, int y0, int x1, int y1, int x2, int y2)
{
    int tmp;
    if (y0>y1) { SWAP(y0,y1,tmp); SWAP(x0,x1,tmp); }
}
```

```
    if (y0>y2) { SWAP(y0,y2,tmp); SWAP(x0,x2,tmp); }
    if (y1>y2) { SWAP(y1,y2,tmp); SWAP(x1,x2,tmp); }

    sorted_triangle(x0, y0, x1, y1, x2, y2);
}

/*      The scan-conversion routines for coordinates at subpixel resolution.
 *
 *      SubEdgeSetup initializes the Bresenham-like scan conversion for a
 *      single edge (analogous to EdgeSetup).
 *
 *      Double lenght fixpoint is used to compute alpha below. Alpha
 *      is then truncated (this corresponds to quantizing to a
 *      particular subpixel while computing the x-intersection of the edge
 *      with the scanline). It will not produces artifacts except for
 *      degenerate databases, in which adjacent polygons don't necessarily
 *      have coincident vertices (see Lathrop, Kirk, Voorhies, IEEE CG&A 10(5),
 *      1990 for a discussion).
 *
 */
struct edge *
SubEdgeSetup(struct edge *e,
              fixpoint x0, fixpoint y0, fixpoint x1, fixpoint y1)
{
    fixpoint ymin, ymax, alpha, beta, sf, xi, si;
    fixpoint dx = x1-x0, dy = y1-y0;

    ymin = fp_floor(y0);
    ymax = fp_floor(y1);

    if ((dy != 0) && (ymin != ymax)) {

        /* Alpha is related to x-intercept with scanline ymin */
        alpha = fp_trunc(fp_dbladd(fp_dblmultiply(fp_fraction(x0), dy),
fp_dblmultiply(dx, ymin - y0 + fp_fix(1.0))));
        beta = fp_floor_div(alpha, dy);
        xi = fp_floor(x0) + beta;

        /* prevent overflow for v. small dy ('si' is only used if dy>=1) */
        if (dy >= fp_fix(1.0)) {
            si = fp_floor_div(dx, dy);
            sf = dx - fp_multiply(si, dy);

            /* (alpha - beta*dy) = fractional part of the x-intercept. */
            e->r = alpha - fp_multiply(beta, dy) + sf - dy;
            e->si = fp_integer(si);
            e->inc = sf;
            e->dec = sf - dy;
        }
        e->xi = fp_integer(xi);
    }
    e->ymin = fp_integer(ymin);
    e->ymax = fp_integer(ymax);

    return(e);
}

/* Like integer version, but uses SubEdgeSetup */
```

```
void subpixel_sorted_triangle(fixpoint x0, fixpoint y0,
                             fixpoint x1, fixpoint y1,
                             fixpoint x2, fixpoint y2)
{
    struct edge left, right;
    int yi, xmin, xmax;

    /* compute determinant using full precision (64 bits) */
    dblfixpoint a = fp_dbلمultiply(y1-y0,x2-x0);
    dblfixpoint b = fp_dbلمultiply(x1-x0,y2-y0);
    int clockwise = fp_dbلمlessthan(a, b);

    /* Setup first pair of edges */
    if (clockwise) {
        SubEdgeSetup(&left, x0, y0, x2, y2);
        SubEdgeSetup(&right, x0, y0, x1, y1);
    } else {
        SubEdgeSetup(&left, x0, y0, x1, y1);
        SubEdgeSetup(&right, x0, y0, x2, y2);
    }

    /* Scan first pair of edges. */
    for (yi = left.ymin + 1; yi <= min(left.ymax, right.ymax); yi++) {
        xmin = EdgeScan(&left);
        xmax = EdgeScan(&right);
        draw_span(xmin, xmax, yi);
    }

    /* Setup third edge */
    if (!clockwise) SubEdgeSetup(&left, x1, y1, x2, y2);
    else SubEdgeSetup(&right, x1, y1, x2, y2);

    /* Scan remainder of triangle. */
    for (yi = max(left.ymin, right.ymin) + 1; yi <= left.ymax; yi++) {
        xmin = EdgeScan(&left);
        xmax = EdgeScan(&right);
        draw_span(xmin, xmax, yi);
    }
}

/* Like integer version. */
void subpixel_triangle(fixpoint x0, fixpoint y0,
                       fixpoint x1, fixpoint y1,
                       fixpoint x2, fixpoint y2)
{
    fixpoint tmp;
    if (y0>y1) { SWAP(y0,y1,tmp); SWAP(x0,x1,tmp); }
    if (y0>y2) { SWAP(y0,y2,tmp); SWAP(x0,x2,tmp); }
    if (y1>y2) { SWAP(y1,y2,tmp); SWAP(x1,x2,tmp); }

    subpixel_sorted_triangle(x0, y0, x1, y1, x2, y2);
}
```

0.0 1 3  
0 0  
0 1  
1 0  
0 1 2

```
/*----- main() - test_scallops.c -----
* This main program tests the scallop (i.e. an envelope of circles) generating
* function by drawing filled circles with a "fill_circle" function and then
* using an "erode" function to reduce them to their outlines or scallops.
* It allocates a small frame buffer and asks the user to choose the number
* of circles desired, their centers, and their radii. The circles are drawn
* and filled in sequence. A simple "printf" to the standard output displays
* the upper left corner of the frame buffer. The "erode" function is then
* applied to the frame buffer, leaving only the outlines of the filled
* circles (scallops) with their interiors refilled by an erosion replacement
* value. Again the upper left corner of the frame buffer is displayed with
* the results of the eroding the filled circles to scallops.
* When the erosion replacement value is the same as the background, only the
* outlines (the value used for circle filling) will be shown.
*
*      Author:      Eric Furman
*                  General Dynamics/ Convair Division
*                  San Diego, California
*/

#include <stdio.h>

#define XMIN      0                /* limits of the frame buffer */
#define XMAX      63
#define YMIN      0
#define YMAX      63
#define STRIDE    64              /* width of frame buffer (XMAX - XMIN + 1) */

#define MIN(j,k)  (j)<(k) ? (j) : (k)
#define MAX(j,k)  (j)>(k) ? (j) : (k)

unsigned char  *Pt_Frame;          /* Global pointer to start of frame buffer */

main()
{
    int  xc, yc, r, ny, nc, k, j;
    unsigned char  val_fill, val_erode;
    void  fill_circle(), erode();

/* Values (i.e. VLT index) for filling the circles and erosion replacement. */
    val_fill = 8;
    val_erode = 1;

/* Allocate space initialized to zero for the test frame buffer: */
    ny = YMAX - YMIN + 1;          /* number of rasters */
    Pt_Frame = (unsigned char *)calloc(STRIDE * ny, sizeof(unsigned char));

/* Obtain user input and loop through circles filling each one. */
    printf(" Number of circles: ");
    scanf("%d", &nc);              /* try:  3 */
    for(k=0; k<nc; k++) {
        printf(" center (x,y) and radius:  ");
        scanf("%d%d%d", &xc, &yc, &r);      /* try:  5 5 6,  15 12 7,  28 6 5 */

/* Check for total clipping before filling circle: */
        if(xc+r < XMIN || xc-r > XMAX || yc+r < YMIN || yc-r > YMAX) {
            printf("Circle x,y,r:  %d  %d  %d total clip.\n", xc, yc, r);
        }
        else {

```

```
        fill_circle(xc, yc, r, val_fill);
    }
} /* end for loop through circles */

/* Print the Upper Left corner of the buffer after filling circles: */
printf("UL corner of filled buffer.\n");
for(k=0; k<23; k++) {
    printf("%2d: ",k);
    for(j=0; j<35; j++) {
        printf("%2d", *(Pt_Frame + j + k * STRIDE));
    }
    printf("\n");
}

erode(val_fill, val_erode); /* erode the filled circles */

/* Print the Upper Left corner of the buffer after eroding filled circles: */
printf("UL corner of eroded buffer.\n");
for(k=0; k<23; k++) {
    printf("%2d: ",k);
    for(j=0; j<35; j++) {
        printf("%2d", *(Pt_Frame + j + k * STRIDE));
    }
    printf("\n");
}

free((char*) Pt_Frame); /* free the frame buffer memory */
}

/*----- fill_circle() -----
* A midpoint circle generating/filling algorithm using second order partial
* differences to compute cartesian increments. Given a circles center
* point (xc,yc), its radius (r), and a fill value (value); this will
* draw the filled circle.
*/

void fill_circle(xc, yc, r, value)
    int xc, yc, r; /* center & radius */
    unsigned char value; /* filling value */
{
    int x, y, d, de, dse ;
    void raster_fill();

    x = 0;
    y = r; /* initialization */
    d = 1 - r;
    de = 3;
    dse = -2 * r + 5;
    while(y >= x) { /* thru 2nd octant, others handled in raster_fill() */
        if(d < 0) { /* only move +x in octant 2 */
            d += de;
            dse += 2;
            raster_fill(xc, yc, y, x, value); /* rasters in octants 1-4 & 5-8 */
        }
        else { /* move +x and -y in octant 2 */
            d += dse;
            dse += 4;
            raster_fill(xc, yc, y, x, value); /* rasters in octants 1-4 & 5-8 */
            raster_fill(xc, yc, x, y, value); /* rasters in octants 2-3 & 6-7 */
            y --;
        }
        de += 2;
    }
}
```

```
        x++;
    }
    /* end while(y >= x) */
}

/*----- raster_fill() -----
* Rasters filling for octants 2 to 3 and 6 to 7. x and y will be reversed in
* the calling sequence for octants 1 to 4 and 5 to 8.
* Requires the global pointer to a frame buffer (Pt_Frame), the defined
* parameters for the frame extents (XMIN, XMAX, YMIN, YMAX), and the defined
* parameter for the frame width (STRIDE).
* Called by the fill_circle function.
*/

void raster_fill(xc, yc, x, y, value)
    int xc, yc, x, y;
    unsigned char value;
{
    int xl, xr, yt, yb;
    register int k;
    register unsigned char *ptb;

    xr = MAX(xc + x, XMIN); /* raster(s) x limits */
    xl = MIN(xc - x, XMAX);

    /* if raster segment on frame in x, get y values of top and bottom raster */
    if(xr >= xl) {
        yt = yc + y;
        yb = yc - y;

    /* if raster segment on frame in y, point to its start and fill segment */
        if(yt <= YMAX && yt >= YMIN) {
            ptb = Pt_Frame + yt * STRIDE + xl;
            for(k=xl; k<=xr; k++) *ptb++ = value;
        }
        if(yb != yt && yb <= YMAX && yb >= YMIN) { /* if on & not same raster */
            ptb = Pt_Frame + yb * STRIDE + xl;
            for(k=xl; k<=xr; k++) *ptb++ = value;
        }
    }
    /* end of if(xr >= xl) */
}

/*----- erode() -----
* Erode the filled circles to their envelope (scallop) only. Erases existing
* val_filled pixels to val_erode when all 4 nearest neighbor pixels are also
* set to val_filled.
* Requires the value used to fill the circles (val_filled) and a value to
* which their interiors will be eroded (val_erode).
* Also requires the global pointer to a frame buffer (Pt_Frame), the defined
* parameters for the frame extents (XMIN, XMAX, YMIN, YMAX), and the defined
* parameter for the frame width (STRIDE).
*/

void erode(val_filled, val_erode)
    unsigned char val_filled, val_erode; /* filled and erode to values */
{
    int k, j;
    unsigned char *buf0, *buf1, *buf2, *b0, *b1, *b2;
    int n;
    unsigned char val4, *ptl, *ptr, *ptb, *ptt, *ptc, *ptf;

    val4 = val_filled * 4; /* for nearest neighbor sum test below */

    /* Allocate space for working raster buffers were erosion test/evaluation will
```



```
* be performed. These include a one pixel border beyond the frame buffer. */
buf0 = b0 = (unsigned char*)malloc((STRIDE+2) * sizeof(unsigned char));
buf1 = b1 = (unsigned char*)malloc((STRIDE+2) * sizeof(unsigned char));
buf2 = b2 = (unsigned char*)malloc((STRIDE+2) * sizeof(unsigned char));
ptt = buf0;
ptc = buf1; /* set center, top, and bottom buffer pointers */
ptb = buf2;

/* Initial buffer values to drawing value for one pixel border at top: */
for(k=0; k<STRIDE+2; k++) *ptt++ = *ptc++ = *ptb++ = val_filled;






/* Initialize first working raster buffer from the Frame buffer */
ptf = Pt_Frame;
ptb = buf2 + 1;
for(k=0; k<STRIDE; k++) *ptb++ = *ptf++;

/* Working through the Frame buffer raster by raster with line counter n: */
for(k=YMIN, n=0; k<=YMAX; k++, n++) {
    ptf = buf0;
    buf0 = buf1;
    buf1 = buf2; /* roll pointers for circular buffering */
    buf2 = ptf;
    ptb = buf2 + 1; /* "new" buffer line to fill */
    if(k == YMAX) { /* if below frame, use filled values */
        for(j=0; j<STRIDE+2; j++) *ptb++ = val_filled;
    }
    else { /* else on frame buffer, so load from frame */
        ptf = Pt_Frame + (n+1) * STRIDE;
        for(j=0; j<STRIDE; j++) *ptb++ = *ptf++; /* add next raster */
    }
}

/* Doing erosion by moving across the three raster buffers by pointers to
* the nearest four neighbors, the center pixel, and the output raster: */
ptl = buf1; /* left of test point */
ptr = buf1 + 2; /* right */
ptt = buf0 + 1; /* top */
ptb = buf2 + 1; /* bottom */
ptc = buf1 + 1; /* center */
ptf = Pt_Frame + n * STRIDE; /* output */
for(j=0; j<STRIDE; j++) { /* across buffers pixel by pixel */
    if(*ptc++ == val_filled) { /* is center pixel value filled */
        if((*ptl + *ptr + *ptt + *ptb) == val4) *ptf = val_erode;
    }
    ptl++; ptr++; ptt++; ptb++; ptf++; /* increment pointers */
} /* end for j loop across raster buffers */
} /* end for k loop thru frame buffer rasters */

free((char*)b0);
free((char*)b1);
free((char*)b2);
}
/*-----*/
```

# Index of /pubs/tog/GraphicsGems/gemsiii/exttest/

Name	Last modified	Size	Description
 <a href="#">_Parent Directory</a>			
 <a href="#">_Makefile</a>	29-Jun-00 08:16	1K	
 <a href="#">_ehtest1.C</a>	29-Jun-00 08:16	1K	
 <a href="#">_exthit.C</a>	29-Jun-00 08:16	7K	
 <a href="#">_exthit.h</a>	29-Jun-00 08:16	4K	

```
# code is C++
CFLAGS =

ehtest1:      ehtest1.C exthit.o exthit.h
              CC $(CFLAGS) -o ehtest1 ehtest1.C exthit.o

exthit.o:      exthit.C exthit.h
              CC $(CFLAGS) -c exthit.C -o exthit.o

clean:
              rm -rf ehtest1 exthit.o
```

```
//
// ExtHit minimal tester
//
// Len Wanger - Fri Aug 30 10:07:01 PDT 1991
//

#include "iostream.h"
#include "exthit.h"

void func_tst (Ptr data, Ptr p1, Ptr p2 )
{
    cout << "Objects " << (Ptr) p1 << " and " << (Ptr) p2 << " intersect.\n";
}

main()
{
    ExtHit eh(5);
    Extent ext;
    Ptr ptr;

    ext.min[0] = 1; ext.max[0] = 3;
    ext.min[1] = 2; ext.max[1] = 4;
    ext.min[2] = 4; ext.max[2] = 6;
    ext.min[3] = 2; ext.max[3] = 8;

    ptr = (Ptr) 0x1;

    if ( ! eh.add ( ext, ptr ) )
        cout << "Fillnext failed\n";

    ext.min[0] = 5; ext.max[0] = 7;
    ext.min[1] = 6; ext.max[1] = 8;
    ext.min[2] = 2; ext.max[2] = 8;
    ext.min[3] = 1; ext.max[3] = 3;

    ptr = (Ptr) 0x2;

    if ( ! eh.add ( ext, ptr ) )
        cout << "Fillnext failed\n";

    ext.min[0] = 2; ext.max[0] = 6;
    ext.min[1] = 3; ext.max[1] = 7;
    ext.min[2] = 7; ext.max[2] = 7;
    ext.min[3] = 2; ext.max[3] = 4;

    ptr = (Ptr) 0x3;

    if ( ! eh.add ( ext, ptr ) )
        cout << "Fillnext failed\n";

    ext.min[0] = 2; ext.max[0] = 6;
    ext.min[1] = 5; ext.max[1] = 5;
    ext.min[2] = 1; ext.max[2] = 5;
    ext.min[3] = 1; ext.max[3] = 5;

    ptr = (Ptr) 0x4;

    if ( ! eh.add ( ext, ptr ) )
        cout << "Fillnext failed\n";

    eh.test ( func_tst, (Ptr) 0x5);
}
```

}

```

/*****
exthit.C

```

This code implements a C++ class for fast n-dimensional extent overlap checking. An instance of the class is initialized with a call to the constructor. For details on the usage of the ExtHit class see the header for the ExtHit header file

Authors: Len Wanger and Mike Fusco

SimGraphics Engineering

```

*****/

```

```

#ifndef _EXTHIT_H
#include "exthit.h"
#endif

```

```

#include <stdlib.h>

```

```

static int dim;

```

```

/*****

```

```

    ExtHit - Class constructor

```

Inputs:

size: Maximum number of extents that can be held by this instance.

Outputs:

None.

```

*****/

```

```

ExtHit::ExtHit (int size)

```

```

{
    numColRecs = 0;
    maxSize = size;
    activeList = NULL;

    // Allocate internal tables
    collideList = new CollideRecord[size];
    overlapList = new MinMaxRecordPtr[2*size];
    overlapTable = (BOOL *) calloc ( (size*size), sizeof(BOOL) );
}

```

```

/*****

```

```

    ~ExtHit - Class destructor

```

Inputs:

None.

Outputs:

None.

```

*****/

```

```

ExtHit::~~ExtHit ()

```

```

{
    // Reclaim the memory allocated for the internal tables
    delete ( overlapList );
    delete ( overlapTable );
    delete ( collideList );
}

```

```

/*****

```

```

    add - Add an extent to be considered for overlap checking.

```

Inputs:

extent: The extent to consider for extent overlap checking.  
obj: A pointer to the parent object for the extent.

Outputs:

A Boolean value: TRUE if the information was added successfully,  
FALSE otherwise.

\*\*\*\*\*/

BOOL ExtHit::add( Extent &extent, Ptr obj )

```
{
    CollideRecord *cr = &collideList[numColRecs];

    // Make sure there is room to add the extent
    if ( numColRecs >= maxSize )
        return (FALSE);

    // Add the new CollideRecord to the activeList
    if (numColRecs == 0)
    {
        cr->prevActive = NULL;
        activeList = cr;
    }
    else
    {
        cr->prevActive = &collideList[numColRecs-1];
        cr->prevActive->nextActive = cr;
    }
}
```

// Initialize the new CollideRecord

```
cr->nextActive = NULL;
cr->index = numColRecs++;
cr->obj = obj;
cr->prevOpen = cr->nextOpen = NULL;
cr->active = TRUE;
cr->open = FALSE;
```

// add the extent values to the CollideRecord

```
for ( int i=0; i<num_dimensions; i++ )
{
    cr->min[i].value = extent.min[i];
    cr->max[i].value = extent.max[i];
    cr->min[i].ptr = cr->max[i].ptr = cr;
}
```

```
return (TRUE);
}
```

\*\*\*\*\*/

test - Test for overlapping extents.

Inputs:

func: A user supplied routine to be called for all pairs of  
overlapping extents.  
data: User supplied data to be passed as the d argument in user  
supplied routine func.

Outputs:

None.

\*\*\*\*\*/

void ExtHit::test (void (\*func)(Ptr d, Ptr obj1, Ptr obj2), Ptr data)

```
{
```

```
for ( dim=0; dim<num_dimensions; dim++ )
{
    // Add the min and max values of each active extent to the overlapList
    numOverlaps = 0;
    CollideRecord *activeCr = activeList;

    while ( activeCr )
    {
        // Add the min extent value to the overlapList
        MinMaxRecord *mmr = &(activeCr->min[dim]);
        overlapList[numOverlaps++] = mmr;

        // Add the max extent value to the overlapList
        mmr = &(activeCr->max[dim]);
        overlapList[numOverlaps++] = mmr;

        // Reset the active and open flags
        mmr->ptr->active = mmr->ptr->open = FALSE;

        // Mark extent as currently not open
        mmr->ptr->prevOpen = mmr->ptr->nextOpen = NULL;

        activeCr = activeCr->nextActive;
    }

    // Sort the OverlapList by MinMaxRecord value
    qsort( overlapList, numOverlaps, sizeof(MinMaxRecordPtr), minMaxCompare );

    // Process all overlapping pairs in the overlapList
    subtest ( dim, numOverlaps, func, data );
}
}
```

```
/******
minMaxCompare - Compare the value of two MinMaxRecords for sorting.
```

Inputs:

rec1: A MinMaxRecord to compare.  
rec2: A MinMaxRecord to compare.

Outputs:

-1: if rec1's MMR value is less than rec2's MMR value.  
0: if the two MMR values are equal  
1: if rec1's MMR value is greater than rec2's MMR value.

```
*****/
```

```
int minMaxCompare( const void* rec1, const void* rec2)
{
    MinMaxRecord *mmr1 = *(MinMaxRecord **) rec1;
    MinMaxRecord *mmr2 = *(MinMaxRecord **) rec2;

    if ( mmr1->value > mmr2->value )
        return (1);
    else if ( mmr1->value < mmr2->value )
        return (-1);
    else // mmr1->value == mmr2->value
    {
        // Put minimum extent values before maximum extent value to
        if ( mmr1->value == mmr1->ptr->min[dim].value )
            return (-1);
        else
            return (1);
    }
}
```



```

    }
}

/*****
subtest - Find and process each set of overlapping pairs of extents
for a dimension (i.e. in the overlapList).

Inputs:
    dim:          The extent dimension being tested.
    oLstSize:     The number of MinMaxRecords in the overlapList.
    func:         User supplied function to be called for each overlapping pair.
    data:         User supplied data to be passed in the call to func.

Outputs:
    None.
*****/
void ExtHit::subtest ( int dim, int oLstSize, void (*func)(Ptr d, Ptr p1,
                                                             Ptr p2), Ptr data )
{
    // Reset the list of currently open CollideRecords.
    CollideRecord *openList = NULL;

    for (int i=0; i<oLstSize; i++)
    {
        // Get CollideRecord for the overlapList entry
        CollideRecord *rec = overlapList[i]->ptr;

        if ( !rec->open )
        {
            // Set rec's open flag
            rec->open = TRUE;

            // Set the overlapTable entry for all objects on the openList
            CollideRecord *cr = openList;

            while ( cr )
            {
                int min = MIN(cr->index, rec->index);
                int max = MAX(cr->index, rec->index);
                int index = (min * maxSize) + max;

                // check whether the entry for cr overlapping rec is set in
                // the overlapTable.
                if ( overlapTable[index] == dim )
                {
                    if ( dim < (num_dimensions-1) )
                    {
                        // Set the entry in the top half of the
                        overlapTable[index] = dim+1;
                        cr->active = rec->active = TRUE;
                    }
                    else
                    {
                        // The extents overlapped in all of the dimensions,
                        // the user supplied overlap function.
                        (*func)(data, cr->obj, rec->obj );
                    }
                }

                cr = cr->nextOpen;
            }
        }
    }
}

```

```
        // Add rec at the head of the openList
        rec->nextOpen = openList;
        rec->prevOpen = NULL;

        if ( openList )
            openList->prevOpen = rec;

        // Update the head of the openList
        openList = rec;
    }
else // the record is already on the openList
{
    // remove the record from the openList
    if ( rec->prevOpen )
        rec->prevOpen->nextOpen = rec->nextOpen;
    else
        openList = rec->nextOpen;

    if ( rec->nextOpen )
        rec->nextOpen->prevOpen = rec->prevOpen;

    rec->open = FALSE;

    // If the record is not active remove it from the activeList
    if ( ! rec->active )
    {
        if ( rec->prevActive == NULL )
        {
            activeList = rec->nextActive;

            if (rec->nextActive)
                rec->nextActive->prevActive = NULL;
        }
        else
        {
            rec->prevActive->nextActive = rec->nextActive;

            if (rec->nextActive)
                rec->nextActive->prevActive = rec->prevActive;
        }
    }
}

}

//-----
// End of exthit.C
//-----
```

```
/******  
exthit.h
```

This is the header file for a C++ class for fast n-dimensional extent overlap checking. An instance of the class is initialized with a call to the constructor.

ExtHit(size) - where size is the maximum number of extents that can be held (and hence tested against each other) in the ExtHit class instance.

There are three other methods in the ExtHit class:

~ExtHit () - the ExtHit class destructor frees the memory allocated for the class instance's internal tables, and destroys the class instance.

BOOL add( Extent &extent, Ptr obj ) - This method adds an extent to the ExtHit class instance. The extent argument is a structure containing the values for the extent, and the obj argument is a pointer to the object for the new extent.

void test (void (\*func)(Ptr d, Ptr e1, Ptr e2), Ptr data) - This method performs the actual extent overlap checking. The func argument is a pointer to a function to be called for each pair of objects whose extents overlap. The data argument allows user specified data to be passed to func. Func is called with three arguments; the user supplied data d, and pointers to the overlapping objects (the pointer passed as the obj argument in the add method).

```
*****/
```

```
#ifndef _EXTHIT_H  
#define _EXTHIT_H
```

```
#ifndef NULL  
#define NULL 0  
#endif
```

```
#ifndef TRUE  
#define FALSE 0  
#define TRUE 1  
#endif
```

```
#ifndef MIN  
#define MIN(a,b) (((a)<(b))?(a):(b))  
#define MAX(a,b) (((a)>(b))?(a):(b))  
#endif
```

```
typedef void* Ptr;  
typedef short BOOL;
```

```
// The number of dimensions to be used in extent checking  
// This constant should be changed to reflect the number  
// of dimensions being checked.  
const int num_dimensions=4;
```

```
typedef struct tagExtent {  
    int min [num_dimensions];  
    int max [num_dimensions];  
} Extent;
```

```
typedef struct tagCollideRecord CollideRecord;
```

```
typedef struct tagMinMaxRecord
{
    CollideRecord *ptr;           // The parent CollideRecord for this MMR
    int value;                   // The extent dimension value for this
MMR
} MinMaxRecord;

typedef MinMaxRecord *MinMaxRecordPtr;

typedef struct tagCollideRecord
{
    int index;                   // The index number of the extent represented
    Ptr obj;                     // A pointer to the extent's parent
geometry
    CollideRecord *nextOpen;     // The next element on the open list
    CollideRecord *prevOpen;     // The previous element on the open list
    CollideRecord *nextActive;   // The next element on the active list
    CollideRecord *prevActive;   // The previous element on the active list
    MinMaxRecord min [num_dimensions]; // MMR's for the minimum extent values
    MinMaxRecord max [num_dimensions]; // MMR's for the maximum extent values
    BOOL active;                 // Flag specifying CR overlapped with another CR
    BOOL open;                   // Flag specifying CR is currently on the open list
} CollideRecord;

/*****
Definition for the ExtHit class
*****/
class ExtHit
{
public:
    ExtHit (int size);           // Class
constructor
    ~ExtHit ();                  // Class
destructor
    BOOL add( Extent &extent, Ptr obj ); // Adds an extent to the collideList
    void test (void (*func)(Ptr d, Ptr e1, Ptr e2), Ptr data);

// Perform extent overlap testing

private:
    int      maxSize;           // Maximum number of extents that
can be held
    int      numColRecs;        // Number of CollideRecords in
the CollideList
    int      numOverlaps;       // Number of MMRs in the overlapList
    MinMaxRecord **overlapList; // Extent values for a single dimension
    BOOL *overlapTable;         // Table specifying which extents
overlapped
    CollideRecord *collideList; // Array holding the extent information
    CollideRecord *activeList;  // The head of the list of active extents

    void subtest ( int dim, int oLstSize, void (*func)(Ptr d, Ptr p1, Ptr p2),
Ptr data);
};

int minMaxCompare( const void* rec1, const void* rec2);

#endif

/*****
```

End of exthit.h

\*\*\*\*\* /

```

/*****
Signed Distance from Point to Plane
Author: Priamos Georgiades
*****/
#include<stdio.h>

typedef struct vect3str {
    float x, y, z;
} vect3;
typedef struct vect4str {
    float x, y, z, w;
} vect4;

#define Vect3Dot(v1, v2) ((v1).x * (v2).x + (v1).y * (v2).y + (v1).z * (v2).z)

/*
Compute the distance from a point to a plane. The plane is
pleq->x * X + pleq->y * Y + pleq->z * Z + pleq->w = 0.
The distance is positive if on same side as the normal, otherwise negative.
Assume the plane normal to be of unit length.
*/
float Pt2Plane(vect3 *pt, vect4 *pleq)
{
    float dist;

    dist = Vect3Dot(*pleq, *pt) + pleq->w;
    return(dist);
}

```

```

/*****
Plane-to-Plane Intersection
Author: Priamos Georgiades
*****/
#include <stdio.h>
#include <math.h>

#define TRUE 1
#define FALSE 0
#define MRG_ZERO 1.0e-8

typedef struct vect3str {
    float x, y, z;
} vect3;
typedef struct vect4str {
    float x, y, z, w;
} vect4;

#define Vect3Init(a, b, c, v) { \
    (v).x = a; (v).y = b; (v).z = c; \
}
#define Vect3Muls(s, v1, v2) { \
    (v2).x = s * (v1).x; \
    (v2).y = s * (v1).y; \
    (v2).z = s * (v1).z; \
}
#define Vect3Cross(v1, v2, v3) { \
    (v3).x = (v1).y * (v2).z - (v1).z * (v2).y; \
    (v3).y = (v1).z * (v2).x - (v1).x * (v2).z; \
    (v3).z = (v1).x * (v2).y - (v1).y * (v2).x; \
}

/*
Calculate the line of intersection between two planes. The two planes are
specified by their equations in the form
    P->x * X + P->y * Y + P->z * Z + P->w = 0.
Initialize the unit direction vector of the line of intersection in xdir.
Pick the point on the line of intersection on the coordinate plane most normal
to xdir. Return TRUE if successful, FALSE otherwise (indicating that the planes
don't intersect). The order in which the planes are given determines the choice
of direction of xdir.
*/
int GetXLine(vect4 *pl1, vect4 *pl2, vect3 *xdir, vect3 *xpt)
{
    float invdet; /* inverse of 2x2 matrix determinant */
    vect3 dir2; /* holds the squares of the coordinates of xdir */

    Vect3Cross(*pl1, *pl2, *xdir)
    dir2.x = xdir->x * xdir->x;
    dir2.y = xdir->y * xdir->y;
    dir2.z = xdir->z * xdir->z;

    if (dir2.z > dir2.y && dir2.z > dir2.x && dir2.z > MRG_ZERO)
    {
        /* then get a point on the XY plane */
        invdet = 1.0 / xdir->z;
        /* solve < pl1.x * xpt.x + pl1.y * xpt.y = - pl1.w >
           < pl2.x * xpt.x + pl2.y * xpt.y = - pl2.w > */
        Vect3Init(pl1->y * pl2->w - pl2->y * pl1->w,
            pl2->x * pl1->w - pl1->x * pl2->w, 0.0, *xpt)
    }
}

```

```
    }
else if (dir2.y > dir2.x && dir2.y > MRG_ZERO)
{
    /* then get a point on the XZ plane */
    invdet = 1.0 / xdir->y;
    /* solve < pl1.x * xpt.x + pl1.z * xpt.z = -pl1.w >
       < pl2.x * xpt.x + pl2.z * xpt.z = -pl2.w > */
    Vect3Init(pl1->z * pl2->w - pl2->z * pl1->w, 0.0,
              pl2->x * pl1->w - pl1->x * pl2->w, *xpt)
}
else if (dir2.x > MRG_ZERO)
{
    /* then get a point on the YZ plane */
    invdet = 1.0 / xdir->x;
    /* solve < pl1.y * xpt.y + pl1.z * xpt.z = - pl1.w >
       < pl2.y * xpt.y + pl2.z * xpt.z = - pl2.w > */
    Vect3Init(0.0, pl1->z * pl2->w - pl2->z * pl1->w,
              pl2->y * pl1->w - pl1->y * pl2->w, *xpt)
}
else /* xdir is zero, then no point of intersection exists */
    return FALSE;
Vect3Muls(invdet, *xpt, *xpt)
invdet = 1.0 / (float)sqrt(dir2.x + dir2.y + dir2.z);
Vect3Muls(invdet, *xdir, *xdir)
return TRUE;
}
```



```
/*
 * GraphicsGems.h
 * Version 1.0 - Andrew Glassner
 * from "Graphics Gems", Academic Press, 1990
 */

#ifndef GG_H

#define GG_H 1

/*****
 * 2d geometry types */
*****/

typedef struct Point2Struct {    /* 2d point */
    double x, y;
} Point2;
typedef Point2 Vector2;

typedef struct IntPoint2Struct {    /* 2d integer point */
    int x, y;
} IntPoint2;

typedef struct Matrix3Struct {    /* 3-by-3 matrix */
    double element[3][3];
} Matrix3;

typedef struct Box2dStruct {    /* 2d box */
    Point2 min, max;
} Box2;

/*****
 * 3d geometry types */
*****/

typedef struct Point3Struct {    /* 3d point */
    double x, y, z;
} Point3;
typedef Point3 Vector3;

typedef struct IntPoint3Struct {    /* 3d integer point */
    int x, y, z;
} IntPoint3;

typedef struct Matrix4Struct {    /* 4-by-4 matrix */
    double element[4][4];
} Matrix4;

typedef struct Box3dStruct {    /* 3d box */
    Point3 min, max;
} Box3;

/*****
 * one-argument macros */
*****/

/* absolute value of a */
```

```
#define ABS(a)          (((a)<0) ? -(a) : (a))

/* round a to nearest int */
#define ROUND(a)        ((a)>0 ? (int)((a)+0.5) : -(int)(0.5-(a)))

/* take sign of a, either -1, 0, or 1 */
#define ZSGN(a)          (((a)<0) ? -1 : (a)>0 ? 1 : 0)

/* take binary sign of a, either -1, or 1 if >= 0 */
#define SGN(a)           (((a)<0) ? -1 : 1)

/* shout if something that should be true isn't */
#define ASSERT(x) \
if (!(x)) fprintf(stderr, " Assert failed: x\n");

/* square a */
#define SQR(a)           ((a)*(a))

/*****/
/* two-argument macros */
/*****/

/* find minimum of a and b */
#define MIN(a,b)         (((a)<(b))?(a):(b))

/* find maximum of a and b */
#define MAX(a,b)         (((a)>(b))?(a):(b))

/* swap a and b (see Gem by Wyvill) */
#define SWAP(a,b)        { a^=b; b^=a; a^=b; }

/* linear interpolation from l (when a=0) to h (when a=1) */
/* (equal to (a*h)+((1-a)*l) */
#define LERP(a,l,h)      ((l)+(((h)-(l))*(a)))

/* clamp the input to the specified range */
#define CLAMP(v,l,h)      ((v)<(l) ? (l) : (v) > (h) ? (h) : v)

/*****/
/* memory allocation macros */
/*****/

/* create a new instance of a structure (see Gem by Hultquist) */
#define NEWSTRUCT(x)      (struct x *) (malloc((unsigned)sizeof(struct x)))

/* create a new instance of a type */
#define NEWTYPE(x)        (x *) (malloc((unsigned)sizeof(x)))

/*****/
/* useful constants */
/*****/

#define PI                3.141592          /* the venerable pi */
#define PITIMES2          6.283185          /* 2 * pi */
#define PIOVER2           1.570796          /* pi / 2 */
#define E                 2.718282          /* the venerable e */
#define SQR2              1.414214          /* sqrt(2) */
#define SQR3              1.732051          /* sqrt(3) */
```

```
#define GOLDEN          1.618034          /* the golden ratio */
#define DTOR            0.017453          /* convert degrees to radians */
#define RTOD            57.29578          /* convert radians to degrees */

/*****/
/* booleans */
/*****/

#define TRUE            1
#define FALSE           0
#define ON              1
#define OFF             0
typedef int boolean;    /* boolean data type */
typedef boolean flag;   /* flag data type */

extern double V2SquaredLength(), V2Length();
extern double V2Dot(), V2DistanceBetween2Points();
extern Vector2 *V2Negate(), *V2Normalize(), *V2Scale(), *V2Add(), *V2Sub();
extern Vector2 *V2Lerp(), *V2Combine(), *V2Mul(), *V2MakePerpendicular();
extern Vector2 *V2New(), *V2Duplicate();
extern Point2 *V2MulPointByMatrix();
extern Matrix3 *V2MatMul();

extern double V3SquaredLength(), V3Length();
extern double V3Dot(), V3DistanceBetween2Points();
extern Vector3 *V3Normalize(), *V3Scale(), *V3Add(), *V3Sub();
extern Vector3 *V3Lerp(), *V3Combine(), *V3Mul(), *V3Cross();
extern Vector3 *V3New(), *V3Duplicate();
extern Point3 *V3MulPointByMatrix();
extern Matrix4 *V3MatMul();

extern double RegulaFalsi(), NewtonRaphson(), findroot();

#endif
```

```
/*
2d and 3d Vector C Library
by Andrew Glassner
from "Graphics Gems", Academic Press, 1990
*/

#include <math.h>
#include "GraphicsGems.h"

/*****
/*    2d Library    */
*****/

/* returns squared length of input vector */
double V2SquaredLength(a)
Vector2 *a;
{
    return((a->x * a->x)+(a->y * a->y));
}

/* returns length of input vector */
double V2Length(a)
Vector2 *a;
{
    return(sqrt(V2SquaredLength(a)));
}

/* negates the input vector and returns it */
Vector2 *V2Negate(v)
Vector2 *v;
{
    v->x = -v->x;  v->y = -v->y;
    return(v);
}

/* normalizes the input vector and returns it */
Vector2 *V2Normalize(v)
Vector2 *v;
{
    double len = V2Length(v);
    if (len != 0.0) { v->x /= len;  v->y /= len; }
    return(v);
}

/* scales the input vector to the new length and returns it */
Vector2 *V2Scale(v, newlen)
Vector2 *v;
double newlen;
{
    double len = V2Length(v);
    if (len != 0.0) { v->x *= newlen/len;  v->y *= newlen/len; }
    return(v);
}

/* return vector sum c = a+b */
Vector2 *V2Add(a, b, c)
Vector2 *a, *b, *c;
{
    c->x = a->x+b->x;  c->y = a->y+b->y;
    return(c);
}
```

```
/* return vector difference c = a-b */
Vector2 *V2Sub(a, b, c)
Vector2 *a, *b, *c;
{
    c->x = a->x-b->x;  c->y = a->y-b->y;
    return(c);
}

/* return the dot product of vectors a and b */
double V2Dot(a, b)
Vector2 *a, *b;
{
    return((a->x*b->x)+(a->y*b->y));
}

/* linearly interpolate between vectors by an amount alpha */
/* and return the resulting vector. */
/* When alpha=0, result=lo.  When alpha=1, result=hi. */
Vector2 *V2Lerp(lo, hi, alpha, result)
Vector2 *lo, *hi, *result;
double alpha;
{
    result->x = LERP(alpha, lo->x, hi->x);
    result->y = LERP(alpha, lo->y, hi->y);
    return(result);
}

/* make a linear combination of two vectors and return the result. */
/* result = (a * ascl) + (b * bscl) */
Vector2 *V2Combine (a, b, result, ascl, bscl)
Vector2 *a, *b, *result;
double ascl, bscl;
{
    result->x = (ascl * a->x) + (bscl * b->x);
    result->y = (ascl * a->y) + (bscl * b->y);
    return(result);
}

/* multiply two vectors together component-wise */
Vector2 *V2Mul (a, b, result)
Vector2 *a, *b, *result;
{
    result->x = a->x * b->x;
    result->y = a->y * b->y;
    return(result);
}

/* return the distance between two points */
double V2DistanceBetween2Points(a, b)
Point2 *a, *b;
{
    double dx = a->x - b->x;
    double dy = a->y - b->y;
    return(sqrt((dx*dx)+(dy*dy)));
}

/* return the vector perpendicular to the input vector a */
Vector2 *V2MakePerpendicular(a, ap)
Vector2 *a, *ap;
```

```
{
    ap->x = -a->y;
    ap->y = a->x;
    return(ap);
}

/* create, initialize, and return a new vector */
Vector2 *V2New(x, y)
double x, y;
{
    Vector2 *v = NEWTYPE(Vector2);
    v->x = x;  v->y = y;
    return(v);
}

/* create, initialize, and return a duplicate vector */
Vector2 *V2Duplicate(a)
Vector2 *a;
{
    Vector2 *v = NEWTYPE(Vector2);
    v->x = a->x;  v->y = a->y;
    return(v);
}

/* multiply a point by a matrix and return the transformed point */
Point2 *V2MulPointByMatrix(p, m)
Point2 *p;
Matrix3 *m;
{
    double w;
    Point2 ptmp;
    ptmp.x = (p->x * m->element[0][0]) +
        (p->y * m->element[1][0]) + m->element[2][0];
    ptmp.y = (p->x * m->element[0][1]) +
        (p->y * m->element[1][1]) + m->element[2][1];
    w = (p->x * m->element[0][2]) +
        (p->y * m->element[1][2]) + m->element[2][2];
    if (w != 0.0) { ptmp.x /= w;  ptmp.y /= w; }
    *p = ptmp;
    return(p);
}

/* multiply together matrices c = ab */
/* note that c must not point to either of the input matrices */
Matrix3 *V2MatMul(a, b, c)
Matrix3 *a, *b, *c;
{
    int i, j, k;
    for (i=0; i<3; i++) {
        for (j=0; j<3; j++) {
            c->element[i][j] = 0;
            for (k=0; k<3; k++) c->element[i][j] +=
                a->element[i][k] * b->element[k][j];
        }
    }
    return(c);
}
```

```

/*****
/*    3d Library    */
*****/

/* returns squared length of input vector */
double V3SquaredLength(a)
Vector3 *a;
{
    return((a->x * a->x)+(a->y * a->y)+(a->z * a->z));
}

/* returns length of input vector */
double V3Length(a)
Vector3 *a;
{
    return(sqrt(V3SquaredLength(a)));
}

/* negates the input vector and returns it */
Vector3 *V3Negate(v)
Vector3 *v;
{
    v->x = -v->x;  v->y = -v->y;  v->z = -v->z;
    return(v);
}

/* normalizes the input vector and returns it */
Vector3 *V3Normalize(v)
Vector3 *v;
{
    double len = V3Length(v);
    if (len != 0.0) { v->x /= len;  v->y /= len; v->z /= len; }
    return(v);
}

/* scales the input vector to the new length and returns it */
Vector3 *V3Scale(v, newlen)
Vector3 *v;
double newlen;
{
    double len = V3Length(v);
    if (len != 0.0) {
        v->x *= newlen/len;  v->y *= newlen/len;  v->z *= newlen/len;
    }
    return(v);
}

/* return vector sum c = a+b */
Vector3 *V3Add(a, b, c)
Vector3 *a, *b, *c;
{
    c->x = a->x+b->x;  c->y = a->y+b->y;  c->z = a->z+b->z;
    return(c);
}

/* return vector difference c = a-b */
Vector3 *V3Sub(a, b, c)
Vector3 *a, *b, *c;
{

```

```
    c->x = a->x-b->x;   c->y = a->y-b->y;   c->z = a->z-b->z;
    return(c);
}
```

/\* return the dot product of vectors a and b \*/

```
double V3Dot(a, b)
Vector3 *a, *b;
{
    return((a->x*b->x)+(a->y*b->y)+(a->z*b->z));
}
```

/\* linearly interpolate between vectors by an amount alpha \*/

/\* and return the resulting vector. \*/

/\* When alpha=0, result=lo. When alpha=1, result=hi. \*/

```
Vector3 *V3Lerp(lo, hi, alpha, result)
Vector3 *lo, *hi, *result;
double alpha;
{
    result->x = LERP(alpha, lo->x, hi->x);
    result->y = LERP(alpha, lo->y, hi->y);
    result->z = LERP(alpha, lo->z, hi->z);
    return(result);
}
```

/\* make a linear combination of two vectors and return the result. \*/

/\* result = (a \* ascl) + (b \* bscl) \*/

```
Vector3 *V3Combine (a, b, result, ascl, bscl)
Vector3 *a, *b, *result;
double ascl, bscl;
{
    result->x = (ascl * a->x) + (bscl * b->x);
    result->y = (ascl * a->y) + (bscl * b->y);
    result->z = (ascl * a->z) + (bscl * b->z);
    return(result);
}
```

/\* multiply two vectors together component-wise and return the result \*/

```
Vector3 *V3Mul (a, b, result)
Vector3 *a, *b, *result;
{
    result->x = a->x * b->x;
    result->y = a->y * b->y;
    result->z = a->z * b->z;
    return(result);
}
```

/\* return the distance between two points \*/

```
double V3DistanceBetween2Points(a, b)
Point3 *a, *b;
{
    double dx = a->x - b->x;
    double dy = a->y - b->y;
    double dz = a->z - b->z;
    return(sqrt((dx*dx)+(dy*dy)+(dz*dz)));
}
```

/\* return the cross product c = a cross b \*/

```
Vector3 *V3Cross(a, b, c)
Vector3 *a, *b, *c;
{
```



```
c->x = (a->y*b->z) - (a->z*b->y);
c->y = (a->z*b->x) - (a->x*b->z);
c->z = (a->x*b->y) - (a->y*b->x);
return(c);
}
```

/\* create, initialize, and return a new vector \*/

Vector3 \*V3New(x, y, z)

double x, y, z;

```
{
Vector3 *v = NEWTYPE(Vector3);
v->x = x; v->y = y; v->z = z;
return(v);
}
```

/\* create, initialize, and return a duplicate vector \*/

Vector3 \*V3Duplicate(a)

Vector3 \*a;

```
{
Vector3 *v = NEWTYPE(Vector3);
v->x = a->x; v->y = a->y; v->z = a->z;
return(v);
}
```

/\* multiply a point by a matrix and return the transformed point \*/

Point3 \*V3MulPointByMatrix(p, m)

Point3 \*p;

Matrix4 \*m;

```
{
double w;
Point3 ptmp;
ptmp.x = (p->x * m->element[0][0]) + (p->y * m->element[1][0]) +
          (p->z * m->element[2][0]) + m->element[3][0];
ptmp.y = (p->x * m->element[0][1]) + (p->y * m->element[1][1]) +
          (p->z * m->element[2][1]) + m->element[3][1];
ptmp.z = (p->x * m->element[0][2]) + (p->y * m->element[1][2]) +
          (p->z * m->element[2][2]) + m->element[3][2];
w =      (p->x * m->element[0][3]) + (p->y * m->element[1][3]) +
          (p->z * m->element[2][3]) + m->element[3][3];
if (w != 0.0) { ptmp.x /= w; ptmp.y /= w; ptmp.z /= w; }
*p = ptmp;
return(p);
}
```

/\* multiply together matrices c = ab \*/

/\* note that c must not point to either of the input matrices \*/

Matrix4 \*V3MatMul(a, b, c)

Matrix4 \*a, \*b, \*c;

```
{
int i, j, k;
for (i=0; i<4; i++) {
    for (j=0; j<4; j++) {
        c->element[i][j] = 0;
        for (k=0; k<4; k++) c->element[i][j] +=
            a->element[i][k] * b->element[k][j];
    }
}
return(c);
}
```

```
/* binary greatest common divisor by Silver and Terzian.  See Knuth */
/* both inputs must be >= 0 */
```

```
gcd(u, v)
int u, v;
{
int t, f;
    if ((u<0) || (v<0)) return(1); /* error if u<0 or v<0 */
    f = 1;
    while ((0 == (u%2)) && (0 == (v%2))) {
        u>>=1; v>>=1, f*=2;
    }
    if (u&01) { t = -v; goto B4; } else { t = u; }
B3: if (t > 0) { t >>= 1; } else { t = -((-t) >> 1); }
B4: if (0 == (t%2)) goto B3;

    if (t > 0) u = t; else v = -t;
    if (0 != (t = u - v)) goto B3;
    return(u*f);
}
```

```
/******
/*    Useful Routines    */
/******
```

```
/* return roots of ax^2+bx+c */
/* stable algebra derived from Numerical Recipes by Press et al.*/
```

```
int quadraticRoots(a, b, c, *roots)
double a, b, c, *roots;
{
double d, q;
int count = 0;
    d = (b*b)-(4*a*c);
    if (d < 0.0) { *roots = *(roots+1) = 0.0; return(0); }
    q = -0.5 * (b + (SGN(b)*sqrt(d)));
    if (a != 0.0) { *roots++ = q/a; count++; }
    if (q != 0.0) { *roots++ = c/q; count++; }
    return(count);
}
```

```
/* generic 1d regula-falsi step. f is function to evaluate */
/* interval known to contain root is given in left, right */
/* returns new estimate */
```

```
double RegulaFalsi(f, left, right)
double (*f)(), left, right;
{
double d = (*f)(right) - (*f)(left);
    if (d != 0.0) return (right - (*f)(right)*(right-left)/d);
    return((left+right)/2.0);
}
```

```
/* generic 1d Newton-Raphson step. f is function, df is derivative */
/* x is current best guess for root location. Returns new estimate */
```

```
double NewtonRaphson(f, df, x)
double (*f)(), (*df)(), x;
{
double d = (*df)(x);
    if (d != 0.0) return (x-((*f)(x)/d));
    return(x-1.0);
}
```

```
/* hybrid 1d Newton-Raphson/Regula Falsi root finder. */
/* input function f and its derivative df, an interval */
/* left, right known to contain the root, and an error tolerance */
/* Based on Blinn */
double findroot(left, right, tolerance, f, df)
double left, right, tolerance;
double (*f)(), (*df)();
{
double newx = left;
    while (ABS((*f)(newx)) > tolerance) {
        newx = NewtonRaphson(f, df, newx);
        if (newx < left || newx > right)
            newx = RegulaFalsi(f, left, right);
        if ((*f)(newx) * (*f)(left) <= 0.0) right = newx;
        else left = newx;
    }
return(newx);
}
```

```
/*
 * GraphicsGems.h
 * Version 1.0 - Andrew Glassner
 * from "Graphics Gems", Academic Press, 1990
 */

#ifndef GG_H

#define GG_H 1

/*****
 * 2d geometry types */
*****/

typedef struct Point2Struct {    /* 2d point */
    double x, y;
} Point2;
typedef Point2 Vector2;

typedef struct IntPoint2Struct {    /* 2d integer point */
    int x, y;
} IntPoint2;

typedef struct Matrix3Struct {    /* 3-by-3 matrix */
    double element[3][3];
} Matrix3;

typedef struct Box2dStruct {    /* 2d box */
    Point2 min, max;
} Box2;

/*****
 * 3d geometry types */
*****/

typedef struct Point3Struct {    /* 3d point */
    double x, y, z;
} Point3;
typedef Point3 Vector3;

typedef struct IntPoint3Struct {    /* 3d integer point */
    int x, y, z;
} IntPoint3;

typedef struct Matrix4Struct {    /* 4-by-4 matrix */
    double element[4][4];
} Matrix4;

typedef struct Box3dStruct {    /* 3d box */
    Point3 min, max;
} Box3;

/*****
 * one-argument macros */
*****/

/* absolute value of a */
```

```
#define ABS(a)          (((a)<0) ? -(a) : (a))

/* round a to nearest int */
#define ROUND(a)        floor((a)+0.5)

/* take sign of a, either -1, 0, or 1 */
#define ZSGN(a)         (((a)<0) ? -1 : (a)>0 ? 1 : 0)

/* take binary sign of a, either -1, or 1 if >= 0 */
#define SGN(a)          (((a)<0) ? -1 : 1)

/* shout if something that should be true isn't */
#define ASSERT(x) \
if (!(x)) fprintf(stderr, " Assert failed: x\n");

/* square a */
#define SQR(a)          ((a)*(a))

/*****/
/* two-argument macros */
/*****/

/* find minimum of a and b */
#define MIN(a,b)        (((a)<(b))?(a):(b))

/* find maximum of a and b */
#define MAX(a,b)        (((a)>(b))?(a):(b))

/* swap a and b (see Gem by Wyvill) */
#define SWAP(a,b)        { a^=b; b^=a; a^=b; }

/* linear interpolation from l (when a=0) to h (when a=1) */
/* (equal to (a*h)+((1-a)*l) */
#define LERP(a,l,h)      ((l)+(((h)-(l))*(a)))

/* clamp the input to the specified range */
#define CLAMP(v,l,h)     ((v)<(l) ? (l) : (v) > (h) ? (h) : v)

/*****/
/* memory allocation macros */
/*****/

/* create a new instance of a structure (see Gem by Hultquist) */
#define NEWSTRUCT(x)     (struct x *) (malloc((unsigned)sizeof(struct x)))

/* create a new instance of a type */
#define NEWTYPE(x)        (x *) (malloc((unsigned)sizeof(x)))

/*****/
/* useful constants */
/*****/

#define PI                3.141592          /* the venerable pi */
#define PITIMES2          6.283185          /* 2 * pi */
#define PIOVER2           1.570796          /* pi / 2 */
#define E                 2.718282          /* the venerable e */
#define SQR2              1.414214          /* sqrt(2) */
#define SQR3              1.732051          /* sqrt(3) */
```

```
#define GOLDEN          1.618034          /* the golden ratio */
#define DTOR            0.017453          /* convert degrees to radians */
#define RTOD            57.29578          /* convert radians to degrees */

/*****/
/* booleans */
/*****/

#define TRUE            1
#define FALSE          0
#define ON              1
#define OFF            0
typedef int boolean;    /* boolean data type */
typedef boolean flag;   /* flag data type */

extern double V2SquaredLength(), V2Length();
extern double V2Dot(), V2DistanceBetween2Points();
extern Vector2 *V2Negate(), *V2Normalize(), *V2Scale(), *V2Add(), *V2Sub();
extern Vector2 *V2Lerp(), *V2Combine(), *V2Mul(), *V2MakePerpendicular();
extern Vector2 *V2New(), *V2Duplicate();
extern Point2 *V2MulPointByProjMatrix();
extern Matrix3 *V2MatMul(), *TransposeMatrix3();

extern double V3SquaredLength(), V3Length();
extern double V3Dot(), V3DistanceBetween2Points();
extern Vector3 *V3Normalize(), *V3Scale(), *V3Add(), *V3Sub();
extern Vector3 *V3Lerp(), *V3Combine(), *V3Mul(), *V3Cross();
extern Vector3 *V3New(), *V3Duplicate();
extern Point3 *V3MulPointByMatrix(), *V3MulPointByProjMatrix();
extern Matrix4 *V3MatMul();

extern double RegulaFalsi(), NewtonRaphson(), findroot();

#endif
```

```
/* GGVecLib.c */
/*
2d and 3d Vector C Library
by Andrew Glassner
from "Graphics Gems", Academic Press, 1990
*/

#include <math.h>
#include "GraphicsGems.h"

/*****/
/*    2d Library    */
/*****/

/* returns squared length of input vector */
double V2SquaredLength(a)
Vector2 *a;
{
    return((a->x * a->x)+(a->y * a->y));
}

/* returns length of input vector */
double V2Length(a)
Vector2 *a;
{
    return(sqrt(V2SquaredLength(a)));
}

/* negates the input vector and returns it */
Vector2 *V2Negate(v)
Vector2 *v;
{
    v->x = -v->x;  v->y = -v->y;
    return(v);
}

/* normalizes the input vector and returns it */
Vector2 *V2Normalize(v)
Vector2 *v;
{
    double len = V2Length(v);
    if (len != 0.0) { v->x /= len;  v->y /= len; }
    return(v);
}

/* scales the input vector to the new length and returns it */
Vector2 *V2Scale(v, newlen)
Vector2 *v;
double newlen;
{
    double len = V2Length(v);
    if (len != 0.0) { v->x *= newlen/len;  v->y *= newlen/len; }
    return(v);
}

/* return vector sum c = a+b */
Vector2 *V2Add(a, b, c)
Vector2 *a, *b, *c;
{
    c->x = a->x+b->x;  c->y = a->y+b->y;
    return(c);
}
```

```
}
```

```
/* return vector difference c = a-b */
```

```
Vector2 *V2Sub(a, b, c)
```

```
Vector2 *a, *b, *c;
```

```
{
    c->x = a->x-b->x;  c->y = a->y-b->y;
    return(c);
}
```

```
/* return the dot product of vectors a and b */
```

```
double V2Dot(a, b)
```

```
Vector2 *a, *b;
```

```
{
    return((a->x*b->x)+(a->y*b->y));
}
```

```
/* linearly interpolate between vectors by an amount alpha */
```

```
/* and return the resulting vector. */
```

```
/* When alpha=0, result=lo.  When alpha=1, result=hi. */
```

```
Vector2 *V2Lerp(lo, hi, alpha, result)
```

```
Vector2 *lo, *hi, *result;
```

```
double alpha;
```

```
{
    result->x = LERP(alpha, lo->x, hi->x);
    result->y = LERP(alpha, lo->y, hi->y);
    return(result);
}
```

```
/* make a linear combination of two vectors and return the result. */
```

```
/* result = (a * ascl) + (b * bscl) */
```

```
Vector2 *V2Combine (a, b, result, ascl, bscl)
```

```
Vector2 *a, *b, *result;
```

```
double ascl, bscl;
```

```
{
    result->x = (ascl * a->x) + (bscl * b->x);
    result->y = (ascl * a->y) + (bscl * b->y);
    return(result);
}
```

```
/* multiply two vectors together component-wise */
```

```
Vector2 *V2Mul (a, b, result)
```

```
Vector2 *a, *b, *result;
```

```
{
    result->x = a->x * b->x;
    result->y = a->y * b->y;
    return(result);
}
```

```
/* return the distance between two points */
```

```
double V2DistanceBetween2Points(a, b)
```

```
Point2 *a, *b;
```

```
{
    double dx = a->x - b->x;
    double dy = a->y - b->y;
    return(sqrt((dx*dx)+(dy*dy)));
}
```

```
/* return the vector perpendicular to the input vector a */
```

```
Vector2 *V2MakePerpendicular(a, ap)
```



```
Vector2 *a, *ap;
{
    ap->x = -a->y;
    ap->y = a->x;
    return(ap);
}

/* create, initialize, and return a new vector */
Vector2 *V2New(x, y)
double x, y;
{
    Vector2 *v = NEWTYPE(Vector2);
    v->x = x;  v->y = y;
    return(v);
}

/* create, initialize, and return a duplicate vector */
Vector2 *V2Duplicate(a)
Vector2 *a;
{
    Vector2 *v = NEWTYPE(Vector2);
    v->x = a->x;  v->y = a->y;
    return(v);
}

/* multiply a point by a projective matrix and return the transformed point */
Point2 *V2MulPointByProjMatrix(pin, m, pout)
Point2 *pin, *pout;
Matrix3 *m;
{
    double w;
    pout->x = (pin->x * m->element[0][0]) +
        (pin->y * m->element[1][0]) + m->element[2][0];
    pout->y = (pin->x * m->element[0][1]) +
        (pin->y * m->element[1][1]) + m->element[2][1];
    w = (pin->x * m->element[0][2]) +
        (pin->y * m->element[1][2]) + m->element[2][2];
    if (w != 0.0) { pout->x /= w;  pout->y /= w; }
    return(pout);
}

/* multiply together matrices c = ab */
/* note that c must not point to either of the input matrices */
Matrix3 *V2MatMul(a, b, c)
Matrix3 *a, *b, *c;
{
    int i, j, k;
    for (i=0; i<3; i++) {
        for (j=0; j<3; j++) {
            c->element[i][j] = 0;
            for (k=0; k<3; k++) c->element[i][j] +=
                a->element[i][k] * b->element[k][j];
        }
    }
    return(c);
}

/* transpose matrix a, return b */
Matrix3 *TransposeMatrix3(a, b)
Matrix3 *a, *b;
```

```
{
int i, j;
    for (i=0; i<3; i++) {
        for (j=0; j<3; j++)
            b->element[i][j] = a->element[j][i];
    }
    return(b);
}

/*****/
/*    3d Library    */
/*****/

/* returns squared length of input vector */
double V3SquaredLength(a)
Vector3 *a;
{
    return((a->x * a->x)+(a->y * a->y)+(a->z * a->z));
}

/* returns length of input vector */
double V3Length(a)
Vector3 *a;
{
    return(sqrt(V3SquaredLength(a)));
}

/* negates the input vector and returns it */
Vector3 *V3Negate(v)
Vector3 *v;
{
    v->x = -v->x;  v->y = -v->y;  v->z = -v->z;
    return(v);
}

/* normalizes the input vector and returns it */
Vector3 *V3Normalize(v)
Vector3 *v;
{
    double len = V3Length(v);
    if (len != 0.0) { v->x /= len;  v->y /= len; v->z /= len; }
    return(v);
}

/* scales the input vector to the new length and returns it */
Vector3 *V3Scale(v, newlen)
Vector3 *v;
double newlen;
{
    double len = V3Length(v);
    if (len != 0.0) {
        v->x *= newlen/len;  v->y *= newlen/len;  v->z *= newlen/len;
    }
    return(v);
}

/* return vector sum c = a+b */
```

```
Vector3 *V3Add(a, b, c)
Vector3 *a, *b, *c;
{
    c->x = a->x+b->x;  c->y = a->y+b->y;  c->z = a->z+b->z;
    return(c);
}

/* return vector difference c = a-b */
Vector3 *V3Sub(a, b, c)
Vector3 *a, *b, *c;
{
    c->x = a->x-b->x;  c->y = a->y-b->y;  c->z = a->z-b->z;
    return(c);
}

/* return the dot product of vectors a and b */
double V3Dot(a, b)
Vector3 *a, *b;
{
    return((a->x*b->x)+(a->y*b->y)+(a->z*b->z));
}

/* linearly interpolate between vectors by an amount alpha */
/* and return the resulting vector. */
/* When alpha=0, result=lo.  When alpha=1, result=hi. */
Vector3 *V3Lerp(lo, hi, alpha, result)
Vector3 *lo, *hi, *result;
double alpha;
{
    result->x = LERP(alpha, lo->x, hi->x);
    result->y = LERP(alpha, lo->y, hi->y);
    result->z = LERP(alpha, lo->z, hi->z);
    return(result);
}

/* make a linear combination of two vectors and return the result. */
/* result = (a * ascl) + (b * bscl) */
Vector3 *V3Combine (a, b, result, ascl, bscl)
Vector3 *a, *b, *result;
double ascl, bscl;
{
    result->x = (ascl * a->x) + (bscl * b->x);
    result->y = (ascl * a->y) + (bscl * b->y);
    result->z = (ascl * a->z) + (bscl * b->z);
    return(result);
}

/* multiply two vectors together component-wise and return the result */
Vector3 *V3Mul (a, b, result)
Vector3 *a, *b, *result;
{
    result->x = a->x * b->x;
    result->y = a->y * b->y;
    result->z = a->z * b->z;
    return(result);
}

/* return the distance between two points */
double V3DistanceBetween2Points(a, b)
Point3 *a, *b;
```

```
{
double dx = a->x - b->x;
double dy = a->y - b->y;
double dz = a->z - b->z;
    return(sqrt((dx*dx)+(dy*dy)+(dz*dz)));
}

/* return the cross product c = a cross b */
Vector3 *V3Cross(a, b, c)
Vector3 *a, *b, *c;
{
    c->x = (a->y*b->z) - (a->z*b->y);
    c->y = (a->z*b->x) - (a->x*b->z);
    c->z = (a->x*b->y) - (a->y*b->x);
    return(c);
}

/* create, initialize, and return a new vector */
Vector3 *V3New(x, y, z)
double x, y, z;
{
Vector3 *v = NEWTYPE(Vector3);
    v->x = x;  v->y = y;  v->z = z;
    return(v);
}

/* create, initialize, and return a duplicate vector */
Vector3 *V3Duplicate(a)
Vector3 *a;
{
Vector3 *v = NEWTYPE(Vector3);
    v->x = a->x;  v->y = a->y;  v->z = a->z;
    return(v);
}

/* multiply a point by a matrix and return the transformed point */
Point3 *V3MulPointByMatrix(pin, m, pout)
Point3 *pin, *pout;
Matrix3 *m;
{
    pout->x = (pin->x * m->element[0][0]) + (pin->y * m->element[1][0]) +
        (pin->z * m->element[2][0]);
    pout->y = (pin->x * m->element[0][1]) + (pin->y * m->element[1][1]) +
        (pin->z * m->element[2][1]);
    pout->z = (pin->x * m->element[0][2]) + (pin->y * m->element[1][2]) +
        (pin->z * m->element[2][2]);
    return(pout);
}

/* multiply a point by a projective matrix and return the transformed point */
Point3 *V3MulPointByProjMatrix(pin, m, pout)
Point3 *pin, *pout;
Matrix4 *m;
{
double w;
    pout->x = (pin->x * m->element[0][0]) + (pin->y * m->element[1][0]) +
        (pin->z * m->element[2][0]) + m->element[3][0];
    pout->y = (pin->x * m->element[0][1]) + (pin->y * m->element[1][1]) +
        (pin->z * m->element[2][1]) + m->element[3][1];
    pout->z = (pin->x * m->element[0][2]) + (pin->y * m->element[1][2]) +
```

```
        (pin->z * m->element[2][2]) + m->element[3][2];
w =      (pin->x * m->element[0][3]) + (pin->y * m->element[1][3]) +
        (pin->z * m->element[2][3]) + m->element[3][3];
if (w != 0.0) { pout->x /= w; pout->y /= w; pout->z /= w; }
return(pout);
}
```

```
/* multiply together matrices c = ab */
/* note that c must not point to either of the input matrices */
Matrix4 *V3MatMul(a, b, c)
Matrix4 *a, *b, *c;
{
int i, j, k;
for (i=0; i<4; i++) {
for (j=0; j<4; j++) {
c->element[i][j] = 0;
for (k=0; k<4; k++) c->element[i][j] +=
a->element[i][k] * b->element[k][j];
}
}
return(c);
}
```

```
/* binary greatest common divisor by Silver and Terzian. See Knuth */
/* both inputs must be >= 0 */
gcd(u, v)
int u, v;
{
int t, f;
if ((u<0) || (v<0)) return(1); /* error if u<0 or v<0 */
f = 1;
while ((0 == (u%2)) && (0 == (v%2))) {
u>>=1; v>>=1, f*=2;
}
if (u&01) { t = -v; goto B4; } else { t = u; }
B3: if (t > 0) { t >>= 1; } else { t = -((-t) >> 1); }
B4: if (0 == (t%2)) goto B3;

if (t > 0) u = t; else v = -t;
if (0 != (t = u - v)) goto B3;
return(u*f);
}
```

```
/******
/* Useful Routines */
/******
```

```
/* return roots of ax^2+bx+c */
/* stable algebra derived from Numerical Recipes by Press et al.*/
int quadraticRoots(a, b, c, roots)
double a, b, c, *roots;
{
double d, q;
int count = 0;
d = (b*b)-(4*a*c);
if (d < 0.0) { *roots = *(roots+1) = 0.0; return(0); }
q = -0.5 * (b + (SGN(b)*sqrt(d)));
if (a != 0.0) { *roots++ = q/a; count++; }
if (q != 0.0) { *roots++ = c/q; count++; }
return(count);
}
```






```
/* generic 1d regula-falsi step. f is function to evaluate */
/* interval known to contain root is given in left, right */
/* returns new estimate */
double RegulaFalsi(f, left, right)
double (*f)(), left, right;
{
double d = (*f)(right) - (*f)(left);
    if (d != 0.0) return (right - (*f)(right)*(right-left)/d);
    return((left+right)/2.0);
}

/* generic 1d Newton-Raphson step. f is function, df is derivative */
/* x is current best guess for root location. Returns new estimate */
double NewtonRaphson(f, df, x)
double (*f)(), (*df)(), x;
{
double d = (*df)(x);
    if (d != 0.0) return (x-((*f)(x)/d));
    return(x-1.0);
}

/* hybrid 1d Newton-Raphson/Regula Falsi root finder. */
/* input function f and its derivative df, an interval */
/* left, right known to contain the root, and an error tolerance */
/* Based on Blinn */
double findroot(left, right, tolerance, f, df)
double left, right, tolerance;
double (*f)(), (*df)();
{
double newx = left;
    while (ABS((*f)(newx)) > tolerance) {
        newx = NewtonRaphson(f, df, newx);
        if (newx < left || newx > right)
            newx = RegulaFalsi(f, left, right);
        if ((*f)(newx) * (*f)(left) <= 0.0) right = newx;
        else left = newx;
    }
return(newx);
}
```

# Index of

## /pubs/tog/GraphicsGems/gemsiv/vert\_norm/

Name	Last modified	Size	Description
 <a href="#">Parent Directory</a>			
 <a href="#">Makefile</a>	29-Jun-00 08:21	1K	
 <a href="#">smooth.c</a>	29-Jun-00 08:21	8K	
 <a href="#">smooth.h</a>	29-Jun-00 08:21	2K	
 <a href="#">test.c</a>	29-Jun-00 08:21	3K	

```
CC = cc
CFLAGS = -I/usr/ph/include -DSTANDALONE_TEST

test: test.o smooth.o
    $(CC) -o test test.o smooth.o -lm

clean:
    rm -f *.o test

smooth.o: smooth.h
test.o: smooth.h
```



```
/*
 * ANSI C code from the article
 * "Building Vertex Normals from an Unstructured Polygon List"
 * by Andrew Glassner, glassner@parc.xerox.com
 * in "Graphics Gems IV", Academic Press, 1994
 */
```

```
/* smooth.c - Compute vertex normals for polygons.
   Andrew S. Glassner / Xerox PARC
```

The general idea is to 1) initialize the tables, 2) add polygons one by one, 3) optionally enable edge preservation, 4) optionally set the fuzz factor, 5) compute the normals, 6) do something with the new normals, then 7) free the new storage. The calls to do this are:

```
1) smooth = initAllTables();
2) includePolygon(int numVerts, Point3 *verts, Smooth smooth);
3) (optional) enableEdgePreservation(Smooth smooth, float minDot);
4) (optional) setFuzzFraction(smooth Smooth, float fuzzFraction);
5) makeVertexNormals(smooth);
6) YOUR CODE
7) freeSmooth();
```

Edge preservation is used to retain sharp creases in the model. If it is enabled, then the dot product of each pair of polygons sharing a vertex is computed. If this value is below the value of 'minDot' (that is, the two polygons are a certain distance away from flatness), then the polygons are not averaged together for the vertex normal.

If you want to re-compute the results without edge preservation, call `disableEdgePreservation(smooth);`

The general flow of the algorithm is:

```
1. currentHash = scan hashTable
2. while (any unmarked) {
    3. firstVertex = first unmarked vertex. set to MARKWORKING
    4. normal = firstVertex->polygon->normal
    5. scan currentHash. If vertex = firstVertex
        6. normal += vertex->polygon->normal
        7. set vertex to MARKWORKING
        (end of scan)
    8. set normal to unit length
    9. scan currentHash. If vertex set to MARKWORKING
        10. set vertex->normal = normal
        11. set to MARKDONE
        (end of scan)
    (end while)
```

The HASH macro must always return a non-negative value, even for negative inputs. The size of the hash table can be adjusted to taste.

The fuzz for comparison needs to be matched to the resolution of the model.

```
*/
```

```
#include "smooth.h"
```

```
void      addVertexToTable(Point3 *pt, Polygon polygon, int vNum, Smooth smooth);
void      makePolyNormal(Polygon polygon);
void      writeSmooth(FILE *fp, int numPolys);
HashNode  getFirstWaitingNode(HashNode node);
void      processHashNode(HashNode headNode, HashNode firstNode, Smooth smooth);
int       hashPolys(boolean phase);
```

```
void      writeGeom(int numPolys);
void      freeSmooth(Smooth smooth);
boolean   compareVerts(Point3 *v0, Point3 *v1, Smooth smooth);
void      computeFuzz(Smooth smooth);

/***** ENTRY PROCS *****/

/* add this polygon to the tables */
void includePolygon(int numVerts, Point3 *verts, Smooth smooth, void *user) {
int i;
Point3 *vp, *ovp;
    Polygon polygon = NEWTYPE(Polygon_def);
    polygon->next = NULL;
    if (smooth->polyTail != NULL) {
        smooth->polyTail->next = polygon;
    } else {
        smooth->polyonTable = polygon;
    };
    smooth->polyTail = polygon;
    polygon->vertices = NEWA(struct Point3Struct, numVerts);
    polygon->normals = NEWA(struct Point3Struct, numVerts);
    polygon->user = user;
    vp = polygon->vertices;
    ovp = verts;
    polygon->numVerts = numVerts;

    for (i=0; i<numVerts; i++) {
        vp->x = ovp->x;
        vp->y = ovp->y;
        vp->z = ovp->z;
        addVertexToTable(vp, polygon, i, smooth);
        vp++;
        ovp++;
    };
    makePolyNormal(polygon);
}

void enableEdgePreservation(Smooth smooth, float minDot) {
    smooth->edgeTest = TRUE;
    smooth->minDot = minDot;
}

void disableEdgePreservation(Smooth smooth) {
    smooth->edgeTest = FALSE;
}

void setFuzzFraction(Smooth smooth, float fuzzFraction) {
    smooth->fuzzFraction = fuzzFraction;
}

/***** PROCEDURES *****/

/* set all the hash-table linked lists to NULL */
Smooth initAllTables() {
int i;
Smooth smooth = NEWTYPE(Smooth_def);
    for (i=0; i<HASH_TABLE_SIZE; i++) smooth->hashTable[i] = NULL;
    smooth->polyonTable = NULL;
    smooth->polyTail = NULL;
    smooth->edgeTest = FALSE;
    smooth->minDot = 0.2;
}
```

```
smooth->fuzzFraction = 0.001;
smooth->fuzz = 0.001;
return(smooth);
}
```

```
/* hash this vertex and add it into the linked list */
void addVertexToTable(Point3 *pt, Polygon polygon, int vNum, Smooth smooth) {
int hash = HASH(pt);
HashNode newNode = NEWTYPE(HashNode_def);
newNode->next = smooth->hashTable[hash];
smooth->hashTable[hash] = newNode;
newNode->polygon = polygon;
newNode->vertexNum = vNum;
newNode->marked = MARKWAITING;
}
```

```
/* compute the normal for this polygon using Newell's method */
/* (see Tampieri, Gems III, pg 517) */
void makePolyNormal(Polygon polygon) {
Point3 *vp, *p0, *p1;
int i;
polygon->normal.x = 0.0; polygon->normal.y = 0.0; polygon->normal.z = 0.0;
vp = polygon->vertices;
for (i=0; i<polygon->numVerts; i++) {
p0 = vp++;
p1 = vp;
if (i == polygon->numVerts-1) p1 = polygon->vertices;
polygon->normal.x += (p1->y - p0->y) * (p1->z + p0->z);
polygon->normal.y += (p1->z - p0->z) * (p1->x + p0->x);
polygon->normal.z += (p1->x - p0->x) * (p1->y + p0->y);
};
(void) V3Normalize(&(polygon->normal));
}
```

```
/* scan each list at each hash table entry until all nodes are marked */
void makeVertexNormals(Smooth smooth) {
HashNode currentHashNode;
HashNode firstNode;
int i;
computeFuzz(smooth);
for (i=0; i<HASH_TABLE_SIZE; i++) {
currentHashNode = smooth->hashTable[i];
do {
firstNode = getFirstWaitingNode(currentHashNode);
if (firstNode != NULL) {
processHashNode(currentHashNode, firstNode, smooth);
};
} while (firstNode != NULL);
};
}
```

```
void computeFuzz(Smooth smooth) {
Point3 min, max;
double od, d;
Point3 *v;
int i;
Polygon poly = smooth->polygonTable;
min.x = max.x = poly->vertices->x;
min.y = max.y = poly->vertices->y;
min.z = max.z = poly->vertices->z;
while (poly != NULL) {
```

```
v = poly->vertices;
for (i=0; i<poly->numVerts; i++) {
    if (v->x < min.x) min.x = v->x;
    if (v->y < min.y) min.y = v->y;
    if (v->z < min.z) min.z = v->z;
    if (v->x > max.x) max.x = v->x;
    if (v->y > max.y) max.y = v->y;
    if (v->z > max.z) max.z = v->z;
    v++;
};
poly = poly->next;
};
d = fabs(max.x - min.x);
od = fabs(max.y - min.y); if (od > d) d = od;
od = fabs(max.z - min.z); if (od > d) d = od;
smooth->fuzz = od * smooth->fuzzFraction;
}

/* get first node in this list that isn't marked as done */
HashNode getFirstWaitingNode(HashNode node) {
    while (node != NULL) {
        if (node->marked != MARKDONE) return(node);
        node = node->next;
    };
    return(NULL);
}

/* are these two vertices the same to within the tolerance? */
boolean compareVerts(Point3 *v0, Point3 *v1, Smooth smooth) {
    float q0, q1;
    q0 = QUANT(v0->x); q1 = QUANT(v1->x); if (!FUZZEQ(q0, q1)) return(FALSE);
    q0 = QUANT(v0->y); q1 = QUANT(v1->y); if (!FUZZEQ(q0, q1)) return(FALSE);
    q0 = QUANT(v0->z); q1 = QUANT(v1->z); if (!FUZZEQ(q0, q1)) return(FALSE);
    return(TRUE);
}

/* compute the normal for an unmarked vertex */
void processHashNode(HashNode headNode, HashNode firstNode, Smooth smooth) {
    HashNode scanNode = firstNode->next;
    Point3 *firstVert = &(firstNode->polygon->vertices[firstNode->vertexNum]);
    Point3 *headNorm = &(firstNode->polygon->normal);
    Point3 *testVert, *testNorm;
    Point3 normal;
    float ndot;

    firstNode->marked = MARKWORKING;
    normal.x = firstNode->polygon->normal.x;
    normal.y = firstNode->polygon->normal.y;
    normal.z = firstNode->polygon->normal.z;

    while (scanNode != NULL) {
        testVert = &(scanNode->polygon->vertices[scanNode->vertexNum]);
        if (compareVerts(testVert, firstVert, smooth)) {
            testNorm = &(scanNode->polygon->normal);
            ndot = V3Dot(testNorm, headNorm);

            if (((smooth->edgeTest)) || (ndot > smooth->minDot)) {
                V3Add(&normal, testNorm, &normal);
                scanNode->marked = MARKWORKING;
            };
        };
    };
}
```

```
scanNode = scanNode->next;
};
```

```
V3Normalize(&normal);
```

```
scanNode = firstNode;
while (scanNode != NULL) {
    if (scanNode->marked == MARKWORKING) {
        testNorm = &(scanNode->polygon->normals[scanNode->vertexNum]);
        testNorm->x = normal.x;
        testNorm->y = normal.y;
        testNorm->z = normal.z;
        scanNode->marked = MARKDONE;
        testVert = &(scanNode->polygon->vertices[scanNode->vertexNum]);
    };
    scanNode = scanNode->next;
};
}
```

```
/* free up all the memory */
```

```
void freeSmooth(Smooth smooth) {
    HashNode headNode;
    HashNode nextNode;
    Polygon poly;
    Polygon nextPoly;
    int i;
    for (i=0; i<HASH_TABLE_SIZE; i++) {
        headNode = smooth->hashTable[i];
        while (headNode != NULL) {
            nextNode = headNode->next;
            free(headNode);
            headNode = nextNode;
        };
    };
    poly = smooth->polygonTable;
    while (poly != NULL) {
        nextPoly = poly->next;
        freePoly(poly);
        poly = nextPoly;
    };
    smooth->polygonTable = NULL;
    free(smooth);
}
```

```
freePoly(polygon) Polygon polygon; {
    if (polygon->vertices != NULL) free(polygon->vertices);
    if (polygon->normals != NULL) free(polygon->normals);
    polygon->next = NULL;
    free(polygon);
}
```

```
/* smooth.h */
/* header file for polygon smoothing */
/* Andrew S. Glassner / Xerox PARC */

#include <stdio.h>
#include <math.h>

#ifdef STANDALONE_TEST

#define NEWTYPE(x) (x *)malloc((unsigned)(sizeof(x)))

typedef struct Point3Struct {
    double x, y, z;
} Point3;
typedef Point3 Vector3;
typedef int boolean;
#define TRUE 1
#define FALSE 0

Vector3 *V3Normalize(Vector3 *v);
Vector3 *V3Add(Vector3 *a, Vector3 *b, Vector3 *c);
double V3Dot(Vector3 *a, Vector3 *b);
#else
#include "GraphicsGems.h"
#endif

/***** MACROS and CONSTANTS *****/

/* new array creator */
#define NEWA(x, num) (x *)malloc((unsigned)((num) * sizeof(x)))

#define MARKWAITING 0
#define MARKWORKING 1
#define MARKDONE 2

/* fuzzy comparison macro */
#define FUZZEQ(x,y) (fabs((x)-(y))<(smooth->fuzz))

/* hash table size; related to HASH */
#define HASH_TABLE_SIZE 1000

/* quantization increment */
#define QSIZE 1000.0

#define QUANT(x) (((int)((x)*QSIZE))/QSIZE)
#define ABSQUANT(x) (((int)((fabs(x))*QSIZE))/QSIZE)
#define HASH(pt) ( \
    (int)(((3*ABSQUANT(pt->x)) + \
    (5*ABSQUANT(pt->y)) + \
    (7*ABSQUANT(pt->z))) * \
    HASH_TABLE_SIZE) % HASH_TABLE_SIZE

/***** STRUCTS AND TYPES *****/

typedef struct Polygonstruct {
    Point3 *vertices; /* polygon vertices */
    Vector3 *normals; /* normal at each vertex */
    Vector3 normal; /* normal for polygon */
    int numVerts; /* number of vertices */
    void *user; /* user information */
    struct Polygonstruct *next;
```

```
    } Polygon_def;
typedef Polygon_def *Polygon;

typedef struct HashNodestruct {
    Polygon    polygon;        /* polygon for this vertex */
    int        vertexNum;      /* which vertex this is */
    int        marked;         /* vertex status */
    struct     HashNodestruct *next;
} HashNode_def;
typedef HashNode_def *HashNode;

typedef struct SmoothStruct {
    HashNode    hashTable[HASH_TABLE_SIZE];
    Polygon     polygonTable;
    Polygon     polyTail;
    double      fuzz;          /* distance for vertex equality */
    double      fuzzFraction;  /* fraction of model size for fuzz */
    boolean     edgeTest;      /* apply edging test using minDot */
    float       minDot;        /* if > this, make sharp edge; see above */
} Smooth_def;
typedef Smooth_def *Smooth;

/***** public procs *****/
Smooth    initAllTables();
void      includePolygon(int numVerts, Point3 *verts, Smooth smooth, void *user);
void      makeVertexNormals(Smooth smooth);

/***** public option control procs *****/
void      setFuzzFraction(Smooth smooth, float fuzzFraction);
void      enableEdgePreservation(Smooth smooth, float minDot);
void      disableEdgePreservation(Smooth smooth);
```

```
/* test.c - sample driver for polygon smoother */
/* makes a mesh height field of quadrilaterals and triangles */
/* Andrew S. Glassner / Xerox PARC */

#include "smooth.h"

#ifdef STANDALONE_TEST
/* from Graphics Gems library ; for standalone compile */

/* normalizes the input vector and returns it */
Vector3 *V3Normalize(Vector3 *v) {
    double len = sqrt(V3Dot(v, v));
    if (len != 0.0) { v->x /= len; v->y /= len; v->z /= len; }
    return(v);
}

/* return vector sum c = a+b */
Vector3 *V3Add(Vector3 *a, Vector3 *b, Vector3 *c) {
    c->x = a->x+b->x; c->y = a->y+b->y; c->z = a->z+b->z;
    return(c);
}

/* return the dot product of vectors a and b */
double V3Dot(Vector3 *a, Vector3 *b) {
    return((a->x*b->x)+(a->y*b->y)+(a->z*b->z));
}
#endif

/* make a square height field of quadrilaterals and triangles */
main(int ac, char *av[]) {
    int xres, yres;
    Smooth smooth;
    if (ac < 3) { printf("use: test x y\n"); exit(-1); }; /* abrupt, I know */
    xres = atoi(++av);
    yres = atoi(++av);
    smooth = initAllTables(); /* initialize */
    buildMesh(smooth, xres, yres); /* build the mesh (calls includePolygon) */
    enableEdgePreservation(smooth, 0.0); /* 90 degree folds or more stay crisp */
    makeVertexNormals(smooth); /* build the normals */
    savePolys(smooth); /* save the result in a file */
    freeSmooth(smooth); /* take only normals, leave only footprints */
}

/* z=f(x,y) */
double fofxy(double x, double y) {
    double h;
    h = 2.0 * (0.5 - x); if (h < 0) h = -h; h = h * y;
    return(h);
}

buildMesh(Smooth smooth, int xres, int yres) {
    int x, y;
    Point3 *vlist;
    double dx, dy, lx, ly, hx, hy;
    vlist = NEWA(struct Point3Struct, 4);
    dx = 1.0/((double)(xres));
    dy = 1.0/((double)(yres));
    for (y=0; y<yres; y++) {
        ly = y * dy;
        hy = (y+1) * dy;
        for (x=0; x<xres; x++) {
```



```
    lx = x * dx;
    hx = (x+1) * dx;
    if ((x+y)%2 == 0) addTriangles(lx, ly, hx, hy, vlist, smooth);
        else addQuadrilateral(lx, ly, hx, hy, vlist, smooth);
};
};
free(vlist);
}
```

```
addTriangles(double lx, double ly, double hx, double hy,
             Point3 *vlist, Smooth smooth) {
```

```
Point3 *p = vlist;
    /* make the first triangle */
    p->x = lx;  p->y = ly;  p->z = fofxy(p->x, p->y); p++;
    p->x = hx;  p->y = ly;  p->z = fofxy(p->x, p->y); p++;
    p->x = lx;  p->y = hy;  p->z = fofxy(p->x, p->y); p++;
    includePolygon(3, vlist, smooth, NULL); /* add the polygon */
    /* make the other triangle */
    p = vlist;
    p->x = hx;  p->y = ly;  p->z = fofxy(p->x, p->y); p++;
    p->x = hx;  p->y = hy;  p->z = fofxy(p->x, p->y); p++;
    p->x = lx;  p->y = hy;  p->z = fofxy(p->x, p->y); p++;
    includePolygon(3, vlist, smooth, NULL); /* add the polygon */
}
```

```
addQuadrilateral(double lx, double ly, double hx, double hy,
                 Point3 *vlist, Smooth smooth) {
```

```
Point3 *p = vlist;
    p->x = lx;  p->y = ly;  p->z = fofxy(p->x, p->y); p++;
    p->x = hx;  p->y = ly;  p->z = fofxy(p->x, p->y); p++;
    p->x = hx;  p->y = hy;  p->z = fofxy(p->x, p->y); p++;
    p->x = lx;  p->y = hy;  p->z = fofxy(p->x, p->y); p++;
    includePolygon(4, vlist, smooth, NULL); /* add the polygon */
}
```

```
savePolys(Smooth smooth) {
Polygon poly = smooth->polygonTable;
```

```
int i, k;
Point3 *v, *n;
    printf("NQUAD\n"); /* header for point/normal format */
    while (poly != NULL) {
        for (i=0; i<4; i++) {
            k = i; /* we always write 4 points so double 3rd triangle vertex */
            if (i >= poly->numVerts) k = poly->numVerts-1;
            v = &(poly->vertices[k]);
            n = &(poly->normals[k]);
            printf("%f %f %f %f %f %f\n", v->x, v->y, v->z, n->x, n->y, n->z);
        };
        printf("\n");
        poly = poly->next;
    };
}
```

```
/*
2d and 3d Vector C Library
by Andrew Glassner
from "Graphics Gems", Academic Press, 1990
*/

#include <math.h>
#include "GraphicsGems.h"

/*****
/*    2d Library    */
*****/

/* returns squared length of input vector */
double V2SquaredLength(a)
Vector2 *a;
{
    return((a->x * a->x)+(a->y * a->y));
}

/* returns length of input vector */
double V2Length(a)
Vector2 *a;
{
    return(sqrt(V2SquaredLength(a)));
}

/* negates the input vector and returns it */
Vector2 *V2Negate(v)
Vector2 *v;
{
    v->x = -v->x;  v->y = -v->y;
    return(v);
}

/* normalizes the input vector and returns it */
Vector2 *V2Normalize(v)
Vector2 *v;
{
    double len = V2Length(v);
    if (len != 0.0) { v->x /= len;  v->y /= len; }
    return(v);
}

/* scales the input vector to the new length and returns it */
Vector2 *V2Scale(v, newlen)
Vector2 *v;
double newlen;
{
    double len = V2Length(v);
    if (len != 0.0) { v->x *= newlen/len;  v->y *= newlen/len; }
    return(v);
}

/* return vector sum c = a+b */
Vector2 *V2Add(a, b, c)
Vector2 *a, *b, *c;
{
    c->x = a->x+b->x;  c->y = a->y+b->y;
    return(c);
}
```

```
/* return vector difference c = a-b */
Vector2 *V2Sub(a, b, c)
Vector2 *a, *b, *c;
{
    c->x = a->x-b->x;  c->y = a->y-b->y;
    return(c);
}

/* return the dot product of vectors a and b */
double V2Dot(a, b)
Vector2 *a, *b;
{
    return((a->x*b->x)+(a->y*b->y));
}

/* linearly interpolate between vectors by an amount alpha */
/* and return the resulting vector. */
/* When alpha=0, result=lo.  When alpha=1, result=hi. */
Vector2 *V2Lerp(lo, hi, alpha, result)
Vector2 *lo, *hi, *result;
double alpha;
{
    result->x = LERP(alpha, lo->x, hi->x);
    result->y = LERP(alpha, lo->y, hi->y);
    return(result);
}

/* make a linear combination of two vectors and return the result. */
/* result = (a * ascl) + (b * bscl) */
Vector2 *V2Combine (a, b, result, ascl, bscl)
Vector2 *a, *b, *result;
double ascl, bscl;
{
    result->x = (ascl * a->x) + (bscl * b->x);
    result->y = (ascl * a->y) + (bscl * b->y);
    return(result);
}

/* multiply two vectors together component-wise */
Vector2 *V2Mul (a, b, result)
Vector2 *a, *b, *result;
{
    result->x = a->x * b->x;
    result->y = a->y * b->y;
    return(result);
}

/* return the distance between two points */
double V2DistanceBetween2Points(a, b)
Point2 *a, *b;
{
    double dx = a->x - b->x;
    double dy = a->y - b->y;
    return(sqrt((dx*dx)+(dy*dy)));
}

/* return the vector perpendicular to the input vector a */
Vector2 *V2MakePerpendicular(a, ap)
Vector2 *a, *ap;
```

```
{
    ap->x = -a->y;
    ap->y = a->x;
    return(ap);
}

/* create, initialize, and return a new vector */
Vector2 *V2New(x, y)
double x, y;
{
    Vector2 *v = NEWTYPE(Vector2);
    v->x = x;  v->y = y;
    return(v);
}

/* create, initialize, and return a duplicate vector */
Vector2 *V2Duplicate(a)
Vector2 *a;
{
    Vector2 *v = NEWTYPE(Vector2);
    v->x = a->x;  v->y = a->y;
    return(v);
}

/* multiply a point by a projective matrix and return the transformed point */
Point2 *V2MulPointByProjMatrix(pin, m, pout)
Point2 *pin, *pout;
Matrix3 *m;
{
    double w;
    pout->x = (pin->x * m->element[0][0]) +
        (pin->y * m->element[1][0]) + m->element[2][0];
    pout->y = (pin->x * m->element[0][1]) +
        (pin->y * m->element[1][1]) + m->element[2][1];
    w = (pin->x * m->element[0][2]) +
        (pin->y * m->element[1][2]) + m->element[2][2];
    if (w != 0.0) { pout->x /= w;  pout->y /= w; }
    return(pout);
}

/* multiply together matrices c = ab */
/* note that c must not point to either of the input matrices */
Matrix3 *V2MatMul(a, b, c)
Matrix3 *a, *b, *c;
{
    int i, j, k;
    for (i=0; i<3; i++) {
        for (j=0; j<3; j++) {
            c->element[i][j] = 0;
            for (k=0; k<3; k++) c->element[i][j] +=
                a->element[i][k] * b->element[k][j];
        }
    }
    return(c);
}

/* transpose matrix a, return b */
Matrix3 *TransposeMatrix3(a, b)
Matrix3 *a, *b;
{

```

```
int i, j;
    for (i=0; i<3; i++) {
        for (j=0; j<3; j++)
            b->element[i][j] = a->element[j][i];
    }
    return(b);
}

/*****
/*    3d Library    */
*****/

/* returns squared length of input vector */
double V3SquaredLength(a)
Vector3 *a;
{
    return((a->x * a->x)+(a->y * a->y)+(a->z * a->z));
}

/* returns length of input vector */
double V3Length(a)
Vector3 *a;
{
    return(sqrt(V3SquaredLength(a)));
}

/* negates the input vector and returns it */
Vector3 *V3Negate(v)
Vector3 *v;
{
    v->x = -v->x;  v->y = -v->y;  v->z = -v->z;
    return(v);
}

/* normalizes the input vector and returns it */
Vector3 *V3Normalize(v)
Vector3 *v;
{
    double len = V3Length(v);
    if (len != 0.0) { v->x /= len;  v->y /= len; v->z /= len; }
    return(v);
}

/* scales the input vector to the new length and returns it */
Vector3 *V3Scale(v, newlen)
Vector3 *v;
double newlen;
{
    double len = V3Length(v);
    if (len != 0.0) {
        v->x *= newlen/len;  v->y *= newlen/len;  v->z *= newlen/len;
    }
    return(v);
}

/* return vector sum c = a+b */
Vector3 *V3Add(a, b, c)
```

```
Vector3 *a, *b, *c;
{
    c->x = a->x+b->x;   c->y = a->y+b->y;   c->z = a->z+b->z;
    return(c);
}

/* return vector difference c = a-b */
Vector3 *V3Sub(a, b, c)
Vector3 *a, *b, *c;
{
    c->x = a->x-b->x;   c->y = a->y-b->y;   c->z = a->z-b->z;
    return(c);
}

/* return the dot product of vectors a and b */
double V3Dot(a, b)
Vector3 *a, *b;
{
    return((a->x*b->x)+(a->y*b->y)+(a->z*b->z));
}

/* linearly interpolate between vectors by an amount alpha */
/* and return the resulting vector. */
/* When alpha=0, result=lo.  When alpha=1, result=hi. */
Vector3 *V3Lerp(lo, hi, alpha, result)
Vector3 *lo, *hi, *result;
double alpha;
{
    result->x = LERP(alpha, lo->x, hi->x);
    result->y = LERP(alpha, lo->y, hi->y);
    result->z = LERP(alpha, lo->z, hi->z);
    return(result);
}

/* make a linear combination of two vectors and return the result. */
/* result = (a * ascl) + (b * bscl) */
Vector3 *V3Combine (a, b, result, ascl, bscl)
Vector3 *a, *b, *result;
double ascl, bscl;
{
    result->x = (ascl * a->x) + (bscl * b->x);
    result->y = (ascl * a->y) + (bscl * b->y);
    result->z = (ascl * a->z) + (bscl * b->z);
    return(result);
}

/* multiply two vectors together component-wise and return the result */
Vector3 *V3Mul (a, b, result)
Vector3 *a, *b, *result;
{
    result->x = a->x * b->x;
    result->y = a->y * b->y;
    result->z = a->z * b->z;
    return(result);
}

/* return the distance between two points */
double V3DistanceBetween2Points(a, b)
Point3 *a, *b;
{

```

```
double dx = a->x - b->x;
double dy = a->y - b->y;
double dz = a->z - b->z;
    return(sqrt((dx*dx)+(dy*dy)+(dz*dz)));
}

/* return the cross product c = a cross b */
Vector3 *V3Cross(a, b, c)
Vector3 *a, *b, *c;
{
    c->x = (a->y*b->z) - (a->z*b->y);
    c->y = (a->z*b->x) - (a->x*b->z);
    c->z = (a->x*b->y) - (a->y*b->x);
    return(c);
}

/* create, initialize, and return a new vector */
Vector3 *V3New(x, y, z)
double x, y, z;
{
Vector3 *v = NEWTYPE(Vector3);
    v->x = x;  v->y = y;  v->z = z;
    return(v);
}

/* create, initialize, and return a duplicate vector */
Vector3 *V3Duplicate(a)
Vector3 *a;
{
Vector3 *v = NEWTYPE(Vector3);
    v->x = a->x;  v->y = a->y;  v->z = a->z;
    return(v);
}

/* multiply a point by a matrix and return the transformed point */
Point3 *V3MulPointByMatrix(pin, m, pout)
Point3 *pin, *pout;
Matrix3 *m;
{
    pout->x = (pin->x * m->element[0][0]) + (pin->y * m->element[1][0]) +
              (pin->z * m->element[2][0]);
    pout->y = (pin->x * m->element[0][1]) + (pin->y * m->element[1][1]) +
              (pin->z * m->element[2][1]);
    pout->z = (pin->x * m->element[0][2]) + (pin->y * m->element[1][2]) +
              (pin->z * m->element[2][2]);
    return(pout);
}

/* multiply a point by a projective matrix and return the transformed point */
Point3 *V3MulPointByProjMatrix(pin, m, pout)
Point3 *pin, *pout;
Matrix4 *m;
{
double w;
    pout->x = (pin->x * m->element[0][0]) + (pin->y * m->element[1][0]) +
              (pin->z * m->element[2][0]) + m->element[3][0];
    pout->y = (pin->x * m->element[0][1]) + (pin->y * m->element[1][1]) +
              (pin->z * m->element[2][1]) + m->element[3][1];
    pout->z = (pin->x * m->element[0][2]) + (pin->y * m->element[1][2]) +
              (pin->z * m->element[2][2]) + m->element[3][2];
}
```

```
w = (pin->x * m->element[0][3]) + (pin->y * m->element[1][3]) +
    (pin->z * m->element[2][3]) + m->element[3][3];
if (w != 0.0) { pout->x /= w; pout->y /= w; pout->z /= w; }
return(pout);
}
```

```
/* multiply together matrices c = ab */
/* note that c must not point to either of the input matrices */
Matrix4 *V3MatMul(a, b, c)
Matrix4 *a, *b, *c;
{
    int i, j, k;
    for (i=0; i<4; i++) {
        for (j=0; j<4; j++) {
            c->element[i][j] = 0;
            for (k=0; k<4; k++) c->element[i][j] +=
                a->element[i][k] * b->element[k][j];
        }
    }
    return(c);
}
```

```
/* binary greatest common divisor by Silver and Terzian. See Knuth */
/* both inputs must be >= 0 */
gcd(u, v)
int u, v;
{
    int t, f;
    if ((u<0) || (v<0)) return(1); /* error if u<0 or v<0 */
    f = 1;
    while ((0 == (u%2)) && (0 == (v%2))) {
        u>>=1; v>>=1, f*=2;
    }
    if (u&01) { t = -v; goto B4; } else { t = u; }
    B3: if (t > 0) { t >>= 1; } else { t = -((-t) >> 1); }
    B4: if (0 == (t%2)) goto B3;

    if (t > 0) u = t; else v = -t;
    if (0 != (t = u - v)) goto B3;
    return(u*f);
}
```

```
/******
/* Useful Routines */
/******
```

```
/* return roots of ax^2+bx+c */
/* stable algebra derived from Numerical Recipes by Press et al.*/
int quadraticRoots(a, b, c, roots)
double a, b, c, *roots;
{
    double d, q;
    int count = 0;
    d = (b*b)-(4*a*c);
    if (d < 0.0) { *roots = *(roots+1) = 0.0; return(0); }
    q = -0.5 * (b + (SGN(b)*sqrt(d)));
    if (a != 0.0) { *roots++ = q/a; count++; }
    if (q != 0.0) { *roots++ = c/q; count++; }
    return(count);
}
```



```
/* generic 1d regula-falsi step. f is function to evaluate */
/* interval known to contain root is given in left, right */
/* returns new estimate */
double RegulaFalsi(f, left, right)
double (*f)(), left, right;
{
double d = (*f)(right) - (*f)(left);
    if (d != 0.0) return (right - (*f)(right)*(right-left)/d);
    return((left+right)/2.0);
}

/* generic 1d Newton-Raphson step. f is function, df is derivative */
/* x is current best guess for root location. Returns new estimate */
double NewtonRaphson(f, df, x)
double (*f)(), (*df)(), x;
{
double d = (*df)(x);
    if (d != 0.0) return (x-((*f)(x)/d));
    return(x-1.0);
}

/* hybrid 1d Newton-Raphson/Regula Falsi root finder. */
/* input function f and its derivative df, an interval */
/* left, right known to contain the root, and an error tolerance */
/* Based on Blinn */
double findroot(left, right, tolerance, f, df)
double left, right, tolerance;
double (*f)(), (*df)();
{
double newx = left;
    while (ABS((*f)(newx)) > tolerance) {
        newx = NewtonRaphson(f, df, newx);
        if (newx < left || newx > right)
            newx = RegulaFalsi(f, left, right);
        if ((*f)(newx) * (*f)(left) <= 0.0) right = newx;
        else left = newx;
    }
    return(newx);
}
```

```
/*
 * GraphicsGems.h
 * Version 1.0 - Andrew Glassner
 * from "Graphics Gems", Academic Press, 1990
 */

#ifndef GG_H

#define GG_H 1

/*****
 * 2d geometry types */
*****/

typedef struct Point2Struct {    /* 2d point */
    double x, y;
} Point2;
typedef Point2 Vector2;

typedef struct IntPoint2Struct {    /* 2d integer point */
    int x, y;
} IntPoint2;

typedef struct Matrix3Struct {    /* 3-by-3 matrix */
    double element[3][3];
} Matrix3;

typedef struct Box2dStruct {    /* 2d box */
    Point2 min, max;
} Box2;

/*****
 * 3d geometry types */
*****/

typedef struct Point3Struct {    /* 3d point */
    double x, y, z;
} Point3;
typedef Point3 Vector3;

typedef struct IntPoint3Struct {    /* 3d integer point */
    int x, y, z;
} IntPoint3;

typedef struct Matrix4Struct {    /* 4-by-4 matrix */
    double element[4][4];
} Matrix4;

typedef struct Box3dStruct {    /* 3d box */
    Point3 min, max;
} Box3;

/*****
 * one-argument macros */
*****/

/* absolute value of a */
```

```
#define ABS(a)          (((a)<0) ? -(a) : (a))

/* round a to nearest int */
#define ROUND(a)        floor((a)+0.5)

/* take sign of a, either -1, 0, or 1 */
#define ZSGN(a)          (((a)<0) ? -1 : (a)>0 ? 1 : 0)

/* take binary sign of a, either -1, or 1 if >= 0 */
#define SGN(a)           (((a)<0) ? -1 : 1)

/* shout if something that should be true isn't */
#define ASSERT(x) \
if (!(x)) fprintf(stderr," Assert failed: x\n");

/* square a */
#define SQR(a)           ((a)*(a))

/*****/
/* two-argument macros */
/*****/

/* find minimum of a and b */
#define MIN(a,b)         (((a)<(b))?(a):(b))

/* find maximum of a and b */
#define MAX(a,b)         (((a)>(b))?(a):(b))

/* swap a and b (see Gem by Wyvill) */
#define SWAP(a,b)        { a^=b; b^=a; a^=b; }

/* linear interpolation from l (when a=0) to h (when a=1)*/
/* (equal to (a*h)+((1-a)*l) */
#define LERP(a,l,h)      ((l)+(((h)-(l))*(a)))

/* clamp the input to the specified range */
#define CLAMP(v,l,h)      ((v)<(l) ? (l) : (v) > (h) ? (h) : v)

/*****/
/* memory allocation macros */
/*****/

/* create a new instance of a structure (see Gem by Hultquist) */
#define NEWSTRUCT(x)      (struct x *) (malloc((unsigned)sizeof(struct x)))

/* create a new instance of a type */
#define NEWTYPE(x)        (x *) (malloc((unsigned)sizeof(x)))

/*****/
/* useful constants */
/*****/

#define PI                3.141592          /* the venerable pi */
#define PITIMES2          6.283185          /* 2 * pi */
#define PIOVER2           1.570796          /* pi / 2 */
#define E                 2.718282          /* the venerable e */
#define SQR2              1.414214          /* sqrt(2) */
#define SQR3              1.732051          /* sqrt(3) */
```

```
#define GOLDEN          1.618034          /* the golden ratio */
#define DTOR            0.017453          /* convert degrees to radians */
#define RTOD            57.29578          /* convert radians to degrees */

/*****/
/* booleans */
/*****/

#define TRUE            1
#define FALSE          0
#define ON              1
#define OFF             0
typedef int boolean;    /* boolean data type */
typedef boolean flag;   /* flag data type */

extern double V2SquaredLength(), V2Length();
extern double V2Dot(), V2DistanceBetween2Points();
extern Vector2 *V2Negate(), *V2Normalize(), *V2Scale(), *V2Add(), *V2Sub();
extern Vector2 *V2Lerp(), *V2Combine(), *V2Mul(), *V2MakePerpendicular();
extern Vector2 *V2New(), *V2Duplicate();
extern Point2 *V2MulPointByProjMatrix();
extern Matrix3 *V2MatMul(), *TransposeMatrix3();

extern double V3SquaredLength(), V3Length();
extern double V3Dot(), V3DistanceBetween2Points();
extern Vector3 *V3Normalize(), *V3Scale(), *V3Add(), *V3Sub();
extern Vector3 *V3Lerp(), *V3Combine(), *V3Mul(), *V3Cross();
extern Vector3 *V3New(), *V3Duplicate();
extern Point3 *V3MulPointByMatrix(), *V3MulPointByProjMatrix();
extern Matrix4 *V3MatMul();

extern double RegulaFalsi(), NewtonRaphson(), findroot();

#endif
```

```
/******  
 * movespan()  
 *  
 * Move a span of common values into frame buffer memory. Assume that  
 * the frame buffer is organized as one byte per pixel; consecutive pixels  
 * occupy consecutive bytes. We assume that the longest span has 16 pixels;  
 * longer spans may be handled by coding additional cases. See  
 * movelongspan() for an alternative.  
 *  
 * PARAMETERS:  
 *     here : pointer to the first pixel to be set  
 *     val  : the value (e.g intensity) to be placed at each pixel  
 *     n    : the number of pixels to be illuminated  
 *  
 * AUTHOR: Thom Grace, CS Dept, Illinois Institute of Technology,  
 *         Chicago, IL 60616 (grace@iitmax.iit.edu)  
 *****/
```

```
movespan(unsigned char *here, unsigned char val, int n) {  
    /*This is simple: fall into the proper place in the switch*/  
    switch(n) {  
        case 16: *(here++) = val;  
        case 15: *(here++) = val;  
        case 14: *(here++) = val;  
        case 13: *(here++) = val;  
        case 12: *(here++) = val;  
        case 11: *(here++) = val;  
        case 10: *(here++) = val;  
        case 9:  *(here++) = val;  
        case 8:  *(here++) = val;  
        case 7:  *(here++) = val;  
        case 6:  *(here++) = val;  
        case 5:  *(here++) = val;  
        case 4:  *(here++) = val;  
        case 3:  *(here++) = val;  
        case 2:  *(here++) = val;  
        case 1:  *(here++) = val;  
        case 0: return;  
    }  
} /*end of movespan()*/
```

```
/******  
 * movelongspan()  
 *  
 * This will move an arbitrarily long sequence of constant values into  
 * frame buffer memory. Loops are partially unrolled, requiring about  
 * 1/16 of the overhead of movespan(). Note that the number of switch  
 * cases should be one less than the number of statements in the loop  
 * which should be the same as the constant in the loop condition.  
 *  
 * PARAMETERS: here : pointer to first pixel to be set;  
 *             val  : value to be placed at each pixel;  
 *             n    : number of pixels to be illuminated.  
 *  
 * AUTHOR: Thom Grace, CS Dept, Illinois Institute of Technology,  
 *         Chicago, IL 60616 (grace@iitmax.iit.edu)  
 *****/  
movelongspan(unsigned char *here, unsigned char val, int n) {  
    /*Illuminate 16 pixels at a time, as long as we are able*/  
    while (n >= 16) {
```

```
    *(here++)=val;  *(here++)=val;  *(here++)=val;  *(here++)=val;
    *(here++)=val;  *(here++)=val;  *(here++)=val;  *(here++)=val;
    *(here++)=val;  *(here++)=val;  *(here++)=val;  *(here++)=val;
    *(here++)=val;  *(here++)=val;  *(here++)=val;  *(here++)=val;
    n -= 16;    /*count off those pixels*/
}
/*Now fall into the proper place in the switch*/
switch(n)    {
    case 16: *(here++) = val;
    case 15: *(here++) = val;
    case 14: *(here++) = val;
    case 13: *(here++) = val;
    case 12: *(here++) = val;
    case 11: *(here++) = val;
    case 10: *(here++) = val;
    case 9:  *(here++) = val;
    case 8:  *(here++) = val;
    case 7:  *(here++) = val;
    case 6:  *(here++) = val;
    case 5:  *(here++) = val;
    case 4:  *(here++) = val;
    case 3:  *(here++) = val;
    case 2:  *(here++) = val;
    case 1:  *(here++) = val;
    case 0: return;
}
} /*end of movespan()*/



/*****
* shadespan()
*
* This will move a span of shade values into frame buffer memory.
* The shades are computed by the addition of a shading constant. This
* assumes that the frame buffer is organized as one byte per pixel and
* that consecutive pixels occupy consecutive bytes. This also assumes
* that the longest span is of length 16; see previous code for ideas on
* how to alleviate this limitation.
*
* PARAMETERS:
*     here : pointer to the first pixel to be set
*     val  : the value (e.g intensity) to be placed at the first
*           pixel MINUS the parameter disp (see below)
*     n    : the number of pixels to be illuminated
*     disp : the shading constant
*
* AUTHOR: Thom Grace, CS Dept, Illinois Institute Of Technology,
*         Chicago, IL 60616 (grace@iitmax.iit.edu)
*****/
```

```
shadespan(unsigned char *here, unsigned char val, int n, int disp)
{
    /*Each pixel is filled with the incremented intensity*/
    switch(n)    {
        case 16: *(here++) = (val += disp);
        case 15: *(here++) = (val += disp);
        case 14: *(here++) = (val += disp);
        case 13: *(here++) = (val += disp);
        case 12: *(here++) = (val += disp);
        case 11: *(here++) = (val += disp);
        case 10: *(here++) = (val += disp);
```

```
    case 9: *(here++) = (val += disp);
    case 8: *(here++) = (val += disp);
    case 7: *(here++) = (val += disp);
    case 6: *(here++) = (val += disp);
    case 5: *(here++) = (val += disp);
    case 4: *(here++) = (val += disp);
    case 3: *(here++) = (val += disp);
    case 2: *(here++) = (val += disp);
    case 1: *(here++) = (val += disp);
    case 0: return;
    }
} /*end of movespan()*/
```

# Index of

## /pubs/tog/GraphicsGems/gemsv/ch4-7/

Name	Last modified	Size	Description
 <a href="#">_Parent Directory</a>			
 <a href="#">_bezlen.c</a>	29-Jun-00 08:23	20K	



```
/*
 * BezierLength.c, 1994
 *
 * Jens Gravesen
 * Mathematical Institute
 * Technical University of Denmark
 * email: J.Gravesen@mat.dtu.dk
 *
 * The file contain six functions length<n><x>, (with <n>=1,2,3 and
 * <x>=a, r) which calculate the length of a Bezier curve with a
 * given bound on the absolute error (<x>=a) or the relative error
 * (<x>=r), using the error estimates:
 * 1)  $L_p - L_c$ .
 * 2)  $(L_p - L_c)^2$ .
 * 3)  $(L_a^1 - L_a^0)/15$ .
 *
 * It is assumed that elsewhere there are defined
 * 1) A type: BezierCurve which contains information on a BezierCurve
 *    such as the degree and the coordinates of the control points. Eg.:
 *    typedef struct{
 *        int dim, deg;
 *        double **Q;
 *    } BezierCurve;
 * Functions:
 * 2) double degree(BezierCurve *b);
 *    which returns the degree of *b. It could be a macro:
 *    #define degree(b) (b)->deg
 *
 * 3) BezierCurve *DiffBezierCurve(BezierCurve *b);
 *    which returns a pointer to a BezierCurve containing the
 *    forward differences of *b, or NULL in case of failure.
 *
 * 4) void FreeBezierCurve(BezierCurve *b);
 *    which frees the memory occupied by *b.
 *
 * 5) double length_of_sum(BezierCurve *b);
 *    which returns the length of the sum of the control points of *b.
 *
 * 6) double sum_of_length(BezierCurve *b);
 *    which returns the sum of the length of the control points of *b.
 *
 * 7) BezierCurve *destructive_subdiv(BezierCurve *b);
 *    The function returns a pointer to the forward difference of b
 *    and leaves the forward difference of b in b. In case of failure
 *    the function returns NULL.
 *
 * We assume all this has been declared in a header file say, BezierCode.h
 */
```

```
#include <math.h>
#include "BezierCode.h" /* arbitrary name, see above */
```

```
#define SQRT2      1.4142135623730951
#define ONE_OVER15 0.06666666666666667
#define ABS(x)     (x)<0?-(x):(x)
```

```
/* Forward declarations: */
static double destructive_length_1a(BezierCurve *b, double eps);
static double destructive_length_1r(BezierCurve *b, double eps);
static double destructive_length_2a(BezierCurve *b, double eps);
```

```
static double destructive_length_2r(BezierCurve *b, double eps);
static double destructive_length_3a(BezierCurve *b, double La0, double eps);
static double destructive_length_3r(BezierCurve *b, double La0, double eps);
```

```

/*****
 *   Public functions:
 *****/

/*
 * -----
 * BezierLength1a
 *
 *   Given a BezierCurve produces the arc-length of the curve with
 *   a given bound on the absolut error.
 *   Using Lp-Lc as the error estimate.
 *
 * Results:
 *   The return value is normally the length of the curve, in case
 *   of failure the function returns -HUGE_VAL.
 *
 * Side effects:
 *   None.
 * -----
 */
double BezierLength1a(b, eps)
    BezierCurve *b;          /* The Bezier Curve */
    double eps;              /* The given tolerance */
{
    BezierCurve *db;

    db = DiffBezierCurve(b);
    if (!db) {                /* Something is wrong, */
        return -HUGE_VAL;     /* signal it with negative length */
    }
    return destructive_length_1a(db, eps)/(degree(b)+1);
                                /* destructive_length_1a works with */
                                /* the length multiplied by degree(b)+1 */
}

/*
 * -----
 * BezierLength1r
 *
 *   Given a BezierCurve produces the arc-length of the curve with
 *   a given bound on the relative error.
 *   Using Lp-Lc as the error estimate.
 *
 * Results:
 *   The return value is normally the length of the curve, in case
 *   of failure the function returns -HUGE_VAL.
 *
 * Side effects:
 *   None.
 * -----
 */
```

```
double BezierLength1r(b, eps)
    BezierCurve *b;          /* The Bezier Curve */
    double eps;              /* The given tolerance */
{
    BezierCurve *db;

    db = DiffBezierCurve(b);
    if (!db) {                /* Something is wrong, */
        return -HUGE_VAL;     /* signal it with negative length */
    }
    eps /= (degree(b)+1);     /* destructive_length_1r works with */
                              /* the length multiplied by degree(b)+1 */
    return destructive_length_1r(db, eps)/(degree(b)+1);
}
```

```
/*
 * -----
 * BezierLength2a
 *
 *      Given a BezierCurve produces the arc-length of the curve with
 *      a given bound on the absolut error.
 *      Using  $(L_p - L_c)^2$  as the error estimate.
 *
 * Results:
 *      The return value is normally the length of the curve, in case
 *      of failure the function returns -HUGE_VAL.
 *
 * Side effects:
 *      None.
 * -----
 */
```

```
double BezierLength2a(b, eps)
    BezierCurve *b;          /* The Bezier Curve */
    double eps;              /* The given tolerance */
{
    BezierCurve *db;

    db = DiffBezierCurve(b);
    if (!db) {                /* Something is wrong, */
        return -HUGE_VAL;     /* signal it with negative length */
    }
    eps = sqrt(eps);          /*  $(L_p - L_c)^2 < \text{eps} \Leftrightarrow L_p - L_c < \text{sqrt}(\text{eps})$  */
    return destructive_length_2a(db, eps)/(degree(b)+1);
                              /* destructive_length_1a works with */
                              /* the length multiplied by degree(b)+1 */
}
```

```
/*
 * -----
 * BezierLength2r
 *
 *      Given a BezierCurve produces the arc-length of the curve with
 *      a given bound on the relative error.
 *      Using  $(L_p - L_c)^2$  as the error estimate.
 *
 * Results:
```

```
*      The return value is normally the length of the curve, in case
*      of failure the function returns -HUGE_VAL.
*
```

```
* Side effects:
*      None.
*
```

```
*-----
*/
```

```
double BezierLength2r(b, eps)
    BezierCurve *b;          /* The Bezier Curve */
    double eps;              /* The given tolerance */
{
    BezierCurve *db;

    db = DiffBezierCurve(b);
    if (!db) {                /* Something is wrong, */
        return -HUGE_VAL;     /* signal it with negative length */
    }
    eps /= (degree(b)+1);     /* destructive_length_1r works with */
                              /* the length multiplied by degree(b)+1 */
    return destructive_length_2r(db, eps)/(degree(b)+1);
}
```

```
/*
* -----
* BezierLength3a
*
*      Given a BezierCurve produces the arc-length of the curve with
*      a given bound on the absolut error.
*      Using  $(La^1-La^0)/15$  as the error estimate.
*
* Results:
*      The return value is normally the length of the curve, in case
*      of failure the function returns -HUGE_VAL.
*
* Side effects:
*      None.
*-----
*/
```

```
double BezierLength3a(b, eps)
    BezierCurve *b;          /* The Bezier Curve */
    double eps;              /* The given tolerance */
{
    BezierCurve *db;
    double Lp, Lc, L;

    db = DiffBezierCurve(b);
    if (!db) {                /* Something is wrong, */
        return -HUGE_VAL;     /* signal it with negative length */
    }
    Lp = sum_of_length(db);
    Lc = length_of_sum(db);
    L = 2*Lc + degree(db)*Lp; /*  $La^0*(deg+1)$  of the curve */

    eps *= 30*(degree(b)+1); /*  $(La^1-La)/15 < eps \iff$  */
                              /*  $2*(La^1-La)*(deg+1) < 30*(deg+1)*eps$  */

    return destructive_length_3a(db,L,eps)/(2*degree(b)+2);
}
```

```

    /* destructive_length_3 works with the */
    /* lenght multiplied by degree(b)+1, */
    /* and return works with curves which */
    /* have twice the right size */
}

/*
 * -----
 * BezierLength3r
 *
 * Given a BezierCurve produces the arc-length of the curve with
 * a given bound on the relative error.
 * Using  $(La^1-La^0)/15$  as the error estimate.
 *
 * Results:
 * The return value is normally the length of the curve, in case
 * of failure the function returns -HUGE_VAL.
 *
 * Side effects:
 * None.
 * -----
 */
double BezierLength3r(b, eps)
    BezierCurve *b;          /* The Bezier Curve */
    double eps;              /* The given tolerance */
{
    BezierCurve *db;
    double Lp, Lc, L;

    db = DiffBezierCurve(b);
    if (!db) {                /* Something is wrong, */
        return -HUGE_VAL;     /* signal it with negative length */
    }
    Lp = sum_of_length(db);
    Lc = length_of_sum(db);
    L = 2*Lc + degree(db)*Lp; /*  $La^0*(deg+1)$  of the curve */

    eps *= 15;                /*  $(La^1-La)/15 < eps*La^1 \Leftrightarrow$  */
                             /*  $(La^1-La)*(deg+1) < 15*eps*La^1*(deg+1)$  */

    return destructive_length_3r(db,L,eps)/(2*degree(b)+2);
    /* destructive_length_3 works with the */
    /* length multiplied by degree(b)+1, and */
    /* the size of the curves is twice the */
    /* right size */
}

/*****
 * Private functions:
 *****/

/*
 * -----
 * destructive_length_1a
```

```
*
*      Given the forward differences of a BezierCurve produces the
*      arc-length of the curve multiplied by (n+1), (n = the degree of
*      original curve), with the bound (n+1)*eps on the absolut error.
*      Using Lp-Lc as the error estimate.
*
* Results:
*      The return value is normally the length of the curve, in case
*      of failure the function returns -HUGE_VAL.
*
* Side effects:
*      The memory held by the input curve is freed.
*
*-----
*/
static double destructive_length_1a(b, eps)
    BezierCurve *b;          /* The forward differences of the */
                             /* Bezier Curve */
    double eps;              /* The given tolerance */
{
    BezierCurve *b1;
    double Lp, Lc;

    Lp = sum_of_length(b);
    Lc = length_of_sum(b);
    if ( Lp-Lc < eps ) {
        FreeBezierCurve(b);
        return 2*Lc+degree(b)*Lp;      /* the degree of b is n-1 */
    }
    b1 = destructive_subdiv(b);
    if (!b1) {                      /* Something is wrong, */
        FreeBezierCurve(b);          /* signal it with negative length */
        return -HUGE_VAL;
    }

    /* We don't change eps, instead the two half are twice as big as */
    /* the forward differences of the two halves of the original curve, */
    /* this correspond to putting eps = eps/2, i.e. to distribute the */
    /* error evenly on the two halves */

    return (destructive_length_1a(b1,eps)+destructive_length_1a(b,eps))/2;

    /* We work on the forward differences of the control points so when */
    /* we subdivide the control points should be divide by 2, instead we */
    /* divide the length with 2 */
}

/*
* -----
* destructive_length_1r
*
*      Given the forward differences of a BezierCurve produces the
*      arc-length of the curve multiplied by (n+1), (n = the degree of
*      original curve), with the bound (n+1)*eps on the relative error.
*      Using Lp-Lc as the error estimate.
*
* Results:
*      The return value is normally the length of the curve, in case
*      of failure the function returns -HUGE_VAL.
```

```
*
*   Side effects:
*       The memory held by the input curve is freed.
*
*-----
*/
static double destructive_length_1r(b, eps)
    BezierCurve *b;          /* The forward differences of the */
                             /* Bezier Curve */
    double eps;              /* The tolerance dividede by n+1 */
{
    BezierCurve *b1;
    double Lp, Lc, L;

    Lp = sum_of_length(b);
    Lc = length_of_sum(b);
    L = 2*Lc + degree(b)*Lp; /* the degree of b is n-1 */
    if (Lp-Lc < eps*L) {
        FreeBezierCurve(b);
        return L;
    }
    b1 = destructive_subdiv(b);
    if(!b1) {
        FreeBezierCurve(b); /* Something is wrong, */
        return -HUGE_VAL;   /* signal it with negative length */
    }

    /* Don't change the tolerance */
    return (destructive_length_1r(b1,eps) + destructive_length_1r(b,eps))/2;

    /* We work on the forward differences of the control points so when */
    /* we subdivide the control points should be divide by 2, instead we */
    /* divide the length with 2 */
}

/*
* -----
* destructive_length_2a
*
*   Given the forward differences of a BezierCurve produces the
*   arc-length of the curve multiplied by (n+1), (n = the degree of
*   original curve), with the bound eps^2 on the absolut error.
*   Using (Lp-Lc)^2 as the error estimate.
*
* Results:
*   The return value is normally the length of the curve, in case
*   of failure the function returns -HUGE_VAL.
*
* Side effects:
*   The memory held by the input curve is freed.
*
*-----
*/
static double destructive_length_2a(b, eps)
    BezierCurve *b;          /* The forward differences of the */
                             /* Bezier Curve */
    double eps;              /* The sqaure root of the tolerance */
{
    BezierCurve *b1;
    double Lp, Lc;
```

```

Lp = sum_of_length(b);
Lc = length_of_sum(b);
if ( Lp-Lc < eps ) {
    /* (Lp-Lc)^2 < eps^2 <=> Lp-Lc < eps */
    FreeBezierCurve(b);
    return 2*Lc + degree(b)*Lp; /* the degree of b is n-1 */
}
b1=destructive_subdiv(b);
if(!b1) {
    /* Something is wrong, */
    FreeBezierCurve(b);
    return -HUGE_VAL; /* signal it with negative length */
}
eps *= SQRT2;
/* We distribute the error evenly on the two halves, but the two */
/* halves are twice as big as the forward differences of the two */
/* halves of the original curve, so we have */
/* (Lp/2-Lc/2)^2 < eps^2/2 <=> Lp-Lc < sqrt(2)*eps */

return (destructive_length_2a(b1,eps)+destructive_length_2a(b,eps))/2;

/* We work on the forward differences of the control points so when */
/* we subdivide the control points should be divide by 2, instead we */
/* divide the length with 2 */
}

/*
 * -----
 * destructive_length_2r
 *
 * Given the forward differences of a BezierCurve produces the
 * arc-length of the curve multiplied by (n+1), (n = the degree of
 * original curve), with the bound eps*(n+1) on the relative error.
 * Using (Lp-Lc)^2 as the error estimate.
 *
 * Results:
 * The return value is normally the length of the curve, in case
 * of failure the function returns -HUGE_VAL.
 *
 * Side effects:
 * The memory held by the input curve is freed.
 * -----
 */
static double destructive_length_2r(b, eps)
    BezierCurve *b; /* The forward differences of the */
                   /* Bezier Curve */
    double eps; /* The tolerance dividede by n+1 */
{
    BezierCurve *b1;
    double Lp, Lc, L, err;

    Lp = sum_of_length(b);
    Lc = length_of_sum(b);
    L = 2*Lc + degree(b)*Lp; /* the degree of b is n-1 */
    err = Lp-Lc;
    if ( err*err < eps*L ) {
        FreeBezierCurve(b);
        return L;
    }
}

```



```
b1 = destructive_subdiv(b);
if(!b1) {
    FreeBezierCurve(b);
    return -HUGE_VAL;
}
eps *=2;

/* We want the same tolerance on the two */
/* halves as we had before, but the size of */
/* the control polygon is two times to */
/* large. So we have */
/*  $(L_p/2 - L_c/2)^2 < \text{eps} * L/2 \Leftrightarrow$  */
/*  $(L_p - L_c)^2 < 2 * \text{eps}$  */

return (destructive_length_2r(b1,eps) + destructive_length_2r(b,eps))/2;

/* We work on the forward differences of the control points so when */
/* we subdivide the control points should be divide by 2, instead we */
/* divide the length with 2 */
}

/*
 * -----
 * destructive_length_3a
 *
 * Given the forward differences of a BezierCurve produces the
 * arc-length of the curve multiplied by  $2 * (n+1)$ , ( $n$  = the degree of
 * original curve), with the bound  $\text{eps} / (30 * (n+1))$  on the absolut error.
 * Using  $(L_a^1 - L_a^0) / 15$  as the error estimate.
 *
 * Results:
 * The return value is normally the length of the curve, in case
 * of failure the function returns -HUGE_VAL.
 *
 * Side effects:
 * The memory held by the input curve is freed.
 * -----
 */
static double destructive_length_3a(b, La0, eps)
    BezierCurve *b;
    double La0;
    double eps;
{
    BezierCurve *b1;
    double Lp, Lc, La1, La2, L, err;

    b1 = destructive_subdiv(b);
    if(!b1) {
        FreeBezierCurve(b);
        return -HUGE_VAL;
    }

    Lp = sum_of_length(b1);
    Lc = length_of_sum(b1);
    La1 = 2*Lc + degree(b1)*Lp;
    /*  $2 * \text{Length} * (\text{deg} + 1)$  of first half */

    Lp = sum_of_length(b);
    Lc = length_of_sum(b);
```

```
La2 = 2*Lc + degree(b)*Lp;          /* 2*Length*(deg+1) of second half */

L = La1+La2;                          /* La^1*(deg+1) */

err = L-2*La0;                        /* L = 2*La^1 */
if ( err < eps && err > -eps) {
    FreeBezierCurve(b1);
    FreeBezierCurve(b);
    return L + err*ONE_OVER15;        /* Do the error correction */
}

/* We don't change eps, instead the two half are twice as big as */
/* the forward differences of the two halves of the original curve, */
/* this correspond to putting eps = eps/2, i.e. to distribute the */
/* error evenly on the two halves */

return (destructive_length_3a(b1,La1,eps) +
        destructive_length_3a(b,La2,eps))/2 ;
/* We work on the forward differences of the control points so when */
/* we subdivide the control points should be divide by 2, instead we */
/* divide the length with 2 */
}

/*
 * -----
 * destructive_length_3r
 *
 * Given the forward differences of a BezierCurve produces the
 * arc-length of the curve multiplied by (n+1), (n = the degree of
 * original curve), with the bound eps/15 on the relative error.
 * Using (La^1-La^0)/15 as the error estimate.
 *
 * Results:
 * The return value is normally the length of the curve, in case
 * of failure the function returns -HUGE_VAL.
 *
 * Side effects:
 * The memory held by the input curve is freed.
 * -----
 */
static double destructive_length_3r(b, La0, eps)
    BezierCurve *b;          /* The forward differences of the */
                             /* Bezier Curve */
    double La0;              /* The "average length" of the */
                             /* Bezier curve */
    double eps;              /* The tolerance multiplied by 15 */
{
    BezierCurve *b1;
    double Lp, Lc, La1, La2, L, err;

    b1 = destructive_subdiv(b);
    if(!b1) {                 /* Something is wrong, */
        FreeBezierCurve(b);   /* signal it with negative length */
        return -HUGE_VAL;
    }

    Lp = sum_of_length(b1);
    Lc = length_of_sum(b1);
```

```
La1 = 2*Lc + degree(b1)*Lp;          /* 2*Length*(deg+1) of first half */

Lp = sum_of_length(b);
Lc = length_of_sum(b);
La2 = 2*Lc + degree(b)*Lp;          /* 2*Length*(deg+1) of second half */

L = La1+La2;                        /* 2*La^1*(deg+1) */

err = L-2*La0;
if ( err < eps*L && err > -eps*L ) {
    FreeBezierCurve(b1);
    FreeBezierCurve(b);
    return L + err*ONE_OVER15;      /* Do the error correction */
}









/* Don't change the tolerance */

return (destructive_length_3r(b1,La1,eps) +
        destructive_length_3r(b,La2,eps))/2;
/* We work on the forward differences of the control points so when */
/* we subdivide the control points should be divide by 2, instead we */
/* divide the length with 2 */
}

/* end of file BezierLength.c, 1994 */
```

# Index of

## /pubs/tog/GraphicsGems/gemsv/ch7-2/

Name	Last modified	Size	Description
 <a href="#">_Parent Directory</a>			
 <a href="#">_fpcube.c</a>	29-Jun-00 08:24	6K	
 <a href="#">_makefile</a>	29-Jun-00 08:24	1K	
 <a href="#">_pcube.c</a>	29-Jun-00 08:24	8K	
 <a href="#">_pcube.h</a>	29-Jun-00 08:24	6K	
 <a href="#">_readme</a>	29-Jun-00 08:24	9K	
 <a href="#">_test.c</a>	29-Jun-00 08:24	2K	
 <a href="#">_vec.h</a>	29-Jun-00 08:24	30K	

```
/*
 *          FAST POLYGON-CUBE INTERSECTION TESTS
 *          by Daniel Green
 *          January 1994
 *
 * The original inspiration for this routine comes from the triangle-cube
 * intersection algorithm in Graphics Gems III by Douglas Voorhies.
 * Aside from the fact that the original code was special-cased for triangles
 * only, it suffered from some bugs. Don Hatch re-wrote the non-trivial tests
 * using the same basic ideas, fixed the bugs and also made it more
 * general and efficient. Don's implementation performs just the
 * intersection calculations without any trivial accept or reject tests,
 * and is embodied in the routine polygon_intersects_cube().
 *
 * The function implemented here is simply a wrapper that begins with a
 * slightly more efficient version of Voorhies' original trivial reject tests
 * and only calls polygon_intersects_cube() when those tests fail.
 * The result is a fast and robust polygon-cube tester.
 *
 * Also included here is trivial_vertex_tests() which is used by
 * fast_polygon_intersects_cube(). It can be used to quickly test an entire
 * set of vertices for trivial reject or accept. This is useful for testing
 * polyhedra or polygon meshes.
 *
 * WARNING: When used to test intersection of a polyhedron with the unit cube,
 * remember that these routines only test *surfaces* and not volumes.
 * If polygon_intersects_cube() reports no intersection with any of the faces
 * of the polyhedron, the caller should be aware that the polyhedron
 * could still contain the whole unit cube which would then need to be checked
 * with a point-within-polyhedron test. The origin would be a natural point
 * to check in such a test.
 */
```

```
#include "pcube.h"
```

```
#ifndef __cplusplus
#define inline
#endif
```

```
#define TEST_AGAINST_PARALLEL_PLANES(posbit, negbit, value, limit) \
    if (mask & (posbit|negbit)) { \
        register real temp = value; \
        if ((mask & posbit) && temp > limit) \
            outcode |= posbit; \
        else if ((mask & negbit) && temp < -limit) \
            outcode |= negbit; \
    }
```

```
/*
 * Tells which of the six face-planes the given point is outside of.
 * Only tests faces not represented in "mask".
 */
```

```
static inline unsigned long
face_plane(const real p[3], unsigned long mask)
{
```

```
    register unsigned long outcode = 0L;

    TEST_AGAINST_PARALLEL_PLANES(0x001, 0x002, p[0], 0.5)
    TEST_AGAINST_PARALLEL_PLANES(0x004, 0x008, p[1], 0.5)
    TEST_AGAINST_PARALLEL_PLANES(0x010, 0x020, p[2], 0.5)

    return(outcode);
}
```

```
/*
 * Tells which of the twelve edge planes the given point is outside of.
 * Only tests faces not represented in "mask".
 */
```

```
static inline unsigned long
bevel_2d(const real p[3], unsigned long mask)
{
    register unsigned long outcode = 0L;

    TEST_AGAINST_PARALLEL_PLANES(0x001, 0x002, p[0] + p[1], 1.0)
    TEST_AGAINST_PARALLEL_PLANES(0x004, 0x008, p[0] - p[1], 1.0)
    TEST_AGAINST_PARALLEL_PLANES(0x010, 0x020, p[0] + p[2], 1.0)
    TEST_AGAINST_PARALLEL_PLANES(0x040, 0x080, p[0] - p[2], 1.0)
    TEST_AGAINST_PARALLEL_PLANES(0x100, 0x200, p[1] + p[2], 1.0)
    TEST_AGAINST_PARALLEL_PLANES(0x400, 0x800, p[1] - p[2], 1.0)

    return(outcode);
}
```

```
/*
 * Tells which of the eight corner planes the given point is outside of.
 * Only tests faces not represented in "mask".
 */
```

```
static inline unsigned long
bevel_3d(const real p[3], unsigned long mask)
{
    register unsigned long outcode = 0L;

    TEST_AGAINST_PARALLEL_PLANES(0x001, 0x002, p[0] + p[1] + p[2], 1.5)
    TEST_AGAINST_PARALLEL_PLANES(0x004, 0x008, p[0] + p[1] - p[2], 1.5)
    TEST_AGAINST_PARALLEL_PLANES(0x010, 0x020, p[0] - p[1] + p[2], 1.5)
    TEST_AGAINST_PARALLEL_PLANES(0x040, 0x080, p[0] - p[1] - p[2], 1.5)

    return(outcode);
}
```

```
/*
 * Returns 1 if any of the vertices are inside the cube of edge length 1
 * centered at the origin (trivial accept), 0 if all vertices are outside
 * of any testing plane (trivial reject), -1 otherwise (couldn't help).
 */
```

```
extern int
trivial_vertex_tests(int nverts, const real verts[][3],
```

```

        int already_know_verts_are_outside_cube)
{
    register unsigned long cum_and; /* cumulative logical ANDs */
    register int i;

    /*
     * Compare the vertices with all six face-planes.
     * If it's known that no vertices are inside the unit cube
     * we can exit the loop early if we run out of bounding
     * planes that all vertices might be outside of. That simply means
     * that this test failed and we can go on to the next one.
     * If we must test for vertices within the cube, the loop is slightly
     * different in that we can trivially accept if we ever do find a
     * vertex within the cube, but we can't break the loop early if we run
     * out of planes to reject against because subsequent vertices might
     * still be within the cube.
     */
    cum_and = ~0L; /* Set to all "1" bits */
    if(already_know_verts_are_outside_cube) {
        for(i=0; i<nverts; i++)
            if(0L == (cum_and = face_plane(verts[i], cum_and)))
                break; /* No planes left to trivially reject */
    }
    else {
        for(i=0; i<nverts; i++) {
            /* Note the ~0L mask below to always test all planes */
            unsigned long face_bits = face_plane(verts[i], ~0L);
            if(0L == face_bits) /* vertex is inside the cube */
                return 1; /* trivial accept */
            cum_and &= face_bits;
        }
    }
    if(cum_and != 0L) /* All vertices outside some face plane. */
        return 0; /* Trivial reject */

    /*
     * Now do the just the trivial reject test against the 12 edge planes.
     */
    cum_and = ~0L; /* Set to all "1" bits */
    for(i=0; i<nverts; i++)
        if(0L == (cum_and = bevel_2d(verts[i], cum_and)))
            break; /* No planes left that might trivially reject */
    if(cum_and != 0L) /* All vertices outside some edge plane. */
        return 0; /* Trivial reject */

    /*
     * Now do the trivial reject test against the 8 corner planes.
     */
    cum_and = ~0L; /* Set to all "1" bits */
    for(i=0; i<nverts; i++)
        if(0L == (cum_and = bevel_3d(verts[i], cum_and)))
            break; /* No planes left that might trivially reject */
    if(cum_and != 0L) /* All vertices outside some corner plane. */
        return 0; /* Trivial reject */

    /*
     * By now we know that the polygon is not to the outside of any of the
     * test planes and can't be trivially accepted *or* rejected.
     */
    return -1;
}

```

```
/*
 * This is a version of the same polygon-cube intersection that first calls
 * trivial_vertex_tests() to hopefully skip the more expensive definitive test.
 * It simply calls polygon_intersects_cube() when that fails.
 * Note that after the trivial tests we at least know that all vertices are
 * outside the cube and can therefore pass a true flag to
 * polygon_intersects_cube().
 */
extern int
fast_polygon_intersects_cube(int nverts, const real verts[][3],
                             const real polynormal[3],
                             int already_know_verts_are_outside_cube,
                             int already_know_edges_are_outside_cube)
{
    int quick_test = trivial_vertex_tests(nverts, verts,
                                           already_know_verts_are_outside_cube);
    if(-1 == quick_test)
        return polygon_intersects_cube(nverts, verts, polynormal, 1,
                                        ready_know_edges_are_outside_cube);
    else
        return quick_test;
}
```



```
CC=CC    # can be C++ or C compiler

OBJS= \
    pcube.o \
    fpcube.o \
    test.o

.c.o:
    $(CC) -O -c $< -I.

all: test

test: $(OBJS)
    $(CC) -o test $(OBJS) -lm

clean:
    rm -f $(OBJS) test
```

```
/*
 *
 *      polygon_intersects_cube()
 *      by Don Hatch
 *      January 1994
 *
 *      Algorithm:
 *      1. If any edge intersects the cube, return true.
 *      Testing whether a line segment intersects the cube
 *      is equivalent to testing whether the origin is contained
 *      in the rhombic dodecahedron obtained by dragging
 *      a unit cube from (being centered at) one segment endpoint
 *      to the other.
 *      2. If the polygon interior intersects the cube, return true.
 *      Since we know no vertex or edge intersects the cube,
 *      this amounts to testing whether any of the four cube diagonals
 *      intersects the interior of the polygon. (Same as voorhies's test).
 *      3. Return false.
 */
```

```
#include "pcube.h"
#include "vec.h"
```

```
#define FOR(i,n) for ((i) = 0; (i) < (n); ++(i))
#define MAXDIM2(v) ((v)[0] > (v)[1] ? 0 : 1)
#define MAXDIM3(v) ((v)[0] > (v)[2] ? MAXDIM2(v) : MAXDIM2((v)+1)+1)
#define ABS(x) ((x)<0 ? -(x) : (x))
#define SQR(x) ((x)*(x))
#define SIGN_NONZERO(x) ((x) < 0 ? -1 : 1)
/* note a and b can be in the reverse order and it still works! */
#define IN_CLOSED_INTERVAL(a,x,b) (((x)-(a)) * ((x)-(b)) <= 0)
#define IN_OPEN_INTERVAL(a,x,b) (((x)-(a)) * ((x)-(b)) < 0)
```

```
#define seg_contains_point(a,b,x) (((b)>(x)) - ((a)>(x)))
/*
 * Tells whether a given polygon with nonzero area
 * contains a point which is assumed to lie in the plane of the polygon.
 * Actually returns the multiplicity of containment.
 * This will always be 1 or 0 for non-self-intersecting planar
 * polygons with the normal in the standard direction
 * (towards the eye when looking at the polygon so that it's CCW).
 */
```

```
extern int
polygon_contains_point_3d(int nverts, const real verts[/* nverts */][3],
                        const real polynormal[3],
                        real point[3])
{
    real abspolynormal[3];
    int zaxis, xaxis, yaxis, i, count;
    int xdirection;
    const real *v, *w;

    /*
     * Determine which axis to ignore
     * (the one in which the polygon normal is largest)
     */
    FOR(i,3)
        abspolynormal[i] = ABS(polynormal[i]);
```

```
zaxis = MAXDIM3(abspolynormal);

if (polynormal[zaxis] < 0) {
    xaxis = (zaxis+2)%3;
    yaxis = (zaxis+1)%3;
} else {
    xaxis = (zaxis+1)%3;
    yaxis = (zaxis+2)%3;
}

count = 0;
FOR(i,nverts) {
    v = verts[i];
    w = verts[(i+1)%nverts];
    if (xdirection = seg_contains_point(v[xaxis], w[xaxis], point[xaxis])) {
        if (seg_contains_point(v[yaxis], w[yaxis], point[yaxis])) {
            if (xdirection * (point[xaxis]-v[xaxis])*(w[yaxis]-v[yaxis]) <=
                xdirection * (point[yaxis]-v[yaxis])*(w[xaxis]-v[xaxis]))
                count += xdirection;
        } else {
            if (v[yaxis] <= point[yaxis])
                count += xdirection;
        }
    }
}
return count;
}
```

```
/*
 * A segment intersects the unit cube centered at the origin
 * iff the origin is contained in the solid obtained
 * by dragging a unit cube from one segment endpoint to the other.
 * (This solid is a warped rhombic dodecahedron.)
 * This amounts to 12 sidedness tests.
 * Also, this test works even if one or both of the segment endpoints is
 * inside the cube.
 */
extern int
segment_intersects_cube(const real v0[3], const real v1[3])
{
    int i, iplus1, iplus2, edgevec_signs[3];
    real edgevec[3];

    VMV3(edgevec, v1, v0);

    FOR(i,3)
        edgevec_signs[i] = SIGN_NONZERO(edgevec[i]);

    /*
     * Test the three cube faces on the v1-ward side of the cube--
     * if v0 is outside any of their planes then there is no intersection.
     * Also test the three cube faces on the v0-ward side of the cube--
     * if v1 is outside any of their planes then there is no intersection.
     */

    FOR(i,3) {
        if (v0[i] * edgevec_signs[i] > .5) return 0;
        if (v1[i] * edgevec_signs[i] < -.5) return 0;
    }
}
```

```
}

/*
 * Okay, that's the six easy faces of the rhombic dodecahedron
 * out of the way. Six more to go.
 * The remaining six planes bound an infinite hexagonal prism
 * joining the petrie polygons (skew hexagons) of the two cubes
 * centered at the endpoints.
 */

FOR(i,3) {
    real rhomb_normal_dot_v0, rhomb_normal_dot_cubedge;

    iplus1 = (i+1)%3;
    iplus2 = (i+2)%3;

#ifdef THE_EASY_TO_UNDERSTAND_WAY

    {
        real rhomb_normal[3], cubedge_midpoint[3];

        /*
         * rhomb_normal = VXV3(edgevec, unit vector in direction i),
         * being cavalier about which direction it's facing
         */
        rhomb_normal[i] = 0;
        rhomb_normal[iplus1] = edgevec[iplus2];
        rhomb_normal[iplus2] = -edgevec[iplus1];

        /*
         * We now are describing a plane parallel to
         * both segment and the cube edge in question.
         * if |DOT3(rhomb_normal, an arbitrary point on the segment)| >
         * |DOT3(rhomb_normal, an arbitrary point on the cube edge in question)|
         * then the origin is outside this pair of opposite faces.
         * (This is equivalent to saying that the line
         * containing the segment is "outside" (i.e. further away from the
         * origin than) the line containing the cube edge.
         */

        cubedge_midpoint[i] = 0;
        cubedge_midpoint[iplus1] = edgevec_signs[iplus1]*.5;
        cubedge_midpoint[iplus2] = -edgevec_signs[iplus2]*.5;

        rhomb_normal_dot_v0 = DOT3(rhomb_normal, v0);
        rhomb_normal_dot_cubedge = DOT3(rhomb_normal, cubedge_midpoint);
    }

#else /* the efficient way */

    rhomb_normal_dot_v0 = edgevec[iplus2] * v0[iplus1]
        - edgevec[iplus1] * v0[iplus2];

    rhomb_normal_dot_cubedge = .5 *
        (edgevec[iplus2] * edgevec_signs[iplus1] +
         edgevec[iplus1] * edgevec_signs[iplus2]);

#endif /* the efficient way */

    if (SQR(rhomb_normal_dot_v0) > SQR(rhomb_normal_dot_cubedge))
        return 0; /* origin is outside this pair of opposite planes */
}
```

```
    }
    return 1;
}

/*
 * Tells whether a given polygon intersects the cube of edge length 1
 * centered at the origin.
 * Always returns 1 if a polygon edge intersects the cube;
 * returns the multiplicity of containment otherwise.
 * (See explanation of polygon_contains_point_3d() above).
 */
extern int
polygon_intersects_cube(int nverts, const real verts[/* nverts */][3],
                        const real polynormal[3],
                        int already_know_vertices_are_outside_cube, /*unused*/
                        int already_know_edges_are_outside_cube)
{
    int i, best_diagonal[3];
    real p[3], t;

    /*
     * If any edge intersects the cube, return 1.
     */
    if (!already_know_edges_are_outside_cube)
        FOR(i,nverts)
            if (segment_intersects_cube(verts[i], verts[(i+1)%nverts]))
                return 1;

    /*
     * If the polygon normal is zero and none of its edges intersect the
     * cube, then it doesn't intersect the cube
     */
    if (ISZEROVEC3(polynormal))
        return 0;

    /*
     * Now that we know that none of the polygon's edges intersects the cube,
     * deciding whether the polygon intersects the cube amounts
     * to testing whether any of the four cube diagonals intersects
     * the interior of the polygon.
     *
     * Notice that we only need to consider the cube diagonal that comes
     * closest to being perpendicular to the plane of the polygon.
     * If the polygon intersects any of the cube diagonals,
     * it will intersect that one.
     */

    FOR(i,3)
        best_diagonal[i] = SIGN_NONZERO(polynormal[i]);

    /*
     * Okay, we have the diagonal of interest.
     * The plane containing the polygon is the set of all points p satisfying
     *     DOT3(polynormal, p) == DOT3(polynormal, verts[0])
     * So find the point p on the cube diagonal of interest
     * that satisfies this equation.
     * The line containing the cube diagonal is described parametrically by

```

```
*      t * best_diagonal
* so plug this into the previous equation and solve for t.
*      DOT3(polynormal, t * best_diagonal) == DOT3(polynormal, verts[0])
* i.e.
*      t = DOT3(polynormal, verts[0]) / DOT3(polynormal, best_diagonal)
*
* (Note that the denominator is guaranteed to be nonzero, since
* polynormal is nonzero and best_diagonal was chosen to have the largest
* magnitude dot-product with polynormal)
*/
t = DOT3(polynormal, verts[0])
  / DOT3(polynormal, best_diagonal);

if (!IN_CLOSED_INTERVAL(-.5, t, .5))
    return 0; /* intersection point is not in cube */

SXV3(p, t, best_diagonal); /* p = t * best_diagonal */

return polygon_contains_point_3d(nverts, verts, polynormal, p);
}
```

```
#ifndef __PCUBE__
#define __PCUBE__
```

```

* * * * *
*
*               POLYGON-CUBE INTERSECTION
*             by Don Hatch & Daniel Green
*           January 1994
*
*   CONTENTS:
*       polygon_intersects_cube()
*       fast_polygon_intersects_cube()
*       trivial_vertex_tests()
*       segment_intersects_cube()
*       polygon_contains_point_3d()
*
*   This module contains routines that test points, segments and polygons
*   for intersections with the unit cube defined as the axially aligned
*   cube of edge length 1 centered at the origin. Polygons may be convex,
*   concave or self-intersecting. Also contained is a routine that tests
*   whether a point is within a polygon. All routines are intended to be
*   fast and robust. Note that the cube and polygons are defined to include
*   their boundaries.
*
*   The fast_polygon_intersects_cube routine is meant to replace the
*   triangle-cube intersection routine in Graphics Gems III by Douglas
*   Voorhies. While that original algorithm is still sound, it is
*   specialized for triangles and the implementation contained several
*   bugs and inefficiencies. The trivial_vertex_tests routine defined here
*   is almost an exact copy of the trivial point-plane tests from the
*   beginning of Voorhies' algorithm but broken out into a separate routine
*   which is called by fast_polygon_intersects_cube. The segment-cube and
*   polygon-cube intersection algorithms have been completely rewritten.
*
*   Notice that trivial_vertex_tests can be used to quickly test an entire
*   set of vertices for trivial reject or accept. This can be useful for
*   testing polyhedra or entire polygon meshes. When used to test
*   polyhedra, remember that these routines only test points, edges and
*   surfaces, not volumes. If no such intersection is reported, the caller
*   should be aware that the volume of the polyhedra could still contain
*   the entire unit box which would then need to be checked for with an
*   additional point-within-polyhedron test. The origin would be a natural
*   point to check in such a test.
*
* * * * *

```

```
#ifndef real
#define real double
#endif
```

```

/*
 *
 *          POLYGON INTERSECTS CUBE
 *
 * Tells how the given polygon intersects the cube of edge length 1 centered
 * at the origin.
 * If any vertex or edge of the polygon intersects the cube,
 * a value of 1 will be returned.
 * Otherwise the value returned will be the multiplicity of containment
 * of the cross-section of the cube in the polygon; this may

```

```
* be interpreted as a boolean value in any of the standard
* ways; e.g. the even-odd rule (it's inside the polygon iff the
* result is odd) or the winding rule (it's inside the polygon iff
* the result is nonzero).
*
* The "polynormal" argument is a vector perpendicular to the polygon. It
* need not be of unit length. It is suggested that Newell's method be used
* to calculate polygon normals (See Graphics Gems III). Zero-lengthed normals
* are quite acceptable for degenerate polygons but are not acceptable
* otherwise. In particular, beware of zero-length normals which Newell's
* method can return for certain self-intersecting polygons (for example
* a bow-tie quadrilateral).
*
* The already_know_verts_are_outside_cube flag is unused by this routine
* but may be useful for alternate implementations.
*
* The already_know_edges_are_outside_cube flag is useful when testing polygon
* meshes with shared edges in order to not test the same edge more than once.
*
* Note: usually users of this module would not want to call this routine
* directly unless they have previously tested the vertices with the trivial
* vertex test below. Normally one would call the fast_polygon_intersects_cube
* utility instead which combines both of these tests.
*/
```

```
extern int
polygon_intersects_cube(int nverts, const real verts[/* nverts */][3],
                        const real polynormal[3],
                        int already_know_verts_are_outside_cube,
                        int already_know_edges_are_outside_cube);
```

```
/*
*
* FAST POLYGON INTERSECTS CUBE
*
* This is a version of the same polygon-cube intersection that first calls
* trivial_vertex_tests() to hopefully skip the more expensive definitive test.
* It simply calls polygon_intersects_cube() when that fails.
* Note that unlike polygon_intersects_cube(), this routine does use the
* already_know_verts_are_outside_cube argument.
*/
```

```
extern int
fast_polygon_intersects_cube(int nverts, const real verts[/* nverts */][3],
                             const real polynormal[3],
                             int already_know_verts_are_outside_cube,
                             int already_know_edges_are_outside_cube);
```

```
/*
*
* TRIVIAL VERTEX TESTS
*
* Returns 1 if any of the vertices are inside the cube of edge length 1
* centered at the origin (trivial accept), 0 if all vertices are outside
* of any testing plane (trivial reject), -1 otherwise (couldn't help).
*/
```

```
extern int
trivial_vertex_tests(int nverts, const real verts[/* nverts */][3],
                     int already_know_verts_are_outside_cube);
```

```
/*
*
* SEGMENT INTERSECTS CUBE
```



```
*
* Returns 1 if the given line segment intersects the cube of edge length 1
* centered at the origin, 0 otherwise.
*/
extern int
segment_intersects_cube(const real v0[3], const real v1[3]);

/*
* POLYGON CONTAINS POINT 3D
*
* Tells whether a given polygon with nonzero area contains a point which is
* assumed to lie in the plane of the polygon.
* Actually returns the multiplicity of containment. This will always be 1
* or 0 for non-self-intersecting planar polygons with the normal in the
* standard direction (towards the eye when looking at the polygon so that
* it's CCW).
*/
extern int
polygon_contains_point_3d(int nverts, const real verts[/* nverts */][3],
                          const real polynormal[3],
                          real point[3]);

#endif
```

EFFICIENT INTERSECTION TESTING OF  
GENERAL POLYGONS AND POLYHEDRA  
AGAINST AN AXIALLY-ALIGNED CUBE

Authors:

Don Hatch (hatch@sgi.com)

Daniel Green (danielg@autodesk.com)

January 1994

Published routines:

```
polygon_intersects_cube  
fast_polygon_intersects_cube  
trivial_vertex_tests  
segment_intersects_cube  
polygon_contains_point_3d
```

## BACKGROUND

In Graphics Gems III, Douglas Voorhies gives an algorithm that tests whether a given triangle intersects the axially-aligned cube of edge length 1 centered at the origin. The algorithm presented here extends that work in several ways. The most important difference is that it is generalized to handle arbitrary polygons which may be convex, concave or self-intersecting. The implementation is also more efficient in several places and fixes bugs in the original implementation which can generate both false positive and false negative results. The new efficiency and robustness come mostly from completely rewritten core routines.

In Graphics Gems IV, Ned Greene describes an efficient algorithm for testing convex polyhedra against axially aligned boxes. That algorithm works by attempting to find a plane separating the two figures. Greene mentions that the intuitive approach is inefficient because of the number of possible intersection calculations. (The intuitive approach contains an intersection test of each polygon edge with each cube face, followed by intersecting each cube diagonal with the polygon body). Our approach is something of a hybrid. It contains only a single intersection calculation which is rarely performed because of the trivial tests that precede it. The rest of the calculations are of the same sort of fast inequality tests that Greene uses.

## DESCRIPTION

Voorhies's original approach is elegant and sound, and we've kept the general approach which proceeds from cheap trivial accept and reject tests through more expensive edge and face intersection tests. We've also broken these individual tests out into separate routines in order to allow higher level routines to be built on top of them - such as general polyhedra and polygon mesh tests - without having to suffer redundant tests on shared vertices or edges.

The composite fast\_polygon\_intersects\_cube routine presented here is meant to replace Voorhies' triangle-cube intersection routine. It begins by calling the trivial\_vertex\_tests routine which is almost an exact copy of the trivial point-plane tests from the beginning of Voorhies' implementation and only calls the definitive polygon\_intersects\_cube routine when that fails.

The main algorithmic difference in our point-plane tests is that in a number of places we avoid testing against planes which cannot possibly give useful information. For example, when a point is found to be to the left of the left face plane of the cube, we don't need to also test whether it might be to the right of the right face plane.

The `trivial_vertex_tests` routine can be used to quickly test an entire set of vertices for trivial reject or accept. This can be useful for testing polyhedra or entire polygon meshes. Useful applications for polyhedra testing include more than just the faster rendering of polyhedra; another important use is in testing for trivial rejection of polyhedral bounding volumes (described more fully in the last section).

Note that when `trivial_vertex_tests` is used to simultaneously test a large set of vertices against a set of bounding planes it's sometimes possible for the function to stop testing vertices early when it can be determined that there are no planes left that all of the vertices might be outside of. For example, if at least one vertex has been found to be to the right of the left face plane, and at least one is found to be below the top face plane and likewise for the other four face planes, then there is no point in classifying all the remaining vertices because it's impossible that as a set they could all lie outside any one of those planes. We do this by keeping a running "cumulative AND" mask representing the set of cube face planes which still have a possibility of trivially rejecting the entire figure while looping through all the vertices. We do the same when testing against the sets of bevel and corner planes (i.e. the 12 planes adjacent to the cube edges and the 8 planes adjacent to the corners).

[NB: A graphic here showing these three sets of planes would be very useful here especially because one was not included in Voorhies' paper.]

As stated previously, when `trivial_vertex_tests` fails to classify a given polygon, `fast_polygon_intersects_cube` then calls the definitive `polygon_intersects_cube` routine. Just like in Voorhies' version the general algorithm is:

1. If any edge intersects the cube, return true.
2. If the polygon interior intersects the cube, return true.
3. Return false.

Our implementation, however, is very different. In the first step, testing whether a line segment intersects the cube is equivalent to testing whether the origin is contained in the solid obtained by dragging a unit cube from (being centered at) one segment endpoint to the other. This solid is a warped rhombic dodecahedron. The code to implement this consists of 12 sidedness tests, which is more efficient and less error-prone than the original six line-plane intersections plus six point-within-polygon tests.

[NB: a graphic might be useful here but it may be difficult to generate one that's any clearer than the above textual description.]

In the second step, since we know no vertex or edge intersects the cube, this amounts to testing whether any of the four cube diagonals intersects the interior of the polygon. (This is the same as Voorhies' test). The difference in our implementation is that we recognize that we really only need to test against the diagonal that comes closest to being perpendicular to the plane of the polygon; if

the polygon intersects any of the cube diagonals, it will intersect that one. Finding that diagonal is trivial, so this part of our implementation is roughly four times as fast as the original.

The last part of the second step is a test to see whether the polygon contains the point which is the intersection of the polygon's plane with the chosen diagonal. We provide the `polygon_contains_point_3d` for that purpose, and are publishing it because it may be useful for other purposes.

## VECTOR MATH MACRO LIBRARY

Another way we squeeze performance out of our implementation is through the use of the linear algebra library "vec.h". This library is implemented purely as a set of C macros, thereby avoiding a function call for every operation. Another big advantage of using a macro implementation is that it is completely type independent. The source and destination types of vectors and matrices may be any types that can be indexed into and are assignment compatible. All integer and floating point types may be freely mixed; in addition, any C++ classes that support arithmetic operations may be used.

The `vec.h` header file is automatically generated by the program `vec_h.c`. This program takes a single integer argument and outputs a `vec.h` file containing macros that handle vectors and matrices up to the dimension specified. The library includes N-dimensional determinants and cross products in addition to the simpler operations.

## POLYHEDRON-CUBE INTERSECTION TESTING

When used to test polyhedra, remember that the routines included in our module only test for intersections with points, edges and surfaces, not volumes. If no such intersection is reported, the caller should be aware that the volume of the polyhedra could still contain the entire unit box. That condition would then need to be checked for with an additional point-within-polyhedron test. The origin would be a natural point to check in such a test. Below is C-like pseudo-code that puts all the pieces together for a fast, complete polyhedron-cube intersection test.

```
switch(trivial_vertex_tests(verts))
{
    case 1: return true /* trivial accept */
    case 0: return false /* trivial reject */
    case -1: for each edge
               if(segment_intersects_cube(edge))
                   return true
             for each face
               if(fast_polygon_intersects_cube(..., true, true))
                   return true
    return polyhedra_contains_point(polyhedra, origin)
}
```

It's useful to notice that when a box is used as a modeling-space bounding polyhedron, testing its intersection against a view volume can often be performed in either direction. In other words, not only can the box be transformed by the viewing transformation that takes the view volume to the unit cube and then tested there, but the the view volume can also be transformed by the transformation that takes the

bounding box to be the unit cube and the test performed there. In the latter case it is the world-space truncated pyramid of the view volume that becomes the polyhedron being tested.

[Note that our implementation contains "const" arguments which may need to be removed for those old compilers that do not understand const.]

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "vec.h"
#include "pcube.h"

/*
 * Calculate a vector perpendicular to a planar polygon.
 * If the polygon is non-planar, a "best fit" plane will be used.
 * The polygon may be concave or even self-intersecting,
 * but it should have nonzero area or the result will be a zero vector
 * (e.g. the "bowtie" quad).
 * The length of vector will be twice the area of the polygon.
 * NOTE: This algorithm gives the same answer as Newell's method
 * (see Graphics Gems III) but is slightly more efficient than Newell's
 * for triangles and quads (slightly less efficient for higher polygons).
 */
static real *
get_polygon_normal(real normal[3],
                  int nverts, const real verts[/* nverts */][3])
{
    int i;
    real tothis[3], toprev[3], cross[3];

    /*
     * Triangulate the polygon and sum up the nverts-2 triangle normals.
     */
    ZEROVEC3(normal);
    VMV3(toprev, verts[1], verts[0]); /* 3 subtracts */
    for (i = 2; i <= nverts-1; ++i) { /* n-2 times... */
        VMV3(tothis, verts[i], verts[0]); /* 3 subtracts */
        VXV3(cross, toprev, tothis); /* 3 subtracts, 6 multiplies */
        VPV3(normal, normal, cross); /* 3 adds */
        SET3(toprev, tothis);
    }
    return normal;
}

main(int argc, char *argv[])
{
    int i, j, k, ntris, ins=0;
    real tri[3][3], normal[3];
    long fast, full;

    if(argc != 2)
        exit(sprintf("usage: %s <ntris>\n", argv[0]));
    ntris = atoi(argv[1]);
    for(i=0; i<ntris; i++) {
        for(j=0; j<3; j++)
            for(k=0; k<3; k++)
                tri[j][k] = (drand48() - .5) * 5.0;
        get_polygon_normal(normal, 3, tri);
        fast = fast_polygon_intersects_cube(3, tri, normal, 0, 0);
        full = polygon_intersects_cube(3, tri, normal, 0, 0);
        if(fast != full)
        {
            printf("fast = %d, full = %d\n", fast, full);
            printf("\t(%f,%f,%f)\n\t(%f,%f,%f)\n\t(%f,%f,%f)\n",
                tri[0][0], tri[0][1], tri[0][2],

```

```

                                tri[1][0], tri[1][1], tri[1][2],
                                tri[2][0], tri[2][1], tri[2][2]);
        }
        if(fast)
            ins++;
    }
    printf("%f percent intersected\n", (float)ins / ntris * 100);
}
```

```
/*
 * vec.h -- Vector macros for 2,3, and 4 dimensions,
 *          for any combination of C scalar types.
 *
 * Author:      Don Hatch (hatch@sgi.com)
 * Last modified: Fri Sep 30 03:23:02 PDT 1994
 *
 * General description:
 *
 * The macro name describes its arguments; e.g.
 *     MXS3 is "matrix times scalar in 3 dimensions";
 *     VMV2 is "vector minus vector in 2 dimensions".
 *
 * If the result of an operation is a scalar, then the macro "returns"
 * the value; e.g.
 *     result = DOT3(v,w);
 *     result = DET4(m);
 *
 * If the result of an operation is a vector or matrix, then
 * the first argument is the destination; e.g.
 *     SET2(tovec, fromvec);
 *     MXM3(result, m1, m2);
 *
 * WARNING: For the operations that are not done "componentwise"
 * (e.g. vector cross products and matrix multiplies)
 * the destination should not be either of the arguments,
 * for obvious reasons. For example, the following is wrong:
 *     VXM2(v,v,m);
 * For such "unsafe" macros, there are safe versions provided,
 * but you have to specify a type for the temporary
 * result vector or matrix. For example, the safe versions
 * of VXM2 are:
 *     VXM2d(v,v,m)    if v's scalar type is double or float
 *     VXM2i(v,v,m)    if v's scalar type is int or char
 *     VXM2l(v,v,m)    if v's scalar type is long
 *     VXM2r(v,v,m)    if v's scalar type is real
 *     VXM2safe(type,v,v,m) for other scalar types.
 * These "safe" macros do not evaluate to C expressions
 * (so, for example, they can't be used inside the parentheses of
 * a for(...)).
 *
 * Specific descriptions:
 *
 * The "?"'s in the following can be 2, 3, or 4.
 *
 * SET?(to,from)           to = from
 * SETMAT?(to,from)        to = from
 * ROUNDVEC?(to,from)       to = from with entries rounded
 *                           to nearest integer
 * ROUNDMAT?(to,from)       to = from with entries rounded
 *                           to nearest integer
 * FILLVEC?(v,s)            set each entry of vector v to be s
 * FILLMAT?(m,s)            set each entry of matrix m to be s
 * ZEROVEC?(v)              v = 0
 * ISZEROVEC?(v)            v == 0
 * EQVEC?(v,w)              v == w
 * EQMAT?(m1,m2)            m1 == m2
 * ZEROMAT?(m)              m = 0
 * IDENTMAT?(m)             m = 1
 * TRANSPOSE?(to,from)      (matrix to) = (transpose of matrix from)
 * ADJOINT?(to,from)        (matrix to) = (adjoint of matrix from)
```



```

*                                     i.e. its determinant times its inverse
*
* V{P,M}V?(to,v,w)                  to = v {+,-} w
* M{P,M}M?(to,m1,m2)                to = m1 {+,-} m2
* SX{V,M}?(to,s,from)               to = s * from
* M{V,M}?(to,from)                  to = -from
* {V,M}{X,D}S?(to,from,s)           to = from {*,/} s
* MXM?(to,m1,m2)                    to = m1 * m2
* VXM?(to,v,m)                      (row vec to) = (row vec v) * m
* MXV?(to,m,v)                      (column vec to) = m * (column vec v)
* LERP?(to,v0,v1,t)                 to = v0 + t*(v1-v0)
*
* DET?(m)                           determinant of m
* TRACE?(m)                          trace (sum of diagonal entries) of m
* DOT?(v,w)                          dot (scalar) product of v and w
* NORMSQRD?(v)                       square of |v|
* DISTSQRD?(v,w)                     square of |v-w|
*
* XV2(to,v)                          to = v rotated by 90 degrees
* VXV3(to,v1,v2)                     to = cross (vector) product of v1 and v2
* VXVXV4(to,v1,v2,v3)                to = 4-dimensional vector cross product
*                                     of v1,v2,v3 (a vector orthogonal to
*                                     v1,v2,v3 whose length equals the
*                                     volume of the spanned parallelotope)
*
* VXV2(v0,v1)                        determinant of matrix with rows v0,v1
* VXVXV3(v0,v1,v2)                  determinant of matrix with rows v0,v1,v2
* VXVXVXV4(v0,v1,v2,v3)             determinant of matrix with rows v0,...,v3

```

The following macros mix objects from different dimensions.

For example, V3XM4 would be used to apply a composite  
4x4 rotation-and-translation matrix to a 3d vector.

```

* SET3from2(to,from,pad)              (3d vec to) = (2d vec from) with pad
* SET4from3(to,from,pad)              (4d vec to) = (3d vec from) with pad
* SETMAT3from2(to,from,pad0,pad1)     (3x3 mat to) = (2x2 mat from)
*                                     padded with pad0 on the sides
*                                     and pad1 in the corner
* SETMAT4from3(to,from,pad0,pad1)     (4x4 mat to) = (3x3 mat from)
*                                     padded with pad0 on the sides
*                                     and pad1 in the corner
*
* V2XM3(to2,v2,m3)                    (2d row vec to2) = (2d row vec v2) * (3x3 mat m3)
* V3XM4(to3,v3,m4)                    (3d row vec to3) = (3d row vec v2) * (4x4 mat m4)
* M3XV2(to2,m3,v2)                    (2d col vec to2) = (3x3 mat m3) * (2d col vec v2)
* M4XV3(to3,m4,v3)                    (3d col vec to3) = (4x4 mat m4) * (3d col vec v3)
* M2XM3(to3,m2,m3)                    (3x3 mat to3) = (2x2 mat m2) * (3x3 mat m3)
* M3XM4(to4,m3,m4)                    (4x4 mat to4) = (3x3 mat m3) * (4x4 mat m4)
* M3XM2(to3,m3,m2)                    (3x3 mat to3) = (3x3 mat m3) * (2x2 mat m2)
* M4XM3(to4,m4,m3)                    (4x4 mat to4) = (4x4 mat m4) * (3x3 mat m3)

```

This file is machine-generated and can be regenerated  
for any number of dimensions.

The program that generated it is available upon request.

\*/

```

#ifndef VEC_H
#define VEC_H 4
#include <math.h> /* for definition of floor() */
#define SET2(to,from) \
    ((to)[0] = (from)[0], \
     (to)[1] = (from)[1])

```

```
#define SETMAT2(to,from) \
    (SET2((to)[0], (from)[0]), \
     SET2((to)[1], (from)[1]))
#define ROUNDVEC2(to,from) \
    ((to)[0] = floor((from)[0]+.5), \
     (to)[1] = floor((from)[1]+.5))
#define ROUNDMAT2(to,from) \
    (ROUNDVEC2((to)[0], (from)[0]), \
     ROUNDVEC2((to)[1], (from)[1]))
#define FILLVEC2(v,s) \
    ((v)[0] = (s), \
     (v)[1] = (s))
#define FILLMAT2(m,s) \
    (FILLVEC2((m)[0], s), \
     FILLVEC2((m)[1], s))
#define ZEROVEC2(v) \
    ((v)[0] = 0, \
     (v)[1] = 0)
#define ISZEROVEC2(v) \
    ((v)[0] == 0 && \
     (v)[1] == 0)
#define EQVEC2(v,w) \
    ((v)[0] == (w)[0] && \
     (v)[1] == (w)[1])
#define EQMAT2(m1,m2) \
    (EQVEC2((m1)[0], (m2)[0]) && \
     EQVEC2((m1)[1], (m2)[1]))
#define ZEROMAT2(m) \
    (ZEROVEC2((m)[0]), \
     ZEROVEC2((m)[1]))
#define IDENTMAT2(m) \
    (ZEROVEC2((m)[0]), (m)[0][0]=1, \
     ZEROVEC2((m)[1]), (m)[1][1]=1)
#define TRANSPOSE2(to,from) \
    (_SETcol2((to)[0], from, 0), \
     _SETcol2((to)[1], from, 1))
#define VPV2(to,v,w) \
    ((to)[0] = (v)[0] + (w)[0], \
     (to)[1] = (v)[1] + (w)[1])
#define VMV2(to,v,w) \
    ((to)[0] = (v)[0] - (w)[0], \
     (to)[1] = (v)[1] - (w)[1])
#define MPM2(to,m1,m2) \
    (VPV2((to)[0], (m1)[0], (m2)[0]), \
     VPV2((to)[1], (m1)[1], (m2)[1]))
#define MMM2(to,m1,m2) \
    (VMV2((to)[0], (m1)[0], (m2)[0]), \
     VMV2((to)[1], (m1)[1], (m2)[1]))
#define SXV2(to,s,from) \
    ((to)[0] = (s) * (from)[0], \
     (to)[1] = (s) * (from)[1])
#define SXM2(to,s,from) \
    (SXV2((to)[0], s, (from)[0]), \
     SXV2((to)[1], s, (from)[1]))
#define MV2(to,from) \
    ((to)[0] = -(from)[0], \
     (to)[1] = -(from)[1])
#define MM2(to,from) \
    (MV2((to)[0], (from)[0]), \
     MV2((to)[1], (from)[1]))
#define VXS2(to,from,s) \
```

```
        ((to)[0] = (from)[0] * (s), \
        (to)[1] = (from)[1] * (s))
#define VDS2(to,from,s) \
        ((to)[0] = (from)[0] / (s), \
        (to)[1] = (from)[1] / (s))
#define MXS2(to,from,s) \
        (VXS2((to)[0], (from)[0], s), \
        VXS2((to)[1], (from)[1], s))
#define MDS2(to,from,s) \
        (VDS2((to)[0], (from)[0], s), \
        VDS2((to)[1], (from)[1], s))
#define MXM2(to,m1,m2) \
        (VXM2((to)[0], (m1)[0], m2), \
        VXM2((to)[1], (m1)[1], m2))
#define VXM2(to,v,m) \
        ((to)[0] = _DOTcol2(v, m, 0), \
        (to)[1] = _DOTcol2(v, m, 1))
#define MXV2(to,m,v) \
        ((to)[0] = DOT2((m)[0], v), \
        (to)[1] = DOT2((m)[1], v))
#define LERP2(to,v0,v1,t) \
        ((to)[0]=(v0)[0]+(t)*((v1)[0]-(v0)[0]), \
        (to)[1]=(v0)[1]+(t)*((v1)[1]-(v0)[1]))
#define TRACE2(m) \
        ((m)[0][0] + \
        (m)[1][1])
#define DOT2(v,w) \
        ((v)[0] * (w)[0] + \
        (v)[1] * (w)[1])
#define NORMSQRD2(v) \
        ((v)[0] * (v)[0] + \
        (v)[1] * (v)[1])
#define DISTSQRD2(v,w) \
        (((v)[0]-(w)[0])*((v)[0]-(w)[0]) + \
        ((v)[1]-(w)[1])*((v)[1]-(w)[1]))
#define _DOTcol2(v,m,j) \
        ((v)[0] * (m)[0][j] + \
        (v)[1] * (m)[1][j])
#define _SETcol2(v,m,j) \
        ((v)[0] = (m)[0][j], \
        (v)[1] = (m)[1][j])
#define _MXVcol2(to,m,M,j) \
        ((to)[0][j] = _DOTcol2((m)[0],M,j), \
        (to)[1][j] = _DOTcol2((m)[1],M,j))
#define _DET2(v0,v1,i0,i1) \
        ((v0)[i0]* _DET1(v1,i1) + \
        (v0)[i1]*-_DET1(v1,i0))
#define XV2(to,v1) \
        ((to)[0] = -_DET1(v1, 1), \
        (to)[1] = _DET1(v1, 0))
#define V2XM3(to2,v2,m3) \
        ((to2)[0] = _DOTcol2(v2,m3,0) + (m3)[2][0], \
        (to2)[1] = _DOTcol2(v2,m3,1) + (m3)[2][1])
#define M3XV2(to2,m3,v2) \
        ((to2)[0] = DOT2((m3)[0],v2) + (m3)[0][2], \
        (to2)[1] = DOT2((m3)[1],v2) + (m3)[1][2])
#define _DET1(v0,i0) \
        ((v0)[i0])
#define VXV2(v0,v1) \
        (_DET2(v0,v1,0,1))
#define DET2(m) \
```

```
(VXV2((m)[0],(m)[1]))
#define ADJOINT2(to,m) \
    ( _ADJOINTcol2(to,0,m,1), \
      __ADJOINTcol2(to,1,m,0))
#define _ADJOINTcol2(to,col,m,i1) \
    ((to)[0][col] = _DET1(m[i1], 1), \
     (to)[1][col] = -_DET1(m[i1], 0))
#define __ADJOINTcol2(to,col,m,i1) \
    ((to)[0][col] = -_DET1(m[i1], 1), \
     (to)[1][col] = _DET1(m[i1], 0))
#define SET3(to,from) \
    ((to)[0] = (from)[0], \
     (to)[1] = (from)[1], \
     (to)[2] = (from)[2])
#define SETMAT3(to,from) \
    (SET3((to)[0], (from)[0]), \
     SET3((to)[1], (from)[1]), \
     SET3((to)[2], (from)[2]))
#define ROUNDVEC3(to,from) \
    ((to)[0] = floor((from)[0]+.5), \
     (to)[1] = floor((from)[1]+.5), \
     (to)[2] = floor((from)[2]+.5))
#define ROUNDMAT3(to,from) \
    (ROUNDVEC3((to)[0], (from)[0]), \
     ROUNDVEC3((to)[1], (from)[1]), \
     ROUNDVEC3((to)[2], (from)[2]))
#define FILLVEC3(v,s) \
    ((v)[0] = (s), \
     (v)[1] = (s), \
     (v)[2] = (s))
#define FILLMAT3(m,s) \
    (FILLVEC3((m)[0], s), \
     FILLVEC3((m)[1], s), \
     FILLVEC3((m)[2], s))
#define ZEROVEC3(v) \
    ((v)[0] = 0, \
     (v)[1] = 0, \
     (v)[2] = 0)
#define ISZEROVEC3(v) \
    ((v)[0] == 0 && \
     (v)[1] == 0 && \
     (v)[2] == 0)
#define EQVEC3(v,w) \
    ((v)[0] == (w)[0] && \
     (v)[1] == (w)[1] && \
     (v)[2] == (w)[2])
#define EQMAT3(m1,m2) \
    (EQVEC3((m1)[0], (m2)[0]) && \
     EQVEC3((m1)[1], (m2)[1]) && \
     EQVEC3((m1)[2], (m2)[2]))
#define ZEROMAT3(m) \
    (ZEROVEC3((m)[0]), \
     ZEROVEC3((m)[1]), \
     ZEROVEC3((m)[2]))
#define IDENTMAT3(m) \
    (ZEROVEC3((m)[0]), (m)[0][0]=1, \
     ZEROVEC3((m)[1]), (m)[1][1]=1, \
     ZEROVEC3((m)[2]), (m)[2][2]=1)
#define TRANSPOSE3(to,from) \
    (_SETcol3((to)[0], from, 0), \
     _SETcol3((to)[1], from, 1), \
```

```
        _SETcol3((to)[2], from, 2))
#define VPV3(to,v,w) \
    ((to)[0] = (v)[0] + (w)[0], \
     (to)[1] = (v)[1] + (w)[1], \
     (to)[2] = (v)[2] + (w)[2])
#define VMV3(to,v,w) \
    ((to)[0] = (v)[0] - (w)[0], \
     (to)[1] = (v)[1] - (w)[1], \
     (to)[2] = (v)[2] - (w)[2])
#define MPM3(to,m1,m2) \
    (VPV3((to)[0], (m1)[0], (m2)[0]), \
     VPV3((to)[1], (m1)[1], (m2)[1]), \
     VPV3((to)[2], (m1)[2], (m2)[2]))
#define MMM3(to,m1,m2) \
    (VMV3((to)[0], (m1)[0], (m2)[0]), \
     VMV3((to)[1], (m1)[1], (m2)[1]), \
     VMV3((to)[2], (m1)[2], (m2)[2]))
#define SXV3(to,s,from) \
    ((to)[0] = (s) * (from)[0], \
     (to)[1] = (s) * (from)[1], \
     (to)[2] = (s) * (from)[2])
#define SXM3(to,s,from) \
    (SXV3((to)[0], s, (from)[0]), \
     SXV3((to)[1], s, (from)[1]), \
     SXV3((to)[2], s, (from)[2]))
#define MV3(to,from) \
    ((to)[0] = -(from)[0], \
     (to)[1] = -(from)[1], \
     (to)[2] = -(from)[2])
#define MM3(to,from) \
    (MV3((to)[0], (from)[0]), \
     MV3((to)[1], (from)[1]), \
     MV3((to)[2], (from)[2]))
#define VXS3(to,from,s) \
    ((to)[0] = (from)[0] * (s), \
     (to)[1] = (from)[1] * (s), \
     (to)[2] = (from)[2] * (s))
#define VDS3(to,from,s) \
    ((to)[0] = (from)[0] / (s), \
     (to)[1] = (from)[1] / (s), \
     (to)[2] = (from)[2] / (s))
#define MXS3(to,from,s) \
    (VXS3((to)[0], (from)[0], s), \
     VXS3((to)[1], (from)[1], s), \
     VXS3((to)[2], (from)[2], s))
#define MDS3(to,from,s) \
    (VDS3((to)[0], (from)[0], s), \
     VDS3((to)[1], (from)[1], s), \
     VDS3((to)[2], (from)[2], s))
#define MXM3(to,m1,m2) \
    (VXM3((to)[0], (m1)[0], m2), \
     VXM3((to)[1], (m1)[1], m2), \
     VXM3((to)[2], (m1)[2], m2))
#define VXM3(to,v,m) \
    ((to)[0] = _DOTcol3(v, m, 0), \
     (to)[1] = _DOTcol3(v, m, 1), \
     (to)[2] = _DOTcol3(v, m, 2))
#define MXV3(to,m,v) \
    ((to)[0] = DOT3((m)[0], v), \
     (to)[1] = DOT3((m)[1], v), \
     (to)[2] = DOT3((m)[2], v))
```

```
#define LERP3(to,v0,v1,t) \
    ((to)[0]=(v0)[0]+(t)*((v1)[0]-(v0)[0]), \
     (to)[1]=(v0)[1]+(t)*((v1)[1]-(v0)[1]), \
     (to)[2]=(v0)[2]+(t)*((v1)[2]-(v0)[2]))

#define TRACE3(m) \
    ((m)[0][0] + \
     (m)[1][1] + \
     (m)[2][2])

#define DOT3(v,w) \
    ((v)[0] * (w)[0] + \
     (v)[1] * (w)[1] + \
     (v)[2] * (w)[2])

#define NORMSQRD3(v) \
    ((v)[0] * (v)[0] + \
     (v)[1] * (v)[1] + \
     (v)[2] * (v)[2])

#define DISTSQRD3(v,w) \
    (((v)[0]-(w)[0])*((v)[0]-(w)[0]) + \
     ((v)[1]-(w)[1])*((v)[1]-(w)[1]) + \
     ((v)[2]-(w)[2])*((v)[2]-(w)[2]))

#define _DOTcol3(v,m,j) \
    ((v)[0] * (m)[0][j] + \
     (v)[1] * (m)[1][j] + \
     (v)[2] * (m)[2][j])

#define _SETcol3(v,m,j) \
    ((v)[0] = (m)[0][j], \
     (v)[1] = (m)[1][j], \
     (v)[2] = (m)[2][j])

#define _MXVcol3(to,m,M,j) \
    ((to)[0][j] = _DOTcol3((m)[0],M,j), \
     (to)[1][j] = _DOTcol3((m)[1],M,j), \
     (to)[2][j] = _DOTcol3((m)[2],M,j))

#define _DET3(v0,v1,v2,i0,i1,i2) \
    ((v0)[i0]*_DET2(v1,v2,i1,i2) + \
     (v0)[i1]*-_DET2(v1,v2,i0,i2) + \
     (v0)[i2]*_DET2(v1,v2,i0,i1))

#define VXV3(to,v1,v2) \
    ((to)[0] = _DET2(v1,v2, 1,2), \
     (to)[1] = -_DET2(v1,v2, 0,2), \
     (to)[2] = _DET2(v1,v2, 0,1))

#define SET3from2(to,from,pad) \
    ((to)[0] = (from)[0], \
     (to)[1] = (from)[1], \
     (to)[2] = (pad))

#define SETMAT3from2(to,from,pad0,pad1) \
    (SET3from2((to)[0], (from)[0], pad0), \
     SET3from2((to)[1], (from)[1], pad0), \
     FILLVEC2((to)[2], (pad0)), (to)[2][2] = (pad1))

#define M2XM3(to3,m2,m3) \
    (_MXVcol2(to3,m2,m3,0), (to3)[2][0]=(m3)[2][0], \
     _MXVcol2(to3,m2,m3,1), (to3)[2][1]=(m3)[2][1], \
     _MXVcol2(to3,m2,m3,2), (to3)[2][2]=(m3)[2][2])

#define M3XM2(to3,m3,m2) \
    (VXM2((to3)[0],(m3)[0],m2), (to3)[0][2]=(m3)[0][2], \
     VXM2((to3)[1],(m3)[1],m2), (to3)[1][2]=(m3)[1][2], \
     VXM2((to3)[2],(m3)[2],m2), (to3)[2][2]=(m3)[2][2])

#define V3XM4(to3,v3,m4) \
    ((to3)[0] = _DOTcol3(v3,m4,0) + (m4)[3][0], \
     (to3)[1] = _DOTcol3(v3,m4,1) + (m4)[3][1], \
     (to3)[2] = _DOTcol3(v3,m4,2) + (m4)[3][2])

#define M4XV3(to3,m4,v3) \
```

```
        ((to3)[0] = DOT3((m4)[0],v3) + (m4)[0][3], \
        (to3)[1] = DOT3((m4)[1],v3) + (m4)[1][3], \
        (to3)[2] = DOT3((m4)[2],v3) + (m4)[2][3])
#define VXVXV3(v0,v1,v2) \
    (_DET3(v0,v1,v2,0,1,2))
#define DET3(m) \
    (VXVXV3((m)[0],(m)[1],(m)[2]))
#define ADJOINT3(to,m) \
    (_ADJOINTcol3(to,0,m,1,2), \
    __ADJOINTcol3(to,1,m,0,2), \
    _ADJOINTcol3(to,2,m,0,1))
#define _ADJOINTcol3(to,col,m,i1,i2) \
    ((to)[0][col] = _DET2(m[i1],m[i2], 1,2), \
    (to)[1][col] = -_DET2(m[i1],m[i2], 0,2), \
    (to)[2][col] = _DET2(m[i1],m[i2], 0,1))
#define __ADJOINTcol3(to,col,m,i1,i2) \
    ((to)[0][col] = -_DET2(m[i1],m[i2], 1,2), \
    (to)[1][col] = _DET2(m[i1],m[i2], 0,2), \
    (to)[2][col] = -_DET2(m[i1],m[i2], 0,1))
#define SET4(to,from) \
    ((to)[0] = (from)[0], \
    (to)[1] = (from)[1], \
    (to)[2] = (from)[2], \
    (to)[3] = (from)[3])
#define SETMAT4(to,from) \
    (SET4((to)[0], (from)[0]), \
    SET4((to)[1], (from)[1]), \
    SET4((to)[2], (from)[2]), \
    SET4((to)[3], (from)[3]))
#define ROUNDVEC4(to,from) \
    ((to)[0] = floor((from)[0]+.5), \
    (to)[1] = floor((from)[1]+.5), \
    (to)[2] = floor((from)[2]+.5), \
    (to)[3] = floor((from)[3]+.5))
#define ROUNDMAT4(to,from) \
    (ROUNDVEC4((to)[0], (from)[0]), \
    ROUNDVEC4((to)[1], (from)[1]), \
    ROUNDVEC4((to)[2], (from)[2]), \
    ROUNDVEC4((to)[3], (from)[3]))
#define FILLVEC4(v,s) \
    ((v)[0] = (s), \
    (v)[1] = (s), \
    (v)[2] = (s), \
    (v)[3] = (s))
#define FILLMAT4(m,s) \
    (FILLVEC4((m)[0], s), \
    FILLVEC4((m)[1], s), \
    FILLVEC4((m)[2], s), \
    FILLVEC4((m)[3], s))
#define ZEROVEC4(v) \
    ((v)[0] = 0, \
    (v)[1] = 0, \
    (v)[2] = 0, \
    (v)[3] = 0)
#define ISZEROVEC4(v) \
    ((v)[0] == 0 && \
    (v)[1] == 0 && \
    (v)[2] == 0 && \
    (v)[3] == 0)
#define EQVEC4(v,w) \
    ((v)[0] == (w)[0] && \
```

```
(v)[1] == (w)[1] && \
(v)[2] == (w)[2] && \
(v)[3] == (w)[3])
#define EQMAT4(m1,m2) \
    (EQVEC4((m1)[0], (m2)[0]) && \
    EQVEC4((m1)[1], (m2)[1]) && \
    EQVEC4((m1)[2], (m2)[2]) && \
    EQVEC4((m1)[3], (m2)[3]))
#define ZEROMAT4(m) \
    (ZEROVEC4((m)[0]), \
    ZEROVEC4((m)[1]), \
    ZEROVEC4((m)[2]), \
    ZEROVEC4((m)[3]))
#define IDENTMAT4(m) \
    (ZEROVEC4((m)[0]), (m)[0][0]=1, \
    ZEROVEC4((m)[1]), (m)[1][1]=1, \
    ZEROVEC4((m)[2]), (m)[2][2]=1, \
    ZEROVEC4((m)[3]), (m)[3][3]=1)
#define TRANSPOSE4(to,from) \
    (_SETcol4((to)[0], from, 0), \
    _SETcol4((to)[1], from, 1), \
    _SETcol4((to)[2], from, 2), \
    _SETcol4((to)[3], from, 3))
#define VPV4(to,v,w) \
    ((to)[0] = (v)[0] + (w)[0], \
    (to)[1] = (v)[1] + (w)[1], \
    (to)[2] = (v)[2] + (w)[2], \
    (to)[3] = (v)[3] + (w)[3])
#define VMV4(to,v,w) \
    ((to)[0] = (v)[0] - (w)[0], \
    (to)[1] = (v)[1] - (w)[1], \
    (to)[2] = (v)[2] - (w)[2], \
    (to)[3] = (v)[3] - (w)[3])
#define MPM4(to,m1,m2) \
    (VPV4((to)[0], (m1)[0], (m2)[0]), \
    VPV4((to)[1], (m1)[1], (m2)[1]), \
    VPV4((to)[2], (m1)[2], (m2)[2]), \
    VPV4((to)[3], (m1)[3], (m2)[3]))
#define MMM4(to,m1,m2) \
    (VMV4((to)[0], (m1)[0], (m2)[0]), \
    VMV4((to)[1], (m1)[1], (m2)[1]), \
    VMV4((to)[2], (m1)[2], (m2)[2]), \
    VMV4((to)[3], (m1)[3], (m2)[3]))
#define SXV4(to,s,from) \
    ((to)[0] = (s) * (from)[0], \
    (to)[1] = (s) * (from)[1], \
    (to)[2] = (s) * (from)[2], \
    (to)[3] = (s) * (from)[3])
#define SXM4(to,s,from) \
    (SXV4((to)[0], s, (from)[0]), \
    SXV4((to)[1], s, (from)[1]), \
    SXV4((to)[2], s, (from)[2]), \
    SXV4((to)[3], s, (from)[3]))
#define MV4(to,from) \
    ((to)[0] = -(from)[0], \
    (to)[1] = -(from)[1], \
    (to)[2] = -(from)[2], \
    (to)[3] = -(from)[3])
#define MM4(to,from) \
    (MV4((to)[0], (from)[0]), \
    MV4((to)[1], (from)[1]), \
```



```
        MV4((to)[2], (from)[2]), \
        MV4((to)[3], (from)[3]))
#define VXS4(to,from,s) \
    ((to)[0] = (from)[0] * (s), \
    (to)[1] = (from)[1] * (s), \
    (to)[2] = (from)[2] * (s), \
    (to)[3] = (from)[3] * (s))
#define VDS4(to,from,s) \
    ((to)[0] = (from)[0] / (s), \
    (to)[1] = (from)[1] / (s), \
    (to)[2] = (from)[2] / (s), \
    (to)[3] = (from)[3] / (s))
#define MXS4(to,from,s) \
    (VXS4((to)[0], (from)[0], s), \
    VXS4((to)[1], (from)[1], s), \
    VXS4((to)[2], (from)[2], s), \
    VXS4((to)[3], (from)[3], s))
#define MDS4(to,from,s) \
    (VDS4((to)[0], (from)[0], s), \
    VDS4((to)[1], (from)[1], s), \
    VDS4((to)[2], (from)[2], s), \
    VDS4((to)[3], (from)[3], s))
#define MXM4(to,m1,m2) \
    (VXM4((to)[0], (m1)[0], m2), \
    VXM4((to)[1], (m1)[1], m2), \
    VXM4((to)[2], (m1)[2], m2), \
    VXM4((to)[3], (m1)[3], m2))
#define VXM4(to,v,m) \
    ((to)[0] = _DOTcol4(v, m, 0), \
    (to)[1] = _DOTcol4(v, m, 1), \
    (to)[2] = _DOTcol4(v, m, 2), \
    (to)[3] = _DOTcol4(v, m, 3))
#define MXV4(to,m,v) \
    ((to)[0] = DOT4((m)[0], v), \
    (to)[1] = DOT4((m)[1], v), \
    (to)[2] = DOT4((m)[2], v), \
    (to)[3] = DOT4((m)[3], v))
#define LERP4(to,v0,v1,t) \
    ((to)[0]=(v0)[0]+(t)*((v1)[0]-(v0)[0]), \
    (to)[1]=(v0)[1]+(t)*((v1)[1]-(v0)[1]), \
    (to)[2]=(v0)[2]+(t)*((v1)[2]-(v0)[2]), \
    (to)[3]=(v0)[3]+(t)*((v1)[3]-(v0)[3]))
#define TRACE4(m) \
    ((m)[0][0] + \
    (m)[1][1] + \
    (m)[2][2] + \
    (m)[3][3])
#define DOT4(v,w) \
    ((v)[0] * (w)[0] + \
    (v)[1] * (w)[1] + \
    (v)[2] * (w)[2] + \
    (v)[3] * (w)[3])
#define NORMSQRD4(v) \
    ((v)[0] * (v)[0] + \
    (v)[1] * (v)[1] + \
    (v)[2] * (v)[2] + \
    (v)[3] * (v)[3])
#define DISTSQRD4(v,w) \
    (((v)[0]-(w)[0])*((v)[0]-(w)[0]) + \
    ((v)[1]-(w)[1])*((v)[1]-(w)[1]) + \
    ((v)[2]-(w)[2])*((v)[2]-(w)[2]) + \
```

```
((v)[3]-(w)[3])*((v)[3]-(w)[3]))
#define _DOTcol4(v,m,j) \
    ((v)[0] * (m)[0][j] + \
     (v)[1] * (m)[1][j] + \
     (v)[2] * (m)[2][j] + \
     (v)[3] * (m)[3][j])
#define _SETcol4(v,m,j) \
    ((v)[0] = (m)[0][j], \
     (v)[1] = (m)[1][j], \
     (v)[2] = (m)[2][j], \
     (v)[3] = (m)[3][j])
#define _MXVcol4(to,m,M,j) \
    ((to)[0][j] = _DOTcol4((m)[0],M,j), \
     (to)[1][j] = _DOTcol4((m)[1],M,j), \
     (to)[2][j] = _DOTcol4((m)[2],M,j), \
     (to)[3][j] = _DOTcol4((m)[3],M,j))
#define _DET4(v0,v1,v2,v3,i0,i1,i2,i3) \
    ((v0)[i0]*_DET3(v1,v2,v3,i1,i2,i3) + \
     (v0)[i1]*-_DET3(v1,v2,v3,i0,i2,i3) + \
     (v0)[i2]*_DET3(v1,v2,v3,i0,i1,i3) + \
     (v0)[i3]*-_DET3(v1,v2,v3,i0,i1,i2))
#define VXVXV4(to,v1,v2,v3) \
    ((to)[0] = -_DET3(v1,v2,v3, 1,2,3), \
     (to)[1] = _DET3(v1,v2,v3, 0,2,3), \
     (to)[2] = -_DET3(v1,v2,v3, 0,1,3), \
     (to)[3] = _DET3(v1,v2,v3, 0,1,2))
#define SET4from3(to,from,pad) \
    ((to)[0] = (from)[0], \
     (to)[1] = (from)[1], \
     (to)[2] = (from)[2], \
     (to)[3] = (pad))
#define SETMAT4from3(to,from,pad0,pad1) \
    (SET4from3((to)[0], (from)[0], pad0), \
     SET4from3((to)[1], (from)[1], pad0), \
     SET4from3((to)[2], (from)[2], pad0), \
     FILLVEC3((to)[3], (pad0)), (to)[3][3] = (pad1))
#define M3XM4(to4,m3,m4) \
    (_MXVcol3(to4,m3,m4,0), (to4)[3][0]=(m4)[3][0], \
     _MXVcol3(to4,m3,m4,1), (to4)[3][1]=(m4)[3][1], \
     _MXVcol3(to4,m3,m4,2), (to4)[3][2]=(m4)[3][2], \
     _MXVcol3(to4,m3,m4,3), (to4)[3][3]=(m4)[3][3])
#define M4XM3(to4,m4,m3) \
    (VXM3((to4)[0],(m4)[0],m3), (to4)[0][3]=(m4)[0][3], \
     VXM3((to4)[1],(m4)[1],m3), (to4)[1][3]=(m4)[1][3], \
     VXM3((to4)[2],(m4)[2],m3), (to4)[2][3]=(m4)[2][3], \
     VXM3((to4)[3],(m4)[3],m3), (to4)[3][3]=(m4)[3][3])
#define VXVXVXV4(v0,v1,v2,v3) \
    (_DET4(v0,v1,v2,v3,0,1,2,3))
#define DET4(m) \
    (VXVXVXV4((m)[0],(m)[1],(m)[2],(m)[3]))
#define ADJOINT4(to,m) \
    (_ADJOINTcol4(to,0,m,1,2,3), \
     _ADJOINTcol4(to,1,m,0,2,3), \
     _ADJOINTcol4(to,2,m,0,1,3), \
     _ADJOINTcol4(to,3,m,0,1,2))
#define _ADJOINTcol4(to,col,m,i1,i2,i3) \
    ((to)[0][col] = _DET3(m[i1],m[i2],m[i3], 1,2,3), \
     (to)[1][col] = -_DET3(m[i1],m[i2],m[i3], 0,2,3), \
     (to)[2][col] = _DET3(m[i1],m[i2],m[i3], 0,1,3), \
     (to)[3][col] = -_DET3(m[i1],m[i2],m[i3], 0,1,2))
#define __ADJOINTcol4(to,col,m,i1,i2,i3) \
```

```
        ((to)[0][col] = -_DET3(m[i1],m[i2],m[i3], 1,2,3), \
        (to)[1][col] =  _DET3(m[i1],m[i2],m[i3], 0,2,3), \
        (to)[2][col] = -_DET3(m[i1],m[i2],m[i3], 0,1,3), \
        (to)[3][col] =  _DET3(m[i1],m[i2],m[i3], 0,1,2))
#define TRANSPOSE2safe(type,to,from) \
        do {type _vec_h_temp_[2][2]; \
            TRANSPOSE2(_vec_h_temp_,from); \
            SETMAT2(to, _vec_h_temp_); \
        } while (0)
#define TRANSPOSE2d(to,from) TRANSPOSE2safe(double,to,from)
#define TRANSPOSE2i(to,from) TRANSPOSE2safe(int,to,from)
#define TRANSPOSE2l(to,from) TRANSPOSE2safe(long,to,from)
#define TRANSPOSE2r(to,from) TRANSPOSE2safe(real,to,from)
#define MXM2safe(type,to,m1,m2) \
        do {type _vec_h_temp_[2][2]; \
            MXM2(_vec_h_temp_,m1,m2); \
            SETMAT2(to, _vec_h_temp_); \
        } while (0)
#define MXM2d(to,m1,m2) MXM2safe(double,to,m1,m2)
#define MXM2i(to,m1,m2) MXM2safe(int,to,m1,m2)
#define MXM2l(to,m1,m2) MXM2safe(long,to,m1,m2)
#define MXM2r(to,m1,m2) MXM2safe(real,to,m1,m2)
#define VXM2safe(type,to,v,m) \
        do {type _vec_h_temp_[2]; \
            VXM2(_vec_h_temp_,v,m); \
            SET2(to, _vec_h_temp_); \
        } while (0)
#define VXM2d(to,v,m) VXM2safe(double,to,v,m)
#define VXM2i(to,v,m) VXM2safe(int,to,v,m)
#define VXM2l(to,v,m) VXM2safe(long,to,v,m)
#define VXM2r(to,v,m) VXM2safe(real,to,v,m)
#define MXV2safe(type,to,m,v) \
        do {type _vec_h_temp_[2]; \
            MXV2(_vec_h_temp_,m,v); \
            SET2(to, _vec_h_temp_); \
        } while (0)
#define MXV2d(to,m,v) MXV2safe(double,to,m,v)
#define MXV2i(to,m,v) MXV2safe(int,to,m,v)
#define MXV2l(to,m,v) MXV2safe(long,to,m,v)
#define MXV2r(to,m,v) MXV2safe(real,to,m,v)
#define XV2safe(type,to,v1) \
        do {type _vec_h_temp_[2]; \
            XV2(_vec_h_temp_,v1); \
            SET2(to, _vec_h_temp_); \
        } while (0)
#define XV2d(to,v1) XV2safe(double,to,v1)
#define XV2i(to,v1) XV2safe(int,to,v1)
#define XV2l(to,v1) XV2safe(long,to,v1)
#define XV2r(to,v1) XV2safe(real,to,v1)
#define V2XM3safe(type,to2,v2,m3) \
        do {type _vec_h_temp_[2]; \
            V2XM3(_vec_h_temp_,v2,m3); \
            SET2(to2, _vec_h_temp_); \
        } while (0)
#define V2XM3d(to2,v2,m3) V2XM3safe(double,to2,v2,m3)
#define V2XM3i(to2,v2,m3) V2XM3safe(int,to2,v2,m3)
#define V2XM3l(to2,v2,m3) V2XM3safe(long,to2,v2,m3)
#define V2XM3r(to2,v2,m3) V2XM3safe(real,to2,v2,m3)
#define M3XV2safe(type,to2,m3,v2) \
        do {type _vec_h_temp_[2]; \
            M3XV2(_vec_h_temp_,m3,v2); \
```

```
        SET2(to2, _vec_h_temp_); \
    } while (0)
#define M3XV2d(to2,m3,v2) M3XV2safe(double,to2,m3,v2)
#define M3XV2i(to2,m3,v2) M3XV2safe(int,to2,m3,v2)
#define M3XV2l(to2,m3,v2) M3XV2safe(long,to2,m3,v2)
#define M3XV2r(to2,m3,v2) M3XV2safe(real,to2,m3,v2)
#define ADJOINT2safe(type,to,m) \
    do {type _vec_h_temp_[2][2]; \
        ADJOINT2(_vec_h_temp_,m); \
        SETMAT2(to, _vec_h_temp_); \
    } while (0)
#define ADJOINT2d(to,m) ADJOINT2safe(double,to,m)
#define ADJOINT2i(to,m) ADJOINT2safe(int,to,m)
#define ADJOINT2l(to,m) ADJOINT2safe(long,to,m)
#define ADJOINT2r(to,m) ADJOINT2safe(real,to,m)
#define TRANSPPOSE3safe(type,to,from) \
    do {type _vec_h_temp_[3][3]; \
        TRANSPPOSE3(_vec_h_temp_,from); \
        SETMAT3(to, _vec_h_temp_); \
    } while (0)
#define TRANSPPOSE3d(to,from) TRANSPPOSE3safe(double,to,from)
#define TRANSPPOSE3i(to,from) TRANSPPOSE3safe(int,to,from)
#define TRANSPPOSE3l(to,from) TRANSPPOSE3safe(long,to,from)
#define TRANSPPOSE3r(to,from) TRANSPPOSE3safe(real,to,from)
#define MXM3safe(type,to,m1,m2) \
    do {type _vec_h_temp_[3][3]; \
        MXM3(_vec_h_temp_,m1,m2); \
        SETMAT3(to, _vec_h_temp_); \
    } while (0)
#define MXM3d(to,m1,m2) MXM3safe(double,to,m1,m2)
#define MXM3i(to,m1,m2) MXM3safe(int,to,m1,m2)
#define MXM3l(to,m1,m2) MXM3safe(long,to,m1,m2)
#define MXM3r(to,m1,m2) MXM3safe(real,to,m1,m2)
#define VXM3safe(type,to,v,m) \
    do {type _vec_h_temp_[3]; \
        VXM3(_vec_h_temp_,v,m); \
        SET3(to, _vec_h_temp_); \
    } while (0)
#define VXM3d(to,v,m) VXM3safe(double,to,v,m)
#define VXM3i(to,v,m) VXM3safe(int,to,v,m)
#define VXM3l(to,v,m) VXM3safe(long,to,v,m)
#define VXM3r(to,v,m) VXM3safe(real,to,v,m)
#define MXV3safe(type,to,m,v) \
    do {type _vec_h_temp_[3]; \
        MXV3(_vec_h_temp_,m,v); \
        SET3(to, _vec_h_temp_); \
    } while (0)
#define MXV3d(to,m,v) MXV3safe(double,to,m,v)
#define MXV3i(to,m,v) MXV3safe(int,to,m,v)
#define MXV3l(to,m,v) MXV3safe(long,to,m,v)
#define MXV3r(to,m,v) MXV3safe(real,to,m,v)
#define VXV3safe(type,to,v1,v2) \
    do {type _vec_h_temp_[3]; \
        VXV3(_vec_h_temp_,v1,v2); \
        SET3(to, _vec_h_temp_); \
    } while (0)
#define VXV3d(to,v1,v2) VXV3safe(double,to,v1,v2)
#define VXV3i(to,v1,v2) VXV3safe(int,to,v1,v2)
#define VXV3l(to,v1,v2) VXV3safe(long,to,v1,v2)
#define VXV3r(to,v1,v2) VXV3safe(real,to,v1,v2)
#define M2XM3safe(type,to3,m2,m3) \
```

```
        do {type _vec_h_temp_[3][3]; \
            M2XM3(_vec_h_temp_,m2,m3); \
            SETMAT3(to3, _vec_h_temp_); \
        } while (0)
#define M2XM3d(to3,m2,m3) M2XM3safe(double,to3,m2,m3)
#define M2XM3i(to3,m2,m3) M2XM3safe(int,to3,m2,m3)
#define M2XM3l(to3,m2,m3) M2XM3safe(long,to3,m2,m3)
#define M2XM3r(to3,m2,m3) M2XM3safe(real,to3,m2,m3)
#define M3XM2safe(type,to3,m3,m2) \
        do {type _vec_h_temp_[3][3]; \
            M3XM2(_vec_h_temp_,m3,m2); \
            SETMAT3(to3, _vec_h_temp_); \
        } while (0)
#define M3XM2d(to3,m3,m2) M3XM2safe(double,to3,m3,m2)
#define M3XM2i(to3,m3,m2) M3XM2safe(int,to3,m3,m2)
#define M3XM2l(to3,m3,m2) M3XM2safe(long,to3,m3,m2)
#define M3XM2r(to3,m3,m2) M3XM2safe(real,to3,m3,m2)
#define V3XM4safe(type,to3,v3,m4) \
        do {type _vec_h_temp_[3]; \
            V3XM4(_vec_h_temp_,v3,m4); \
            SET3(to3, _vec_h_temp_); \
        } while (0)
#define V3XM4d(to3,v3,m4) V3XM4safe(double,to3,v3,m4)
#define V3XM4i(to3,v3,m4) V3XM4safe(int,to3,v3,m4)
#define V3XM4l(to3,v3,m4) V3XM4safe(long,to3,v3,m4)
#define V3XM4r(to3,v3,m4) V3XM4safe(real,to3,v3,m4)
#define M4XV3safe(type,to3,m4,v3) \
        do {type _vec_h_temp_[3]; \
            M4XV3(_vec_h_temp_,m4,v3); \
            SET3(to3, _vec_h_temp_); \
        } while (0)
#define M4XV3d(to3,m4,v3) M4XV3safe(double,to3,m4,v3)
#define M4XV3i(to3,m4,v3) M4XV3safe(int,to3,m4,v3)
#define M4XV3l(to3,m4,v3) M4XV3safe(long,to3,m4,v3)
#define M4XV3r(to3,m4,v3) M4XV3safe(real,to3,m4,v3)
#define ADJOINT3safe(type,to,m) \
        do {type _vec_h_temp_[3][3]; \
            ADJOINT3(_vec_h_temp_,m); \
            SETMAT3(to, _vec_h_temp_); \
        } while (0)
#define ADJOINT3d(to,m) ADJOINT3safe(double,to,m)
#define ADJOINT3i(to,m) ADJOINT3safe(int,to,m)
#define ADJOINT3l(to,m) ADJOINT3safe(long,to,m)
#define ADJOINT3r(to,m) ADJOINT3safe(real,to,m)
#define TRANSPPOSE4safe(type,to,from) \
        do {type _vec_h_temp_[4][4]; \
            TRANSPPOSE4(_vec_h_temp_,from); \
            SETMAT4(to, _vec_h_temp_); \
        } while (0)
#define TRANSPPOSE4d(to,from) TRANSPPOSE4safe(double,to,from)
#define TRANSPPOSE4i(to,from) TRANSPPOSE4safe(int,to,from)
#define TRANSPPOSE4l(to,from) TRANSPPOSE4safe(long,to,from)
#define TRANSPPOSE4r(to,from) TRANSPPOSE4safe(real,to,from)
#define MXM4safe(type,to,m1,m2) \
        do {type _vec_h_temp_[4][4]; \
            MXM4(_vec_h_temp_,m1,m2); \
            SETMAT4(to, _vec_h_temp_); \
        } while (0)
#define MXM4d(to,m1,m2) MXM4safe(double,to,m1,m2)
#define MXM4i(to,m1,m2) MXM4safe(int,to,m1,m2)
#define MXM4l(to,m1,m2) MXM4safe(long,to,m1,m2)
```

```
#define MXM4r(to,m1,m2) MXM4safe(real,to,m1,m2)
#define VXM4safe(type,to,v,m) \
    do {type _vec_h_temp_[4]; \
        VXM4(_vec_h_temp_,v,m); \
        SET4(to, _vec_h_temp_); \
    } while (0)
#define VXM4d(to,v,m) VXM4safe(double,to,v,m)
#define VXM4i(to,v,m) VXM4safe(int,to,v,m)
#define VXM4l(to,v,m) VXM4safe(long,to,v,m)
#define VXM4r(to,v,m) VXM4safe(real,to,v,m)
#define MXV4safe(type,to,m,v) \
    do {type _vec_h_temp_[4]; \
        MXV4(_vec_h_temp_,m,v); \
        SET4(to, _vec_h_temp_); \
    } while (0)
#define MXV4d(to,m,v) MXV4safe(double,to,m,v)
#define MXV4i(to,m,v) MXV4safe(int,to,m,v)
#define MXV4l(to,m,v) MXV4safe(long,to,m,v)
#define MXV4r(to,m,v) MXV4safe(real,to,m,v)
#define VXVXV4safe(type,to,v1,v2,v3) \
    do {type _vec_h_temp_[4]; \
        VXVXV4(_vec_h_temp_,v1,v2,v3); \
        SET4(to, _vec_h_temp_); \
    } while (0)
#define VXVXV4d(to,v1,v2,v3) VXVXV4safe(double,to,v1,v2,v3)
#define VXVXV4i(to,v1,v2,v3) VXVXV4safe(int,to,v1,v2,v3)
#define VXVXV4l(to,v1,v2,v3) VXVXV4safe(long,to,v1,v2,v3)
#define VXVXV4r(to,v1,v2,v3) VXVXV4safe(real,to,v1,v2,v3)
#define M3XM4safe(type,to4,m3,m4) \
    do {type _vec_h_temp_[4][4]; \
        M3XM4(_vec_h_temp_,m3,m4); \
        SETMAT4(to4, _vec_h_temp_); \
    } while (0)
#define M3XM4d(to4,m3,m4) M3XM4safe(double,to4,m3,m4)
#define M3XM4i(to4,m3,m4) M3XM4safe(int,to4,m3,m4)
#define M3XM4l(to4,m3,m4) M3XM4safe(long,to4,m3,m4)
#define M3XM4r(to4,m3,m4) M3XM4safe(real,to4,m3,m4)
#define M4XM3safe(type,to4,m4,m3) \
    do {type _vec_h_temp_[4][4]; \
        M4XM3(_vec_h_temp_,m4,m3); \
        SETMAT4(to4, _vec_h_temp_); \
    } while (0)
#define M4XM3d(to4,m4,m3) M4XM3safe(double,to4,m4,m3)
#define M4XM3i(to4,m4,m3) M4XM3safe(int,to4,m4,m3)
#define M4XM3l(to4,m4,m3) M4XM3safe(long,to4,m4,m3)
#define M4XM3r(to4,m4,m3) M4XM3safe(real,to4,m4,m3)
#define ADJOINT4safe(type,to,m) \
    do {type _vec_h_temp_[4][4]; \
        ADJOINT4(_vec_h_temp_,m); \
        SETMAT4(to, _vec_h_temp_); \
    } while (0)
#define ADJOINT4d(to,m) ADJOINT4safe(double,to,m)
#define ADJOINT4i(to,m) ADJOINT4safe(int,to,m)
#define ADJOINT4l(to,m) ADJOINT4safe(long,to,m)
#define ADJOINT4r(to,m) ADJOINT4safe(real,to,m)
#endif /* VEC_H */
```

```
/* Ray-Convex Polyhedron Intersection Test by Eric Haines, erich@eye.com
 *
 * This test checks the ray against each face of a polyhedron, checking whether
 * the set of intersection points found for each ray-plane intersection
 * overlaps the previous intersection results.  If there is no overlap (i.e.
 * no line segment along the ray that is inside the polyhedron), then the
 * ray misses and returns 0; else 1 is returned if the ray is entering the
 * polyhedron, -1 if the ray originates inside the polyhedron.  If there is
 * an intersection, the distance and the normal of the face hit is returned.
 */

#include <math.h>
#include "GraphicsGems.h"

#ifndef HUGE_VAL
#define HUGE_VAL      1.7976931348623157e+308
#endif

typedef struct Point4Struct {    /* 4d point */
    double x, y, z, w;
} Point4;

/* fast macro version of V3Dot, usable with Point4 */
#define DOT3( a, b )    ( (a)->x*(b)->x + (a)->y*(b)->y + (a)->z*(b)->z )

/* return codes */
#define MISSED          0
#define FRONTFACE      1
#define BACKFACE       -1

int RayCvxPolyhedronInt( org, dir, tmax, phdrn, ph_num, tresult, norm )
Point3 *org, *dir ;    /* origin and direction of ray */
double tmax ;          /* maximum useful distance along ray */
Point4 *phdrn ;        /* list of planes in convex polyhedron */
int ph_num ;           /* number of planes in convex polyhedron */
double *tresult ;       /* returned: distance of intersection along ray */
Point3 *norm ;          /* returned: normal of face hit */
{
    Point4 *pln ;        /* plane equation */
    double tnear, tfar, t, vn, vd ;
    int fnorm_num, bnorm_num ; /* front/back face # hit */

    tnear = -HUGE_VAL ;
    tfar = tmax ;

    /* Test each plane in polyhedron */
    for ( pln = &phdrn[ph_num-1] ; ph_num-- ; pln-- ) {
        /* Compute intersection point T and sidedness */
        vd = DOT3( dir, pln ) ;
        vn = DOT3( org, pln ) + pln->w ;
        if ( vd == 0.0 ) {
            /* ray is parallel to plane - check if ray origin is inside plane's
             half-space */
            if ( vn > 0.0 )
                /* ray origin is outside half-space */
                return ( MISSED ) ;
        } else {
            /* ray not parallel - get distance to plane */
            t = -vn / vd ;
            if ( vd < 0.0 ) {
                /* front face - T is a near point */
                tnear = t ;
                fnorm_num = ph_num ;
            } else {
                /* back face - T is a far point */
                tfar = t ;
                bnorm_num = ph_num ;
            }
        }
    }

    if ( tnear < tfar )
        return ( fnorm_num ) ;
    else
        return ( bnorm_num ) ;
}
```









```
        if ( t > tfar ) return ( MISSED ) ;
        if ( t > tnear ) {
            /* hit near face, update normal */
            fnorm_num = ph_num ;
            tnear = t ;
        }
    } else {
        /* back face - T is a far point */
        if ( t < tnear ) return ( MISSED ) ;
        if ( t < tfar ) {
            /* hit far face, update normal */
            bnorm_num = ph_num ;
            tfar = t ;
        }
    }
}

/* survived all tests */
/* Note: if ray originates on polyhedron, may want to change 0.0 to some
 * epsilon to avoid intersecting the originating face.
 */
if ( tnear >= 0.0 ) {
    /* outside, hitting front face */
    *norm = *(Point3 *)&phdrn[fnorm_num] ;
    *tresult = tnear ;
    return ( FRONTFACE ) ;
} else {
    if ( tfar < tmax ) {
        /* inside, hitting back face */
        *norm = *(Point3 *)&phdrn[bnorm_num] ;
        *tresult = tfar ;
        return ( BACKFACE ) ;
    } else {
        /* inside, but back face beyond tmax */
        return ( MISSED ) ;
    }
}
}
```



# Index of

## /pubs/tog/GraphicsGems/gemsiv/ptpoly\_haines/

Name	Last modified	Size	Description
 <a href="#">_ Parent Directory</a>			
 <a href="#">_ Makefile</a>	29-Jun-00 08:20	1K	
 <a href="#">_ README</a>	29-Jun-00 08:20	1K	
 <a href="#">_ p_test.c</a>	29-Jun-00 08:20	24K	
 <a href="#">_ ptinpoly.c</a>	29-Jun-00 08:20	55K	
 <a href="#">_ ptinpoly.h</a>	29-Jun-00 08:20	6K	
 <a href="#">_ statrun.tst</a>	29-Jun-00 08:20	7K	
 <a href="#">_ table.awk</a>	29-Jun-00 08:20	1K	

```
# Valid options: -DTIMER -DDISPLAY -DCONVEX -DHYBRID -DSORT -DRANDOM -DWINDING
# You can define $MAKEOPTS outside the program for testing purposes.
#
# for testing purposes we leave all options off and invoke through $MAKEOPTS
CCFLAGS=-O
# good default if not testing:
# CCFLAGS=-O -DTIMER -DDISPLAY -DSORT -DRANDOM
```

```
# use this one for making one which will display the tests being done on an HP
# and do:
# export LDOPTS="-a shared"
```

```
p_test:          ptinpoly.o ptinpoly.h p_test.c
                  cc -o p_test p_test.c $(CCFLAGS) $(MAKEOPTS) ptinpoly.o -lm
# include these lines for linking in HP Starbase, for display version
#                  -L /usr/lib/X11R4 \
#                  -lXwindow \
#                  -lsb \
#                  -lXhp11 -lX11 -ldld \
#                  -lm
```

```
ptinpoly.o:      ptinpoly.c ptinpoly.h
                  cc -o ptinpoly.o -c ptinpoly.c $(CCFLAGS) $(MAKEOPTS)
```

```
clean:

                  rm -f ptinpoly.o p_test
```

ANSI C code from the article  
"Point in Polygon Strategies"  
by Eric Haines, [erich@eye.com](mailto:erich@eye.com)  
in "Graphics Gems IV", Academic Press, 1994

files:

- Makefile - makefile for code
- ptinpoly.h ptinpoly.c p\_test.c - the distribution code
- statrun.tst - the basic set of tests run for the book
- table.awk - an awk processor for the output files from statrun.tst

```
/* p_test.c - point in polygon inside/outside tester.  This is
 * simply testing and display code, the actual algorithms are in ptinpoly.c.
 *
 * Probably the most important thing to set for timings is TEST_RATIO (too
 * low and the timings are untrustworthy, too high and you wait forever).
 * Start low and see how consistent separate runs appear to be.
 *
 * To add a new algorithm to the test suite, add code at the spots marked
 * with '+++'.
 *
 * See Usage() for command line options (or just do "p_test -?").
 *
 * by Eric Haines, 3D/Eye Inc, erich@eye.com
 */

/* Define TIMER to perform timings on code (need system timing function) */
/* #define TIMER */

/* Number of times to try a single point vs. a polygon, per vertex.
 * This should be greater than 1 / ( HZ * approx. single test time in seconds )
 * in order to get a meaningful timings difference.  200 is reasonable for a
 * IBM PC 25 MHz 386 with no FPU, 50000 is good for an HP 720 workstation.
 * Start low and see how consistent separate runs appear to be.
 */

#ifndef M_PI
#define M_PI      3.14159265358979323846
#endif

#ifdef TIMER
#define MACHINE_TEST_RATIO      50000
#else
#define MACHINE_TEST_RATIO      1
#endif

/* ===== that's all the easy stuff than can be changed ===== */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "ptinpoly.h"

#ifdef TIMER
#ifdef IBM_PC
#include <bios.h>
#include <time.h>
#define HZ      CLK_TCK
#define mGetTime(t)      (t) = biostime(0,0L) ;

#else /* UNIX */
#include <sys/types.h>
#include <sys/param.h>
#include <sys/times.h>
struct tms      Timebuf ;

#define mGetTime(t)      (t) = times(&Timebuf) ;
#if (!(defined(sparc)||defined(__sparc__)))
#define mGetTime(t)      (t) = times(&Timebuf) ;
#else /* sparc (and others, you figure it out) */
/* for Sun and others do something like: */
#define mGetTime(t)      times(&Timebuf) ;
\
```

```
(t) = Timebuf.tms_utime + Timebuf.tms_stime ;
#endif /* sparc */

#endif
#endif

#ifdef DISPLAY
#include <starbase.c.h>          /* HP display */
#endif

#ifndef TRUE
#define TRUE      1
#define FALSE     0
#endif

#ifndef HUGE
#define HUGE      1.79769313486232e+308
#endif

#define X         0
#define Y         1

#ifdef DISPLAY
int      Display_Tests = 0 ;
double   Display_Scale ;
double   Display_OffsetX ;
double   Display_OffsetY ;
int      Fd ;
#endif

typedef struct {
    double  time_total ;
    int     test_ratio ;
    int     test_times ;
    int     work ;
    char    *name ;
    int     flag ;
} Statistics, *pStatistics ;

#define ANGLE_TEST           0
#define BARYCENTRIC_TEST    1
#define CROSSINGS_TEST      2
#define EXTERIOR_TEST       3
#define GRID_TEST           4
#define INCLUSION_TEST      5
#define CROSSMULT_TEST      6
#define PLANE_TEST          7
#define SPACKMAN_TEST       8
#define TRAPEZOID_TEST      9
#define WEILER_TEST        10
/* +++ add new name here and increment TOT_NUM_TESTS +++ */
#define TOT_NUM_TESTS      11

Statistics      St[TOT_NUM_TESTS] ;

char    *TestName[] = {
    "angle",
    "barycentric",
    "crossings",
    "exterior",
    "grid",
```

```
"inclusion",
"cross-mult",
"plane",
"spackman",
"trapezoid",
"weiler" } ;
```

```
/* +++ add new name here +++ */
```

```
/* minimum & maximum number of polygon vertices to generate */
```

```
#define TOT_VERTS      1000
static int      Min_Verts = 3 ;
static int      Max_Verts = 6 ;
```

```
/* Test polygons are generated by going CCW in a circle around the origin from
 * the X+ axis around and generating vertices.  The radius is how big the
 * circumscribing circle is, the perturbation is how much each vertex is varied.
 * So, radius 1 and perturbation 0 gives a regular, inscribed polygon;
 * radius 0 and perturbation 1 gives a totally random polygon in the
 * space [-1,1)
 */
```

```
static double    Vertex_Radius = 1.0 ;
static double    Vertex_Perturbation = 0.0 ;
```

```
/* A box is circumscribed around the test polygon.  Making this box bigger
 * is useful for a higher rejection rate.  For example, a ray tracing bounding
 * box usually contains a few polygons, so making the box ratio say 2 or so
 * could simulate this type of box.
 */
```

```
static double    Box_Ratio = 1.0 ;
```

```
/* for debugging purposes, you might want to set Test_Polygons and Test_Points
 * high (say 1000), and then set the *_Test_Times to 1.  The timings will be
 * useless, but you'll test 1000 polygons each with 1000 points.  You'll also
 * probably want to set the Vertex_Perturbation to something > 0.0.
 */
```

```
/* number of different polygons to try - I like 50 or so; left low for now */
static int      Test_Polygons = 20 ;
```

```
/* number of different intersection points to try - I like 50 or so */
static int      Test_Points = 20 ;
```

```
/* for debugging or constrained test purposes, this value constrains the value
 * of the polygon points and the test points to those on a grid emanating from
 * the origin with this increment.  Points are shifted to the closest grid
 * point.  0.0 means no grid.  NOTE: by setting this value, a large number
 * of points will be generated exactly on interior (triangle fan) or exterior
 * edges.  Interior edge points will cause problems for the algorithms that
 * generate interior edges (triangle fan).  "On edge" points are arbitrarily
 * determined to be inside or outside the polygon, so results can differ.
 */
```

```
static double    Constraint_Increment = 0.0 ;
```

```
/* default resolutions */
```

```
static int      Grid_Resolution = 20 ;
static int      Trapezoid_Bins = 20 ;
```

```
#define Max(a,b)      (((a)>(b))?(a):(b))
```

```
#define FPRINTF_POLYGON
    fprintf( stderr, "point %g %g\n",
    (float)point[X], (float)point[Y] ) ;
```

```
        fprintf( stderr, "polygon (%d vertices):\n",      \
                numverts ) ;                               \
    for ( n = 0 ; n < numverts ; n++ ) {                  \
        fprintf( stderr, "    %g %g\n",                  \
                (float)pgon[n][X], (float)pgon[n][Y]);    \
    }
```

/\* timing functions \*/

#ifdef TIMER

```
#define START_TIMER( test_id )                               \
    /* do the test a bunch of times to get a useful time reading */ \
    mGetTime( timestart ) ;                                   \
    for ( tcnt = St[test_id].test_times+1 ; --tcnt ; )
```

```
#define STOP_TIMER( test_id )                               \
    mGetTime( timestop ) ;                                   \
    /* time in microseconds */                               \
    St[test_id].time_total +=                                 \
        1000000.0 * (double)(timestop - timestart) /        \
        (double)(HZ * St[test_id].test_times) ;
```

#else

#define START\_TIMER( test\_id )

#define STOP\_TIMER( test\_id )

#endif

```
char    *getenv() ;
void     Usage() ;
void     ScanOpts() ;
void     ConstrainPoint() ;
void     BreakString() ;
#ifdef DISPLAY
void     DisplayPolygon() ;
void     DisplayPoint() ;
#endif
```

/\* test program - see Usage() for command line options \*/

main(argc,argv)

int argc; char \*argv[];

```
{
#ifdef TIMER
register long    tcnt ;
long    timestart ;
long    timestop ;
#endif
```

```
int    i, j, k, n, numverts, inside_flag, inside_tot, numrec ;
double pgon[TOT_VERTS][2], point[2], angle, ran_offset ;
double rangex, rangey, scale, minx, maxx, diffx, miny, maxy, diffy ;
double offx, offy ;
char    str[256], *strplus ;
GridSet grid_set ;
pPlaneSet    p_plane_set ;
pSpackmanSet    p_spackman_set ;
TrapezoidSet    trap_set ;
```

#ifdef CONVEX

```
pPlaneSet    p_ext_set ;
pInclusionAnchor    p_inc_anchor ;
#endif
```

```
SRAN() ;

ScanOpts( argc, argv ) ;

for ( i = 0 ; i < TOT_NUM_TESTS ; i++ ) {
    St[i].time_total = 0.0 ;
    if ( i == ANGLE_TEST ) {
        /* angle test is real slow, so test it fewer times */
        St[i].test_ratio = MACHINE_TEST_RATIO / 10 ;
    } else {
        St[i].test_ratio = MACHINE_TEST_RATIO ;
    }
    St[i].name = TestName[i] ;
    St[i].flag = 0 ;
}

inside_tot = 0 ;

#ifdef CONVEX
    if ( Vertex_Perturbation > 0.0 && Max_Verts > 3 ) {
        fprintf( stderr,
            "warning: vertex perturbation is > 0.0, which is exciting\n");
        fprintf( stderr,
            "    when using convex-only algorithms!\n" ) ;
    }
#endif

    if ( Min_Verts == Max_Verts ) {
        sprintf( str, "\nPolygons with %d vertices, radius %g, "
            "perturbation +/- %g, bounding box scale %g",
            Min_Verts, Vertex_Radius, Vertex_Perturbation, Box_Ratio ) ;
    } else {
        sprintf( str, "\nPolygons with %d to %d vertices, radius %g, "
            "perturbation +/- %g, bounding box scale %g",
            Min_Verts, Max_Verts, Vertex_Radius, Vertex_Perturbation,
            Box_Ratio ) ;
    }
    strplus = &str[strlen(str)] ;
    if ( St[TRAPEZOID_TEST].work ) {
        sprintf( strplus, ", %d trapezoid bins", Trapezoid_Bins ) ;
        strplus = &str[strlen(str)] ;
    }
    if ( St[GRID_TEST].work ) {
        sprintf( strplus, ", %d grid resolution", Grid_Resolution ) ;
        strplus = &str[strlen(str)] ;
    }
#ifdef CONVEX
    sprintf( strplus, ", convex" ) ;
    strplus = &str[strlen(str)] ;
#endif
#ifdef HYBRID
    sprintf( strplus, ", hybrid" ) ;
    strplus = &str[strlen(str)] ;
#endif
#ifdef SORT
    if ( St[PLANE_TEST].work || St[SPACKMAN_TEST].work ) {
        sprintf( strplus, ", using triangles sorted by edge lengths" ) ;
        strplus = &str[strlen(str)] ;
    }
#endif
#ifdef CONVEX
    sprintf( strplus, " and areas" ) ;
```



```
        strplus = &str[strlen(str)] ;
#endif
    }
#endif
#ifdef RANDOM
    if ( St[EXTERIOR_TEST].work ) {
        sprintf( strplus, ", exterior edges' order randomized" ) ;
        strplus = &str[strlen(str)] ;
    }
#endif
    sprintf( strplus, ".\n" ) ;
    strplus = &str[strlen(str)] ;
    BreakString( str ) ;
    printf( "%s", str ) ;

    printf(
        " Testing %d polygons with %d points\n", Test_Polygons, Test_Points ) ;

#ifdef TIMER
    printf( "doing timings" ) ;
    fflush( stdout ) ;
#endif
    for ( i = 0 ; i < Test_Polygons ; i++ ) {

        /* make an arbitrary polygon fitting 0-1 range in x and y */
        numverts = Min_Verts +
            (int)(RAN01() * (double)(Max_Verts-Min_Verts+1)) ;

        /* add a random offset to the angle so that each polygon is not in
         * some favorable (or unfavorable) alignment.
         */
        ran_offset = 2.0 * M_PI * RAN01() ;
        minx = miny = 99999.0 ;
        maxx = maxy = -99999.0 ;
        for ( j = 0 ; j < numverts ; j++ ) {
            angle = 2.0 * M_PI * (double)j / (double)numverts + ran_offset ;
            pgon[j][X] = cos(angle) * Vertex_Radius +
                ( RAN01() * 2.0 - 1.0 ) * Vertex_Perturbation ;
            pgon[j][Y] = sin(angle) * Vertex_Radius +
                ( RAN01() * 2.0 - 1.0 ) * Vertex_Perturbation ;

            ConstrainPoint( pgon[j] ) ;

            if ( pgon[j][X] < minx ) minx = pgon[j][X] ;
            if ( pgon[j][X] > maxx ) maxx = pgon[j][X] ;
            if ( pgon[j][Y] < miny ) miny = pgon[j][Y] ;
            if ( pgon[j][Y] > maxy ) maxy = pgon[j][Y] ;
        }

        offx = ( maxx + minx ) / 2.0 ;
        offy = ( maxy + miny ) / 2.0 ;
        if ( (diffx = maxx - minx ) > ( diffy = maxy - miny ) ) {
            scale = 2.0 / (Box_Ratio * diffx) ;
            rangex = 1.0 ;
            rangey = diffy / diffx ;
        } else {
            scale = 2.0 / (Box_Ratio * diffy) ;
            rangex = diffx / diffy ;
            rangey = 1.0 ;
        }
    }
```

```
for ( j = 0 ; j < numverts ; j++ ) {
    pgon[j][X] = ( pgon[j][X] - offx ) * scale ;
    pgon[j][Y] = ( pgon[j][Y] - offy ) * scale ;
}

/* Set up number of times to test a point against a polygon, for
 * the sake of getting a reasonable timing.  We already know how
 * most of these will perform, so scale their # tests accordingly.
 */
for ( j = 0 ; j < TOT_NUM_TESTS ; j++ ) {
    if ( ( j == GRID_TEST ) || ( j == TRAPEZOID_TEST ) ) {
        St[j].test_times = Max( St[j].test_ratio /
            (int)sqrt((double)numverts), 1 ) ;
    } else {
        St[j].test_times = Max( St[j].test_ratio / numverts, 1 ) ;
    }
}

/* set up tests */
#ifdef CONVEX
if ( St[EXTERIOR_TEST].work ) {
    p_ext_set = ExteriorSetup( pgon, numverts ) ;
}
#endif

if ( St[GRID_TEST].work ) {
    GridSetup( pgon, numverts, Grid_Resolution, &grid_set ) ;
}

#ifdef CONVEX
if ( St[INCLUSION_TEST].work ) {
    p_inc_anchor = InclusionSetup( pgon, numverts ) ;
}
#endif

if ( St[PLANE_TEST].work ) {
    p_plane_set = PlaneSetup( pgon, numverts ) ;
}

if ( St[SPACKMAN_TEST].work ) {
    p_spackman_set = SpackmanSetup( pgon, numverts, &numrec ) ;
}

if ( St[TRAPEZOID_TEST].work ) {
    TrapezoidSetup( pgon, numverts, Trapezoid_Bins, &trap_set ) ;
}

#ifdef DISPLAY
if ( Display_Tests ) {
    DisplayPolygon( pgon, numverts, i ) ;
}
#endif

/* now try # of points against it */
for ( j = 0 ; j < Test_Points ; j++ ) {
    point[X] = RAN01() * rangex * 2.0 - rangex ;
    point[Y] = RAN01() * rangey * 2.0 - rangey ;

    ConstrainPoint( point ) ;
}
```

#ifdef DISPLAY

```
    if ( Display_Tests ) {
        DisplayPoint( point, TRUE ) ;
    }
#endif

    if ( St[ANGLE_TEST].work ) {
        START_TIMER( ANGLE_TEST )
        St[ANGLE_TEST].flag = AngleTest( pgon, numverts, point ) ;
        STOP_TIMER( ANGLE_TEST )
    }
    if ( St[BARYCENTRIC_TEST].work ) {
        START_TIMER( BARYCENTRIC_TEST )
        St[BARYCENTRIC_TEST].flag =
            BarycentricTest( pgon, numverts, point ) ;
        STOP_TIMER( BARYCENTRIC_TEST )
    }
    if ( St[CROSSINGS_TEST].work ) {
        START_TIMER( CROSSINGS_TEST )
        St[CROSSINGS_TEST].flag =
            CrossingsTest( pgon, numverts, point ) ;
        STOP_TIMER( CROSSINGS_TEST )
    }
#ifdef CONVEX
    if ( St[EXTERIOR_TEST].work ) {
        START_TIMER( EXTERIOR_TEST )
        St[EXTERIOR_TEST].flag =
            ExteriorTest( p_ext_set, numverts, point ) ;
        STOP_TIMER( EXTERIOR_TEST )
    }
#endif

    if ( St[GRID_TEST].work ) {
        START_TIMER( GRID_TEST )
        St[GRID_TEST].flag = GridTest( &grid_set, point ) ;
        STOP_TIMER( GRID_TEST )
    }
#ifdef CONVEX
    if ( St[INCLUSION_TEST].work ) {
        START_TIMER( INCLUSION_TEST )
        St[INCLUSION_TEST].flag =
            InclusionTest( p_inc_anchor, point ) ;
        STOP_TIMER( INCLUSION_TEST )
    }
#endif

    if ( St[CROSSMULT_TEST].work ) {
        START_TIMER( CROSSMULT_TEST )
        St[CROSSMULT_TEST].flag = CrossingsMultiplyTest(
            pgon, numverts, point ) ;
        STOP_TIMER( CROSSMULT_TEST )
    }
    if ( St[PLANE_TEST].work ) {
        START_TIMER( PLANE_TEST )
        St[PLANE_TEST].flag =
            PlaneTest( p_plane_set, numverts, point ) ;
        STOP_TIMER( PLANE_TEST )
    }
    if ( St[SPACKMAN_TEST].work ) {
        START_TIMER( SPACKMAN_TEST )
        St[SPACKMAN_TEST].flag =
            SpackmanTest( pgon[0], p_spackman_set, numrec, point ) ;
        STOP_TIMER( SPACKMAN_TEST )
    }
}
```

```
    if ( St[TRAPEZOID_TEST].work ) {
        START_TIMER( TRAPEZOID_TEST )
        St[TRAPEZOID_TEST].flag =
            TrapezoidTest( pgon, numverts, &trap_set, point ) ;
        STOP_TIMER( TRAPEZOID_TEST )
    }
    if ( St[WEILER_TEST].work ) {
        START_TIMER( WEILER_TEST )
        St[WEILER_TEST].flag =
            WeilerTest( pgon, numverts, point ) ;
        STOP_TIMER( WEILER_TEST )
    }
/* +++ add new procedure call here +++ */

/* reality check if crossings test is used */
if ( St[CROSSINGS_TEST].work ) {
    for ( k = 0 ; k < TOT_NUM_TESTS ; k++ ) {
        if ( St[k].work &&
            ( St[k].flag != St[CROSSINGS_TEST].flag ) ) {
            fprintf( stderr,
                "%s test says %s, crossings test says %s\n",
                St[k].name,
                St[k].flag ? "INSIDE" : "OUTSIDE",
                St[CROSSINGS_TEST].flag ? "INSIDE" : "OUTSIDE" ) ;
            FPRINTF_POLYGON ;
        }
    }
}

/* see if any flag is TRUE (i.e. the test point is inside) */
for ( k = 0, inside_flag = 0
    ; k < TOT_NUM_TESTS && !inside_flag
    ; k++ ) {
    inside_flag = St[k].flag ;
}
inside_tot += inside_flag ;

/* turn off highlighting for this point */
#ifdef DISPLAY
    if ( Display_Tests ) {
        DisplayPoint( point, FALSE ) ;
    }
#endif

/* clean up test structures */
#ifdef CONVEX
    if ( St[EXTERIOR_TEST].work ) {
        ExteriorCleanup( p_ext_set ) ;
        p_ext_set = NULL ;
    }
#endif

    if ( St[GRID_TEST].work ) {
        GridCleanup( &grid_set ) ;
    }

#ifdef CONVEX
    if ( St[INCLUSION_TEST].work ) {
        InclusionCleanup( p_inc_anchor ) ;
        p_inc_anchor = NULL ;
    }
}
```

```
    }
#endif

    if ( St[PLANE_TEST].work ) {
        PlaneCleanup( p_plane_set ) ;
        p_plane_set = NULL ;
    }

    if ( St[SPACKMAN_TEST].work ) {
        SpackmanCleanup( p_spackman_set ) ;
        p_spackman_set = NULL ;
    }

    if ( St[TRAPEZOID_TEST].work ) {
        TrapezoidCleanup( &trap_set ) ;
    }

#ifdef TIMER
    /* print a "." every polygon done to give the user a warm feeling */
    printf( "." ) ;
    fflush( stdout ) ;
#endif

    }

    printf( "\n%g %% of all points were inside polygons\n",
        (float)inside_tot * 100.0 / (float)(Test_Points*Test_Polygons) ) ;

#ifdef TIMER
    for ( i = 0 ; i < TOT_NUM_TESTS ; i++ ) {
        if ( St[i].work ) {
            printf( " %s test time: %g microseconds per test\n",
                St[i].name,
                (float)( St[i].time_total/(double)(Test_Points*Test_Polygons) ) ) ;
        }
    }
#endif

    return 0 ;
}

void Usage()
{
    /* +++ add new routine here +++ */
    printf("p_test [options] -{ABCEGIMPSTW}\n");
    printf("  -v minverts [maxverts] = variation in number of polygon vertices\n");
    printf("  -r radius = radius of polygon vertices generated\n");
    printf("  -p perturbation = perturbation of polygon vertices generated\n");
    printf("      These first three determine the type of polygon tested.\n");
    printf("      No perturbation gives regular polygons, while no radius\n");
    printf("      gives random polygons, and a mix gives semi-random polygons\n");
    printf("  -s size = scale of test point box around polygon (1.0 default)\n");
    printf("      A larger size means more points generated outside the\n");
    printf("      polygon. By default test points are in the bounding box.\n");
    printf("  -b bins = number of y bins for trapezoid test\n");
    printf("  -g resolution = grid resolution for grid test\n");
    printf("  -n polygons = number of polygons to test (default %d)\n",
        Test_Polygons);
    printf("  -i points = number of points to test per polygon (default %d)\n",
        Test_Points);
    printf("  -c increment = constrain polygon and test points to grid\n");
    /* +++ add new routine here +++ */
}
```

```
printf("  -{ABCEGIMPSTW} = angle/bary/crossings/exterior/grid/inclusion/cross-mult/\n");
printf("      plane/spackman/trapezoid (bin)/weiler test (default is all)\n");
printf("  -d = display polygons and points using starbase\n");
}

void ScanOpts( argc, argv )
int argc; char *argv[];
{
float    f1 ;
int      i1 ;
int      test_flag = FALSE ;

for ( argc--, argv++; argc > 0; argc--, argv++ ) {
    if ( **argv == '-' ) {
        switch ( *++(*argv) ) {

            case 'v':    /* vertex min & max */
                argv++ ; argc-- ;
                if ( argc && sscanf ( *argv, "%d", &i1 ) == 1 ) {
                    Min_Verts = i1 ;
                    argv++ ; argc-- ;
                    if ( argc && sscanf ( *argv, "%d", &i1 ) == 1 ) {
                        Max_Verts = i1 ;
                    } else {
                        argv-- ; argc++ ;
                        Max_Verts = Min_Verts ;
                    }
                } else {
                    Usage() ;
                    exit(1) ;
                }
                break;

            case 'r':    /* vertex radius */
                argv++ ; argc-- ;
                if ( argc && sscanf ( *argv, "%f", &f1 ) == 1 ) {
                    Vertex_Radius = (double)f1 ;
                } else {
                    Usage() ;
                    exit(1) ;
                }
                break;

            case 'p':    /* vertex perturbation */
                argv++ ; argc-- ;
                if ( argc && sscanf ( *argv, "%f", &f1 ) == 1 ) {
                    Vertex_Perturbation = (double)f1 ;
                } else {
                    Usage() ;
                    exit(1) ;
                }
                break;

            case 's':    /* centered box size ratio - higher is bigger */
                argv++ ; argc-- ;
                if ( argc && sscanf ( *argv, "%f", &f1 ) == 1 ) {
                    Box_Ratio = (double)f1 ;
                    if ( Box_Ratio < 1.0 ) {
                        fprintf(stderr,"warning: ratio is smaller than 1.0\n");
                    }
                } else {

```

```
        Usage() ;
        exit(1) ;
    }
    break;
```

```
case 'b':    /* number of bins for trapezoid test */
    argv++ ; argc-- ;
    if ( argc && sscanf ( *argv, "%d", &i1 ) == 1 ) {
        Trapezoid_Bins = i1 ;
    } else {
        Usage() ;
        exit(1) ;
    }
    break;
```

```
case 'g':    /* grid resolution for grid test */
    argv++ ; argc-- ;
    if ( argc && sscanf ( *argv, "%d", &i1 ) == 1 ) {
        Grid_Resolution = i1 ;
    } else {
        Usage() ;
        exit(1) ;
    }
    break;
```

```
case 'n':    /* number of polygons to test */
    argv++ ; argc-- ;
    if ( argc && sscanf ( *argv, "%d", &i1 ) == 1 ) {
        Test_Polygons = i1 ;
    } else {
        Usage() ;
        exit(1) ;
    }
    break;
```

```
case 'i':    /* number of intersections per polygon to test */
    argv++ ; argc-- ;
    if ( argc && sscanf ( *argv, "%d", &i1 ) == 1 ) {
        Test_Points = i1 ;
    } else {
        Usage() ;
        exit(1) ;
    }
    break;
```

```
case 'c':    /* constrain increment (0 means don't use) */
    argv++ ; argc-- ;
    if ( argc && sscanf ( *argv, "%f", &f1 ) == 1 ) {
        Constraint_Increment = (double)f1 ;
    } else {
        Usage() ;
        exit(1) ;
    }
    break;
```

```
case 'd':    /* display polygon & test points */
```

```
#ifdef DISPLAY
```

```
    Display_Tests = 1 ;
```

```
#else
```

```
    fprintf( stderr,
        "warning: display mode not compiled in - ignored\n" ) ;
```

```
#endif

        break;

/* +++ add new symbol here +++ */
case 'A': /* do tests specified */
case 'B':
case 'C':
case 'E':
case 'G':
case 'I':
case 'M':
case 'P':
case 'S':
case 'T':
case 'W':
    test_flag = TRUE ;
    if ( strchr( *argv, 'A' ) ) {
        St[ANGLE_TEST].work = 1 ;
    }
    if ( strchr( *argv, 'B' ) ) {
        St[BARYCENTRIC_TEST].work = 1 ;
    }
    if ( strchr( *argv, 'C' ) ) {
        St[CROSSINGS_TEST].work = 1 ;
    }
    if ( strchr( *argv, 'E' ) ) {
#ifdef CONVEX
        St[EXTERIOR_TEST].work = 1 ;
    }
    else
        fprintf( stderr,
            "warning: exterior test for -DCONVEX only - ignored\n" ) ;
    }
    if ( strchr( *argv, 'G' ) ) {
        St[GRID_TEST].work = 1 ;
    }
    if ( strchr( *argv, 'I' ) ) {
#ifdef CONVEX
        St[INCLUSION_TEST].work = 1 ;
    }
    else
        fprintf( stderr,
            "warning: inclusion test for -DCONVEX only - ignored\n" ) ;
    }
    if ( strchr( *argv, 'M' ) ) {
        St[CROSSMULT_TEST].work = 1 ;
    }
    if ( strchr( *argv, 'P' ) ) {
        St[PLANE_TEST].work = 1 ;
    }
    if ( strchr( *argv, 'S' ) ) {
        St[SPACKMAN_TEST].work = 1 ;
    }
    if ( strchr( *argv, 'T' ) ) {
        St[TRAPEZOID_TEST].work = 1 ;
    }
    if ( strchr( *argv, 'W' ) ) {
        St[WEILER_TEST].work = 1 ;
    }
    /* +++ add new symbol test here +++ */
    break;
```



```
        default:
            Usage() ;
            exit(1) ;
            break ;
    }

    } else {
        Usage() ;
        exit(1) ;
    }
}

if ( !test_flag ) {
    fprintf( stderr,
        "error: no point in polygon tests were specified, e.g. -PCS\n" ) ;
    Usage() ;
    exit(1) ;
}

}

void ConstrainPoint( pt )
double *pt ;
{
    double val ;

    if ( Constraint_Increment > 0.0 ) {
        pt[X] -=
            ( val = fmod( pt[X], Constraint_Increment ) ) ;
        if ( fabs(val) > Constraint_Increment * 0.5 ) {
            pt[X] += (val > 0.0) ? Constraint_Increment :
                -Constraint_Increment ;
        }
        pt[Y] -=
            ( val = fmod( pt[Y], Constraint_Increment ) ) ;
        if ( fabs(val) > Constraint_Increment * 0.5 ) {
            pt[Y] += (val > 0.0) ? Constraint_Increment :
                -Constraint_Increment ;
        }
    }
}

/* break long strings into 80 or less character output.  Not foolproof, but
 * good enough.
 */
void BreakString( str )
char *str ;
{
    int length, i, last_space, col ;

    length = strlen( str ) ;
    last_space = 0 ;
    col = 0 ;
    for ( i = 0 ; i < length ; i++ ) {
        if ( str[i] == ' ' ) {
            last_space = i ;
        }
        if ( col == 79 ) {
            str[last_space] = '\n' ;
            col = i - last_space ;
            last_space = 0 ;
        }
    }
}
```

```
        } else {
            col++ ;
        }
    }
}

#ifdef DISPLAY
/* ===== display routines ===== */
/* Currently for HP Starbase - pretty easy to modify */
void DisplayPolygon( pgon, numverts, id )
double pgon[][2] ;
int numverts ;
{
    static int init_flag = 0 ;
    int i ;
    char str[256] ;

    if ( !init_flag ) {
        init_flag = 1 ;
        /* make things big enough to avoid clipping */
        Display_Scale = 0.45 / ( Vertex_Radius + Vertex_Perturbation ) ;
        Display_OffsetX = Display_Scale + 0.05 ;
        Display_OffsetY = Display_Scale + 0.10 ;

        Fd = DevOpen(OUTDEV,INIT) ;
        shade_mode( Fd, CMAP_FULL|INIT, 0 ) ;
        background_color( Fd, 0.1, 0.2, 0.4 ) ;
        line_color( Fd, 1.0, 0.9, 0.7 ) ;
        text_color( Fd, 1.0, 1.0, 0.2 ) ;
        character_height( Fd, 0.08 ) ;
        marker_type( Fd, 3 ) ;
    }
    clear_view_surface( Fd ) ;

    move2d( Fd,
        (float)( pgon[numverts-1][X] * Display_Scale + Display_OffsetX ),
        (float)( pgon[numverts-1][Y] * Display_Scale + Display_OffsetY ) ) ;
    for ( i = 0 ; i < numverts ; i++ ) {
        draw2d( Fd,
            (float)( pgon[i][X] * Display_Scale + Display_OffsetX ),
            (float)( pgon[i][Y] * Display_Scale + Display_OffsetY ) ) ;
    }

    sprintf( str, "%4d sides, %3d of %d\n", numverts, id+1, Test_Polygons ) ;
    text2d( Fd, 0.01, 0.01, str, VDC_TEXT, 0 ) ;
    flush_buffer( Fd ) ;
}

void DisplayPoint( point, hilit )
double point[2] ;
int hilit ;
{
    float clist[2] ;

    if ( hilit ) {
        marker_color( Fd, 1.0, 0.0, 0.0 ) ;
    } else {
        marker_color( Fd, 0.2, 1.0, 1.0 ) ;
    }

    clist[0] = (float)( point[0] * Display_Scale + Display_OffsetX ) ;
```

```
        clist[1] = (float)( point[1] * Display_Scale + Display_OffsetY ) ;
        polymarker2d( Fd, clist, 1, 0 ) ;
        flush_buffer( Fd ) ;
    }

int DevOpen(dev_kind,init_mode)
int dev_kind,init_mode;
{
    char      *dev, *driver;
    int       fildes ;

    if ( dev_kind == OUTDEV ) {
        dev = getenv("OUTINDEV");
        if (!dev) dev = getenv("OUTDEV");
        if (!dev) dev = "/dev/crt" ;
        driver = getenv("OUTDRIVER");
        if (!driver) driver = "hp98731" ;
    } else {
        dev = getenv("OUTINDEV");
        if (!dev) dev = getenv("INDEV");
        if (!dev) dev = "/dev/hil2" ;
        driver = getenv("INDRIVER");
        if (!driver) driver = "hp-hil" ;
    }

    /* driver?  we don't need no stinking driver... */
    fildes = gopen(dev,dev_kind,NULL,init_mode);

    return(fildes) ;
}
#endif
```

/\* ptinpoly.c - point in polygon inside/outside code.

by Eric Haines, 3D/Eye Inc, erich@eye.com

This code contains the following algorithms:

- crossings - count the crossing made by a ray from the test point
- crossings-multiply - as above, but avoids a division; often a bit faster
- angle summation - sum the angle formed by point and vertex pairs
- weiler angle summation - sum the angles using quad movements
- half-plane testing - test triangle fan using half-space planes
- barycentric coordinates - test triangle fan w/barycentric coords
- spackman barycentric - preprocessed barycentric coordinates
- trapezoid testing - bin sorting algorithm
- grid testing - grid imposed on polygon
- exterior test - for convex polygons, check exterior of polygon
- inclusion test - for convex polygons, use binary search for edge.

\*/

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "ptinpoly.h"
```

```
#define X      0
#define Y      1
```

```
#ifndef TRUE
#define TRUE    1
#define FALSE  0
#endif
```

```
#ifndef HUGE
#define HUGE    1.797693134862315e+308
#endif
```

```
#ifndef M_PI
#define M_PI    3.14159265358979323846
#endif
```

```
/* test if a & b are within epsilon.  Favors cases where a < b */
#define Near(a,b,eps)  ( ((b)-(eps)<(a)) && ((a)-(eps)<(b)) )
```

```
#define MALLOC_CHECK( a )      if ( !(a) ) { \
                                fprintf( stderr, "out of memory\n" ) ; \
                                exit(1) ; \
                                }
```

```
/* ===== Crossings algorithm ===== */
```

```
/* Shoot a test ray along +X axis.  The strategy, from MacMartin, is to
 * compare vertex Y values to the testing point's Y and quickly discard
 * edges which are entirely to one side of the test ray.
 *
 * Input 2D polygon _pgon_ with _numverts_ number of vertices and test point
 * _point_, returns 1 if inside, 0 if outside.  WINDING and CONVEX can be
 * defined for this test.
 */
```

```
int CrossingsTest( pgon, numverts, point )
double pgon[][2] ;
int      numverts ;
```

```
double point[2] ;
{
#ifdef WINDING
register int crossings ;
#endif
register int j, yflag0, yflag1, inside_flag, xflag0 ;
register double ty, tx, *vtx0, *vtx1 ;
#ifdef CONVEX
register int line_flag ;
#endif

    tx = point[X] ;
    ty = point[Y] ;

    vtx0 = pgon[numverts-1] ;
    /* get test bit for above/below X axis */
    yflag0 = ( vtx0[Y] >= ty ) ;
    vtx1 = pgon[0] ;

#ifdef WINDING
    crossings = 0 ;
#else
    inside_flag = 0 ;
#endif
#ifdef CONVEX
    line_flag = 0 ;
#endif
    for ( j = numverts+1 ; --j ; ) {

        yflag1 = ( vtx1[Y] >= ty ) ;
        /* check if endpoints straddle (are on opposite sides) of X axis
         * (i.e. the Y's differ); if so, +X ray could intersect this edge.
         */
        if ( yflag0 != yflag1 ) {
            xflag0 = ( vtx0[X] >= tx ) ;
            /* check if endpoints are on same side of the Y axis (i.e. X's
             * are the same); if so, it's easy to test if edge hits or misses.
             */
            if ( xflag0 == ( vtx1[X] >= tx ) ) {

                /* if edge's X values both right of the point, must hit */
#ifdef WINDING
                if ( xflag0 ) crossings += ( yflag0 ? -1 : 1 ) ;
#else
                if ( xflag0 ) inside_flag = !inside_flag ;
#endif
            } else {
                /* compute intersection of pgon segment with +X ray, note
                 * if >= point's X; if so, the ray hits it.
                 */
                if ( (vtx1[X] - (vtx1[Y]-ty)*
                    ( vtx0[X]-vtx1[X])/(vtx0[Y]-vtx1[Y])) >= tx ) {
#ifdef WINDING
                    crossings += ( yflag0 ? -1 : 1 ) ;
#else
                    inside_flag = !inside_flag ;
#endif
                }
            }
        }
    }
#ifdef CONVEX
    /* if this is second edge hit, then done testing */

```

```
        if ( line_flag ) goto Exit ;

        /* note that one edge has been hit by the ray's line */
        line_flag = TRUE ;
#endif
    }

    /* move to next pair of vertices, retaining info as possible */
    yflag0 = yflag1 ;
    vtx0 = vtx1 ;
    vtx1 += 2 ;
}
#endif CONVEX
    Exit: ;
#endif
#endif WINDING
    /* test if crossings is not zero */
    inside_flag = (crossings != 0) ;
#endif

    return( inside_flag ) ;
}

/* ===== Angle summation algorithm ===== */

/* Sum angles made by (vtxN to test point to vtxN+1), check if in proper
 * range to be inside.  VERY SLOW, included for tutorial reasons (i.e.
 * to show why one should never use this algorithm).
 *
 * Input 2D polygon _pgon_ with _numverts_ number of vertices and test point
 * _point_, returns 1 if inside, 0 if outside.
 */
int AngleTest( pgon, numverts, point )
double  pgon[][2] ;
int      numverts ;
double  point[2] ;
{
    register double *vtx0, *vtx1, angle, len, vec0[2], vec1[2], vec_dot ;
    register int      j ;
    int      inside_flag ;

    /* sum the angles and see if answer mod 2*PI > PI */
    vtx0 = pgon[numverts-1] ;
    vec0[X] = vtx0[X] - point[X] ;
    vec0[Y] = vtx0[Y] - point[Y] ;
    len = sqrt( vec0[X] * vec0[X] + vec0[Y] * vec0[Y] ) ;
    if ( len <= 0.0 ) {
        /* point and vertex coincide */
        return( 1 ) ;
    }
    vec0[X] /= len ;
    vec0[Y] /= len ;

    angle = 0.0 ;
    for ( j = 0 ; j < numverts ; j++ ) {
        vtx1 = pgon[j] ;
        vec1[X] = vtx1[X] - point[X] ;
        vec1[Y] = vtx1[Y] - point[Y] ;
        len = sqrt( vec1[X] * vec1[X] + vec1[Y] * vec1[Y] ) ;
        if ( len <= 0.0 ) {
            /* point and vertex coincide */

```

```

        return( 1 ) ;
    }
    vec1[X] /= len ;
    vec1[Y] /= len ;

    /* check if vec1 is to "left" or "right" of vec0 */
    vec_dot = vec0[X] * vec1[X] + vec0[Y] * vec1[Y] ;
    if ( vec_dot < -1.0 ) {
        /* point is on edge, so always add 180 degrees.  always
        * adding is not necessarily the right answer for
        * concave polygons and subtractive triangles.
        */
        angle += M_PI ;
    } else if ( vec_dot < 1.0 ) {
        if ( vec0[X] * vec1[Y] - vec1[X] * vec0[Y] >= 0.0 ) {
            /* add angle due to dot product of vectors */
            angle += acos( vec_dot ) ;
        } else {
            /* subtract angle due to dot product of vectors */
            angle -= acos( vec_dot ) ;
        }
    } /* if vec_dot >= 1.0, angle does not change */

    /* get to next point */
    vtx0 = vtx1 ;
    vec0[X] = vec1[X] ;
    vec0[Y] = vec1[Y] ;
}
/* test if between PI and 3*PI, 5*PI and 7*PI, etc */
inside_flag = fmod( fabs(angle) + M_PI, 4.0*M_PI ) > 2.0*M_PI ;

return( inside_flag ) ;
}

/* ===== Weiler algorithm ===== */

/* Track quadrant movements using Weiler's algorithm (elsewhere in Graphics
 * Gems IV).  Algorithm has been optimized for testing purposes, but the
 * crossings algorithm is still faster.  Included to provide timings.
 *
 * Input 2D polygon _pgon_ with _numverts_ number of vertices and test point
 * _point_, returns 1 if inside, 0 if outside.  WINDING can be defined for
 * this test.
 */

#define QUADRANT( vtx, x, y ) \
    (((vtx)[X]>(x)) ? ( ((vtx)[Y]>(y)) ? 0:3 ) : ( ((vtx)[Y]>(y)) ? 1:2 ))

#define X_INTERCEPT( v0, v1, y ) \
    ( (((v1)[X]-(v0)[X])/((v1)[Y]-(v0)[Y])) * ((y)-(v0)[Y]) + (v0)[X] )

int WeilerTest( pgon, numverts, point )
double pgon[][2] ;
int numverts ;
double point[2] ;
{
    register int angle, qd, next_qd, delta, j ;
    register double ty, tx, *vtx0, *vtx1 ;
    int inside_flag ;

    tx = point[X] ;

```

```
ty = point[Y] ;

vtx0 = pgon[numverts-1] ;
qd = QUADRANT( vtx0, tx, ty ) ;
angle = 0 ;

vtx1 = pgon[0] ;

for ( j = numverts+1 ; --j ; ) {
    /* calculate quadrant and delta from last quadrant */
    next_qd = QUADRANT( vtx1, tx, ty ) ;
    delta = next_qd - qd ;

    /* adjust delta and add it to total angle sum */
    switch ( delta ) {
        case 0:      /* do nothing */
            break ;
        case -1:
        case 3:
            angle-- ;
            qd = next_qd ;
            break ;

        case 1:
        case -3:
            angle++ ;
            qd = next_qd ;
            break ;

        case 2:
        case -2:
            if (X_INTERCEPT( vtx0, vtx1, ty ) > tx ) {
                angle -= delta ;
            } else {
                angle += delta ;
            }
            qd = next_qd ;
            break ;
    }

    /* increment for next step */
    vtx0 = vtx1 ;
    vtx1 += 2 ;
}

#ifdef WINDING
    /* simple windings test:  if angle != 0, then point is inside */
    inside_flag = ( angle != 0 ) ;
#else
    /* Jordan test:  if angle is +-4, 12, 20, etc then point is inside */
    inside_flag = ( (angle/4) & 0x1 ) ;
#endif

return (inside_flag) ;
}

#undef QUADRANT
#undef Y_INTERCEPT

/* ===== Triangle half-plane algorithm ===== */
```



```
/* Split the polygon into a fan of triangles and for each triangle test if
 * the point is inside of the three half-planes formed by the triangle's edges.
 *
 * Call setup with 2D polygon _pgon_ with _numverts_ number of vertices,
 * which returns a pointer to a plane set array.
 * Call testing procedure with a pointer to this array, _numverts_, and
 * test point _point_, returns 1 if inside, 0 if outside.
 * Call cleanup with pointer to plane set array to free space.
 *
 * SORT and CONVEX can be defined for this test.
 */
```

```
/* split polygons along set of x axes - call preprocess once */
pPlaneSet      PlaneSetup( pgon, numverts )
double  pgon[][2] ;
int      numverts ;
{
    int      i, p1, p2 ;
    double  tx, ty, vx0, vy0 ;
    pPlaneSet      pps, pps_return ;
#ifdef  SORT
    double  len[3], len_temp ;
    int      j ;
    PlaneSet      ps_temp ;
#ifdef  CONVEX
    pPlaneSet      pps_new ;
    pSizePlanePair p_size_pair ;
#endif
#endif

    pps = pps_return =
        (pPlaneSet)malloc( 3 * (numverts-2) * sizeof( PlaneSet ) ) ;
    MALLOC_CHECK( pps ) ;
#ifdef  CONVEX
#ifdef  SORT
    p_size_pair =
        (pSizePlanePair)malloc( (numverts-2) * sizeof( SizePlanePair ) ) ;
    MALLOC_CHECK( p_size_pair ) ;
#endif
#endif
#ifdef  SORT
    len[0] = pps->vx * pps->vx + pps->vy * pps->vy ;
#endif
#ifdef  CONVEX
#ifdef  HYBRID
    pps->ext_flag = ( p1 == 1 ) ;
#endif
#endif

    /* Sort triangles by areas, so compute (twice) the area here */
    p_size_pair[p1-1].pps = pps ;
    p_size_pair[p1-1].size =
        ( pgon[0][X] * pgon[p1][Y] ) +
        ( pgon[p1][X] * pgon[p2][Y] ) +
        ( pgon[p2][X] * pgon[0][Y] ) -
        ( pgon[p1][X] * pgon[0][Y] ) -
```

```

                ( pgon[p2][X] * pgon[p1][Y] ) -
                ( pgon[0][X] * pgon[p2][Y] ) ;
#endif
#endif

    pps++ ;
    pps->vx = pgon[p1][Y] - pgon[p2][Y] ;
    pps->vy = pgon[p2][X] - pgon[p1][X] ;
    pps->c = pps->vx * pgon[p1][X] + pps->vy * pgon[p1][Y] ;
#ifdef SORT
    len[1] = pps->vx * pps->vx + pps->vy * pps->vy ;
#endif
#ifdef CONVEX
#ifdef HYBRID
    pps->ext_flag = TRUE ;
#endif
#endif
#endif

    pps++ ;
    pps->vx = pgon[p2][Y] - vy0 ;
    pps->vy = vx0 - pgon[p2][X] ;
    pps->c = pps->vx * pgon[p2][X] + pps->vy * pgon[p2][Y] ;
#ifdef SORT
    len[2] = pps->vx * pps->vx + pps->vy * pps->vy ;
#endif
#ifdef CONVEX
#ifdef HYBRID
    pps->ext_flag = ( p2 == numverts-1 ) ;
#endif
#endif
#endif

    /* find an average point which must be inside of the triangle */
    tx = ( vx0 + pgon[p1][X] + pgon[p2][X] ) / 3.0 ;
    ty = ( vy0 + pgon[p1][Y] + pgon[p2][Y] ) / 3.0 ;

    /* check sense and reverse if test point is not thought to be inside
    * first triangle
    */
    if ( pps->vx * tx + pps->vy * ty >= pps->c ) {
        /* back up to start of plane set */
        pps -= 2 ;
        /* point is thought to be outside, so reverse sense of edge
        * normals so that it is correctly considered inside.
        */
        for ( i = 0 ; i < 3 ; i++ ) {
            pps->vx = -pps->vx ;
            pps->vy = -pps->vy ;
            pps->c = -pps->c ;
            pps++ ;
        }
    } else {
        pps++ ;
    }
}

#ifdef SORT
/* sort the planes based on the edge lengths */
pps -= 3 ;
for ( i = 0 ; i < 2 ; i++ ) {
    for ( j = i+1 ; j < 3 ; j++ ) {
        if ( len[i] < len[j] ) {
            ps_temp = pps[i] ;
            pps[i] = pps[j] ;
            pps[j] = ps_temp ;
        }
    }
}
#endif
```

```

        len_temp = len[i] ;
        len[i] = len[j] ;
        len[j] = len_temp ;
    }
}
}
pps += 3 ;
#endif
}

#ifdef CONVEX
#ifdef SORT
/* sort the triangles based on their areas */
qsort( p_size_pair, numverts-2,
        sizeof( SizePlanePair ), CompareSizePlanePairs ) ;

/* make the plane sets match the sorted order */
for ( i = 0, pps = pps_return
      ; i < numverts-2
      ; i++ ) {

    pps_new = p_size_pair[i].pps ;
    for ( j = 0 ; j < 3 ; j++, pps++, pps_new++ ) {
        ps_temp = *pps ;
        *pps = *pps_new ;
        *pps_new = ps_temp ;
    }
}
free( p_size_pair ) ;
#endif
#endif

return( pps_return ) ;
}

#ifdef CONVEX
#ifdef SORT
int CompareSizePlanePairs( p_sp0, p_sp1 )
pSizePlanePair p_sp0, p_sp1 ;
{
    if ( p_sp0->size == p_sp1->size ) {
        return( 0 ) ;
    } else {
        return( p_sp0->size > p_sp1->size ? -1 : 1 ) ;
    }
}
#endif
#endif

/* check point for inside of three "planes" formed by triangle edges */
int PlaneTest( p_plane_set, numverts, point )
pPlaneSet p_plane_set ;
int numverts ;
double point[2] ;
{
    register pPlaneSet ps ;
    register int p2 ;
#ifdef CONVEX
    register int inside_flag ;
#endif

```

```
register double tx, ty ;

    tx = point[X] ;
    ty = point[Y] ;

#ifdef CONVEX
    inside_flag = 0 ;
#endif

    for ( ps = p_plane_set, p2 = numverts-1 ; --p2 ; ) {

        if ( ps->vx * tx + ps->vy * ty < ps->c ) {
            ps++ ;
            if ( ps->vx * tx + ps->vy * ty < ps->c ) {
                ps++ ;
                /* note: we make the third edge have a slightly different
                 * equality condition, since this third edge is in fact
                 * the next triangle's first edge.  Not fool-proof, but
                 * it doesn't hurt (better would be to keep track of the
                 * triangle's area sign so we would know which kind of
                 * triangle this is).  Note that edge sorting nullifies
                 * this special inequality, too.
                 */
                if ( ps->vx * tx + ps->vy * ty <= ps->c ) {
                    /* point is inside polygon */
#ifdef CONVEX
                    return( 1 ) ;
#else
                    inside_flag = !inside_flag ;
#endif
                }
            }
        }

        ps++ ;
    } else {
#ifdef CONVEX
#ifdef HYBRID
        /* check if outside exterior edge */
        else if ( ps->ext_flag ) return( 0 ) ;
#endif
#endif

        ps++ ;
    } else {
#ifdef CONVEX
#ifdef HYBRID
        /* check if outside exterior edge */
        if ( ps->ext_flag ) return( 0 ) ;
#endif
#endif

        /* get past last two plane tests */
        ps += 2 ;
    }
    } else {
#ifdef CONVEX
#ifdef HYBRID
        /* check if outside exterior edge */
        if ( ps->ext_flag ) return( 0 ) ;
#endif
#endif

        /* get past all three plane tests */
        ps += 3 ;
    }
    }
}

#ifdef CONVEX
```

```
    /* for convex, if we make it to here, all triangles were missed */
    return( 0 ) ;
#else
    return( inside_flag ) ;
#endif
}

void PlaneCleanup( p_plane_set )
pPlaneSet      p_plane_set ;
{
    free( p_plane_set ) ;
}

/* ===== Barycentric algorithm ===== */

/* Split the polygon into a fan of triangles and for each triangle test if
 * the point has barycentric coordinates which are inside the triangle.
 * Similar to Badouel's code in Graphics Gems, with a little more efficient
 * coding.
 *
 * Input 2D polygon _pgon_ with _numverts_ number of vertices and test point
 * _point_, returns 1 if inside, 0 if outside.
 */
int BarycentricTest( pgon, numverts, point )
double pgon[][2] ;
int     numverts ;
double point[2] ;
{
    register double *pg1, *pg2, *pgend ;
    register double tx, ty, u0, u1, u2, v0, v1, vx0, vy0, alpha, beta, denom ;
    int     inside_flag ;

    tx = point[X] ;
    ty = point[Y] ;
    vx0 = pgon[0][X] ;
    vy0 = pgon[0][Y] ;
    u0 = tx - vx0 ;
    v0 = ty - vy0 ;

    inside_flag = 0 ;
    pgend = pgon[numverts-1] ;
    for ( pg1 = pgon[1], pg2 = pgon[2] ; pg1 != pgend ; pg1+=2, pg2+=2 ) {

        u1 = pg1[X] - vx0 ;
        if ( u1 == 0.0 ) {

            /* 0 and 1 vertices have same X value */

            /* zero area test - can be removed for convex testing */
            u2 = pg2[X] - vx0 ;
            if ( ( u2 == 0.0 ) ||

                /* compute beta and check bounds */
                /* we use "<= 0.0" so that points on the shared interior
                 * edge will (generally) be inside only one polygon.
                 */
                ( ( beta = u0 / u2 ) <= 0.0 ) ||
                ( beta > 1.0 ) ||

                /* zero area test - remove for convex testing */
                ( ( v1 = pg1[Y] - vy0 ) == 0.0 ) ||
```

```

        /* compute alpha and check bounds */
        ( ( alpha = ( v0 - beta *
            ( pg2[Y] - vy0 ) ) / v1 ) < 0.0 ) ) {

        /* whew! missed! */
        goto NextTri ;
    }

} else {
    /* 0 and 1 vertices have different X value */

    /* compute denom and check for zero area triangle - check
    * is not needed for convex polygon testing
    */
    u2 = pg2[X] - vx0 ;
    v1 = pg1[Y] - vy0 ;
    denom = ( pg2[Y] - vy0 ) * u1 - u2 * v1 ;
    if ( ( denom == 0.0 ) ||

        /* compute beta and check bounds */
        /* we use "<= 0.0" so that points on the shared interior
        * edge will (generally) be inside only one polygon.
        */
        ( ( beta = ( v0 * u1 - u0 * v1 ) / denom ) <= 0.0 ) ||
        ( beta > 1.0 ) ||

        /* compute alpha & check bounds */
        ( ( alpha = ( u0 - beta * u2 ) / u1 ) < 0.0 ) ) {

        /* whew! missed! */
        goto NextTri ;
    }

}

/* check gamma */
if ( alpha + beta <= 1.0 ) {
    /* survived */
    inside_flag = !inside_flag ;
}

NextTri: ;
}
return( inside_flag ) ;
}

/* ===== Barycentric precompute (Spackman) algorithm ===== */

/* Split the polygon into a fan of triangles and for each triangle test if
* the point has barycentric coordinates which are inside the triangle.
* Use Spackman's normalization method to precompute various parameters.
*
* Call setup with 2D polygon _pgon_ with _numverts_ number of vertices,
* which returns a pointer to the array of the parameters records and the
* number of parameter records created.
* Call testing procedure with the first vertex in the polygon _pgon[0]_,
* a pointer to this array, the number of parameter records, and test point
* _point_, returns 1 if inside, 0 if outside.
* Call cleanup with pointer to parameter record array to free space.
*
* SORT can be defined for this test.

```

```
* (CONVEX could be added: see PlaneSetup and PlaneTest for method)
*/
pSpackmanSet    SpackmanSetup( pgon, numverts, p_numrec )
double  pgon[][2] ;
int     numverts ;
int     *p_numrec ;
{
    int     p1, p2, degen ;
    double  denom, u1, v1, *pv[3] ;
    pSpackmanSet    pss, pss_return ;
#ifdef  SORT
    double  u[2], v[2], len[2], *pv_temp ;
#endif

    pss = pss_return =
        (pSpackmanSet)malloc( (numverts-2) * sizeof( SpackmanSet ) ) ;
    MALLOC_CHECK( pss ) ;

    degen = 0 ;

    for ( p1 = 1, p2 = 2 ; p2 < numverts ; p1++, p2++ ) {

        pv[0] = pgon[0] ;
        pv[1] = pgon[p1] ;
        pv[2] = pgon[p2] ;

#ifdef  SORT
        /* Note that sorting can cause a mismatch of alpha/beta inequality
         * tests.  In other words, test points on an interior line between
         * test triangles will often then be wrong.
         */
        u[0] = pv[1][X] - pv[0][X] ;
        u[1] = pv[2][X] - pv[0][X] ;
        v[0] = pv[1][Y] - pv[0][Y] ;
        v[1] = pv[2][Y] - pv[0][Y] ;
        len[0] = u[0] * u[0] + v[0] * v[0] ;
        len[1] = u[1] * u[1] + v[1] * v[1] ;

        /* compare two edges touching anchor point and put longest first */
        /* we don't sort all three edges because the anchor point and
         * values computed from it gets used for all triangles in the fan.
         */
        if ( len[0] < len[1] ) {
            pv_temp = pv[1] ;
            pv[1] = pv[2] ;
            pv[2] = pv_temp ;
        }
#endif

        u1 = pv[1][X] - pv[0][X] ;
        pss->u2 = pv[2][X] - pv[0][X] ;
        v1 = pv[1][Y] - pv[0][Y] ;
        pss->v2 = pv[2][Y] - pv[0][Y] ;
        pss->u1_nonzero = !( u1 == 0.0 ) ;
        if ( pss->u1_nonzero ) {
            /* not zero, so compute inverse */
            pss->inv_u1 = 1.0 / u1 ;
            denom = pss->v2 * u1 - pss->u2 * v1 ;
            if ( denom == 0.0 ) {
                /* degenerate triangle, ignore it */
                degen++ ;
            }
        }
    }
}

#endif
```

```
        goto Skip ;
    } else {
        pss->ulp = u1 / denom ;
        pss->vlp = v1 / denom ;
    }
} else {
    if ( pss->u2 == 0.0 ) {
        /* degenerate triangle, ignore it */
        degen++ ;
        goto Skip ;
    } else {
        /* not zero, so compute inverse */
        pss->inv_u2 = 1.0 / pss->u2 ;
        if ( v1 == 0.0 ) {
            /* degenerate triangle, ignore it */
            degen++ ;
            goto Skip ;
        } else {
            pss->inv_v1 = 1.0 / v1 ;
        }
    }
}

pss++ ;
Skip: ;
}

/* number of Spackman records */
*p_numrec = numverts - degen - 2 ;
if ( degen ) {
    pss = pss_return =
        (pSpackmanSet)realloc( pss_return,
                               (numverts-2-degen) * sizeof( SpackmanSet )) ;
}

return( pss_return ) ;
}

/* barycentric, a la Gems I and Spackman's normalization precompute */
int SpackmanTest( anchor, p_spackman_set, numrec, point )
double anchor[2] ;
pSpackmanSet p_spackman_set ;
int numrec ;
double point[2] ;
{
    register pSpackmanSet pss ;
    register int inside_flag ;
    register int nr ;
    register double tx, ty, vx0, vy0, u0, v0, alpha, beta ;

    tx = point[X] ;
    ty = point[Y] ;
    /* note that we really need only the first vertex of the polygon,
     * so do not really need to keep the whole polygon around.
     */
    vx0 = anchor[X] ;
    vy0 = anchor[Y] ;
    u0 = tx - vx0 ;
    v0 = ty - vy0 ;

    inside_flag = 0 ;
```



```

for ( pss = p_spackman_set, nr = numrec+1 ; --nr ; pss++ ) {

    if ( pss->ul_nonzero ) {
        /* 0 and 2 vertices have different X value */

        /* compute beta and check bounds */
        /* we use "<= 0.0" so that points on the shared interior edge
         * will (generally) be inside only one polygon.
         */
        beta = ( v0 * pss->ulp - u0 * pss->vlp ) ;
        if ( ( beta <= 0.0 ) || ( beta > 1.0 ) ||

            /* compute alpha & check bounds */
            ( ( alpha = ( u0 - beta * pss->u2 ) * pss->inv_u1 )
              < 0.0 ) ) {

            /* whew! missed! */
            goto NextTri ;
        }
    } else {
        /* 0 and 2 vertices have same X value */

        /* compute beta and check bounds */
        /* we use "<= 0.0" so that points on the shared interior edge
         * will (generally) be inside only one polygon.
         */
        beta = u0 * pss->inv_u2 ;
        if ( ( beta <= 0.0 ) || ( beta >= 1.0 ) ||

            /* compute alpha and check bounds */
            ( ( alpha = ( v0 - beta * pss->v2 ) * pss->inv_v1 )
              < 0.0 ) ) {

            /* whew! missed! */
            goto NextTri ;
        }
    }

    /* check gamma */
    if ( alpha + beta <= 1.0 ) {
        /* survived */
        inside_flag = !inside_flag ;
    }

    NextTri: ;
}

```

```

return( inside_flag ) ;

```

```

void SpackmanCleanup( p_spackman_set )
pSpackmanSet      p_spackman_set ;
{
    free( p_spackman_set ) ;
}

```

```

/* ===== Trapezoid (bin) algorithm ===== */

```

```

/* Split polygons along set of y bins and sorts the edge fragments.  Testing
 * is done against these fragments.

```

```
*
* Call setup with 2D polygon _pgon_ with _numverts_ number of vertices, the
* number of bins desired _bins_, and a pointer to a trapezoid structure
* _p_trap_set_.
* Call testing procedure with 2D polygon _pgon_ with _numverts_ number of
* vertices, _p_trap_set_ pointer to trapezoid structure, and test point
* _point_, returns 1 if inside, 0 if outside.
* Call cleanup with pointer to trapezoid structure to free space.
*/
```

```
void TrapezoidSetup( pgon, numverts, bins, p_trap_set )
double  pgon[][2] ;
int      numverts ;
int      bins ;
pTrapezoidSet  p_trap_set ;
{
double  *vtx0, *vtx1, *vtxa, *vtxb, slope ;
int      i, j, bin_tot[TOT_BINS], ba, bb, id, full_cross, count ;
double  fba, fbb, vx0, vx1, dy, vy0 ;

    p_trap_set->bins = bins ;
    p_trap_set->trapz = (pTrapezoid)malloc( p_trap_set->bins *
        sizeof(Trapezoid)) ;
    MALLOC_CHECK( p_trap_set->trapz ) ;

    p_trap_set->minx =
    p_trap_set->maxx = pgon[0][X] ;
    p_trap_set->miny =
    p_trap_set->maxy = pgon[0][Y] ;

    for ( i = 1 ; i < numverts ; i++ ) {
        if ( p_trap_set->minx > (vx0 = pgon[i][X]) ) {
            p_trap_set->minx = vx0 ;
        } else if ( p_trap_set->maxx < vx0 ) {
            p_trap_set->maxx = vx0 ;
        }

        if ( p_trap_set->miny > (vy0 = pgon[i][Y]) ) {
            p_trap_set->miny = vy0 ;
        } else if ( p_trap_set->maxy < vy0 ) {
            p_trap_set->maxy = vy0 ;
        }
    }

    /* add a little to the bounds to ensure everything falls inside area */
    p_trap_set->miny -= EPSILON * (p_trap_set->maxy-p_trap_set->miny) ;
    p_trap_set->maxy += EPSILON * (p_trap_set->maxy-p_trap_set->miny) ;

    p_trap_set->ydelta =
        (p_trap_set->maxy-p_trap_set->miny) / (double)p_trap_set->bins ;
    p_trap_set->inv_ydelta = 1.0 / p_trap_set->ydelta ;

    /* find how many locations to allocate for each bin */
    for ( i = 0 ; i < p_trap_set->bins ; i++ ) {
        bin_tot[i] = 0 ;
    }

    vtx0 = pgon[numverts-1] ;
    for ( i = 0 ; i < numverts ; i++ ) {
        vtx1 = pgon[i] ;

        /* skip if Y's identical (edge has no effect) */
```

```
    if ( vtx0[Y] != vtx1[Y] ) {

        if ( vtx0[Y] < vtx1[Y] ) {
            vtxa = vtx0 ;
            vtxb = vtx1 ;
        } else {
            vtxa = vtx1 ;
            vtxb = vtx0 ;
        }
        ba = (int)(( vtxa[Y]-p_trap_set->miny ) * p_trap_set->inv_ydelta) ;
        fbb = ( vtxb[Y] - p_trap_set->miny ) * p_trap_set->inv_ydelta ;
        bb = (int)fbb ;
        /* if high vertex ends on a boundary, don't go into next boundary */
        if ( fbb - (double)bb == 0.0 ) {
            bb-- ;
        }

        /* mark the bins with this edge */
        for ( j = ba ; j <= bb ; j++ ) {
            bin_tot[j]++ ;
        }
    }

    vtx0 = vtx1 ;
}

/* allocate the bin contents and fill in some basics */
for ( i = 0 ; i < p_trap_set->bins ; i++ ) {
    p_trap_set->trapz[i].edge_set =
        (pEdge*)malloc( bin_tot[i] * sizeof(pEdge) ) ;
    MALLOC_CHECK( p_trap_set->trapz[i].edge_set ) ;
    for ( j = 0 ; j < bin_tot[i] ; j++ ) {
        p_trap_set->trapz[i].edge_set[j] =
            (pEdge)malloc( sizeof(Edge) ) ;
        MALLOC_CHECK( p_trap_set->trapz[i].edge_set[j] ) ;
    }

    /* start these off at some awful values; refined below */
    p_trap_set->trapz[i].minx = p_trap_set->maxx ;
    p_trap_set->trapz[i].maxx = p_trap_set->minx ;
    p_trap_set->trapz[i].count = 0 ;
}

/* now go through list yet again, putting edges in bins */
vtx0 = pgon[numverts-1] ;
id = numverts-1 ;
for ( i = 0 ; i < numverts ; i++ ) {
    vtx1 = pgon[i] ;

    /* we can skip edge if Y's are equal */
    if ( vtx0[Y] != vtx1[Y] ) {
        if ( vtx0[Y] < vtx1[Y] ) {
            vtxa = vtx0 ;
            vtxb = vtx1 ;
        } else {
            vtxa = vtx1 ;
            vtxb = vtx0 ;
        }
        fba = ( vtxa[Y] - p_trap_set->miny ) * p_trap_set->inv_ydelta ;
        ba = (int)fba ;
        fbb = ( vtxb[Y] - p_trap_set->miny ) * p_trap_set->inv_ydelta ;
```

```
bb = (int)fbb ;
/* if high vertex ends on a boundary, don't go into it */
if ( fbb == (double)bb ) {
    bb-- ;
}

vx0 = vtxa[X] ;
dy = vtxa[Y] - vtxb[Y] ;
slope = p_trap_set->ydelta * ( vtxa[X] - vtxb[X] ) / dy ;

/* set vx1 in case loop is not entered */
vx1 = vx0 ;
full_cross = 0 ;

for ( j = ba ; j < bb ; j++, vx0 = vx1 ) {
    /* could increment vx1, but for greater accuracy recompute it */
    vx1 = vtxa[X] + ( (double)(j+1) - fba ) * slope ;

    count = p_trap_set->trapz[j].count++ ;
    p_trap_set->trapz[j].edge_set[count]->id = id ;
    p_trap_set->trapz[j].edge_set[count]->full_cross = full_cross ;
    TrapBound( j, count, vx0, vx1, p_trap_set ) ;
    full_cross = 1 ;
}

/* at last bin - fill as above, but with vx1 = vtxb[X] */
vx0 = vx1 ;
vx1 = vtxb[X] ;
count = p_trap_set->trapz[bb].count++ ;
p_trap_set->trapz[bb].edge_set[count]->id = id ;
/* the last bin is never a full crossing */
p_trap_set->trapz[bb].edge_set[count]->full_cross = 0 ;
TrapBound( bb, count, vx0, vx1, p_trap_set ) ;
}
```

```
vtx0 = vtx1 ;
id = i ;
}
```

```
/* finally, sort the bins' contents by minx */
for ( i = 0 ; i < p_trap_set->bins ; i++ ) {
    qsort( p_trap_set->trapz[i].edge_set, p_trap_set->trapz[i].count,
        sizeof(pEdge), CompareEdges ) ;
}
```

```
}
```

```
void TrapBound( j, count, vx0, vx1, p_trap_set )
int    j, count ;
double vx0, vx1 ;
pTrapezoidSet  p_trap_set ;
{
double  xt ;
```

```
    if ( vx0 > vx1 ) {
        xt = vx0 ;
        vx0 = vx1 ;
        vx1 = xt ;
    }
```

```
    if ( p_trap_set->trapz[j].minx > vx0 ) {
        p_trap_set->trapz[j].minx = vx0 ;
```

```
}
if ( p_trap_set->trapz[j].maxx < vx1 ) {
    p_trap_set->trapz[j].maxx = vx1 ;
}
p_trap_set->trapz[j].edge_set[count]->minx = vx0 ;
p_trap_set->trapz[j].edge_set[count]->maxx = vx1 ;
}

/* used by qsort to sort */
int CompareEdges( u, v )
pEdge *u, *v ;
{
    if ( (*u)->minx == (*v)->minx ) {
        return( 0 ) ;
    } else {
        return( (*u)->minx < (*v)->minx ? -1 : 1 ) ;
    }
}

int TrapezoidTest( pgon, numverts, p_trap_set, point )
double pgon[][2] ;
int numverts ;
pTrapezoidSet p_trap_set ;
double point[2] ;
{
    int j, b, count, id ;
    double tx, ty, *vtx0, *vtx1 ;
    pEdge *pp_bin ;
    pTrapezoid p_trap ;
    int inside_flag ;

    inside_flag = 0 ;

    /* first, is point inside bounding rectangle? */
    if ( ( ty = point[Y] ) < p_trap_set->miny ||
        ty >= p_trap_set->maxy ||
        ( tx = point[X] ) < p_trap_set->minx ||
        tx >= p_trap_set->maxx ) {

        /* outside of box */
        return( 0 ) ;
    }

    /* what bin are we in? */
    b = ( ty - p_trap_set->miny ) * p_trap_set->inv_ydelta ;

    /* find if we're inside this bin's bounds */
    if ( tx < (p_trap = &p_trap_set->trapz[b])->minx ||
        tx > p_trap->maxx ) {

        /* outside of box */
        return( 0 ) ;
    }

    /* now search bin for crossings */
    pp_bin = p_trap->edge_set ;
    count = p_trap->count ;
    for ( j = 0 ; j < count ; j++, pp_bin++ ) {
        if ( tx < (*pp_bin)->minx ) {

            /* all remaining edges are to right of point, so test them */
```

```

do {
    if ( (*pp_bin)->full_cross ) {
        inside_flag = !inside_flag ;
    } else {
        id = (*pp_bin)->id ;
        if ( ( ty <= pgon[id][Y] ) !=
            ( ty <= pgon[(id+1)%numverts][Y] ) ) {

            /* point crosses edge in Y, so must cross */
            inside_flag = !inside_flag ;
        }
    }
    pp_bin++ ;
} while ( ++j < count ) ;
goto Exit;

```

```

} else if ( tx < (*pp_bin)->maxx ) {
    /* edge is overlapping point in X, check it */
    id = (*pp_bin)->id ;
    vtx0 = pgon[id] ;
    vtx1 = pgon[(id+1)%numverts] ;

    if ( (*pp_bin)->full_cross ||
        ( ty <= vtx0[Y] ) != ( ty <= vtx1[Y] ) ) {

        /* edge crosses in Y, so have to do full crossings test */
        if ( (vtx0[X] -
            (vtx0[Y] - ty ) *
            ( vtx1[X]-vtx0[X])/(vtx1[Y]-vtx0[Y])) >= tx ) {
            inside_flag = !inside_flag ;
        }
    }
}

} /* else edge is to left of point, ignore it */

```

```

Exit:
return( inside_flag ) ;

```

```

void TrapezoidCleanup( p_trap_set )
pTrapezoidSet    p_trap_set ;
{
    int    i, j, count ;

```

```

    for ( i = 0 ; i < p_trap_set->bins ; i++ ) {
        /* all of these should have bin sets, but check just in case */
        if ( p_trap_set->trapz[i].edge_set ) {
            count = p_trap_set->trapz[i].count ;
            for ( j = 0 ; j < count ; j++ ) {
                if ( p_trap_set->trapz[i].edge_set[j] ) {
                    free( p_trap_set->trapz[i].edge_set[j] ) ;
                }
            }
            free( p_trap_set->trapz[i].edge_set ) ;
        }
    }
    free( p_trap_set->trapz ) ;
}

```

```

/* ===== Grid algorithm ===== */

```

```
/* Impose a grid upon the polygon and test only the local edges against the
 * point.
 *
 * Call setup with 2D polygon _pgon_ with _numverts_ number of vertices,
 * grid resolution _resolution_ and a pointer to a grid structure _p_gs_.
 * Call testing procedure with a pointer to this array and test point _point_,
 * returns 1 if inside, 0 if outside.
 * Call cleanup with pointer to grid structure to free space.
 */

/* Strategy for setup:
 *   Get bounds of polygon, allocate grid.
 *   "Walk" each edge of the polygon and note which edges have been crossed
 *   and what cells are entered (points on a grid edge are always considered
 *   to be above that edge). Keep a record of the edges overlapping a cell.
 *   For cells with edges, determine if any cell border has no edges passing
 *   through it and so can be used for shooting a test ray.
 *   Keep track of the parity of the x (horizontal) grid cell borders for
 *   use in determining whether the grid corners are inside or outside.
 */

void GridSetup( pgon, numverts, resolution, p_gs )
double pgon[][2] ;
int numverts ;
int resolution ;
pGridSet p_gs ;
{
double *vtx0, *vtx1, *vtxa, *vtxb, *p_gl ;
int i, j, gc_clear_flags ;
double vx0, vx1, vy0, vy1, gxdiff, gydiff, eps ;
pGridCell p_gc, p_ngc ;
double xdiff, ydiff, tmax, inv_x, inv_y, xdir, ydir, t_near, tx, ty ;
double tgcx, tgcy ;
int gcx, gcy, sign_x ;
int y_flag, io_state ;

p_gs->xres = p_gs->yres = resolution ;
p_gs->tot_cells = p_gs->xres * p_gs->yres ;
p_gs->glx = (double *)malloc( (p_gs->xres+1) * sizeof(double)) ;
MALLOC_CHECK( p_gs->glx ) ;
p_gs->gly = (double *)malloc( (p_gs->yres+1) * sizeof(double)) ;
MALLOC_CHECK( p_gs->gly ) ;
p_gs->gc = (pGridCell)malloc( p_gs->tot_cells * sizeof(GridCell)) ;
MALLOC_CHECK( p_gs->gc ) ;

p_gs->minx =
p_gs->maxx = pgon[0][X] ;
p_gs->miny =
p_gs->maxy = pgon[0][Y] ;

/* find bounds of polygon */
for ( i = 1 ; i < numverts ; i++ ) {
vx0 = pgon[i][X] ;
if ( p_gs->minx > vx0 ) {
p_gs->minx = vx0 ;
} else if ( p_gs->maxx < vx0 ) {
p_gs->maxx = vx0 ;
}

vy0 = pgon[i][Y] ;
if ( p_gs->miny > vy0 ) {
```

```
        p_gs->miny = vy0 ;
    } else if ( p_gs->maxy < vy0 ) {
        p_gs->maxy = vy0 ;
    }
}

/* add a little to the bounds to ensure everything falls inside area */
gxdiff = p_gs->maxx - p_gs->minx ;
gydiff = p_gs->maxy - p_gs->miny ;
p_gs->minx -= EPSILON * gxdiff ;
p_gs->maxx += EPSILON * gxdiff ;
p_gs->miny -= EPSILON * gydiff ;
p_gs->maxy += EPSILON * gydiff ;

/* avoid roundoff problems near corners by not getting too close to them */
eps = 1e-9 * ( gxdiff + gydiff ) ;

/* use the new bounds to compute cell widths */
TryAgain:
p_gs->xdelta =
    (p_gs->maxx-p_gs->minx) / (double)p_gs->xres ;
p_gs->inv_xdelta = 1.0 / p_gs->xdelta ;

p_gs->ydelta =
    (p_gs->maxy-p_gs->miny) / (double)p_gs->yres ;
p_gs->inv_ydelta = 1.0 / p_gs->ydelta ;

for ( i = 0, p_gl = p_gs->glx ; i < p_gs->xres ; i++ ) {
    *p_gl++ = p_gs->minx + i * p_gs->xdelta ;
}
/* make last grid corner precisely correct */
*p_gl = p_gs->maxx ;

for ( i = 0, p_gly = p_gs->gly ; i < p_gs->yres ; i++ ) {
    *p_gly++ = p_gs->miny + i * p_gs->ydelta ;
}
*p_gly = p_gs->maxy ;

for ( i = 0, p_gc = p_gs->gc ; i < p_gs->tot_cells ; i++, p_gc++ ) {
    p_gc->tot_edges = 0 ;
    p_gc->gc_flags = 0x0 ;
    p_gc->gr = NULL ;
}

/* loop through edges and insert into grid structure */
vtx0 = pgon[numverts-1] ;
for ( i = 0 ; i < numverts ; i++ ) {
    vtx1 = pgon[i] ;

    if ( vtx0[Y] < vtx1[Y] ) {
        vtxa = vtx0 ;
        vtxb = vtx1 ;
    } else {
        vtxa = vtx1 ;
        vtxb = vtx0 ;
    }

    /* Set x variable for the direction of the ray */
    xdifff = vtxb[X] - vtxa[X] ;
    ydifff = vtxb[Y] - vtxa[Y] ;
    tmax = sqrt( xdifff * xdifff + ydifff * ydifff ) ;
```



```
/* if edge is of 0 length, ignore it (useless edge) */
if ( tmax == 0.0 ) goto NextEdge ;

xdir = xdiff / tmax ;
ydir = ydiff / tmax ;

gcx = (int)(( vtxa[X] - p_gs->minx ) * p_gs->inv_xdelta) ;
gcy = (int)(( vtxa[Y] - p_gs->miny ) * p_gs->inv_ydelta) ;

/* get information about slopes of edge, etc */
if ( vtxa[X] == vtxb[X] ) {
    sign_x = 0 ;
    tx = HUGE ;
} else {
    inv_x = tmax / xdiff ;
    tx = p_gs->xdelta * (double)gcx + p_gs->minx - vtxa[X] ;
    if ( vtxa[X] < vtxb[X] ) {
        sign_x = 1 ;
        tx += p_gs->xdelta ;
        tgcx = p_gs->xdelta * inv_x ;
    } else {
        sign_x = -1 ;
        tgcx = -p_gs->xdelta * inv_x ;
    }
    tx *= inv_x ;
}

if ( vtxa[Y] == vtxb[Y] ) {
    ty = HUGE ;
} else {
    inv_y = tmax / ydiff ;
    ty = (p_gs->ydelta * (double)(gcy+1) + p_gs->miny - vtxa[Y])
        * inv_y ;
    tgcy = p_gs->ydelta * inv_y ;
}

p_gc = &p_gs->gc[gcy*p_gs->xres+gcx] ;

vx0 = vtxa[X] ;
vy0 = vtxa[Y] ;

t_near = 0.0 ;

do {
    /* choose the next boundary, but don't move yet */
    if ( tx <= ty ) {
        gcx += sign_x ;

        ty -= tx ;
        t_near += tx ;
        tx = tgcx ;

        /* note which edge is hit when leaving this cell */
        if ( t_near < tmax ) {
            if ( sign_x > 0 ) {
                p_gc->gc_flags |= GC_R_EDGE_HIT ;
                vx1 = p_gs->glx[gcx] ;
            } else {
                p_gc->gc_flags |= GC_L_EDGE_HIT ;
                vx1 = p_gs->glx[gcx+1] ;
            }
        }
    }
}
```

```
    }

    /* get new location */
    vy1 = t_near * ydir + vtxa[Y] ;
} else {
    /* end of edge, so get exact value */
    vx1 = vtxb[X] ;
    vy1 = vtxb[Y] ;
}

y_flag = FALSE ;

} else {

    gcy++ ;

    tx -= ty ;
    t_near += ty ;
    ty = tgcy ;

    /* note top edge is hit when leaving this cell */
    if ( t_near < tmax ) {
        p_gc->gc_flags |= GC_T_EDGE_HIT ;
        /* this toggles the parity bit */
        p_gc->gc_flags ^= GC_T_EDGE_PARITY ;

        /* get new location */
        vx1 = t_near * xdir + vtxa[X] ;
        vy1 = p_gs->gly[gcy] ;
    } else {
        /* end of edge, so get exact value */
        vx1 = vtxb[X] ;
        vy1 = vtxb[Y] ;
    }

    y_flag = TRUE ;
}

/* check for corner crossing, then mark the cell we're in */
if ( !AddGridRecAlloc( p_gc, vx0, vy0, vx1, vy1, eps ) ) {
    /* warning, danger - we have just crossed a corner.
    * There are all kinds of topological messiness we could
    * do to get around this case, but they're a headache.
    * The simplest recovery is just to change the extents a bit
    * and redo the meshing, so that hopefully no edges will
    * perfectly cross a corner. Since it's a preprocess, we
    * don't care too much about the time to do it.
    */

    /* clean out all grid records */
    for ( i = 0, p_gc = p_gs->gc
          ; i < p_gs->tot_cells
          ; i++, p_gc++ ) {

        if ( p_gc->gr ) {
            free( p_gc->gr ) ;
        }
    }

    /* make the bounding box ever so slightly larger, hopefully
    * changing the alignment of the corners.
    */
}
```

```
        */
        p_gs->minx -= EPSILON * gxdiff * 0.24 ;
        p_gs->miny -= EPSILON * gydiff * 0.10 ;

        /* yes, it's the dreaded goto - run in fear for your lives! */
        goto TryAgain ;
    }

    if ( t_near < tmax ) {
        /* note how we're entering the next cell */
        /* TBD: could be done faster by incrementing index in the
         * incrementing code, above */
        p_gc = &p_gs->gc[gcy*p_gs->xres+gcx] ;

        if ( y_flag ) {
            p_gc->gc_flags |= GC_B_EDGE_HIT ;
            /* this toggles the parity bit */
            p_gc->gc_flags ^= GC_B_EDGE_PARITY ;
        } else {
            p_gc->gc_flags |=
                ( sign_x > 0 ) ? GC_L_EDGE_HIT : GC_R_EDGE_HIT ;
        }
    }

    vx0 = vx1 ;
    vy0 = vy1 ;
}
/* have we gone further than the end of the edge? */
while ( t_near < tmax ) ;

NextEdge:
vtx0 = vtx1 ;
}

/* the grid is all set up, now set up the inside/outside value of each
 * corner.
 */
p_gc = p_gs->gc ;
p_ngc = &p_gs->gc[p_gs->xres] ;

/* we know the bottom and top rows are all outside, so no flag is set */
for ( i = 1; i < p_gs->yres ; i++ ) {
    /* start outside */
    io_state = 0x0 ;

    for ( j = 0; j < p_gs->xres ; j++ ) {

        if ( io_state ) {
            /* change cell left corners to inside */
            p_gc->gc_flags |= GC_TL_IN ;
            p_ngc->gc_flags |= GC_BL_IN ;
        }

        if ( p_gc->gc_flags & GC_T_EDGE_PARITY ) {
            io_state = !io_state ;
        }

        if ( io_state ) {
            /* change cell right corners to inside */
            p_gc->gc_flags |= GC_TR_IN ;
            p_ngc->gc_flags |= GC_BR_IN ;
        }
    }
}
```

```
    }

    p_gc++ ;
    p_ngc++ ;
}

p_gc = p_gs->gc ;
for ( i = 0; i < p_gs->tot_cells ; i++ ) {

    /* reverse parity of edge clear (1==edge clear) */
    gc_clear_flags = p_gc->gc_flags ^ GC_ALL_EDGE_CLEAR ;
    if ( gc_clear_flags & GC_L_EDGE_CLEAR ) {
        p_gc->gc_flags |= GC_AIM_L ;
    } else
    if ( gc_clear_flags & GC_B_EDGE_CLEAR ) {
        p_gc->gc_flags |= GC_AIM_B ;
    } else
    if ( gc_clear_flags & GC_R_EDGE_CLEAR ) {
        p_gc->gc_flags |= GC_AIM_R ;
    } else
    if ( gc_clear_flags & GC_T_EDGE_CLEAR ) {
        p_gc->gc_flags |= GC_AIM_T ;
    } else {
        /* all edges have something on them, do full test */
        p_gc->gc_flags |= GC_AIM_C ;
    }
    p_gc++ ;
}

int AddGridRecAlloc( p_gc, xa, ya, xb, yb, eps )
pGridCell      p_gc ;
double xa,yb,xb,yb,eps ;
{
    pGridRec      p_gr ;
    double      slope, inv_slope ;

    if ( Near(ya, yb, eps) ) {
        if ( Near(xa, xb, eps) ) {
            /* edge is 0 length, so get rid of it */
            return( FALSE ) ;
        } else {
            /* horizontal line */
            slope = HUGE ;
            inv_slope = 0.0 ;
        }
    } else {
        if ( Near(xa, xb, eps) ) {
            /* vertical line */
            slope = 0.0 ;
            inv_slope = HUGE ;
        } else {
            slope = (xb-xa)/(yb-ya) ;
            inv_slope = (yb-ya)/(xb-xa) ;
        }
    }

    p_gc->tot_edges++ ;
    if ( p_gc->tot_edges <= 1 ) {
        p_gc->gr = (pGridRec)malloc( sizeof(GridRec) ) ;
    }
}
```

```
    } else {
        p_gc->gr = (pGridRec)realloc( p_gc->gr,
            p_gc->tot_edges * sizeof(GridRec) ) ;
    }
    MALLOC_CHECK( p_gc->gr ) ;
    p_gr = &p_gc->gr[p_gc->tot_edges-1] ;

    p_gr->slope = slope ;
    p_gr->inv_slope = inv_slope ;

    p_gr->xa = xa ;
    p_gr->ya = ya ;
    if ( xa <= xb ) {
        p_gr->minx = xa ;
        p_gr->maxx = xb ;
    } else {
        p_gr->minx = xb ;
        p_gr->maxx = xa ;
    }
    if ( ya <= yb ) {
        p_gr->miny = ya ;
        p_gr->maxy = yb ;
    } else {
        p_gr->miny = yb ;
        p_gr->maxy = ya ;
    }

    /* P2 - P1 */
    p_gr->ax = xb - xa ;
    p_gr->ay = yb - ya ;

    return( TRUE ) ;
}

/* Test point against grid and edges in the cell (if any).  Algorithm:
 *   Check bounding box; if outside then return.
 *   Check cell point is inside; if simple inside or outside then return.
 *   Find which edge or corner is considered to be the best for testing and
 *   send a test ray towards it, counting the crossings.  Add in the
 *   state of the edge or corner the ray went to and so determine the
 *   state of the point (inside or outside).
 */
int GridTest( p_gs, point )
register pGridSet      p_gs ;
double point[2] ;
{
    int      j, count, init_flag ;
    pGridCell      p_gc ;
    pGridRec      p_gr ;
    double tx, ty, xcell, ycell, bx,by,cx,cy, cornerx, cornery ;
    double alpha, beta, denom ;
    unsigned short gc_flags ;
    int      inside_flag ;

    /* first, is point inside bounding rectangle? */
    if ( ( ty = point[Y] ) < p_gs->miny ||
        ty >= p_gs->maxy ||
        ( tx = point[X] ) < p_gs->minx ||
        tx >= p_gs->maxx ) {

        /* outside of box */

```

```

    inside_flag = FALSE ;
} else {

    /* what cell are we in? */
    ycell = ( ty - p_gs->miny ) * p_gs->inv_ydelta ;
    xcell = ( tx - p_gs->minx ) * p_gs->inv_xdelta ;
    p_gc = &p_gs->gc[((int)ycell)*p_gs->xres + (int)xcell] ;

    /* is cell simple? */
    count = p_gc->tot_edges ;
    if ( count ) {
        /* no, so find an edge which is free. */
        gc_flags = p_gc->gc_flags ;
        p_gr = p_gc->gr ;
        switch( gc_flags & GC_AIM ) {
        case GC_AIM_L:
            /* left edge is clear, shoot X- ray */
            /* note - this next statement requires that GC_BL_IN is 1 */
            inside_flag = gc_flags & GC_BL_IN ;
            for ( j = count+1 ; --j ; p_gr++ ) {
                /* test if y is between edges */
                if ( ty >= p_gr->miny && ty < p_gr->maxy ) {
                    if ( tx > p_gr->maxx ) {
                        inside_flag = !inside_flag ;
                    } else if ( tx > p_gr->minx ) {
                        /* full computation */
                        if ( ( p_gr->xa -
                            ( p_gr->ya - ty ) * p_gr->slope ) < tx ) {
                            inside_flag = !inside_flag ;
                        }
                    }
                }
            }
            break ;

        case GC_AIM_B:
            /* bottom edge is clear, shoot Y+ ray */
            /* note - this next statement requires that GC_BL_IN is 1 */
            inside_flag = gc_flags & GC_BL_IN ;
            for ( j = count+1 ; --j ; p_gr++ ) {
                /* test if x is between edges */
                if ( tx >= p_gr->minx && tx < p_gr->maxx ) {
                    if ( ty > p_gr->maxy ) {
                        inside_flag = !inside_flag ;
                    } else if ( ty > p_gr->miny ) {
                        /* full computation */
                        if ( ( p_gr->ya - ( p_gr->xa - tx ) *
                            p_gr->inv_slope ) < ty ) {
                            inside_flag = !inside_flag ;
                        }
                    }
                }
            }
            break ;

        case GC_AIM_R:
            /* right edge is clear, shoot X+ ray */
            inside_flag = (gc_flags & GC_TR_IN) ? 1 : 0 ;

            /* TBD: Note, we could have sorted the edges to be tested
             * by miny or somesuch, and so be able to cut testing

```

```

        * short when the list's miny > point.y .
        */
    for ( j = count+1 ; --j ; p_gr++ ) {
        /* test if y is between edges */
        if ( ty >= p_gr->miny && ty < p_gr->maxy ) {
            if ( tx <= p_gr->minx ) {
                inside_flag = !inside_flag ;
            } else if ( tx <= p_gr->maxx ) {
                /* full computation */
                if ( ( p_gr->xa -
                    ( p_gr->ya - ty ) * p_gr->slope ) >= tx ) {
                    inside_flag = !inside_flag ;
                }
            }
        }
    }
    break ;

case GC_AIM_T:
    /* top edge is clear, shoot Y+ ray */
    inside_flag = (gc_flags & GC_TR_IN) ? 1 : 0 ;
    for ( j = count+1 ; --j ; p_gr++ ) {
        /* test if x is between edges */
        if ( tx >= p_gr->minx && tx < p_gr->maxx ) {
            if ( ty <= p_gr->miny ) {
                inside_flag = !inside_flag ;
            } else if ( ty <= p_gr->maxy ) {
                /* full computation */
                if ( ( p_gr->ya - ( p_gr->xa - tx ) *
                    p_gr->inv_slope ) >= ty ) {
                    inside_flag = !inside_flag ;
                }
            }
        }
    }
    break ;

case GC_AIM_C:
    /* no edge is clear, bite the bullet and test
     * against the bottom left corner.
     * We use Franklin Antonio's algorithm (Graphics Gems III).
     */
    /* TBD: Faster yet might be to test against the closest
     * corner to the cell location, but our hope is that we
     * rarely need to do this testing at all.
     */
    inside_flag = ((gc_flags & GC_BL_IN) == GC_BL_IN) ;
    init_flag = TRUE ;

    /* get lower left corner coordinate */
    cornerx = p_gs->glx[(int)xcell] ;
    cornery = p_gs->gly[(int)ycell] ;
    for ( j = count+1 ; --j ; p_gr++ ) {

        /* quick out test: if test point is
         * less than minx & miny, edge cannot overlap.
         */
        if ( tx >= p_gr->minx && ty >= p_gr->miny ) {

            /* quick test failed, now check if test point and
             * corner are on different sides of edge.

```

```
        */
        if ( init_flag ) {
            /* Compute these at most once for test */
            /* P3 - P4 */
            bx = tx - cornerx ;
            by = ty - cornery ;
            init_flag = FALSE ;
        }
        denom = p_gr->ay * bx - p_gr->ax * by ;
        if ( denom != 0.0 ) {
            /* lines are not collinear, so continue */
            /* P1 - P3 */
            cx = p_gr->xa - tx ;
            cy = p_gr->ya - ty ;
            alpha = by * cx - bx * cy ;
            if ( denom > 0.0 ) {
                if ( alpha < 0.0 || alpha >= denom ) {
                    /* test edge not hit */
                    goto NextEdge ;
                }
                beta = p_gr->ax * cy - p_gr->ay * cx ;
                if ( beta < 0.0 || beta >= denom ) {
                    /* polygon edge not hit */
                    goto NextEdge ;
                }
            }
            else {
                if ( alpha > 0.0 || alpha <= denom ) {
                    /* test edge not hit */
                    goto NextEdge ;
                }
                beta = p_gr->ax * cy - p_gr->ay * cx ;
                if ( beta > 0.0 || beta <= denom ) {
                    /* polygon edge not hit */
                    goto NextEdge ;
                }
            }
            inside_flag = !inside_flag ;
        }
    }
    NextEdge: ;
}
break ;
}
} else {
    /* simple cell, so if lower left corner is in,
     * then cell is inside.
     */
    inside_flag = p_gc->gc_flags & GC_BL_IN ;
}
}

return( inside_flag ) ;
}
```

```
void GridCleanup( p_gs )
pGridSet      p_gs ;
{
    int      i ;
    pGridCell p_gc ;
```



```
for ( i = 0, p_gc = p_gs->gc
    ; i < p_gs->tot_cells
    ; i++, p_gc++ ) {

    if ( p_gc->gr ) {
        free( p_gc->gr ) ;
    }
}
free( p_gs->glx ) ;
free( p_gs->gly ) ;
free( p_gs->gc ) ;
}

/* ===== Exterior (convex only) algorithm ===== */

/* Test the edges of the convex polygon against the point.  If the point is
 * outside any edge, the point is outside the polygon.
 *
 * Call setup with 2D polygon _pgon_ with _numverts_ number of vertices,
 * which returns a pointer to a plane set array.
 * Call testing procedure with a pointer to this array, _numverts_, and
 * test point _point_, returns 1 if inside, 0 if outside.
 * Call cleanup with pointer to plane set array to free space.
 *
 * RANDOM can be defined for this test.
 * CONVEX must be defined for this test; it is not usable for general polygons.
 */

#ifdef CONVEX
/* make exterior plane set */
pPlaneSet ExteriorSetup( pgon, numverts )
double pgon[][2] ;
int numverts ;
{
    int p1, p2, flip_edge ;
    pPlaneSet pps, pps_return ;
#ifdef RANDOM
    int i, ind ;
    PlaneSet ps_temp ;
#endif

    pps = pps_return =
        (pPlaneSet)malloc( numverts * sizeof( PlaneSet ) ) ;
    MALLOC_CHECK( pps ) ;

    /* take cross product of vertex to find handedness */
    flip_edge = (pgon[0][X] - pgon[1][X]) * (pgon[1][Y] - pgon[2][Y]) >
        (pgon[0][Y] - pgon[1][Y]) * (pgon[1][X] - pgon[2][X]) ;

    /* Generate half-plane boundary equations now for faster testing later.
     * vx & vy are the edge's normal, c is the offset from the origin.
     */
    for ( p1 = numverts-1, p2 = 0 ; p2 < numverts ; p1 = p2, p2++, pps++ ) {
        pps->vx = pgon[p1][Y] - pgon[p2][Y] ;
        pps->vy = pgon[p2][X] - pgon[p1][X] ;
        pps->c = pps->vx * pgon[p1][X] + pps->vy * pgon[p1][Y] ;

        /* check sense and reverse plane edge if need be */
        if ( flip_edge ) {
            pps->vx = -pps->vx ;
            pps->vy = -pps->vy ;
        }
    }
}
```

```
        pps->c = -pps->c ;
    }
}

#ifdef RANDOM
/* Randomize the order of the edges to improve chance of early out */
/* There are better orders, but the default order is the worst */
for ( i = 0, pps = pps_return
      ; i < numverts
      ; i++ ) {

    ind = (int)(RAN01() * numverts ) ;
    if ( ( ind < 0 ) || ( ind >= numverts ) ) {
        fprintf( stderr,
                  "Yikes, the random number generator is returning values\n" ) ;
        fprintf( stderr,
                  "outside the range [0.0,1.0), so please fix the code!\n" ) ;
        ind = 0 ;
    }

    /* swap edges */
    ps_temp = *pps ;
    *pps = pps_return[ind] ;
    pps_return[ind] = ps_temp ;
}
#endif

return( pps_return ) ;
}

/* Check point for outside of all planes */
/* note that we don't need "pgon", since it's been processed into
 * its corresponding PlaneSet.
 */
int ExteriorTest( p_ext_set, numverts, point )
pPlaneSet      p_ext_set ;
int            numverts ;
double         point[2] ;
{
    register PlaneSet      *pps ;
    register int           p0 ;
    register double tx, ty ;
    int            inside_flag ;

    tx = point[X] ;
    ty = point[Y] ;

    for ( p0 = numverts+1, pps = p_ext_set ; --p0 ; pps++ ) {

        /* test if the point is outside this edge */
        if ( pps->vx * tx + pps->vy * ty > pps->c ) {
            return( 0 ) ;
        }
    }
    /* if we make it to here, we were inside all edges */
    return( 1 ) ;
}

void ExteriorCleanup( p_ext_set )
pPlaneSet      p_ext_set ;
{

```

```
    free( p_ext_set ) ;
}
#endif

/* ===== Inclusion (convex only) algorithm ===== */

/* Create an efficiency structure (see Preparata) for the convex polygon which
 * allows binary searching to find which edge to test the point against. This
 * algorithm is O(log n).
 *
 * Call setup with 2D polygon _pgon_ with _numverts_ number of vertices,
 * which returns a pointer to an inclusion anchor structure.
 * Call testing procedure with a pointer to this structure and test point
 * _point_, returns 1 if inside, 0 if outside.
 * Call cleanup with pointer to inclusion anchor structure to free space.
 *
 * CONVEX must be defined for this test; it is not usable for general polygons.
 */

#ifdef CONVEX
/* make inclusion wedge set */
pInclusionAnchor InclusionSetup( pgon, numverts )
double pgon[][2] ;
int numverts ;
{
    int pc, p1, p2, flip_edge ;
    double ax,ay, qx,qy, wx,wy, len ;
    pInclusionAnchor pia ;
    pInclusionSet pis ;

    /* double the first edge to avoid needing modulo during test search */
    pia = (pInclusionAnchor)malloc( sizeof( InclusionAnchor ) ) ;
    MALLOC_CHECK( pia ) ;
    pis = pia->pis =
        (pInclusionSet)malloc( (numverts+1) * sizeof( InclusionSet ) ) ;
    MALLOC_CHECK( pis ) ;

    pia->hi_start = numverts - 1 ;

    /* get average point to make wedges from */
    qx = qy = 0.0 ;
    for ( p2 = 0 ; p2 < numverts ; p2++ ) {
        qx += pgon[p2][X] ;
        qy += pgon[p2][Y] ;
    }
    pia->qx = qx /= (double)numverts ;
    pia->qy = qy /= (double)numverts ;

    /* take cross product of vertex to find handedness */
    pia->flip_edge = flip_edge =
        (pgon[0][X] - pgon[1][X]) * (pgon[1][Y] - pgon[2][Y]) >
        (pgon[0][Y] - pgon[1][Y]) * (pgon[1][X] - pgon[2][X]) ;

    ax = pgon[0][X] - qx ;
    ay = pgon[0][Y] - qy ;
    len = sqrt( ax * ax + ay * ay ) ;
    if ( len == 0.0 ) {
        fprintf( stderr, "sorry, polygon for inclusion test is defective\n" ) ;
        exit(1) ;
    }
}
```

```
pia->ax = ax /= len ;
pia->ay = ay /= len ;

/* loop through edges, and double last edge */
for ( pc = p1 = 0, p2 = 1
      ; pc <= numverts
      ; pc++, p1 = p2, p2 = (++p2)%numverts, pis++ ) {

    /* wedge border */
    wx = pgon[p1][X] - qx ;
    wy = pgon[p1][Y] - qy ;
    len = sqrt( wx * wx + wy * wy ) ;
    wx /= len ;
    wy /= len ;

    /* cosine of angle from anchor border to wedge border */
    pis->dot = ax * wx + ay * wy ;
    /* sign from cross product */
    if ( ( ax * wy > ay * wx ) == flip_edge ) {
        pis->dot = -2.0 - pis->dot ;
    }

    /* edge */
    pis->ex = pgon[p1][Y] - pgon[p2][Y] ;
    pis->ey = pgon[p2][X] - pgon[p1][X] ;
    pis->ec = pis->ex * pgon[p1][X] + pis->ey * pgon[p1][Y] ;

    /* check sense and reverse plane eqns if need be */
    if ( flip_edge ) {
        pis->ex = -pis->ex ;
        pis->ey = -pis->ey ;
        pis->ec = -pis->ec ;
    }
}
/* set first angle a little > 1.0 and last < -3.0 just to be safe. */
pia->pis[0].dot = -3.001 ;
pia->pis[numverts].dot = 1.001 ;

return( pia ) ;
}

/* Find wedge point is in by binary search, then test wedge */
int InclusionTest( pia, point )
pInclusionAnchor    pia ;
double point[2] ;
{
    register double tx, ty, len, dot ;
    int inside_flag, lo, hi, ind ;
    pInclusionSet    pis ;

    tx = point[X] - pia->qx ;
    ty = point[Y] - pia->qy ;
    len = sqrt( tx * tx + ty * ty ) ;
    /* check if point is exactly at anchor point (which is inside polygon) */
    if ( len == 0.0 ) return( 1 ) ;
    tx /= len ;
    ty /= len ;

    /* get dot product for searching */
    dot = pia->ax * tx + pia->ay * ty ;
    if ( ( pia->ax * ty > pia->ay * tx ) == pia->flip_edge ) {
```

```
        dot = -2.0 - dot ;
    }

    /* binary search through angle list and find matching angle pair */
    lo = 0 ;
    hi = pia->hi_start ;
    while ( lo <= hi ) {
        ind = (lo+hi)/ 2 ;
        if ( dot < pia->pis[ind].dot ) {
            hi = ind - 1 ;
        } else if ( dot > pia->pis[ind+1].dot ) {
            lo = ind + 1 ;
        } else {
            goto Foundit ;
        }
    }
    /* should never reach here, but just in case... */
    fprintf( stderr,
        "Hmmm, something weird happened - bad dot product %lg\n", dot);

Foundit:

    /* test if the point is outside the wedge's exterior edge */
    pis = &pia->pis[ind] ;
    inside_flag = ( pis->ex * point[X] + pis->ey * point[Y] <= pis->ec ) ;

    return( inside_flag ) ;
}

void InclusionCleanup( p_inc_anchor )
pInclusionAnchor p_inc_anchor ;
{
    free( p_inc_anchor->pis ) ;
    free( p_inc_anchor ) ;
}
#endif

/* ===== Crossings Multiply algorithm ===== */

/*
 * This version is usually somewhat faster than the original published in
 * Graphics Gems IV; by turning the division for testing the X axis crossing
 * into a tricky multiplication test this part of the test became faster,
 * which had the additional effect of making the test for "both to left or
 * both to right" a bit slower for triangles than simply computing the
 * intersection each time. The main increase is in triangle testing speed,
 * which was about 15% faster; all other polygon complexities were pretty much
 * the same as before. On machines where division is very expensive (not the
 * case on the HP 9000 series on which I tested) this test should be much
 * faster overall than the old code. Your mileage may (in fact, will) vary,
 * depending on the machine and the test data, but in general I believe this
 * code is both shorter and faster. This test was inspired by unpublished
 * Graphics Gems submitted by Joseph Samosky and Mark Haigh-Hutchinson.
 * Related work by Samosky is in:
 *
 * Samosky, Joseph, "SectionView: A system for interactively specifying and
 * visualizing sections through three-dimensional medical image data",
 * M.S. Thesis, Department of Electrical Engineering and Computer Science,
 * Massachusetts Institute of Technology, 1993.
 *
 */
```

```
*/

/* Shoot a test ray along +X axis. The strategy is to compare vertex Y values
 * to the testing point's Y and quickly discard edges which are entirely to one
 * side of the test ray. Note that CONVEX and WINDING code can be added as
 * for the CrossingsTest() code; it is left out here for clarity.
 */
/* Input 2D polygon _pgon_ with _numverts_ number of vertices and test point
 * _point_, returns 1 if inside, 0 if outside.
 */
int CrossingsMultiplyTest( pgon, numverts, point )
double pgon[][2] ;
int numverts ;
double point[2] ;
{
register int j, yflag0, yflag1, inside_flag ;
register double ty, tx, *vtx0, *vtx1 ;

tx = point[X] ;
ty = point[Y] ;

vtx0 = pgon[numverts-1] ;
/* get test bit for above/below X axis */
yflag0 = ( vtx0[Y] >= ty ) ;
vtx1 = pgon[0] ;

inside_flag = 0 ;
for ( j = numverts+1 ; --j ; ) {

yflag1 = ( vtx1[Y] >= ty ) ;
/* Check if endpoints straddle (are on opposite sides) of X axis
 * (i.e. the Y's differ); if so, +X ray could intersect this edge.
 * The old test also checked whether the endpoints are both to the
 * right or to the left of the test point. However, given the faster
 * intersection point computation used below, this test was found to
 * be a break-even proposition for most polygons and a loser for
 * triangles (where 50% or more of the edges which survive this test
 * will cross quadrants and so have to have the X intersection computed
 * anyway). I credit Joseph Samosky with inspiring me to try dropping
 * the "both left or both right" part of my code.
 */
if ( yflag0 != yflag1 ) {
/* Check intersection of pgon segment with +X ray.
 * Note if >= point's X; if so, the ray hits it.
 * The division operation is avoided for the ">=" test by checking
 * the sign of the first vertex wrto the test point; idea inspired
 * by Joseph Samosky's and Mark Haigh-Hutchinson's different
 * polygon inclusion tests.
 */
if ( ((vtx1[Y]-ty) * (vtx0[X]-vtx1[X]) >=
      (vtx1[X]-tx) * (vtx0[Y]-vtx1[Y])) == yflag1 ) {
inside_flag = !inside_flag ;
}
}

/* Move to the next pair of vertices, retaining info as possible. */
yflag0 = yflag1 ;
vtx0 = vtx1 ;
vtx1 += 2 ;
}
}
```

```
    return( inside_flag ) ;  
}
```

```
/* ptpinpoly.h - point in polygon inside/outside algorithms header file.
 *
 * by Eric Haines, 3D/Eye Inc, erich@eye.com
 */

/* Define CONVEX to compile for testing only convex polygons (when possible,
 * this is faster) */
/* #define CONVEX */

/* Define HYBRID to compile triangle fan test for CONVEX with exterior edges
 * meaning an early exit (faster - recommended).
 */
/* #define HYBRID */

/* Define DISPLAY to display test triangle and test points on screen */
/* #define DISPLAY */

/* Define RANDOM to randomize order of edges for exterior test (faster -
 * recommended). */
/* #define RANDOM */

/* Define SORT to sort triangle edges and areas for half-plane and Spackman
 * tests (faster - recommended).
 * The bad news with SORT for non-convex testing is that this usually messes
 * up any coherence for the triangle fan tests, meaning that points on an
 * interior edge can be mis-classified (very rare, except when -c is used).
 * In other words, if a point lands on an edge between two test triangles,
 * normally it will be inside only one - sorting messes up the test order and
 * makes it so that the point can be inside two.
 */
/* #define SORT */

/* Define WINDING if a non-zero winding number should be used as the criterion
 * for being inside the polygon. Only used by the general crossings test and
 * Weiler test. The winding number computed for each is the number of
 * counter-clockwise loops the polygon makes around the point.
 */
/* #define WINDING */

/* ===== System Related ===== */

/* define your own random number generator, change as needed */
/* SRAN initializes random number generator, if needed */
#define SRAN()          srand48(1)
/* RAN01 returns a double from [0..1) */
#define RAN01()          drand48()
double  drand48() ;

/* On systems without drand48() you might do this instead (though check if
 * rand()'s divisor is correct for your machine):
#define SRAN()          srand(1)
#define RAN01()          ((double)rand() / 32768.0)
*/

/* ===== Grid stuff ===== */

#define GR_FULLL_VERT    0x01    /* line crosses vertically */
#define GR_FULLL_HORZ    0x02    /* line crosses horizontally */

typedef struct {
    double      xa,ya ;
```



```
double      minx, maxx, miny, maxy ;
double      ax, ay ;
double      slope, inv_slope ;
} GridRec, *pGridRec;

#define GC_BL_IN      0x0001 /* bottom left corner is in (else out) */
#define GC_BR_IN      0x0002 /* bottom right corner is in (else out) */
#define GC_TL_IN      0x0004 /* top left corner is in (else out) */
#define GC_TR_IN      0x0008 /* top right corner is in (else out) */
#define GC_L_EDGE_HIT 0x0010 /* left edge is crossed */
#define GC_R_EDGE_HIT 0x0020 /* right edge is crossed */
#define GC_B_EDGE_HIT 0x0040 /* bottom edge is crossed */
#define GC_T_EDGE_HIT 0x0080 /* top edge is crossed */
#define GC_B_EDGE_PARITY 0x0100 /* bottom edge parity */
#define GC_T_EDGE_PARITY 0x0200 /* top edge parity */
#define GC_AIM_L      (0<<10) /* aim towards left edge */
#define GC_AIM_B      (1<<10) /* aim towards bottom edge */
#define GC_AIM_R      (2<<10) /* aim towards right edge */
#define GC_AIM_T      (3<<10) /* aim towards top edge */
#define GC_AIM_C      (4<<10) /* aim towards a corner */
#define GC_AIM        0x1c00

#define GC_L_EDGE_CLEAR GC_L_EDGE_HIT
#define GC_R_EDGE_CLEAR GC_R_EDGE_HIT
#define GC_B_EDGE_CLEAR GC_B_EDGE_HIT
#define GC_T_EDGE_CLEAR GC_T_EDGE_HIT

#define GC_ALL_EDGE_CLEAR      (GC_L_EDGE_HIT | \
                                GC_R_EDGE_HIT | \
                                GC_B_EDGE_HIT | \
                                GC_T_EDGE_HIT )

typedef struct {
    short          tot_edges ;
    unsigned short gc_flags ;
    GridRec        *gr ;
} GridCell, *pGridCell;

typedef struct {
    int      xres, yres ; /* grid size */
    int      tot_cells ; /* xres * yres */
    double   minx, maxx, miny, maxy ; /* bounding box */
    double   xdelta, ydelta ;
    double   inv_xdelta, inv_ydelta ;
    double   *glx, *gly ;
    GridCell *gc ;
} GridSet, *pGridSet ;

#ifdef CONVEX
/* ===== Inclusion stuff ===== */
typedef struct {
    double   dot ; /* angle to beginning of edge */
    double   ex, ey, ec ; /* edge equation */
} InclusionSet, *pInclusionSet ;

typedef struct {
    int      flip_edge ; /* clockwise/counterclockwise */
    int      hi_start ; /* hi start for binary search: numverts-1 */
    double   ax, ay ; /* anchor edge vector */
    double   qx, qy ; /* anchor point */
}
```

```
    pInclusionSet    pis ;
} InclusionAnchor, *pInclusionAnchor ;
#endif /* end CONVEX */

/* ===== Half-Plane stuff ===== */

typedef struct {
    double    vx, vy, c ;    /* edge equation  vx*X + vy*Y + c = 0 */
#ifdef CONVEX
#ifdef HYBRID
    int        ext_flag ;    /* TRUE == exterior edge of polygon */
#endif
#endif
} PlaneSet, *pPlaneSet ;

#ifdef CONVEX
#ifdef SORT
/* Size sorting structure for half-planes */
typedef struct {
    double    size ;
    pPlaneSet pps ;
} SizePlanePair, *pSizePlanePair ;
#endif
#endif

/* ===== Spackman (precomputed barycentric) stuff ===== */
typedef struct {
    double    ulp, u2, vlp, v2, inv_u1, inv_u2, inv_v1 ;
    int        ul_nonzero ;
} SpackmanSet, *pSpackmanSet ;

/* ===== Trapezoid stuff ===== */
/* how many bins shall we put the edges into? */
#define TOT_BINS    1000    /* absolutely the maximum number of bins */

/* add a little to the limits of the polygon bounding box to avoid precision
 * problems.
 */
#define EPSILON    0.00001

/* The following structure is associated with a polygon */
typedef struct {
    int        id ;    /* vertex number of edge */
    int        full_cross ;    /* 1 if extends from top to bottom */
    double    minx, maxx ;    /* X bounds for bin */
} Edge, *pEdge ;

typedef struct {
    pEdge        *edge_set ;
    double    minx, maxx ;    /* min and max for all edges in bin */
    int        count ;
} Trapezoid, *pTrapezoid ;

typedef struct {
    int        bins ;
    double    minx, maxx ;    /* bounding box for polygon */
    double    miny, maxy ;
```

```
double      ydelta ;          /* (maxy - miny)/bins */
double      inv_ydelta ;
Trapezoid   *trapz ;
} TrapezoidSet, *pTrapezoidSet ;
```

```
#ifdef CONVEX
pPlaneSet   ExteriorSetup() ;
void         ExteriorCleanup() ;
```

```
pInclusionAnchor   InclusionSetup() ;
void              InclusionCleanup() ;
```

```
#ifdef SORT
int      CompareSizePlanePairs() ;
#endif
#endif
```

```
pPlaneSet   PlaneSetup() ;
void         PlaneCleanup() ;
```

```
pSpackmanSet   SpackmanSetup() ;
void           SpackmanCleanup() ;
```

```
void         TrapezoidCleanup() ;
void         TrapBound() ;
int          CompareEdges() ;
void         TrapezoidSetup() ;
```

```
void         GridSetup() ;
int          AddGridRecAlloc() ;
void         GridCleanup() ;
```

```
# Statistics generator script.  A good overnight job.  Then use table.awk
# to convert an output stats file into a timings table.
```

```
# General polygon tests
make clean
export MAKEOPTS="-DTIMER -DRANDOM -DSORT"
make
```

```
# test all basic algorithms on random polygons
p_test -v 3 -n 50 -i 50 -r 0 -p 1 -d -ABCMPSW > Rr0.sts
p_test -v 4 -n 50 -i 50 -r 0 -p 1 -d -ABCMPSW >> Rr0.sts
p_test -v 10 -n 50 -i 50 -r 0 -p 1 -d -ABCMPSW >> Rr0.sts
p_test -v 20 -n 50 -i 50 -r 0 -p 1 -d -ABCMPSW >> Rr0.sts
p_test -v 50 -n 50 -i 50 -r 0 -p 1 -d -ABCMPSW >> Rr0.sts
p_test -v 100 -n 50 -i 50 -r 0 -p 1 -d -ABCMPSW >> Rr0.sts
p_test -v 1000 -n 50 -i 50 -r 0 -p 1 -d -ABCMPSW >> Rr0.sts
```

```
# grid and trapezoid tests at resolution 20
p_test -v 3 -n 50 -i 50 -r 0 -p 1 -d -CGT -g 20 -b 20 > Rr0R20.sts
p_test -v 4 -n 50 -i 50 -r 0 -p 1 -d -CGT -g 20 -b 20 >> Rr0R20.sts
p_test -v 10 -n 50 -i 50 -r 0 -p 1 -d -CGT -g 20 -b 20 >> Rr0R20.sts
p_test -v 20 -n 50 -i 50 -r 0 -p 1 -d -CGT -g 20 -b 20 >> Rr0R20.sts
p_test -v 50 -n 50 -i 50 -r 0 -p 1 -d -CGT -g 20 -b 20 >> Rr0R20.sts
p_test -v 100 -n 50 -i 50 -r 0 -p 1 -d -CGT -g 20 -b 20 >> Rr0R20.sts
p_test -v 1000 -n 50 -i 50 -r 0 -p 1 -d -CGT -g 20 -b 20 >> Rr0R20.sts
```

```
# grid and trapezoid tests at resolution 100
p_test -v 3 -n 50 -i 50 -r 0 -p 1 -d -CGT -g 100 -b 100 > Rr0R100.sts
p_test -v 4 -n 50 -i 50 -r 0 -p 1 -d -CGT -g 100 -b 100 >> Rr0R100.sts
p_test -v 10 -n 50 -i 50 -r 0 -p 1 -d -CGT -g 100 -b 100 >> Rr0R100.sts
p_test -v 20 -n 50 -i 50 -r 0 -p 1 -d -CGT -g 100 -b 100 >> Rr0R100.sts
p_test -v 50 -n 50 -i 50 -r 0 -p 1 -d -CGT -g 100 -b 100 >> Rr0R100.sts
p_test -v 100 -n 50 -i 50 -r 0 -p 1 -d -CGT -g 100 -b 100 >> Rr0R100.sts
p_test -v 1000 -n 50 -i 50 -r 0 -p 1 -d -CGT -g 100 -b 100 >> Rr0R100.sts
```

```
# test all basic algorithms on regular polygons
p_test -v 3 -n 50 -i 50 -r 1 -p 0 -d -ABCMPSW > Rr1.sts
p_test -v 4 -n 50 -i 50 -r 1 -p 0 -d -ABCMPSW >> Rr1.sts
p_test -v 10 -n 50 -i 50 -r 1 -p 0 -d -ABCMPSW >> Rr1.sts
p_test -v 20 -n 50 -i 50 -r 1 -p 0 -d -ABCMPSW >> Rr1.sts
p_test -v 50 -n 50 -i 50 -r 1 -p 0 -d -ABCMPSW >> Rr1.sts
p_test -v 100 -n 50 -i 50 -r 1 -p 0 -d -ABCMPSW >> Rr1.sts
p_test -v 1000 -n 50 -i 50 -r 1 -p 0 -d -ABCMPSW >> Rr1.sts
```

```
# grid and trapezoid tests at resolution 20
p_test -v 3 -n 50 -i 50 -r 1 -p 0 -d -CGT -g 20 -b 20 > Rr1R20.sts
p_test -v 4 -n 50 -i 50 -r 1 -p 0 -d -CGT -g 20 -b 20 >> Rr1R20.sts
p_test -v 10 -n 50 -i 50 -r 1 -p 0 -d -CGT -g 20 -b 20 >> Rr1R20.sts
p_test -v 20 -n 50 -i 50 -r 1 -p 0 -d -CGT -g 20 -b 20 >> Rr1R20.sts
p_test -v 50 -n 50 -i 50 -r 1 -p 0 -d -CGT -g 20 -b 20 >> Rr1R20.sts
p_test -v 100 -n 50 -i 50 -r 1 -p 0 -d -CGT -g 20 -b 20 >> Rr1R20.sts
p_test -v 1000 -n 50 -i 50 -r 1 -p 0 -d -CGT -g 20 -b 20 >> Rr1R20.sts
```

```
# grid and trapezoid tests at resolution 100
p_test -v 3 -n 50 -i 50 -r 1 -p 0 -d -CGT -g 100 -b 100 > Rr1R100.sts
p_test -v 4 -n 50 -i 50 -r 1 -p 0 -d -CGT -g 100 -b 100 >> Rr1R100.sts
p_test -v 10 -n 50 -i 50 -r 1 -p 0 -d -CGT -g 100 -b 100 >> Rr1R100.sts
p_test -v 20 -n 50 -i 50 -r 1 -p 0 -d -CGT -g 100 -b 100 >> Rr1R100.sts
p_test -v 50 -n 50 -i 50 -r 1 -p 0 -d -CGT -g 100 -b 100 >> Rr1R100.sts
p_test -v 100 -n 50 -i 50 -r 1 -p 0 -d -CGT -g 100 -b 100 >> Rr1R100.sts
```

```
p_test -v 1000 -n 50 -i 50 -r 1 -p 0 -d -CGT -g 100 -b 100 >> Rr1R100.sts
```

```
# General polygons, but not sorting edges
```

```
make clean
```

```
export MAKEOPTS="-DTIMER -DRANDOM"
```

```
make
```

```
p_test -v 3 -n 50 -i 50 -r 0 -p 1 -d -PS > Rr0no_sort.sts
```

```
p_test -v 4 -n 50 -i 50 -r 0 -p 1 -d -PS >> Rr0no_sort.sts
```

```
p_test -v 10 -n 50 -i 50 -r 0 -p 1 -d -PS >> Rr0no_sort.sts
```

```
p_test -v 20 -n 50 -i 50 -r 0 -p 1 -d -PS >> Rr0no_sort.sts
```

```
p_test -v 50 -n 50 -i 50 -r 0 -p 1 -d -PS >> Rr0no_sort.sts
```

```
p_test -v 100 -n 50 -i 50 -r 0 -p 1 -d -PS >> Rr0no_sort.sts
```

```
p_test -v 1000 -n 50 -i 50 -r 0 -p 1 -d -PS >> Rr0no_sort.sts
```

```
p_test -v 3 -n 50 -i 50 -r 1 -p 0 -d -PS > Rr1no_sort.sts
```

```
p_test -v 4 -n 50 -i 50 -r 1 -p 0 -d -PS >> Rr1no_sort.sts
```

```
p_test -v 10 -n 50 -i 50 -r 1 -p 0 -d -PS >> Rr1no_sort.sts
```

```
p_test -v 20 -n 50 -i 50 -r 1 -p 0 -d -PS >> Rr1no_sort.sts
```

```
p_test -v 50 -n 50 -i 50 -r 1 -p 0 -d -PS >> Rr1no_sort.sts
```

```
p_test -v 100 -n 50 -i 50 -r 1 -p 0 -d -PS >> Rr1no_sort.sts
```

```
p_test -v 1000 -n 50 -i 50 -r 1 -p 0 -d -PS >> Rr1no_sort.sts
```

```
# Convex polygon tests, best algorithms
```

```
make clean
```

```
export MAKEOPTS="-DTIMER -DRANDOM -DSORT -DCONVEX -DHYBRID"
```

```
make
```

```
p_test -v 3 -n 50 -i 50 -r 1 -p 0 -d -EICMP > Rr1convex_hy.sts
```

```
p_test -v 4 -n 50 -i 50 -r 1 -p 0 -d -EICMP >> Rr1convex_hy.sts
```

```
p_test -v 10 -n 50 -i 50 -r 1 -p 0 -d -EICMP >> Rr1convex_hy.sts
```

```
p_test -v 20 -n 50 -i 50 -r 1 -p 0 -d -EICMP >> Rr1convex_hy.sts
```

```
p_test -v 50 -n 50 -i 50 -r 1 -p 0 -d -EICMP >> Rr1convex_hy.sts
```

```
p_test -v 100 -n 50 -i 50 -r 1 -p 0 -d -EICMP >> Rr1convex_hy.sts
```

```
p_test -v 1000 -n 50 -i 50 -r 1 -p 0 -d -EICMP >> Rr1convex_hy.sts
```

```
# random triangles (vs. regular triangles)
```

```
p_test -v 3 -n 50 -i 50 -r 0 -p 1 -d -EICMP > Rr0convex_hy.sts
```

```
# Convex polygon tests, best algorithms
```

```
make clean
```

```
export MAKEOPTS="-DTIMER -DRANDOM -DSORT -DCONVEX"
```

```
make
```

```
p_test -v 3 -n 50 -i 50 -r 1 -p 0 -d -CP > Rr1convex.sts
```

```
p_test -v 4 -n 50 -i 50 -r 1 -p 0 -d -CP >> Rr1convex.sts
```

```
p_test -v 10 -n 50 -i 50 -r 1 -p 0 -d -CP >> Rr1convex.sts
```

```
p_test -v 20 -n 50 -i 50 -r 1 -p 0 -d -CP >> Rr1convex.sts
```

```
p_test -v 50 -n 50 -i 50 -r 1 -p 0 -d -CP >> Rr1convex.sts
```

```
p_test -v 100 -n 50 -i 50 -r 1 -p 0 -d -CP >> Rr1convex.sts
```

```
p_test -v 1000 -n 50 -i 50 -r 1 -p 0 -d -CP >> Rr1convex.sts
```

```
# random triangles (vs. regular triangles)
```

```
p_test -v 3 -n 50 -i 50 -r 0 -p 1 -d -EICMP > Rr0convex.sts
```

```
# convex polygon tests, no randomizing, no sorting (slower)
```

```
make clean
```

```
export MAKEOPTS="-DTIMER -DCONVEX -DHYBRID"
```

```
make
```

```
p_test -v 3 -n 50 -i 50 -r 1 -p 0 -d -ECMP > Rr1convex_no_sort.sts
```

```
p_test -v 4 -n 50 -i 50 -r 1 -p 0 -d -ECMP >> Rr1convex_no_sort.sts
```

```
p_test -v 10 -n 50 -i 50 -r 1 -p 0 -d -ECMP >> Rrlconvex_no_sort.sts
p_test -v 20 -n 50 -i 50 -r 1 -p 0 -d -ECMP >> Rrlconvex_no_sort.sts
p_test -v 50 -n 50 -i 50 -r 1 -p 0 -d -ECMP >> Rrlconvex_no_sort.sts
p_test -v 100 -n 50 -i 50 -r 1 -p 0 -d -ECMP >> Rrlconvex_no_sort.sts
p_test -v 1000 -n 50 -i 50 -r 1 -p 0 -d -ECMP >> Rrlconvex_no_sort.sts
```

```
# random triangles (vs. regular triangles)
```

```
p_test -v 3 -n 50 -i 50 -r 1 -p 0 -d -ECMP > Rr0convex_no_sort.sts
```

```
# Winding Number tests
```

```
make clean
```

```
export MAKEOPTS="-DTIMER -DRANDOM -DSORT -DWINDING"
```

```
make
```

```
# random polygons
```

```
p_test -v 3 -n 50 -i 50 -r 0 -p 1 -d -CW > Rr0winding.sts
p_test -v 4 -n 50 -i 50 -r 0 -p 1 -d -CW >> Rr0winding.sts
p_test -v 10 -n 50 -i 50 -r 0 -p 1 -d -CW >> Rr0winding.sts
p_test -v 20 -n 50 -i 50 -r 0 -p 1 -d -CW >> Rr0winding.sts
p_test -v 50 -n 50 -i 50 -r 0 -p 1 -d -CW >> Rr0winding.sts
p_test -v 100 -n 50 -i 50 -r 0 -p 1 -d -CW >> Rr0winding.sts
p_test -v 1000 -n 50 -i 50 -r 0 -p 1 -d -CW >> Rr0winding.sts
```

```
# regular polygons
```

```
p_test -v 3 -n 50 -i 50 -r 1 -p 0 -d -CW > Rrlwinding.sts
p_test -v 4 -n 50 -i 50 -r 1 -p 0 -d -CW >> Rrlwinding.sts
p_test -v 10 -n 50 -i 50 -r 1 -p 0 -d -CW >> Rrlwinding.sts
p_test -v 20 -n 50 -i 50 -r 1 -p 0 -d -CW >> Rrlwinding.sts
p_test -v 50 -n 50 -i 50 -r 1 -p 0 -d -CW >> Rrlwinding.sts
p_test -v 100 -n 50 -i 50 -r 1 -p 0 -d -CW >> Rrlwinding.sts
p_test -v 1000 -n 50 -i 50 -r 1 -p 0 -d -CW >> Rrlwinding.sts
```

# Run results from statrun.tst through this script, e.g.

# table.awk \*.stats

# to generate tables of timings

```
for dog in $*
do
cat $dog | awk '
BEGIN {
    count = -1
}
{
    if ( $1 == "Polygons" ) {
        count++
        if ( !count ) {
            title = $0
            title_on = 1
        }
        if ( $4 == "to" ) {
            nv[count] = $3"-"$5
        } else {
            nv[count] = $3
        }
    } else if ( title_on ) {
        if ( !count ) {
            if ( $1 == "Testing" ) {
                title_on = 0
            } else {
                title = title "\n" $0
            }
        }
    } else if ( $3 == "time:" ) {
        t[$1,count] = $4
        found[$1] = 1
    } else if ( $2 == "%" ) {
        pc[count] = $1
    }
}
END {
    print title

    printf( "\n\t\t Number of vertices\n\t" )
    for ( i = 0 ; i <= count ; i++ ) {
        printf( "\t\t %s",nv[i] )
    }
    printf( "\n\n" ) ;

    for ( name in found ) {
        printf( "%s\t", name )
        if ( length( name ) < 8 ) {
            printf( "\t" )
        }
        for ( i = 0 ; i <= count ; i++ ) {
            printf( "%8.1f",t[name,i] )
        }
        printf( "\n" ) ;
    }

    printf( "\ninside %\t" )
    for ( i = 0 ; i <= count ; i++ ) {
        printf( "%8.1f",pc[i] )
    }
}
```

```
    printf( "\n\n\n" ) ;  
}',  
done
```



```
/* rolling-ball-gems3.c */
/*
 * Take a 3D graphics object "obj" and the incremental mouse
 * motion data "(dx,dy,state)" and rotate the object
 * using the rolling ball algorithm. A generic interactive
 * graphics interface is assumed that maps onto common protocols
 * such as Xlib in an obvious way.
 *
 * It is assumed that an object "obj" of type "Polyhedron" has
 * already been drawn once on the display, and that it carries
 * a 4x4 frame matrix "obj->frame[i][j]" specifying the position
 * and orientation at which it is to be drawn.
 *
 * Make3DRot is assumed to construct a 4x4 rotation matrix from
 * the angle and axis parameters as shown in the text and in
 * GGI "Rotation Tools," by M. Pique, p.466.
 *
 * Make3DTranslation is assumed to store an (x,y,z) position in a 4x4
 * matrix in such a way that it can be extracted by Tx = obj->frame[3][0]; .
 *
 * CombineMatrices3D performs a left to right matrix multiplication,
 * leaving the result in the location specified by the last argument.
 *
 * CopyMatrix3D copies a matrix into another.
 *
 * Entry: display      - generic graphics device specification.
 *         obj         - the object (typically a wire frame) to be rotated.
 *         dx,dy       - the distance the mouse moved in screen coordinates
 *                       since the previous event was processed.
 *         state        - state of the mouse such as button presses.
 *
 * Exit: Object "obj" erased, rotated, and redrawn.
 */
```

```
#include <math.h>
```

```
#include "defs.h"
```

```
ModifyObject(display, obj, dx, dy, state)
```

```
    Display *display;
```

```
    Polyhedron *obj;
```

```
    int dx, dy, state;
```

```
{double Tx, Ty, Tz, n[3], dr, denom, cos_theta, sin_theta;
```

```
double Matrix3D[4][4], TmpMat[4][4], TransToOrigin[4][4], Rmat[4][4];
```

```
static double Radius=100.0;
```

```
/* Example of interactive method: apply rolling ball to
```

```
 * object's orientation as long as mouse Button1 is
```

```
 * held down.
```

```
*/
```

```
if (state == Button1)
```

```
{
    /* Obtain current object position from its frame. */
```

```
Tx = obj->frame[3][0];
```

```
Ty = obj->frame[3][1];
```

```
Tz = obj->frame[3][2];
```

```
/* Compute the rolling ball axis and angle from the incremental mouse
```

```
 * displacements (dx,dy) and compute corresponding rotation matrix RMat.
```

```
 * See text for full form of Rmat returned by Make3DRot.
```

```
 * NOTE: For window systems using a left-handed screen
```

```
 * coordinate system, the formula (-dy,dx,0) given
```

```

    * in the text for the rotation axis direction must
    * be changed to (+dy,dx,0) to give the desired effect!
    * We explicitly use this coordinate system in the example
    * code because so many systems possess this reversal.
    */
dr = sqrt((double)(dx*dx + dy*dy));
denom = sqrt(Radius*Radius + dr*dr);
cos_theta = Radius/denom;
sin_theta = dr/denom;
n[0] = (double)(dy)/dr;    /* Change sign for right-handed coord system. */
n[1] = (double)(dx)/dr;
n[2] = 0.0;
Make3DRot(cos_theta, sin_theta, n, Rmat);

/* Translate current object frame to origin. */
Make3DTranslation(-Tx, -Ty, -Tz, TransToOrigin);
CombineMatrices3D(TransToOrigin, obj->frame, Matrix3D);

/* Rotate about origin. */
CombineMatrices3D(Rmat, Matrix3D, TmpMat);

/* Translate rotated temporary frame back to original object position. */
Make3DTranslation(Tx, Ty, Tz, TransToOrigin);
CombineMatrices3D(TransToOrigin, TmpMat, Matrix3D);

/* Erase current object. */
SetFunction(display, ERASE);
DrawObject(display, obj);

/* Install new frame in object, draw rotated object. */
CopyMatrix3D(Matrix3D, obj->frame);
SetFunction(display, DRAW);
DrawObject(display, obj);

}}
```

```
/* Dummy include file of definitions to make rolling-ball compile. */
```

```
#define Button1 1
#define ERASE    0
#define DRAW     3
```

```
typedef struct {
    char *name;
    double frame[4][4];} Polyhedron;
```

```
typedef struct {
    char *name;
    int id;} Display;
```

```
/*
Ordered Dithering
by Stephen Hawley
from "Graphics Gems", Academic Press, 1990
*/

/* Program to generate dithering matrices.
 * written by Jim Blandy, Oberlin College, jimb@occs.oberlin.edu
 * Gifted to, documented and revised by Stephen Hawley,
 * sdh@flash.bellcore.com
 *
 * Generates a dithering matrix from the command line arguments.
 * The first argument, size, determines the dimensions of the
 * matrix: 2^size by 2^size
 * The optional range argument is the range of values to be
 * dithered over. By default, it is (2^size)^2, or simply the
 * total number of elements in the matrix.
 * The final output is suitable for inclusion in a C program.
 * A typical dithering function is something like this:
 * extern int dm[], size;
 *
 * int
 * dither(x,y, level)
 * register int x,y, level;
 * {
 *     return(level > dm[(x % size) + size * (y % size)]);
 * }
 */
```

```
main(argc, argv)
int argc;
char **argv;
{
    register int size, range;

    if (argc >= 2) size = atoi(argv[1]);
    else size = 2;

    if (argc == 3) range = atoi(argv[2]);
    else range = (1 << size) * (1 << size);

    prindither (size, range);
}
```

```
prindither (size, range)
register int size, range;
{
    register int l = (1 << size), i;
    /*
    * print a dithering matrix.
    * l is the length on a side.
    */
    range = range / (l * l);
    puts("int dm[] = {");
    for (i=0; i < l*l; i++) {
        if (i % l == 0) /* tab in 4 spaces per row */
            printf("    ");
        /* print the dither value for this location
        * scaled to the given range
        */
    }
}
```

```
        printf("%4d", range * dithervalue(i / l, i % l, size));

        /* commas after all but the last */
        if (i + 1 < l * l)
            putchar(',');
        /* newline at the end of the row */
        if ((i + 1) % l == 0)
            putchar('\n');
    }
    puts("\n}; ");
}

dithervalue(x, y, size)
register int x, y, size;
{
    register int d;
    /*
     * calculate the dither value at a particular
     * (x, y) over the size of the matrix.
     */
    d=0;
    while (size-->0)
    {
        /* Think of d as the density. At every iteration,
         * d is shifted left one and a new bit is put in the
         * low bit based on x and y. If x is odd and y is even,
         * or x is even and y is odd, a bit is put in. This
         * generates the checkerboard seen in dithering.
         * This quantity is shifted left again and the low bit of
         * y is added in.
         * This whole thing interleaves a checkerboard bit pattern
         * and y's bits, which is the value you want.
         */
        d = (d <<1 | (x&1 ^ y&1))<<1 | y&1;
        x >>= 1;
        y >>= 1;
    }
    return(d);
}
```

```
/*
 * Nice Numbers for Graph Labels
 * by Paul Heckbert
 * from "Graphics Gems", Academic Press, 1990
 */

/*
 * label.c: demonstrate nice graph labeling
 *
 * Paul Heckbert          2 Dec 88
 */

#include <stdio.h>
#include <math.h>
#include "GraphicsGems.h"

double nicenum();

/* expt(a,n)=a^n for integer n */

#ifdef POW_NOT_TRUSTWORTHY
/* if roundoff errors in pow cause problems, use this: */

double expt(a, n)
double a;
register int n;
{
    double x;

    x = 1.;
    if (n>0) for (; n>0; n--) x *= a;
    else for (; n<0; n++) x /= a;
    return x;
}

#else
#   define expt(a, n) pow(a, (double)(n))
#endif

#define NTICK 5                /* desired number of tick marks */

main(ac, av)
int ac;
char **av;
{
    double min, max;

    if (ac!=3) {
        fprintf(stderr, "Usage: label <min> <max>\n");
        exit(1);
    }
    min = atof(av[1]);
    max = atof(av[2]);

    loose_label(min, max);
}

/*
 * loose_label: demonstrate loose labeling of data range from min to max.
 * (tight method is similar)
 */
```

```
loose_label(min, max)
double min, max;
{
    char str[6], temp[20];
    int nfrac;
    double d;
    double graphmin, graphmax;
    double range, x;

    /* we expect min!=max */
    range = nicenum(max-min, 0);
    d = nicenum(range/(NTICK-1), 1);
    graphmin = floor(min/d)*d;
    graphmax = ceil(max/d)*d;
    nfrac = MAX(-floor(log10(d)), 0); /* # of fractional digits to show */
    sprintf(str, "%%.%df", nfrac); /* simplest axis labels */









    printf("graphmin=%g graphmax=%g increment=%g\n", graphmin, graphmax, d);
    for (x=graphmin; x<graphmax+.5*d; x+=d) {
        sprintf(temp, str, x);
        printf("(%s)\n", temp);
    }
}

/*
 * nicenum: find a "nice" number approximately equal to x.
 * Round the number if round=1, take ceiling if round=0
 */

double nicenum(x, round)
double x;
int round;
{
    int expv;
    double f;
    double nf;























    expv = floor(log10(x));
    f = x/expt(10., expv);
    if (round)
        if (f<1.5) nf = 1.;
        else if (f<3.) nf = 2.;
        else if (f<7.) nf = 5.;
        else nf = 10.;
    else
        if (f<=1.) nf = 1.;
        else if (f<=2.) nf = 2.;
        else if (f<=5.) nf = 5.;
        else nf = 10.;
    return nf*expt(10., expv);
}
```



# Index of /pubs/tog/GraphicsGems/gems/PolyScan/

Name	Last modified	Size	Description
 <a href="#">_Parent Directory</a>			
 <a href="#">_Makefile</a>	29-Jun-00 08:12	1K	
 <a href="#">_fancytest.c</a>	29-Jun-00 08:12	3K	
 <a href="#">_poly.c</a>	29-Jun-00 08:12	2K	
 <a href="#">_poly.h</a>	29-Jun-00 08:12	2K	
 <a href="#">_poly_clip.c</a>	29-Jun-00 08:12	4K	
 <a href="#">_poly_scan.c</a>	29-Jun-00 08:12	4K	
 <a href="#">_scantest.c</a>	29-Jun-00 08:12	1K	



# Index of /pubs/tog/GraphicsGems/gems/

Name	Last modified	Size	Description
 <a href="#">Parent Directory</a>			
 <a href="#">2DClip/</a>	29-Jun-00 08:12	1K	
 <a href="#">AALines/</a>	29-Jun-00 08:12	1K	
 <a href="#">AAPolyScan.c</a>	29-Jun-00 08:12	7K	
 <a href="#">Albers.c</a>	29-Jun-00 08:12	3K	
 <a href="#">AllGems.TOC</a>	13-Sep-00 22:26	26K	
 <a href="#">BinRec.c</a>	29-Jun-00 08:12	1K	
 <a href="#">BoundSphere.c</a>	03-Jan-01 11:17	3K	
 <a href="#">BoxSphere.c</a>	29-Jun-00 08:12	3K	
 <a href="#">CircleRect.c</a>	29-Jun-00 08:12	1K	
 <a href="#">ConcaveScan.c</a>	29-Jun-00 08:12	4K	
 <a href="#">DigitalLine.c</a>	29-Jun-00 08:12	1K	
 <a href="#">Dissolve.c</a>	29-Jun-00 08:12	4K	
 <a href="#">DoubleLine.c</a>	29-Jun-00 08:12	4K	
 <a href="#">Errata.GraphicsGems</a>	03-Jan-01 10:59	15K	
 <a href="#">FastJitter.c</a>	29-Jun-00 08:12	1K	
 <a href="#">FitCurves.c</a>	29-Jun-00 08:12	14K	
 <a href="#">FixedTrig.c</a>	29-Jun-00 08:12	1K	
 <a href="#">Forms.c</a>	29-Jun-00 08:12	5K	
 <a href="#">GGVecLib.c</a>	29-Jun-00 08:12	9K	
 <a href="#">GraphicsGems.h</a>	29-Jun-00 08:12	3K	
 <a href="#">HSLtoRGB.c</a>	29-Jun-00 08:12	1K	

	<a href="#">Hash3D.c</a>	29-Jun-00 08:12	1K
	<a href="#">HypotApprox.c</a>	29-Jun-00 08:12	1K
	<a href="#">Interleave.c</a>	29-Jun-00 08:12	6K
	<a href="#">Label.c</a>	29-Jun-00 08:12	2K
	<a href="#">LineEdge.c</a>	29-Jun-00 08:12	3K
	<a href="#">MANIFEST</a>	29-Jun-00 08:12	4K
	<a href="#">Makefile</a>	29-Jun-00 08:12	2K
	<a href="#">MatrixInvert.c</a>	29-Jun-00 08:12	4K
	<a href="#">MatrixOrtho.c</a>	29-Jun-00 08:12	1K
	<a href="#">MatrixPost.c</a>	29-Jun-00 08:12	3K
	<a href="#">Median.c</a>	29-Jun-00 08:12	2K
	<a href="#">NearestPoint.c</a>	29-Jun-00 08:12	12K
	<a href="#">OrderDither.c</a>	29-Jun-00 08:12	2K
	<a href="#">PixelInteger.c</a>	29-Jun-00 08:12	1K
	<a href="#">PntOnLine.c</a>	29-Jun-00 08:12	2K
	<a href="#">PolyScan/</a>	29-Jun-00 08:12	1K
	<a href="#">Quaternions.c</a>	29-Jun-00 08:12	2K
	<a href="#">README</a>	29-Jun-00 08:12	1K
	<a href="#">RGBTo4Bits.c</a>	29-Jun-00 08:12	1K
	<a href="#">RayBox.c</a>	29-Jun-00 08:12	1K
	<a href="#">RayPolygon.c</a>	29-Jun-00 08:12	1K
	<a href="#">Roots3And4.c</a>	29-Jun-00 08:12	4K
	<a href="#">SeedFill.c</a>	29-Jun-00 08:12	2K
	<a href="#">SquareRoot.c</a>	29-Jun-00 08:12	2K
	<a href="#">Sturm/</a>	29-Jun-00 08:12	1K
	<a href="#">TransBox.c</a>	29-Jun-00 08:12	1K



[\\_TriPoints.c](#)

29-Jun-00 08:12

2K



[\\_ViewTrans.c](#)

29-Jun-00 08:12

1K

```
# Makefile for scantest, test program for generic convex polygon scan conversion
#
# Note: fancytest.c needs a main routine and several auxiliary routines
# in order to be compiled.
```

```
CFLAGS = $(GENCFLAGS)
```

```
scantest: scantest.o poly_scan.o poly.o
    $(CC) $(CFLAGS) -o scantest scantest.o poly_scan.o poly.o -lm
```

```
clean:
    /bin/rm -f scantest.o poly_clip.o poly_scan.o poly.o scantest
```

```
/*
 * fancytest.c: subroutine illustrating the use of poly_clip and poly_scan
 * for Phong-shading and texture mapping.
 *
 * Note: lines enclosed in angle brackets '<', '>' should be replaced
 * with the code described.
 * Makes calls to hypothetical packages "shade", "image", "texture", "zbuffer".
 *
 * Paul Heckbert          Dec 1989
 */

#include <stdio.h>
#include <math.h>
#include "poly.h"

#define XMAX 1280          /* hypothetical image width */
#define YMAX 1024          /* hypothetical image height */
#define LIGHT_INTEN 255    /* light source intensity */

void pixelproc();

fancytest()
{
    int i;
    double WS[4][4];        /* world space to screen space transform */
    Poly p;                 /* a polygon */
    Poly_vert *v;

    static Poly_box box = {0, XMAX, 0, YMAX, -32678, 32767.99};
        /* 3-D screen clipping box, continuous coordinates */

    static Window win = {0, 0, XMAX-1, YMAX-1};
        /* 2-D screen clipping window, discrete coordinates */

    <initialize world space position (x,y,z), normal (nx,ny,nz), and texture
    position (u,v) at each vertex of p; set p.n>
    <set WS to world-to-screen transform>

    /* transform vertices from world space to homogeneous screen space */
    for (i=0; i<p.n; i++) {
        v = &p.vert[i];
        mx4_transform(v->x, v->y, v->z, 1., WS, &v->sx, &v->sy, &v->sz, &v->sw);
    }

    /* interpolate sx, sy, sz, sw, nx, ny, nz, u, v in poly_clip */
    p.mask = POLY_MASK(sx) | POLY_MASK(sy) | POLY_MASK(sz) | POLY_MASK(sw) |
        POLY_MASK(nx) | POLY_MASK(ny) | POLY_MASK(nz) |
        POLY_MASK(u) | POLY_MASK(v);

    poly_print("before clipping", &p);
    if (poly_clip_to_box(&p, &box) == POLY_CLIP_OUT)    /* clip polygon */
        return;                                          /* quit if off-screen */

    /* do homogeneous division of screen position and texture position */
    for (i=0; i<p.n; i++) {
        v = &p.vert[i];
        v->sx /= v->sw;
        v->sy /= v->sw;
        v->sz /= v->sw;
        v->u /= v->sw;
        v->v /= v->sw;
    }
}
```

```
        v->q = 1./v->sw;
    }
    /*
    * previously we ignored q (assumed q=1), henceforth ignore sw (assume sw=1)
    * Interpolate sx, sy, sz, nx, ny, nz, u, v, q in poly_scan
    */
    p.mask &= ~POLY_MASK(sw);
    p.mask |= POLY_MASK(q);

    poly_print("scan converting the polygon", &p);
    poly_scan(&p, &win, pixelproc);    /* scan convert! */
}

static void pixelproc(x, y, pt)          /* called at each pixel by poly_scan */
int x, y;
Poly_vert *pt;
{
    int sz, u, v, inten;
    double len, nor[3], diff, spec;

    sz = pt->sz;
    if (sz < zbuffer_read(x, y)) {
        len = sqrt(pt->nx*pt->nx + pt->ny*pt->ny + pt->nz*pt->nz);
        nor[0] = pt->nx/len;                /* unitize the normal vector */
        nor[1] = pt->ny/len;
        nor[2] = pt->nz/len;
        shade(nor, &diff, &spec);          /* compute specular and diffuse coeffs*/
        u = pt->u/pt->q;                    /* do homogeneous div. of texture pos */
        v = pt->v/pt->q;
        inten = texture_read(u, v)*diff + LIGHT_INTEN*spec;
        image_write(x, y, inten);
        zbuffer_write(x, y, sz);
    }
}

/* mx4_transform: transform 4-vector p by matrix m yielding q: q = p*m */

mx4_transform(px, py, pz, pw, m, qxp, qyp, qzp, qwp)
double px, py, pz, pw, m[4][4], *qxp, *qyp, *qzp, *qwp;
{
    *qxp = px*m[0][0] + py*m[1][0] + pz*m[2][0] + pw*m[3][0];
    *qyp = px*m[0][1] + py*m[1][1] + pz*m[2][1] + pw*m[3][1];
    *qzp = px*m[0][2] + py*m[1][2] + pz*m[2][2] + pw*m[3][2];
    *qwp = px*m[0][3] + py*m[1][3] + pz*m[2][3] + pw*m[3][3];
}
```

```
/*
 * poly.c: simple utilities for polygon data structures
 */

#include "poly.h"

Poly_vert *poly_dummy;          /* used superficially by POLY_MASK macro */

/*
 * poly_print: print Poly p to stdout, prefixed by the label str
 */

void poly_print(str, p)
char *str;
Poly *p;
{
    int i;

    printf("%s: %d sides\n", str, p->n);
    poly_vert_label("      ", p->mask);
    for (i=0; i<p->n; i++) {
        printf("    v[%d] ", i);
        poly_vert_print(" ", &p->vert[i], p->mask);
    }
}

void poly_vert_label(str, mask)
char *str;
unsigned long mask;
{
    printf("%s", str);
    if (mask&POLY_MASK(sx))    printf("    sx ");
    if (mask&POLY_MASK(sy))    printf("    sy ");
    if (mask&POLY_MASK(sz))    printf("    sz ");
    if (mask&POLY_MASK(sw))    printf("    sw ");
    if (mask&POLY_MASK(x))     printf("    x  ");
    if (mask&POLY_MASK(y))     printf("    y  ");
    if (mask&POLY_MASK(z))     printf("    z  ");
    if (mask&POLY_MASK(u))     printf("    u  ");
    if (mask&POLY_MASK(v))     printf("    v  ");
    if (mask&POLY_MASK(q))     printf("    q  ");
    if (mask&POLY_MASK(r))     printf("    r  ");
    if (mask&POLY_MASK(g))     printf("    g  ");
    if (mask&POLY_MASK(b))     printf("    b  ");
    if (mask&POLY_MASK(nx))    printf("    nx ");
    if (mask&POLY_MASK(ny))    printf("    ny ");
    if (mask&POLY_MASK(nz))    printf("    nz ");
    printf("\n");
}

void poly_vert_print(str, v, mask)
char *str;
Poly_vert *v;
unsigned long mask;
{
    printf("%s", str);
    if (mask&POLY_MASK(sx))    printf(" %6.1f", v->sx);
    if (mask&POLY_MASK(sy))    printf(" %6.1f", v->sy);
    if (mask&POLY_MASK(sz))    printf(" %6.2f", v->sz);
    if (mask&POLY_MASK(sw))    printf(" %6.2f", v->sw);
    if (mask&POLY_MASK(x))     printf(" %6.2f", v->x);
}
```

```
if (mask&POLY_MASK(y)) printf(" %6.2f", v->y);
if (mask&POLY_MASK(z)) printf(" %6.2f", v->z);
if (mask&POLY_MASK(u)) printf(" %6.2f", v->u);
if (mask&POLY_MASK(v)) printf(" %6.2f", v->v);
if (mask&POLY_MASK(q)) printf(" %6.2f", v->q);
if (mask&POLY_MASK(r)) printf(" %6.4f", v->r);
if (mask&POLY_MASK(g)) printf(" %6.4f", v->g);
if (mask&POLY_MASK(b)) printf(" %6.4f", v->b);
if (mask&POLY_MASK(nx)) printf(" %6.3f", v->nx);
if (mask&POLY_MASK(ny)) printf(" %6.3f", v->ny);
if (mask&POLY_MASK(nz)) printf(" %6.3f", v->nz);
printf("\n");
```

```
}
```



```
/* poly.h: definitions for polygon package */

#ifndef POLY_HDR
#define POLY_HDR

#define POLY_NMAX 10          /* max #sides to a polygon; change if needed */
/* note that poly_clip, given an n-gon as input, might output an (n+6)gon */
/* POLY_NMAX=10 is thus appropriate if input polygons are triangles or quads */

typedef struct {              /* A POLYGON VERTEX */
    double sx, sy, sz, sw;    /* screen space position (sometimes homo.) */
    double x, y, z;           /* world space position */
    double u, v, q;           /* texture position (sometimes homogeneous) */
    double r, g, b;           /* (red,green,blue) color */
    double nx, ny, nz;        /* world space normal vector */
} Poly_vert;
/* update poly.c if you change this structure */
/* note: don't put > 32 doubles in Poly_vert struct, or mask will overflow */

typedef struct {              /* A POLYGON */
    int n;                    /* number of sides */
    unsigned long mask;       /* interpolation mask for vertex elems */
    Poly_vert vert[POLY_NMAX]; /* vertices */
} Poly;
/*
 * mask is an interpolation mask whose kth bit indicates whether the kth
 * double in a Poly_vert is relevant.
 * For example, if the valid attributes are sx, sy, and sz, then set
 *     mask = POLY_MASK(sx) | POLY_MASK(sy) | POLY_MASK(sz);
 */

typedef struct {              /* A BOX (TYPICALLY IN SCREEN SPACE) */
    double x0, x1;            /* left and right */
    double y0, y1;            /* top and bottom */
    double z0, z1;            /* near and far */
} Poly_box;

typedef struct {              /* WINDOW: A DISCRETE 2-D RECTANGLE */
    int x0, y0;               /* xmin and ymin */
    int x1, y1;               /* xmax and ymax (inclusive) */
} Window;

#define POLY_MASK(elem) (1 << (&poly_dummy->elem - (double *)poly_dummy))

#define POLY_CLIP_OUT 0       /* polygon entirely outside box */
#define POLY_CLIP_PARTIAL 1    /* polygon partially inside */
#define POLY_CLIP_IN 2        /* polygon entirely inside box */

extern Poly_vert *poly_dummy; /* used superficially by POLY_MASK macro */

void poly_print(/* str, p */);
void poly_vert_label(/* str, mask */);
void poly_vert_print(/* str, v, mask */);
int poly_clip_to_box(/* pl, box */);
void poly_clip_to_halfspace(/* p, q, index, sign, k, name */);
void poly_scan(/* p, win, pixelproc */);

#endif
```

```
/*
 * Generic Convex Polygon Scan Conversion and Clipping
 * by Paul Heckbert
 * from "Graphics Gems", Academic Press, 1990
 */

/*
 * poly_clip.c: homogeneous 3-D convex polygon clipper
 *
 * Paul Heckbert          1985, Dec 1989
 */

#include <stdio.h>
#include "poly.h"

#define SWAP(a, b, temp)      {temp = a; a = b; b = temp;}
#define COORD(vert, i) ((double *) (vert))[i]

#define CLIP_AND_SWAP(elem, sign, k, p, q, r) { \
    poly_clip_to_halfspace(p, q, &v->elem-(double *)v, sign, sign*k); \
    if (q->n==0) {p1->n = 0; return POLY_CLIP_OUT;} \
    SWAP(p, q, r); \
}

/*
 * poly_clip_to_box: Clip the convex polygon p1 to the screen space box
 * using the homogeneous screen coordinates (sx, sy, sz, sw) of each vertex,
 * testing if v->sx/v->sw > box->x0 and v->sx/v->sw < box->x1,
 * and similar tests for y and z, for each vertex v of the polygon.
 * If polygon is entirely inside box, then POLY_CLIP_IN is returned.
 * If polygon is entirely outside box, then POLY_CLIP_OUT is returned.
 * Otherwise, if the polygon is cut by the box, p1 is modified and
 * POLY_CLIP_PARTIAL is returned.
 *
 * Given an n-gon as input, clipping against 6 planes could generate an
 * (n+6)gon, so POLY_NMAX in poly.h must be big enough to allow that.
 */

int poly_clip_to_box(p1, box)
register Poly *p1;
register Poly_box *box;
{
    int x0out = 0, x1out = 0, y0out = 0, y1out = 0, z0out = 0, z1out = 0;
    register int i;
    register Poly_vert *v;
    Poly p2, *p, *q, *r;

    if (p1->n+6>POLY_NMAX) {
        fprintf(stderr, "poly_clip_to_box: too many vertices: %d (max=%d-6)\n",
            p1->n, POLY_NMAX);
        exit(1);
    }
    if (sizeof(Poly_vert)/sizeof(double) > 32) {
        fprintf(stderr, "Poly_vert structure too big; must be <=32 doubles\n");
        exit(1);
    }

    /* count vertices "outside" with respect to each of the six planes */
    for (v=p1->vert, i=p1->n; i>0; i--, v++) {
        if (v->sx < box->x0*v->sw) x0out++;      /* out on left */
        if (v->sx > box->x1*v->sw) x1out++;      /* out on right */
    }
}
```

```
    if (v->sy < box->y0*v->sw) y0out++;      /* out on top */
    if (v->sy > box->y1*v->sw) y1out++;      /* out on bottom */
    if (v->sz < box->z0*v->sw) z0out++;      /* out on near */
    if (v->sz > box->z1*v->sw) z1out++;      /* out on far */
}

/* check if all vertices inside */
if (x0out+x1out+y0out+y1out+z0out+z1out == 0) return POLY_CLIP_IN;

/* check if all vertices are "outside" any of the six planes */
if (x0out==p1->n || x1out==p1->n || y0out==p1->n ||
    y1out==p1->n || z0out==p1->n || z1out==p1->n) {
    p1->n = 0;
    return POLY_CLIP_OUT;
}

/*
 * now clip against each of the planes that might cut the polygon,
 * at each step toggling between polygons p1 and p2
 */
p = p1;
q = &p2;
if (x0out) CLIP_AND_SWAP(sx, -1., box->x0, p, q, r);
if (x1out) CLIP_AND_SWAP(sx, 1., box->x1, p, q, r);
if (y0out) CLIP_AND_SWAP(sy, -1., box->y0, p, q, r);
if (y1out) CLIP_AND_SWAP(sy, 1., box->y1, p, q, r);
if (z0out) CLIP_AND_SWAP(sz, -1., box->z0, p, q, r);
if (z1out) CLIP_AND_SWAP(sz, 1., box->z1, p, q, r);

/* if result ended up in p2 then copy it to p1 */
if (p==&p2)
    bcopy(&p2, p1, sizeof(Poly)-(POLY_NMAX-p2.n)*sizeof(Poly_vert));
return POLY_CLIP_PARTIAL;
}

/*
 * poly_clip_to_halfspace: clip convex polygon p against a plane,
 * copying the portion satisfying sign*s[index] < k*sw into q,
 * where s is a Poly_vert* cast as a double*.
 * index is an index into the array of doubles at each vertex, such that
 * s[index] is sx, sy, or sz (screen space x, y, or z).
 * Thus, to clip against xmin, use
 *     poly_clip_to_halfspace(p, q, XINDEX, -1., -xmin);
 * and to clip against xmax, use
 *     poly_clip_to_halfspace(p, q, XINDEX, 1., xmax);
 */

void poly_clip_to_halfspace(p, q, index, sign, k)
Poly *p, *q;
register int index;
double sign, k;
{
    register unsigned long m;
    register double *up, *vp, *wp;
    register Poly_vert *v;
    int i;
    Poly_vert *u;
    double t, tu, tv;

    q->n = 0;
    q->mask = p->mask;
```

```
/* start with u=vert[n-1], v=vert[0] */
u = &p->vert[p->n-1];
tu = sign*COORD(u, index) - u->sw*k;
for (v= &p->vert[0], i=p->n; i>0; i--, u=v, tu=tv, v++) {
    /* on old polygon (p), u is previous vertex, v is current vertex */
    /* tv is negative if vertex v is in */
    tv = sign*COORD(v, index) - v->sw*k;
    if (tu<=0. ^ tv<=0.) {
        /* edge crosses plane; add intersection point to q */
        t = tu/(tu-tv);
        up = (double *)u;
        vp = (double *)v;
        wp = (double *)&q->vert[q->n];
        for (m=p->mask; m!=0; m>>=1, up++, vp++, wp++)
            if (m&1) *wp = *up+t*(*vp-*up);
        q->n++;
    }
    if (tv<=0.) /* vertex v is in, copy it to q */
        q->vert[q->n++] = *v;
}
}
```

```
/*
 * Generic Convex Polygon Scan Conversion and Clipping
 * by Paul Heckbert
 * from "Graphics Gems", Academic Press, 1990
 */

/*
 * poly_scan.c: point-sampled scan conversion of convex polygons
 *
 * Paul Heckbert          1985, Dec 1989
 */

#include <stdio.h>
#include <math.h>
#include "poly.h"

/*
 * poly_scan: Scan convert a polygon, calling pixelproc at each pixel with an
 * interpolated Poly_vert structure. Polygon can be clockwise or ccw.
 * Polygon is clipped in 2-D to win, the screen space window.
 *
 * Scan conversion is done on the basis of Poly_vert fields sx and sy.
 * These two must always be interpolated, and only they have special meaning
 * to this code; any other fields are blindly interpolated regardless of
 * their semantics.
 *
 * The pixelproc subroutine takes the arguments:
 *
 *     pixelproc(x, y, point)
 *     int x, y;
 *     Poly_vert *point;
 *
 * All the fields of point indicated by p->mask will be valid inside pixelproc
 * except sx and sy. If they were computed, they would have values
 * sx=x+.5 and sy=y+.5, since sampling is done at pixel centers.
 */

void poly_scan(p, win, pixelproc)
register Poly *p;           /* polygon */
Window *win;               /* 2-D screen space clipping window */
void (*pixelproc)();       /* procedure called at each pixel */
{
    register int i, li, ri, y, ly, ry, top, rem;
    register unsigned long mask;
    double ymin;
    Poly_vert l, r, dl, dr;

    if (p->n>POLY_NMAX) {
        fprintf(stderr, "poly_scan: too many vertices: %d\n", p->n);
        return;
    }
    if (sizeof(Poly_vert)/sizeof(double) > 32) {
        fprintf(stderr, "Poly_vert structure too big; must be <=32 doubles\n");
        exit(1);
    }

    ymin = HUGE;
    for (i=0; i<p->n; i++)          /* find top vertex (y points down) */
        if (p->vert[i].sy < ymin) {
            ymin = p->vert[i].sy;
            top = i;
        }
}
```

```
}
```

```
li = ri = top;          /* left and right vertex indices */
rem = p->n;              /* number of vertices remaining */
y = ceil(ymin-.5);      /* current scan line */
ly = ry = y-1;          /* lower end of left & right edges */
mask = p->mask & ~POLY_MASK(sy); /* stop interpolating screen y */
```

```
while (rem>0) {          /* scan in y, activating new edges on left & right */
    /* as scan line passes over new vertices */
```

```
    while (ly<=y && rem>0) { /* advance left edge? */
        rem--;
        i = li-1;            /* step ccw down left side */
        if (i<0) i = p->n-1;
        incrementalize_y(&p->vert[li], &p->vert[i], &l, &dl, y, mask);
        ly = floor(p->vert[i].sy+.5);
        li = i;
    }
```

```
    while (ry<=y && rem>0) { /* advance right edge? */
        rem--;
        i = ri+1;            /* step cw down right edge */
        if (i>=p->n) i = 0;
        incrementalize_y(&p->vert[ri], &p->vert[i], &r, &dr, y, mask);
        ry = floor(p->vert[i].sy+.5);
        ri = i;
    }
```

```
    while (y<ly && y<ry) { /* do scanlines till end of l or r edge */
        if (y>=win->y0 && y<=win->y1)
            if (l.sx<=r.sx) scanline(y, &l, &r, win, pixelproc, mask);
            else scanline(y, &r, &l, win, pixelproc, mask);
        y++;
        increment(&l, &dl, mask);
        increment(&r, &dr, mask);
    }
```

```
/* scanline: output scanline by sampling polygon at Y=y+.5 */
```

```
static scanline(y, l, r, win, pixelproc, mask)
int y;
unsigned long mask;
Poly_vert *l, *r;
Window *win;
void (*pixelproc)();
{
    int x, lx, rx;
    Poly_vert p, dp;

    mask &= ~POLY_MASK(sx); /* stop interpolating screen x */
    lx = ceil(l->sx-.5);
    if (lx<win->x0) lx = win->x0;
    rx = floor(r->sx-.5);
    if (rx>win->x1) rx = win->x1;
    if (lx>rx) return;
    incrementalize_x(l, r, &p, &dp, lx, mask);
    for (x=lx; x<=rx; x++) { /* scan in x, generating pixels */
        (*pixelproc)(x, y, &p);
        increment(&p, &dp, mask);
    }
```

```
    }
}

/*
 * incrementalize_y: put intersection of line Y=y+.5 with edge between points
 * p1 and p2 in p, put change with respect to y in dp
 */

static incrementalize_y(p1, p2, p, dp, y, mask)
register double *p1, *p2, *p, *dp;
register unsigned long mask;
int y;
{
    double dy, frac;

    dy = ((Poly_vert *)p2)->sy - ((Poly_vert *)p1)->sy;
    if (dy==0.) dy = 1.;
    frac = y+.5 - ((Poly_vert *)p1)->sy;

    for (; mask!=0; mask>>=1, p1++, p2++, p++, dp++)
        if (mask&1) {
            *dp = (*p2-*p1)/dy;
            *p = *p1+*dp*frac;
        }
}

/*
 * incrementalize_x: put intersection of line X=x+.5 with edge between points
 * p1 and p2 in p, put change with respect to x in dp
 */

static incrementalize_x(p1, p2, p, dp, x, mask)
register double *p1, *p2, *p, *dp;
register unsigned long mask;
int x;
{
    double dx, frac;

    dx = ((Poly_vert *)p2)->sx - ((Poly_vert *)p1)->sx;
    if (dx==0.) dx = 1.;
    frac = x+.5 - ((Poly_vert *)p1)->sx;

    for (; mask!=0; mask>>=1, p1++, p2++, p++, dp++)
        if (mask&1) {
            *dp = (*p2-*p1)/dx;
            *p = *p1+*dp*frac;
        }
}

static increment(p, dp, mask)
register double *p, *dp;
register unsigned long mask;
{
    for (; mask!=0; mask>>=1, p++, dp++)
        if (mask&1)
            *p += *dp;
}
```

```
/*
 * scantest.c: use poly_scan() for Gouraud shading and z-buffer demo.
 * Given the screen space X, Y, and Z of N-gon on command line,
 * print out all pixels during scan conversion.
 * This code could easily be modified to actually read and write pixels.
 *
 * Paul Heckbert          Dec 1989
 */

#include <stdio.h>
#include <math.h>
#include "poly.h"

#define XMAX 1280          /* hypothetical image width */
#define YMAX 1024          /* hypothetical image height */

#define FRANDOM() ((rand()&32767)/32767.) /* random number between 0 and 1 */

void pixelproc();

main(ac, av)
int ac;
char **av;
{
    int i;
    Poly p;
    static Window win = {0, 0, XMAX-1, YMAX-1}; /* screen clipping window */

    if (ac<2 || ac != 2+3*(p.n = atoi(av[1]))) {
        fprintf(stderr, "Usage: scantest N X1 Y1 Z1 X2 Y2 Z2 ... XN YN ZN\n");
        exit(1);
    }
    for (i=0; i<p.n; i++) {
        p.vert[i].sx = atof(av[2+3*i]); /* set screen space x,y,z */
        p.vert[i].sy = atof(av[3+3*i]);
        p.vert[i].sz = atof(av[4+3*i]);
        p.vert[i].r = FRANDOM();          /* random vertex colors, for kicks */
        p.vert[i].g = FRANDOM();
        p.vert[i].b = FRANDOM();
    }
    /* interpolate sx, sy, sz, r, g, and b in poly_scan */
    p.mask = POLY_MASK(sx) | POLY_MASK(sy) | POLY_MASK(sz) |
        POLY_MASK(r) | POLY_MASK(g) | POLY_MASK(b);

    poly_print("scan converting the polygon", &p);

    poly_scan(&p, &win, pixelproc);      /* scan convert! */
}

static void pixelproc(x, y, point)      /* called at each pixel by poly_scan */
int x, y;
Poly_vert *point;
{
    printf("pixel (%d,%d) screenz=%g rgb=(%g,%g,%g)\n",
        x, y, point->sz, point->r, point->g, point->b);

    /*
     * in real graphics program you could read and write pixels, e.g.:
     *
     * if (point->sz < zbuffer_read(x, y)) {
     *     image_write_rgb(x, y, point->r, point->g, point->b);
     */
}
```



```
    *      zbuffer_write(x, y, point->sz);  
    *    }  
    */  
}
```

```
/*
 * Concave Polygon Scan Conversion
 * by Paul Heckbert
 * from "Graphics Gems", Academic Press, 1990
 */

/*
 * concave: scan convert nvert-sided concave non-simple polygon with vertices at
 * (point[i].x, point[i].y) for i in [0..nvert-1] within the window win by
 * calling spanproc for each visible span of pixels.
 * Polygon can be clockwise or counterclockwise.
 * Algorithm does uniform point sampling at pixel centers.
 * Inside-outside test done by Jordan's rule: a point is considered inside if
 * an emanating ray intersects the polygon an odd number of times.
 * drawproc should fill in pixels from xl to xr inclusive on scanline y,
 * e.g:
 *
 *     drawproc(y, xl, xr)
 *     int y, xl, xr;
 *     {
 *         int x;
 *         for (x=xl; x<=xr; x++)
 *             pixel_write(x, y, pixelvalue);
 *     }
 *
 * Paul Heckbert          30 June 81, 18 Dec 89
 */
```

```
#include <stdio.h>
#include <math.h>
#include "GraphicsGems.h"

#define ALLOC(ptr, type, n)  ASSERT(ptr = (type *)malloc((n)*sizeof(type)))

typedef struct {
    int x0, y0;          /* xmin and ymin */
    int x1, y1;          /* xmax and ymax (inclusive) */
} Window;

typedef struct {
    double x;            /* a polygon edge */
    double dx;           /* x coordinate of edge's intersection with current scanline */
    double dy;           /* change in x with respect to y */
    int i;               /* edge number: edge i goes from pt[i] to pt[i+1] */
} Edge;

static int n;           /* number of vertices */
static Point2 *pt;      /* vertices */

static int nact;        /* number of active edges */
static Edge *active;    /* active edge list: edges crossing scanline y */

int compare_ind(), compare_active();

concave(nvert, point, win, spanproc)
int nvert;              /* number of vertices */
Point2 *point;          /* vertices of polygon */
Window *win;            /* screen clipping window */
void (*spanproc)();     /* called for each span of pixels */
{
    int k, y0, y1, y, i, j, xl, xr;
    int *ind;            /* list of vertex indices, sorted by pt[ind[j]].y */
```

```
n = nvert;
pt = point;
if (n<=0) return;
ALLOC(ind, int, n);
ALLOC(active, Edge, n);

/* create y-sorted array of indices ind[k] into vertex list */
for (k=0; k<n; k++)
    ind[k] = k;
qsort(ind, n, sizeof ind[0], compare_ind); /* sort ind by pt[ind[k]].y */

nact = 0; /* start with empty active list */
k = 0; /* ind[k] is next vertex to process */
y0 = MAX(win->y0, ceil(pt[ind[0]].y-.5)); /* ymin of polygon */
y1 = MIN(win->y1, floor(pt[ind[n-1]].y-.5)); /* ymax of polygon */

for (y=y0; y<=y1; y++) { /* step through scanlines */
    /* scanline y is at y+.5 in continuous coordinates */

    /* check vertices between previous scanline and current one, if any */
    for (; k<n && pt[ind[k]].y<=y+.5; k++) {
        /* to simplify, if pt.y=y+.5, pretend it's above */
        /* invariant: y-.5 < pt[i].y <= y+.5 */
        i = ind[k];
        /*
         * insert or delete edges before and after vertex i (i-1 to i,
         * and i to i+1) from active list if they cross scanline y
         */
        j = i>0 ? i-1 : n-1; /* vertex previous to i */
        if (pt[j].y <= y-.5) /* old edge, remove from active list */
            cdelete(j);
        else if (pt[j].y > y+.5) /* new edge, add to active list */
            cinsert(j, y);
        j = i<n-1 ? i+1 : 0; /* vertex next after i */
        if (pt[j].y <= y-.5) /* old edge, remove from active list */
            cdelete(i);
        else if (pt[j].y > y+.5) /* new edge, add to active list */
            cinsert(i, y);
    }

    /* sort active edge list by active[j].x */
    qsort(active, nact, sizeof active[0], compare_active);

    /* draw horizontal segments for scanline y */
    for (j=0; j<nact; j+=2) { /* draw horizontal segments */
        /* span 'tween j & j+1 is inside, span tween j+1 & j+2 is outside */
        xl = ceil(active[j].x-.5); /* left end of span */
        if (xl<win->x0) xl = win->x0;
        xr = floor(active[j+1].x-.5); /* right end of span */
        if (xr>win->x1) xr = win->x1;
        if (xl<=xr)
            (*spanproc)(y, xl, xr); /* draw pixels in span */
        active[j].x += active[j].dx; /* increment edge coords */
        active[j+1].x += active[j+1].dx;
    }
}

static cdelete(i) /* remove edge i from active list */
int i;
{
```

```
    int j;

    for (j=0; j<nact && active[j].i!=i; j++);
    if (j>=nact) return;          /* edge not in active list; happens at win->y0*/
    nact--;
    bcopy(&active[j+1], &active[j], (nact-j)*sizeof active[0]);
}

static cinsert(i, y)              /* append edge i to end of active list */
int i, y;
{
    int j;
    double dx;
    Point2 *p, *q;

    j = i<n-1 ? i+1 : 0;
    if (pt[i].y < pt[j].y) {p = &pt[i]; q = &pt[j];}
    else {p = &pt[j]; q = &pt[i];}
    /* initialize x position at intersection of edge with scanline y */
    active[nact].dx = dx = (q->x-p->x)/(q->y-p->y);
    active[nact].x = dx*(y+.5-p->y)+p->x;
    active[nact].i = i;
    nact++;
}

/* comparison routines for qsort */
compare_ind(u, v) int *u, *v; {return pt[*u].y <= pt[*v].y ? -1 : 1;}
compare_active(u, v) Edge *u, *v; {return u->x <= v->x ? -1 : 1;}
```

```
/*
 * Digital Line Drawing
 * by Paul Heckbert
 * from "Graphics Gems", Academic Press, 1990
 */

/*
 * digline: draw digital line from (x1,y1) to (x2,y2),
 * calling a user-supplied procedure at each pixel.
 * Does no clipping.  Uses Bresenham's algorithm.
 *
 * Paul Heckbert          3 Sep 85
 */
```

```
#include "GraphicsGems.h"
```

```
digline(x1, y1, x2, y2, dotproc)
int x1, y1, x2, y2;
void (*dotproc)();
{
    int d, x, y, ax, ay, sx, sy, dx, dy;

    dx = x2-x1;  ax = ABS(dx)<<1;  sx = SGN(dx);
    dy = y2-y1;  ay = ABS(dy)<<1;  sy = SGN(dy);

    x = x1;
    y = y1;
    if (ax>ay) {                      /* x dominant */
        d = ay-(ax>>1);
        for (;;) {
            (*dotproc)(x, y);
            if (x==x2) return;
            if (d>=0) {
                y += sy;
                d -= ax;
            }
            x += sx;
            d += ay;
        }
    }
    else {                            /* y dominant */
        d = ax-(ay>>1);
        for (;;) {
            (*dotproc)(x, y);
            if (y==y2) return;
            if (d>=0) {
                x += sx;
                d -= ay;
            }
            y += sy;
            d += ax;
        }
    }
}
```

```
/*
 * Recording Animation in Binary Order for Progressive Temporal Refinement
 * by Paul Heckbert
 * from "Graphics Gems", Academic Press, 1990
 */

/*
 * binrec.c: demonstrate binary recording order
 *
 * Paul Heckbert          Jan 90
 */

#include <stdio.h>

main(ac, av)
int ac;
char **av;
{
    int nframes, i, start_frame, repeat_count;
    if (ac!=2) {
        fprintf(stderr, "Usage: binrec <nframes>\n");
        exit(1);
    }
    nframes = atoi(av[1]);

    printf("step startframe repeatcount\n");
    for (i=0; i<nframes; i++) {
        inside_out(nframes, i, &start_frame, &repeat_count);
        printf(" %2d      %2d          %2d\n", i, start_frame, repeat_count);
    }
}

/*
 * inside_out: turn a number "inside-out": a generalization of bit-reversal.
 * For n = power of two, this is equivalent to bit-reversal.
 *
 * Turn the number a inside-out, yielding b.  If 0<=a<n then 0<=b<n.
 * Also return r = min(n-b, largest power of 2 dividing b)
 */

inside_out(n, a, b, r)
int n, a, *b, *r;
{
    int k, m;

    *r = m = n;
    for (*b=0, k=1; k<n; k<<=1)
        if (a<<1>=m) {
            if (*b==0) *r = k;
            *b += k;
            a -= (m+1)>>1;
            m >= 1;
        }
        else m = (m+1)>>1;
    if (*r>n-*b) *r = n-*b;
}
```

```
/*
 * A Seed Fill Algorithm
 * by Paul Heckbert
 * from "Graphics Gems", Academic Press, 1990
 *
 * user provides pixelread() and pixelwrite() routines
 */

/*
 * fill.c : simple seed fill program
 * Calls pixelread() to read pixels, pixelwrite() to write pixels.
 *
 * Paul Heckbert          13 Sept 1982, 28 Jan 1987
 */

typedef struct {
    int x0, y0;          /* xmin and ymin */
    int x1, y1;          /* xmax and ymax (inclusive) */
} Window;

typedef int Pixel;       /* 1-channel frame buffer assumed */

Pixel pixelread();

typedef struct {short y, xl, xr, dy;} Segment;
/*
 * Filled horizontal segment of scanline y for xl<=x<=xr.
 * Parent segment was on line y-dy. dy=1 or -1
 */

#define MAX 10000        /* max depth of stack */

#define PUSH(Y, XL, XR, DY) /* push new segment on stack */ \
    if (sp<stack+MAX && Y+(DY)>=win->y0 && Y+(DY)<=win->y1) \
    {sp->y = Y; sp->xl = XL; sp->xr = XR; sp->dy = DY; sp++;}

#define POP(Y, XL, XR, DY) /* pop segment off stack */ \
    {sp--; Y = sp->y+(DY = sp->dy); XL = sp->xl; XR = sp->xr;}

/*
 * fill: set the pixel at (x,y) and all of its 4-connected neighbors
 * with the same pixel value to the new pixel value nv.
 * A 4-connected neighbor is a pixel above, below, left, or right of a pixel.
 */

fill(x, y, win, nv)
int x, y;          /* seed point */
Window *win;       /* screen window */
Pixel nv;          /* new pixel value */
{
    int l, x1, x2, dy;
    Pixel ov;      /* old pixel value */
    Segment stack[MAX], *sp = stack; /* stack of filled segments */

    ov = pixelread(x, y); /* read pv at seed point */
    if (ov==nv || x<win->x0 || x>win->x1 || y<win->y0 || y>win->y1) return;
    PUSH(y, x, x, 1); /* needed in some cases */
    PUSH(y+1, x, x, -1); /* seed segment (popped 1st) */








    while (sp>stack) {
        /* pop segment off stack and fill a neighboring scan line */

```

```
POP(y, x1, x2, dy);
/*
 * segment of scan line y-dy for x1<=x<=x2 was previously filled,
 * now explore adjacent pixels in scan line y
 */
for (x=x1; x<=win->x0 && pixelread(x, y)==ov; x--)
    pixelwrite(x, y, nv);
if (x>=x1) goto skip;
l = x+1;
if (l<x1) PUSH(y, l, x1-1, -dy);          /* leak on left? */
x = x1+1;
do {
    for (; x<=win->x1 && pixelread(x, y)==ov; x++)
        pixelwrite(x, y, nv);
    PUSH(y, l, x-1, dy);
    if (x>x2+1) PUSH(y, x2+1, x-1, -dy);    /* leak on right? */
skip:   for (x++; x<=x2 && pixelread(x, y)!=ov; x++);
        l = x;
    } while (x<=x2);
    }
}
```



# Index of /pubs/tog/GraphicsGems/gemsiv/minray/

Name	Last modified	Size	Description
 <a href="#">Parent Directory</a>			
 <a href="#">README</a>	29-Jun-00 08:20	1K	
 <a href="#">minray.c</a>	29-Jun-00 08:20	2K	
 <a href="#">minray.card.c</a>	29-Jun-00 08:20	1K	
 <a href="#">minray.post</a>	29-Jun-00 08:20	52K	
 <a href="#">minray.ps</a>	29-Jun-00 08:20	6K	
 <a href="#">ray.h</a>	29-Jun-00 08:20	1K	

C code from the article

"A Minimal Ray Tracer"

by Paul S. Heckbert, [ph@cs.cmu.edu](mailto:ph@cs.cmu.edu)

in "Graphics Gems IV", Academic Press, 1994

files:

README	- this file
minray.c	- "readable" minimal ray tracer
minray.card.c	- minimal ray tracer for business card
ray.h	- header file for test scene
minray.ps	- John Hartman's postscript ray tracer (on UNIX, run "gs" or "lpr" on it)
minray.post	- messages posted to comp.graphics announcing contest and results. Contains three shell archives of all the contest entries.

Note that there may be some tricks used in this code that are not portable to your machine!

```
/* minimal ray tracer, hybrid version - 888 tokens
 * Paul Heckbert, ucbvax!pixar!ph, 13 Jun 87
 * Using tricks from Darwyn Peachey and Joe Cychosz. */

#define TOL 1e-7
#define AMBIENT vec U, black, amb
#define SPHERE struct sphere {vec cen, color; double rad, kd, ks, kt, kl, ir} \
    *s, *best, sph[]
typedef struct {double x, y, z} vec;
#include "ray.h"
yx;
double u, b, tmin, sqrt(), tan();

double vdot(A, B)
vec A, B;
{
    return A.x*B.x + A.y*B.y + A.z*B.z;
}

vec vcomb(a, A, B)      /* aA+B */
double a;
vec A, B;
{
    B.x += a*A.x;
    B.y += a*A.y;
    B.z += a*A.z;
    return B;
}

vec vunit(A)
vec A;
{
    return vcomb(1./sqrt(vdot(A, A)), A, black);
}

struct sphere *intersect(P, D)
vec P, D;
{
    best = 0;
    tmin = 1e30;
    s = sph+NSPHERE;
    while (s-->sph)
        b = vdot(D, U = vcomb(-1., P, s->cen)),
        u = b*b-vdot(U, U)+s->rad*s->rad,
        u = u>0 ? sqrt(u) : 1e31,
        u = b-u>TOL ? b-u : b+u,
        tmin = u>=TOL && u<tmin ?
            best = s, u : tmin;
    return best;
}

vec trace(level, P, D)
vec P, D;
{
    double d, eta, e;
    vec N, color;
    struct sphere *s, *l;

    if (!level--) return black;
    if (s = intersect(P, D));
    else return amb;
}
```

```
color = amb;
eta = s->ir;
d = -vdot(D, N = vunit(vcomb(-1., P = vcomb(tmin, D, P), s->cen)));
if (d<0)
    N = vcomb(-1., N, black),
    eta = 1/eta,
    d = -d;
l = sph+NSPHERE;
while (l-->sph)
    if ((e = l->kl*vdot(N, U = vunit(vcomb(-1., P, l->cen)))) > 0 &&
        intersect(P, U)==l)
        color = vcomb(e, l->color, color);
U = s->color;
color.x *= U.x;
color.y *= U.y;
color.z *= U.z;
e = 1-eta*eta*(1-d*d);
/* the following is non-portable: we assume right to left arg evaluation.
 * (use U before call to trace, which modifies U) */
return vcomb(s->kt,
    e>0 ? trace(level, P, vcomb(eta, D, vcomb(eta*d-sqrt(e), N, black)))
        : black,
    vcomb(s->ks, trace(level, P, vcomb(2*d, N, D)),
    vcomb(s->kd, color, vcomb(s->kl, U, black))));
}
```

```
main()
{
    printf("%d %d\n", SIZE, SIZE);
    while (yx<SIZE*SIZE)
        U.x = yx%SIZE-SIZE/2,
        U.z = SIZE/2-yx++/SIZE,
        U.y = SIZE/2/tan(AOV/114.5915590261), /* 360/PI~114 */
        U = vcomb(255., trace(DEPTH, black, vunit(U)), black),
        printf("%.0f %.0f %.0f\n", U); /* yowsa! non-portable! */
}
```

```
typedef struct{double x,y,z}vec;vec U,black,amb={.02,.02,.02};struct sphere{
vec cen,color;double rad,kd,ks,kt,kl,ir}*s,*best,sph[]={0.,6.,.5,1.,1.,1.,.9,
.05,.2,.85,0.,1.7,-1.,8.,-.5,1.,.5,.2,1.,.7,.3,0.,.05,1.2,1.,8.,-.5,.1,.8,.8,
1.,.3,.7,0.,0.,1.2,3.,-6.,15.,1.,.8,1.,7.,0.,0.,0.,.6,1.5,-3.,-3.,12.,.8,1.,
1.,5.,0.,0.,0.,.5,1.5,};yx;double u,b,tmin,sqrt(),tan();double vdot(A,B)vec A
,B;{return A.x*B.x+A.y*B.y+A.z*B.z;}vec vcomb(a,A,B)double a;vec A,B;{B.x+=a*
A.x;B.y+=a*A.y;B.z+=a*A.z;return B;}vec vunit(A)vec A;{return vcomb(1./sqrt(
vdot(A,A)),A,black);}struct sphere*intersect(P,D)vec P,D;{best=0;tmin=1e30;s=
sph+5;while(s-->sph)b=vdot(D,U=vcomb(-1.,P,s->cen)),u=b*b-vdot(U,U)+s->rad*s
->rad,u=u>0?sqrt(u):1e31,u=b-u>1e-7?b-u:b+u,tmin=u>=1e-7&&u<tmin?best=s,u:
tmin;return best;}vec trace(level,P,D)vec P,D;{double d,eta,e;vec N,color;
struct sphere*s,*l;if(!level--)return black;if(s=intersect(P,D));else return
amb;color=amb;eta=s->ir;d= -vdot(D,N=vunit(vcomb(-1.,P=vcomb(tmin,D,P),s->cen
)));if(d<0)N=vcomb(-1.,N,black),eta=1/eta,d= -d;l=sph+5;while(l-->sph)if((e=l
->kl*vdot(N,U=vunit(vcomb(-1.,P,l->cen))))>0&&intersect(P,U)==l)color=vcomb(e
,l->color,color);U=s->color;color.x*=U.x;color.y*=U.y;color.z*=U.z;e=1-eta*
eta*(1-d*d);return vcomb(s->kt,e>0?trace(level,P,vcomb(eta,D,vcomb(eta*d-sqrt
(e),N,black))):black,vcomb(s->ks,trace(level,P,vcomb(2*d,N,D)),vcomb(s->kd,
color,vcomb(s->kl,U,black))));}main(){printf("%d %d\n",32,32);while(yx<32*32)
U.x=yx%32-32/2,U.z=32/2-yx++/32,U.y=32/2/tan(25/114.5915590261),U=vcomb(255.,
trace(3,black,vunit(U)),black),printf("%.0f %.0f %.0f\n",U);}/*minray!*/
```

```
Path: minray.ph
From: jbm@leisr.CDCP (Paul Heckbert)
Newsgroup: comp.graphics
Subject: Announcement: MINIMAL RAY TRACER PROGRAMMING CONTEST
Date: 4 May 87 21:52:33 GMT
Organization: Pixar -- Marin County, California

# to unpack, cut here and run the following through sh
# shell archive of: rules.squease.c ntok.cah ray.h test1.ap
#
cat << EOF1582 >>rules
*****
MINIMAL RAY TRACER PROGRAMMING CONTEST
*****

The goal: write the shortest Whitted-style ray tracing program in C.
Countless people have written basic sphere ray tracers and made pictures of
planes and chrome balls, so I can't it time we found out just how short such a
program can be?
```

The algorithms required are described in the classic article:

```
Turner Whitted
"An Improved Illumination Model for Shaded Display"
Communications of the ACM, June 1980, pp. 343-349
```

Given Kajiya's article, Whitted's article, and a basic knowledge of 3-D graphics, anyone should be able to enter this contest. Winning the contest will require intimate knowledge of C and considerable ingenuity, however.

Briefly, the rules are as follows: you mail me a C source to a ray tracer which renders spheres with specular and transmitted rays and hard shadows but no antialiasing. The scene is compiled in, and the picture is output to stdout. Speed is no object. The winner will be the shortest C program which produces the "correct" output, where shortest is measured by the number of tokens after the C preprocessor. The deadline is 15 June 1987.

Files in this shell archive are:

- rules contact announcement and rules
- squease.c program for token counting
- ntok.cah C shell alias for token counting
- ray.h sphere list for a test scene
- test1.ap a correct image of that scene, for reference

\*\*\*\* CONTEST RULES \*\*\*\*

An entry is considered valid if it meets all of the following conditions:

1. program consists of a single C source file (called ray.c, ray)
2. compile with no errors or warnings on 4.3bsd on my VAX 780 with a command of the form 'cc -o ray ray.c -la' (sorry, but this is the only way I can verify entries consistently). Thus, "recursive" C features such as structure passing and union are legal.
3. ray traces a list of spheres using Whitted's recursive shading model for the specular and transmitted (refracted) components.
4. the sphere list, camera, and other parameters (listed below) are compiled into the program from a header file (called minray.h). The program must work for any set of such parameters, except where noted below (each entry will be compiled and tested with several different header files); the following are specified in the header file:
  - a sphere has at least the following attributes
    - center point (x,y,z)
    - color (r,g,b) with 0<=r,g,b<=1
    - radius
    - diffuse, specular, transmitted, and luminosity coefficients
    - call them kd, ks, kt, kl respectively
    - index of refraction ir
  - and the following constants are #defined:
    - DEPTH maximum ray tree depth
    - depth=0 means return black (0,0,0)
    - depth=1 means shade only primary rays
    - depth=2 means primary and secondary, etc.
    - SIZE resolution of picture in pixels (it's square)
    - AOV total angle of view in degrees
    - NSPHERE number of spheres in scene

The sphere list and ambient color are initialized in ray.h using the identifiers 'SPHERE' and 'AMBIENT', which you will need to #define before #including that file. SPHERE and AMBIENT can be arrays of doubles, structures, or whatever you like. For example:

```
#define SPHERE struct sphere sph[NSPHERE]
```

Your program should compile with the enclosed ray.h.

5. use the following shading model:

```
if a ray intersects sphere i, then the color returned along that ray is:
C = kd[i]*SURFCOLOR[i]
  + ks[i]*CS + kt[i]*CT + kl[i]*AMBIENT
if a ray misses all spheres, then the color returned is:
C = AMBIENT
```

where

- capitals denote 3-vectors (xyz or rgb)
- geometric (xyz) vectors R and B should be normalized
- sum[j][a] is the sum of a over all lights j
- lights are modeled as luminous spheres within the scene
- any sphere with kl=0 is considered a light
- lit[i]=1 if the surface point is in shadow with respect to light j, else lit[i]=0. A surface point is "in shadow" if a ray from that point toward the center of the light intersects any spheres before it reaches the light
- (SURFCOLOR, kd, ks, kt, kl[i]) are attributes of the sphere hit
- kl[j] and SURFCOLOR[j] are intensity and color of light being tested
- L is the vector from the surface point to light j
- the normal vector N points in if the ray strikes the surface from within the sphere, else N points out
- AMBIENT is the ambient light color
- CS and CT are the recursive specular and transmitted colors

notes:

- air has index of refraction of 1
- you may assume all spheres have a positive index of refraction
- you may assume that no two transparent spheres intersect
- use CT=0 when refraction causes total internal reflection
- shading formula needs no highlight term because lights are in scene

6. use the following perspective camera model:
  - spheres are specified in right-handed world space
  - eye is at (0,0,0) looking in -y direction of world space
  - screen space x points right, y points down
  - pixels are square
  - the pixel at screen space (x,y) has world space ray direction
  - dx = x-SIZE/2
  - dy = (SIZE/2)/tan(AOV/2) (where tan takes degrees)
  - dx = SIZE/2-y
7. the picture is output to stdout in ascii in the format:
  - A header of two integers, the width and height,
  - followed by width\*height rgb BYTENUM triangles in scanline (row-major) order. These pixel values should be 255 times the intensities computed with the above shading model.
  - It is acceptable for values to exceed 255.
  - Enclosed is test1.ap, an ascii picture file conforming to this format.
8. output of your program must match my pixel values within + or - 10. (Please run your program with the enclosed ray.h and compare your output to test1.ap; only submit if your pixel values nearly match)
9. entries must be postmarked before 15 June 1987

not needed:

1. antialiasing
2. any geometric primitives besides spheres
3. C++
4. any probabilistic ray tracing effects, such as penumbras or the rendering equation
5. program doesn't have to pass lint
6. speed

goal:

The winner will be the valid entry with the minimum number of tokens after running the source through the C preprocessor. (This is a better measure of program length than number of lines or object code size, since it is machine-independent and more hacker-resistant.) Use the enclosed program, squease.c, and the ntok.cah in ntok.cah to count tokens.

Send entries and questions via e-mail to me at:

```
UNCC: {sun,unbwa}@pixar:minray
ARPA: minray@pixar.unccpubvax.berkeley.edu
```

Please put your name, address (electronic and otherwise), number of years of programming, and any remarks about your minimization tricks in a comment at the top of your source file. Note that my token counter ignores comments and unused #ifdefs, so such a comment won't penalize you.

At any time before the deadline you can mail me and I'll tell you the token count of the current leader.

The winning entries will be posted to comp.graphics.

```
- Paul Heckbert
3 May 87
EOF1582
cat << EOF1583 >>squease.c
/*
 * SQUEEZE - Squease a C program.
```

<http://www.acm.org/pubs/toq/GraphicsGems/gemsiv/minray/minray.post> (2 of 13) [2/21/2001 9:42:27 AM]

[illegible]





<http://www.acm.org/pubs/tog/GraphicsGems/gemsiv/minray/minray.post> (5 of 13) [2/21/2001 9:42:27 AM]

5 5 5  
5 5 5  
5 5 5  
5 5 5  
5 8 9  
4 26 31  
5 26 32  
111 111 111 199  
111 112 199  
10 51 19  
9 51 17  
9 49 55  
9 46 52  
121 62 28  
133 87 30  
141 71 32  
147 75 33  
152 78 34  
73 44 18  
72 44 18  
70 42 17  
69 41 16  
66 39 16  
16 11 7  
5 5 5  
5 5 5  
5 5 5  
5 5 5  
5 5 5  
5 5 5  
5 5 5  
5 5 5  
5 5 5  
5 5 5  
4 5 5  
1 2 2  
4 24 29  
4 24 30  
7 27 33  
7 28 34  
7 28 34  
9 48 53  
9 46 51  
8 43 48  
111 57 25  
122 62 28  
130 65 29  
136 66 31  
69 41 17  
68 41 17  
67 41 17  
66 40 17  
63 38 15  
61 36 14  
16 8 3  
17 9 4  
7 6 5  
5 5 5  
5 5 5  
5 5 5  
5 5 5  
5 5 5  
4 5 5  
3 4 4  
1 2 2  
4 22 27  
7 25 30  
7 25 31  
7 26 31  
7 26 31  
8 44 49  
8 42 47  
8 39 44  
98 55 21  
109 55 25  
117 55 27  
123 62 28  
63 38 16  
63 38 16  
62 37 16  
61 36 15  
59 35 15  
57 34 14  
17 9 4  
17 9 4  
17 9 4  
17 9 4  
19 10 4  
5 5 5  
11 8 4  
12 8 4  
12 8 4  
3 5 4  
3 4 4  
3 4 4  
6 22 27  
6 23 28  
6 23 28  
6 24 28  
6 24 28  
6 23 28  
8 37 42  
7 35 39  
91 43 19  
95 48 22  
103 52 23  
109 55 25  
58 34 14  
57 34 14  
56 34 14  
55 33 14  
53 32 13  
51 30 13  
17 9 4  
17 9 4  
17 9 4  
17 9 4  
19 10 4  
5 5 5  
11 8 4  
12 8 4  
12 8 4  
3 5 4  
3 4 4  
3 4 4  
6 20 24  
6 21 25  
6 21 25  
6 21 26  
6 21 26  
6 21 25  
7 33 37  
7 30 34  
66 35 15  
78 40 18  
87 44 20  
93 47 21  
51 30 13  
51 30 13  
50 30 13  
49 29 12  
47 28 12  
45 26 11  
17 9 4  
17 9 4  
17 9 4  
17 9 4  
17 9 4  
5 5 5  
12 9 5  
12 8 4  
12 8 4  
3 4 4  
3 4 4  
3 4 4  
3 4 4  
5 17 21  
5 18 22  
5 19 22  
6 19 22  
6 19 22  
5 19 22  
6 27 31  
6 24 28  
47 25 11  
60 32 14  
70 36 16  
76 39 17  
44 26 11  
44 26 11  
44 26 11  
42 25 11  
40 24 10  
38 22 10  
17 9 4  
17 9 4  
17 9 4  
17 9 4  
17 9 4  
5 5 5  
4 5 5  
12 8 4  
3 5 5  
3 5 4  
3 4 4



PLACE	*****	AUTHOR	NOTES
		*** GENIUS THEATRE ***	
1	916	Yue Cichors, Purdue	(complete-dependent)
1a	916	Yue Cichors, Purdue	(portable)
2	956	Darwyn Parkashov, Kansas	(complete)
2a	956	Darwyn Parkashov, Kansas	(portable)
4	1003	Greg Ward, Berkeley	(portable)
		*** MEMORABLE THEATRE ***	
c1	10	Tony Apollaka, Pkay	(complete)
c2	66	Greg Ward, Berkeley	(cheater)

Interestingly, the entries had nearly identical modularization into six sub-entries: main, trace, inter-episode, vector, normalizes, vector-add, and vector-sub. The entries were also modularized into a complete list of 1000 entries which allowed arbitrarily long cheater strings to come as a single entry. The entries were also modularized into a complete list of 1000 entries which allowed arbitrarily long cheater strings to come as a single entry. I'm not sure if this is a coincidence or if the entries were modularized in this way at all. I'm not sure if this is a coincidence or if the entries were modularized in this way at all.

\* our person obviously missed the spirit of the context (and perhaps of life itself):

"For an up-front payment of \$5000 and a 12% royalty on any sales of the product, I consider submitting an entry by Mr. X. My lawyers are Hughes, Green, Seidlin and Diener of Santa Clara; please have your lawyers contact me at [redacted] for more information."

Those who entered the context also learned some repulsive C coding tricks:

```
* color = point; *struct {float x, y;} --> *struct {float x, y, z};
* pass structures (not just pointers to structs) to allow vector expressions
  like point + 2*point
* no optimizations: trace applied and transmitted rays even if coeff < 0
  (for if (b0; int i; i++) --> "i = n while (...)";
  merge a and y loops into one (see [redacted])
* assume statistics (not initializations)
  "apb[1]" --> "apb[0]"
* choose just the right set of utility routines
* creative use of comma, e.g., "if (a) [b.c];" --> "if (a) b.c;"
* use of "unusual" expressions: b = vdot(d, v vcomb)/1.; r = a ccomb
  (obscure (non-portable) c)
* eliminate semicolons in struct def: "struct def; struct {float x, y, z};"
```

Just if you were expecting a useful ray tracing program out of this, a warning:

As one would expect of any 'minimal' program, the winning program is cryptic, 'tense', inefficient, unmaintainable, and nearly useless, except as a source of C coding tricks. Remember all the limitations of this contest: no antialiasing, spheres only, a bizarre shading model, and i/o formats out of the neanderthal age. So don't be surprised that the program is slow and the output contoured and jaggy. If you want a good ray tracer, start from scratch!

```
# The header files for test programs are included below along with the
# basic file operations to compile and run one of these entries, run .e.g.:
#
g++ test1.h say.h
rm -o say prog.o -lm

Please mail my minimizations of these programs to me and I'll post to the net.
```

---

```
# To unpack, cut here and run the following through
mail archive of gnu.c.dawson: mitchel.c.tony:c gnu.c.test1.h.test2.h.test3.h
or:
cut -c 6-98 <test1.h >say.h
#
say.h = Minimal say tracing program.
#
cc -c sayprog.c
#
Joe Cydonias
#
work: Purdue University CACIAS /
Control Data Corporation
Peters Engineering Center
E Lafayette, IN 47907
#
phone: (317) 494-5544
#
sgsa: Hsamm@dpd.cc.purdue.edu
scsp: pur-as@Hsamm4
#
experience: 13 years programming,
this is my second c program.
```

```

/*
 * strategy:
 *
 * 1) Express shading calculations as functions.
 * 2) All vector expressions are vector triads (a + b*a)
 * 3) Do a critical path analysis of variables and map to
 *    temporary with #define's. This keeps the code
 *    readable.
 * 4) Have Kirk Smith find it more tokens.
 * 5) This version is machine/compiler dependent in that
 *    it assumes function parameters are evaluated from
 *    right to left.
 */

typedef struct { double x, y, z;
                } vector;

typedef struct { vector center; /* center coordinate */
                double color; /* surface color */
                double radius; /* radius */
                kd; /* diffuse coefficient */
                kr; /* coefficient of reflection */
                kt; /* coefficient of transmission */
                ki; /* light source intensity */
                ir; /* index of refraction */
                } sphere;

#define SPHERE sphere spheres[] /* "The Spheres" */
#define AMBIENT vector ambient /* ambient light color */
#define INFP 10000000. /* positive infinity */
#define INFM 20000000. /* second positive infinity */
#define EPS 0.001 /* used to detect self-intersect */

/* include file defining the screen and viewing parameters */
#include "ray.h"

/* global variables */
double sqrt(), tan(), /* math functions from cmath.h */
dist; /* distance to closest sphere */
sized = SIZE/2; /* SIZE / 2 */
s0, s1, s2; /* temporary scalars */
vector origin; /* eye location and zero */
c; /* direction cosine of pial ray */
/* and pial color */
v0, v1; /* temporary vectors */
sphere *sph; /* closest sphere, = 0 if null */
*s; /* intsp - current sphere */

/* basic math routines */
double dot (a,b) /* compute dot product (a.b) */
vector a, b;
{
    return a.x*b.x + a.y*b.y + a.z*b.z;
}

vector vaddn (a,b,s) /* compute vector triad (a+b*s) */
vector a, b;
double s;
{
    a.x += b.x * s;
    a.y += b.y * s;
    a.z += b.z * s;
    return a;
}

vector unitize (a) /* normalize vector (a/|a|) */
vector a;
{
    return vaddn ( origin, a, 1.0/sqrt(dot(a,a)) );
}

vector base, cosine; /* base of the ray */
/* direction cosines for the ray */
{
#define d v0
#define hsq s0
#define dist s1
#define root s2 /* distance to intersected sphere */
    dist = INFM; /* closest sphere is at infinity */
    sph = 0; /* default to no sphere intersected */
    for (s = spheres; s < spheres+NSPHERE; s++)
        hsq = - dot (d = vaddn(base,s->center,-1.),cosine);
        dist = hsq*hsq - dot (d,d) - s->radius*s->radius;
        dist = dist > 0 ? sqrt(dist) : INFM;
        root = hsq - dist;
        root = root < EPS ? hsq + dist : root;
        dist = root + EPS && root < dist ? sph = s, root : dist
    }

vector shade (base,cosine,depth) /* base of the ray */
vector base, cosine; /* direction cosines for the ray */
/* depth in the shade tree */
/* int depth; */
{
    sphere *p; /* visible sphere for the ray */
    /* current light source for shadow tests */
    vector hitpt; /* hit point on the visible sphere */
    norm; /* surface normal in direction of ray */
    sint; /* diffuse illumination intensity */
    double n12; /* relative index of refraction */
#define s v0
#define norm s0
#define cosh s1
    if (!depth) return origin;
    intsp (base,cosine);
    if (p = sph) {
        cosh = - dot (cosine,
            norm = unitize(vaddn(
                hitpt = vaddn(base,cosine,dist),
                p->center,-1.))
            );
        n12 = 1. / p->ir;
        if (cosh < 0.) /* if ray originates from within */
            norm = vaddn(origin,norm,-1.), n12 = p->ir, cosh = -cosh;
        sint = ambient; /* initial intensity = ambient */
        while ( light < spheres+NSPHERE )
            intsp (hitpt,
                s0 = unitize(vaddn(light->center,hitpt,-1.))
                );
            q = dot (norm,norm); /* dot (norm,norm) */
            sint = vaddn (sint,sph->color,
                sph ** light++ && q > 0. ? sph->ki*q : 0.)
            ;
            kf = 1.0 - n12*n12 * dot(s,
                s = vaddn (cosine,norm,cosh)
                );
    }
    /* evaluate the shading function */
    /*
    /* cr = shade (hitpt,N,-depth); /* reflected ray */
    /* cr = shade (hitpt,V, depth); /* refracted ray */
    /* c = p->ki * p->color + (ambient + shade(s)) */
    /* p->ki * cr + p->kt * cr + p->ki * p->color; */
    sint.s = p->color.s;
    sint.y = p->color.y;
    sint.z = p->color.z;
    depth--;
    return vaddn(vaddn(vaddn(vaddn(
        origin,sint,
        p->ki)
        , shade(hitpt,
            vaddn(cosine,norm,2.*cosh),
            depth),
        p->ki)
        , kf > 0. ? shade(hitpt,
            vaddn(vaddn(origin,norm,-sqrt(kf)),s,n12),
            depth) : origin,
        p->kt)
        , p->color.p->ki)
        );
    }
}

```

```

    } else
        return ambient ;
}

main (argc)
/*
int     argc;          /* pixel number          */
{
    argc = 0;
    printf ("h0 h0\n", SIZE, SIZE);
    while ( argc < SIZE*SIZE )
        c = argc % SIZE - size2;
        c = size2 / tan(ADW / 114.59166 ),
        c = size2 - argc + SIZE;
        c = width (origin, shade (origin,unitize(c),DUPTR), 255.);
        printf ("k.0f k.0f k.0f\n", c)
    }
    /* send layers, guns, and money, get me out of this... */
    EDP27902
    cat << "EDP27904" >> darwyn.c
    /*
    * Short ray tracer written with the goal of minimizing the total number
    * of C calls after preprocessing by spm. For Paul Buchheit's contest.
    * May/June 1987. WARNING! this style of coding results in increasing
    * execution time because the simplest brute force algorithms are used.
    * Maintenance of the code is harder than it should be for a program
    * of this size, because of the messy coding style
    *
    * I have 16 years of programming experience, including
    * 12 years in C, but I'm afraid it doesn't show in this program.
    *
    * Darwyn Peachey, University of Saskatchewan, Dept. of Computational Science.
    * 1361 664-4939 peachey@math.usask.ca peachey@math.hinet
    *
    * This is the 956 token version of 13-Jun-87.
    */

#define PI360      8.7266462599716e-3      /* pi/360 */
#define DPHITTRY  1400
#define EPSILON    1e-6

typedef struct { double x, y, z; } triple;
typedef struct {
    triple ctr, color; /* center and RGB coeffs of sphere */
    double radius, hd, kv, kt, kt, kt
} sphere;
#define SPHERE sphere sph[]
#define AMBIENT    triple v, zero, ambient

#include "ray.h"

double tmin(), sgtr(), tmin, a, b, t;
/* int */ row, col, half = SIZE/2; /* row initialized to 0 */

triple combine(v1, a, v2);
triple v1, v2;
double a;
{
    v1.x = a * v1.x;
    v1.y = a * v1.y;
    v1.z = a * v1.z;
    return v1;
}
double v1, v2;
{
    triple v1, v2;
    return v1.x*v2.x + v1.y*v2.y + v1.z*v2.z;
}

triple
norm(v1)
{
    triple vec;
    return combine(vec, 1/sqrt(dot(vec,vec)), zero);
}

/*
* Intersect and trace use the same global variable, tmin.
*/
sphere *
intersect(rayorg, rayvec)
    triple rayorg, rayvec;
{
    sphere *sp, *which = 0;

    tmin = INFINITY;
    for (sp = sph; sp < sph+NSPHERE; sp++) {
        /* determine coefficients of quadratic equation */
        /* at^2 + bt + c = 0 */
        a = dot(rayvec, rayvec);
        v = combine(sp-ctr, -1.0, rayorg);
        b = 2 * dot(rayvec, v);
        /* discriminant: b*b - 4*a*c */
        t = b*b - 4*a*dot(v,v) - sp->radius*sp->radius;
        /* find (closest) root if any */
        if (t > EPSILON) { /* there are two real roots */
            t = sgtr(t);
            t = 0.5 * ((-b + t > EPSILON ? -t : t) - b)/a;
            if (t > EPSILON && t < tmin)
                which = sp, tmin = t;
        }
    }
    return which;
}

triple
trace(M, rayvec, depth, inside) /* N used for ray origin & normal vec */
{
    triple loc, dir, color;
    sphere *which, *sp;
    double rcos1, rcos2, a;
    if (depth > DUPTR)
        return zero;

    if (which = intersect(M, rayvec)) {
        /* having found best "t" value, compute location and normal */
        M = combine(norm(combine(which-ctr, -1.0,
            loc = combine(rayvec, tmin, M))),
            inside ? -1.0 : 1.0, zero);

        /* do shading calculations, and fire subrays as necessary */
        color = ambient; /* accumulates illumination in color */
        for (sp = sph; sp < sph+NSPHERE; sp++)
            if ((a = dot(B,
                dir = norm(combine(loc, -1.0, sp-ctr)))) > 0.0
                && intersect(loc, dir) == sp)
                color = combine(sp->color, a*sp->kl, color);
        /* color is now the total ambient + direct lighting */
        color.x = *which->color.x + *which->hd;
        color.y = *which->color.y + *which->hd;
        color.z = *which->color.z + *which->hd;
        rcos1 = -dot(B, rayvec);
        a = inside ? which->vr : 1/which->vr;
        rcos2 = 1 - a*a * (1 - rcos1*rcos1);
        return combine( /* refracted color */
            trace(loc, combine(rayvec, a,
                combine(M, a * rcos1 - sqrt(rcos2),
                    zero)),
                /* reflected color */
                combine(trace(loc, combine(M, 2 * rcos1, rayvec),
                    depth+1, inside), which->ka,
                    /* luminance + diffuse color */
                    combine(which->color, which->kt,
                        color)))));
    }
    return ambient;
}

main()
{
    printf("h0 h0\n", SIZE, SIZE);
    while (col = 0, row++ < SIZE)
        while (col < SIZE)
            v.x = col * .5 - half;
            v.y = half / tan(ADW * PI360);
            v.z = half - row * 1;
            v = trace(zero, norm(v), 1, 0);
            printf("k.0f k.0f k.0f\n",
                v.x*255, v.y*255, v.z*255);
            col++;
        }
    EDP27904
    cat << "EDP27905" >> michel.c
    /*
    * MINIMAL RAY TRACER PROGRAMMING CONTEST ENTRY
    *
    * By : Michel Burgess
    *      6760, rue St-Charles
    *      Montreal(Quebec)
    *      Canada H2S 2S2
    *
    * Email: CMBest@miralibro.uden.edu
    *         CMBEST@miralibro.uden.edu@ubc.cmcnet
    *         UUCP : ...!uimmo!utpp!total!mcsa!mcgill-viaimmo!michelburgess
    *                                     ...!michelbakin
    *
    * Years programming : 9 (recently finished a M.Sc. C.S. Universite de Montreal)
    *
    * Notes : This program is barely readable due to compacting.
    *          Reduced to fit in 80 columns.
    *          It is based on a program written by Turner Whitted, that raytraces
    */
}
```

<http://www.acm.org/pubs/tog/GraphicsGems/gemsiv/minray/minray.post> (11 of 13) [2/21/2001 9:42:27 AM]



```
KOP27907
cat <<'KOP27908'>test1.h
/* ray.h for test1, first test scene */
#define DEPTH 3 /* max ray tree depth */
#define SIZE 32 /* resolution of picture in x and y */
#define ANV 25 /* total angle of view in degrees */
#define NSPHERES 5 /* number of spheres */

AMBLIGHT = [.02, .02, .02]; /* ambient light color */

/* sphere: x y z r g b rad kd ka kt kl ir */
SPHERE = {
  1., .3, 1., 1., 1., .5, .05, .2, .85, 0., 1.2,
  -1., -.5, 1., .5, .2, 1., .7, .3, 0., .05, 1.2,
  1., .8, -.5, .1, .8, .8, .3, .7, 0., 0., 1.2,
  1., -.8, .15, 1., .8, 1., 7., 0., 0., 0., 0., 1.5,
  -1., -.3, .12, -.8, 1., 1., 5., 0., 0., 0., .5, 1.5,
};

KOP27908
cat <<'KOP27909'>test2.h
/* ray.h for test2, second test scene */
#define DEPTH 4 /* max ray tree depth */
#define SIZE 32 /* resolution of picture in x and y */
#define ANV 30 /* total angle of view in degrees */
#define NSPHERES 3 /* number of spheres */

AMBLIGHT = [.04, .02, .02]; /* ambient light color */

/* sphere: x y z r g b rad kd ka kt kl ir */
SPHERE = {
  .15, 0., 0., 1., 1., 1., 1., 0., .2, .8, .02, 1.1,
  1., 0., .3, .7, 1., .2, 1., .8, .2, 0., 0., 1.5,
  .3, -.3, .4, 1., 1., 1., 4., 0., 0., 0., 1., 1.1,
};

KOP27909
cat <<'KOP27910'>test3.h
/* ray.h for test3, third test scene */
#define DEPTH 3 /* max ray tree depth */
#define SIZE 31 /* resolution of picture in x and y */
#define ANV 35 /* total angle of view in degrees */
#define NSPHERES 7 /* number of spheres */

AMBLIGHT = [.05, .05, .05]; /* ambient light color */

/* sphere: x y z r g b rad kd ka kt kl ir */
SPHERE = {
  .75, 4., 0., 1., 1., 1., .5, .2, .8, 0., 0., 1.2,
  .65, 6., 0., .9, .2, .9, .5, 0., .2, .9, .05, 1.2,
  .55, 8., 0., .2, .7, .5, .5, 0., .2, 0., 0., 1.2,
  -.75, 4., 0., .7, .3, .3, .5, .1, .3, .7, 0., 1.4,
  -.65, 6., 0., 1., .8, .1, .5, .2, .8, 0., 1.2,
  -.55, 8., 0., 0., 0., 1., .5, .5, .5, 0., 0., 1.2,
  0., 0., 0., 1., 1., 1., 2., 0., 0., 0., 1., 1.1,
};

KOP27910
exit

Path: p1minerph
From: jhmiller@UCSD (Paul Heckbert)
Newsgroup: comp.graphics
Subject: Re: Winners of Minimal Ray Tracer Contest
Summary: Minimal ray tracing reaches new lows
Date: 25 Jun 87 07:38:42 GMT
Organization: Pixar -- Marin County, California

Minimal ray tracing reaches new lows!

I've taken some of the tricks from Harvey Peachey's and Joe Cychosz's minimal
ray tracers (posted previously) and combined them with some of my own to
create a hybrid program that is the shortest of all 888 tokens.
Recall that Joe Cychosz's winning entry had 916 tokens.
To compile the enclosed "paul.c", run "cc -o paul paul.c -lm".
If you're able to shorten the enclosed program further, please send me mail.

When my C code compiler program "squeeze.c" (also posted previously)
is run on paul.c,

$./lib/cpp paul.c | sed -i '/^#if/ | squeeze -77 > paul.squeeze.c

we get a rectangular block of C code that fits nicely on a standard 24x80
terminal screen. See the enclosed file paul.squeeze.c.

An even more impressive thing to do with this file is reduce it to fit on
a small email. If you've got the Adobe TeXscript laser printer software, run:

$enscript -B -fCourier5 paul.squeeze.c

and you've got a ray-tracer-on-a-business-card!

Paul Heckbert
Pixar 415-499-3600
P.O. Box 13713 UCSD: [sam,ucbwa]@p1minerph
San Rafael, CA 94913 DDP: pdp1miner.comp@uicwa.berkeley.edu

-----
# to unpack, cut here and run the following shell archive through sh
# contents: paul.c ray.h ray.h paul.squeeze.c
#
and '$/"/' <<'KOP14163'>paul.c
2/* minimal ray tracer, hybrid version - 888 tokens
X Paul Heckbert, ucwva@p1minerph, 13 Jun 87
X "Using tricks from Harvey Peachey and Joe Cychosz.
X
X#define TDS 1e-7
X#define AMBLIGHT vec 0, black, amb
X#define STRUT sstrut sphere {vec om, color; double rad, kd, ka, kt, kl, ir} \
X /*, "test, sphl"
X typedef struct {double x, y, z} vec;
X#include "ray.h"
Xvec
Xdouble u, b, tmin, spt(), tan();
Xdouble vdot(A, B)
Xvec A, B;
X return A.x*B.x + A.y*B.y + A.z*B.z;
X
X
Xvec vcomb(A, B) /* aAB */
Xdouble a;
Xvec A, B;
X B.x += *A.x;
X B.y += *A.y;
X B.z += *A.z;
X return B;
X
X
Xvec vunit(A)
Xvec A;
X if
X return vcomb(1./spt(vdot(A, A)), A, black);
X
X
Xstrut sphere *intersect(P, D)
Xvec P, D;
X if
X best = 0;
X tmin = 1e3;
X s = spt(NSPHERE);
X while (s-->0)
X b = vdot(D, U = vcomb(-1., P, s->xcm));
X u = b?b>vdot(D, U)>-rad?>-rad;
X u = u?0? spt(u) : 1e3;
X u = b?TDS, 1-b : b*u;
X tmin = u>TDS && u?tmin :
X best = s, u : tmin;
X return best;
X
X
Xvec trace(level, P, D)
Xvec P, D;
X if
X double d, eta, s;
X vec R, color;
X strut sphere *s;
X
X if (!level--) return black;
X if (s = intersect(P, D))
X else return amb;
X
X color = amb;
X eta = s->ir;
X s = vdot(D, S = vunit(vcomb(-1., P = vcomb(tmin, D, P), s->xcm)));
X if (d&0)
X S = vcomb(-1., R, black);
X eta = 1/eta;
X d = s;
X l = spt(NSPHERE);
X while (l-->0)
X if ((s = l>kl?vdot(S, U = vunit(vcomb(-1., P, l->xcm)))) > 0 &&
X intersect(P, U)>u)
X color = vcomb(s, l->color, color);
X U = s->color;
X color.x *= U.x;
X color.y *= U.y;
X color.z *= U.z;
X s = l>ka?eta*(1-d*d);
X /* the following is non-portable; we assume right to left arg evaluation.
X (use 0 before call to trace, which modifies U) */
X return vcomb(s->kl,
X s?0? trace(level, P, vcomb(eta, D, vcomb(eta*d-spt(s), R, black)))
X vcomb(s->ka, trace(level, P, vcomb(2*d, R, D)))
X vcomb(s->kd, color, vcomb(s->kl, U, black)));
X
X
X
Xmain()
X{
X printf("hd hPa", SIZE, SIZE);
X while (y>0)TRACE;
X U.x = y*SIZE/2-y*SIZE/2,
X U.z = SIZE/2-y*SIZE/2,

```

[illegible]

```
/* ray.h for test1, first test scene */
#define DEPTH 3          /* max ray tree depth */
#define SIZE 32          /* resolution of picture in x and y */
#define AOV 25           /* total angle of view in degrees */
#define NSPHERE 5        /* number of spheres */

AMBIENT = {.02, .02, .02}; /* ambient light color */

/* sphere: x y z  r g b  rad  kd ks kt kl  ir */
SPHERE = {
    0., 6., .5,    1., 1., 1.,    .9,    .05, .2, .85, 0.,    1.7,
   -1., 8., -.5,   1., .5, .2,    1.,    .7, .3, 0., .05,    1.2,
    1., 8., -.5,   .1, .8, .8,    1.,    .3, .7, 0., 0.,    1.2,
    3., -6., 15.,  1., .8, 1.,    7.,    0., 0., 0., .6,    1.5,
   -3., -3., 12.,  .8, 1., 1.,    5.,    0., 0., 0., .5,    1.5,
};
```

```
/*
 * ANSI C code from the article
 * "Bilinear Coons Patch Image Warping"
 * by Paul S. Heckbert, ph@cs.cmu.edu
 * in "Graphics Gems IV", Academic Press, 1994
 *
 *
 * This code has been written in the most portable way possible.
 * It will not compile and run without some modifications.
 * You'll need to:
 *     write pixel_read and pixel_write subroutines
 *     read or compute a source image
 *     read or compute four boundary curves
 *     pick a destination image size
 *     call resample() to resample the boundary curves to the width and
 *         height of the destination image
 *     call coons_warp() to write the destination image
 *     display the warped destination image
 */

#include <math.h>
#include <assert.h>
#define ALLOC(ptr, type, n)  assert(ptr = (type *)malloc((n)*sizeof(type)))

typedef struct {          /* 2-D POINT OR VECTOR */
    float x, y;
} Point2f;

typedef struct {          /* A CURVE DEFINED BY A SEQUENCE OF POINTS */
    int npt;              /* number of points */
    Point2f *pt;          /* array of npt points */
} Curve;

#define SHIFT 20          /* number of fractional bits in fixed point coords */
#define SCALE (1<<SHIFT)

typedef struct {          /* INTEGER POINT AND VECTOR */
    int px, py;           /* position */
    int dx, dy;           /* incremental displacement */
} Ipoint;

/*
 * coons_warp: warps the picture in source image into a rectangular
 * destination image according to four boundary curves, using a
 * bilinear Coons patch.
 * bound[0] through bound[3] are the top, right, bottom, and left
 * boundary curves, respectively, clockwise from upper left.
 * (These comments are written assuming that y points down on your
 * frame buffer. Otherwise, bound should proceed CCW from lower left.)
 * The lengths of bound[0] and bound[2] are assumed to be the width of
 * the destination rectangle, and the lengths of bound[1] and bound[3]
 * are assumed to be the height of the rectangle.
 * The upper left corner of the destination rectangle is (u0,v0).
 *
 * Paul Heckbert          25 Feb 82, 15 Oct 93
 */

void coons_warp(Pic *source, Pic *dest, Curve *bound, int u0, int v0) {
    register Ipoint *pu;
    register int u, x, y, qx, qy, dqx, dqy;
```

```
int nu, nv, v;
float du, dv, fv;
Point2f p00, p01, p10, p11, *pu0, *pu1, *p0v, *p1v;
Ipoint *pua;

nu = bound[0].npt-1; /* nu = dest_width-1 */
nv = bound[1].npt-1; /* nv = dest_height-1 */
assert(bound[2].npt==nu+1);
assert(bound[3].npt==nv+1);

pu0 = &bound[0].pt[0]; /* top boundary curve */
p1v = &bound[1].pt[0]; /* right */
pu1 = &bound[2].pt[nu]; /* bottom */
p0v = &bound[3].pt[nv]; /* left */
/* arrays pu1 and p0v are in the reverse of the desired order,
   running from right to left and bottom to top, resp., so we
   index them with negative subscripts from their ends (yeeeha!) */

p00.x = (p0v[ 0].x + pu0[ 0].x)/2.; /* upper left patch corner */
p00.y = (p0v[ 0].y + pu0[ 0].y)/2.;
p10.x = (pu0[nu].x + p1v[ 0].x)/2.; /* upper right */
p10.y = (pu0[nu].y + p1v[ 0].y)/2.;
p11.x = (p1v[nv].x + pu1[-nu].x)/2.; /* lower right */
p11.y = (p1v[nv].y + pu1[-nu].y)/2.;
p01.x = (pu1[ 0].x + p0v[-nv].x)/2.; /* lower left */
p01.y = (pu1[ 0].y + p0v[-nv].y)/2.;

du = 1./nu;
dv = 1./nv;

ALLOC(pua, Ipoint, nu+1);
for (pu=pua, u=0; u<=nu; u++, pu++) {
    pu->dx = (pu1[-u].x - pu0[u].x)*dv*SCALE + .5;
    pu->dy = (pu1[-u].y - pu0[u].y)*dv*SCALE + .5;
    pu->px = pu0[u].x*SCALE + .5;
    pu->py = pu0[u].y*SCALE + .5;
}

for (fv=0., v=0; v<=nv; v++, fv+=dv) {
    qx = (p0v[-v].x - (1.-fv)*p00.x - fv*p01.x + .5)*SCALE + .5;
    qy = (p0v[-v].y - (1.-fv)*p00.y - fv*p01.y + .5)*SCALE + .5;
    dqx = (p1v[v].x - p0v[-v].x - (1.-fv)*(p10.x-p00.x) - fv*(p11.x-p01.x))
           *du*SCALE + .5;
    dqy = (p1v[v].y - p0v[-v].y - (1.-fv)*(p10.y-p00.y) - fv*(p11.y-p01.y))
           *du*SCALE + .5;
    for (pu=pua, u=0; u<=nu; u++, pu++) {
        x = pu->px+qx >> SHIFT;
        y = pu->py+qy >> SHIFT;
        pixel_write(dest, u0+u, v0+v, pixel_read(source, x, y));
        qx += dqx;
        qy += dqy;
        pu->px += pu->dx;
        pu->py += pu->dy;
    }
}
free(pua);
}
```

/\* the following routine is the slow way to do a Coons warp, for reference \*/

```
void coons_warpl(Pic *source, Pic *dest, Curve *bound, int u0, int v0) {
```

```
int nu, nv, u, v, x, y;
float fu, fv;
Point2f p00, p01, p10, p11, *pu0, *pul, *p0v, *plv;

nu = bound[0].npt-1;
nv = bound[1].npt-1;

pu0 = &bound[0].pt[0];          /* top boundary curve */
plv = &bound[1].pt[0];          /* right */
pul = &bound[2].pt[nu];         /* bottom */
p0v = &bound[3].pt[nv];         /* left */

p00.x = (p0v[ 0].x + pu0[ 0].x)/2.; /* upper left patch corner */
p00.y = (p0v[ 0].y + pu0[ 0].y)/2.;
p10.x = (pu0[nu].x + plv[ 0].x)/2.; /* upper right */
p10.y = (pu0[nu].y + plv[ 0].y)/2.;
p11.x = (plv[nv].x + pul[-nu].x)/2.; /* lower right */
p11.y = (plv[nv].y + pul[-nu].y)/2.;
p01.x = (pul[ 0].x + p0v[-nv].x)/2.; /* lower left */
p01.y = (pul[ 0].y + p0v[-nv].y)/2.;
```

```
for (v=0; v<=nv; v++) {
    fv = (float)v/nv;
    for (u=0; u<=nu; u++) {
        fu = (float)u/nu;
        x = (1.-fv)*pu0[ u].x + fv*pul[-u].x
            + (1.-fu)*p0v[-v].x + fu*plv[ v].x
            - (1.-fu)*(1.-fv)*p00.x - fu*(1.-fv)*p10.x
            - (1.-fu)*    fv *p01.x - fu*    fv *p11.x + .5;
        y = (1.-fv)*pu0[ u].y + fv*pul[-u].y
            + (1.-fu)*p0v[-v].y + fu*plv[ v].y
            - (1.-fu)*(1.-fv)*p00.y - fu*(1.-fv)*p10.y
            - (1.-fu)*    fv *p01.y - fu*    fv *p11.y + .5;
        pixel_write(dest, u0+u, v0+v, pixel_read(source, x, y));
    }
}
```

/\* resample\_nonuniform: non-uniformly resample curve a to create curve b,  
 \* with n points. Allocates b->pt to have length n \*/

```
void resample_nonuniform(Curve *a, Curve *b, int n) {
    int ai, bi;
    double ax, af;
    Point2f *ap, *bp;

    if (n<2) {
        fprintf(stderr, "Only %d point in new curve\n", n);
        exit(1);
    }
    ALLOC(b->pt, Point2f, n);
    b->npt = n;
    for (bp=b->pt, bi=0; bi<n; bi++, bp++) {
        ax = (float)bi*(a->npt-1)/(n-1);
        ai = ax;
        af = ax-ai;
        ap = &a->pt[ai];
        bp->x = ap[0].x + af*(ap[1].x-ap[0].x);
        bp->y = ap[0].y + af*(ap[1].y-ap[0].y);
    }
}
```

```
static double len(double x, double y) {return sqrt(x*x+y*y);}

/* resample_uniform: uniformly resample curve a to create curve b,
 * with n points.  Allocates b->pt to have length n */

void resample_uniform(Curve *a, Curve *b, int n) {
    int i;
    double step, l, d;
    Point2f *ap, *bp;

    if (a->npt<2) {
        fprintf(stderr, "Only %d point in curve\n", a->npt);
        exit(1);
    }
    for (step=0., ap=a->pt, i=a->npt-1; i>0; i--, ap++)
        step += len(ap[1].x-ap[0].x, ap[1].y-ap[0].y);
    step /= n-1; /* length of each output segment (ideally) */
    ALLOC(b->pt, Point2f, n);
    b->npt = n;
    d = .0001; /* = 0 + tolerance for roundoff error */
    for (ap=a->pt, bp=b->pt, i=a->npt-1; i>0; i--, ap++) {
        l = len(ap[1].x-ap[0].x, ap[1].y-ap[0].y);
        d += l;
        /* d is the remaining length of the line segment from ap[0] to ap[1]
         * that needs to be subdivided into segments of length step */
        while (d>0.) {
            bp->x = ap[1].x - d/l*(ap[1].x-ap[0].x);
            bp->y = ap[1].y - d/l*(ap[1].y-ap[0].y);
            bp++;
            d -= step;
        }
    }
    if (bp-b->pt != n)
        printf("WARNING: requested %d points, created %d, d=%g\n",
            n, bp-b->pt, d);
}



static void resample(Curve *a, Curve *b, int n, int nonuniform) {
    if (nonuniform) resample_nonuniform(a, b, n);
    else resample_uniform(a, b, n);
}

Curve *boundary_resample(Curve *a, int nx, int ny, int nonuniform) {
    /* "nonuniform" is a boolean flag */
    Curve *b;

    ALLOC(b, Curve, 4);
    resample(&a[0], &b[0], nx, nonuniform);
    resample(&a[1], &b[1], ny, nonuniform);
    resample(&a[2], &b[2], nx, nonuniform);
    resample(&a[3], &b[3], ny, nonuniform);
    return b;
}
```

# Index of

## /pubs/tog/GraphicsGems/gemsv/ch1-1/

Name	Last modified	Size	Description
 <a href="#">_Parent Directory</a>			
 <a href="#">_quarcube.c</a>	29-Jun-00 08:22	18K	



```
/*
    doquartic.c

    a test of a quartic solving routine

    Don Herbison-Evans    24 June 1994
*/

#include <stdio.h>

double nought;
double doub1,doub2;
double doub3,doub4;
double doub6,doub12;
double doub24;
double doubmax;      /* approx square root of max double number */
double doubmin;      /* smallest double number */
double doubtol;      /* tolerance of double numbers */
double rt3;
double inv2,inv3,inv4;

void setcns();
int descartes();
double exp(),log(),sqrt(),cos(),acos();
double cubic();
int qudrtc();
int quartic();
int ferrari();
int neumark();
void errors();

main()
{
    double a,b,c,d;
    double rts[4];
    double rterr[4];
    int nr;

    setcns();
    a = -(double)10;
    b = (double)35;
    c = -(double)50;
    d = (double)24;
    nr = descartes(a,b,c,d,rts);
    errors(a,b,c,d,rts,rterr,nr);
    nr = ferrari(a,b,c,d,rts);
    errors(a,b,c,d,rts,rterr,nr);
    nr = neumark(a,b,c,d,rts);
    errors(a,b,c,d,rts,rterr,nr);
    exit(0);
}
/*****/

void setcns()
/*
    set up constants
*/
{
    int j;

    nought = (double)0;
```

```
doub1 = (double)1;
doub2 = (double)2;
doub3 = (double)3;
doub4 = (double)4;
doub6 = (double)6;
doub12 = (double)12;
doub24 = (double)24;
inv2 = doub1/(double)2;
inv3 = doub1/(double)3;
inv4 = doub1/(double)4;
rt3 = sqrt(doub3) ;

doubtol = doub1;
for ( j = 1 ; doub1+doubtol > doub1 ; ++ j )
{
    doubtol *= inv2;
}
doubtol = sqrt(doubtol);

doubmin = inv2 ;
for ( j = 1 ; j <= 100 ; ++ j )
{
    doubmin=doubmin*doubmin ;
    if ((doubmin*doubmin) <= (doubmin*doubmin*inv2))
        break;
}
doubmax=0.7/sqrt(doubmin) ;
```

```
} /* setcns */
/*****
```

```
int quartic(a,b,c,d,rts,rterr)
double a,b,c,d,rts[4],rterr[4];
/*
    Solve quartic equation using either
    quadratic, Ferrari's or Neumark's algorithm.
```

```
called by
calls qudrtc, ferrari, neumark.
```

```
21 Jan 1989 Don Herbison-Evans
```

```
*/
{
    int qudrtc(),ferrari(),neumark();
    int j,k,nq,nr;
    double odd, even;
    double roots[4];

    if (a < nought) odd = -a; else odd = a;
    if (c < nought) odd -= c; else odd += c;
    if (b < nought) even = -b; else even = b;
    if (d < nought) even -= d; else even += d;
    if (odd < even*doubtol)
    {
        nq = qudrtc(b,d,roots,b*b-doub4*d);
        j = 0;
        for (k = 0; k < nq; ++k)
        {
            if (roots[k] > nought)
            {
                rts[j] = sqrt(roots[k]);
```

```

        rts[j+1] = -rts[j];
        ++j; ++j;
    }
}
nr = j;
}
else
{
    if (a < nought) k = 1; else k = 0;
    if (b < nought) k += k+1; else k +=k;
    if (c < nought) k += k+1; else k +=k;
    if (d < nought) k += k+1; else k +=k;
    switch (k)
    {
        case 0 : nr = ferrari(a,b,c,d,rts) ; break;
        case 1 : nr = neumark(a,b,c,d,rts) ; break;
        case 2 : nr = neumark(a,b,c,d,rts) ; break;
        case 3 : nr = ferrari(a,b,c,d,rts) ; break;
        case 4 : nr = ferrari(a,b,c,d,rts) ; break;
        case 5 : nr = neumark(a,b,c,d,rts) ; break;
        case 6 : nr = ferrari(a,b,c,d,rts) ; break;
        case 7 : nr = ferrari(a,b,c,d,rts) ; break;
        case 8 : nr = neumark(a,b,c,d,rts) ; break;
        case 9 : nr = ferrari(a,b,c,d,rts) ; break;
        case 10 : nr = ferrari(a,b,c,d,rts) ; break;
        case 11 : nr = neumark(a,b,c,d,rts) ; break;
        case 12 : nr = ferrari(a,b,c,d,rts) ; break;
        case 13 : nr = ferrari(a,b,c,d,rts) ; break;
        case 14 : nr = ferrari(a,b,c,d,rts) ; break;
        case 15 : nr = ferrari(a,b,c,d,rts) ; break;
    }
}
errors(a,b,c,d,rts,rterr,nr);
return(nr);
} /* quartic */
/*****

int ferrari(a,b,c,d,rts)
    double a,b,c,d,rts[4];
/*
    solve the quartic equation -


$$x^{**4} + a*x^{**3} + b*x^{**2} + c*x + d = 0$$


called by quartic
calls      cubic, qudrtc.

input -
a,b,c,e - coeffs of equation.

output -
nquar - number of real roots.
rts - array of root values.

method :  Ferrari - Lagrange
Theory of Equations, H.W. Turnbull p. 140 (1947)

calls  cubic, qudrtc

*/
{
    double cubic();

```

```
int qudrtc();

int nquar,n1,n2 ;
double asq,ainv2;
double v1[4],v2[4] ;
double p,q,r ;
double y;
double e,f,esq,fsq,ef ;
double g,gg,h,hh;

fprintf(stderr,"\nFerrari %g %g %g %g\n",a,b,c,d);
asq = a*a;

p = b ;
q = a*c-doub4*d ;
r = (asq - doub4*b)*d + c*c ;
y = cubic(p,q,r) ;

esq = inv4*asq - b - y;
if (esq < nought) return(0);
else
{
    fsq = inv4*y*y - d;
    if (fsq < nought) return(0);
    else
    {
        ef = -(inv4*a*y + inv2*c);
        if ( ((a > nought)&&(y > nought)&&(c > nought))
            || ((a > nought)&&(y < nought)&&(c < nought))
            || ((a < nought)&&(y > nought)&&(c < nought))
            || ((a < nought)&&(y < nought)&&(c > nought))
            || (a == nought) || (y == nought) || (c == nought)
        )
        /* use ef - */
        {
            if ((b < nought)&&(y < nought)&&(esq > nought))
            {
                e = sqrt(esq);
                f = ef/e;
            }
            else if ((d < nought) && (fsq > nought))
            {
                f = sqrt(fsq);
                e = ef/f;
            }
            else
            {
                e = sqrt(esq);
                f = sqrt(fsq);
                if (ef < nought) f = -f;
            }
        }
    }
    else
    {
        e = sqrt(esq);
        f = sqrt(fsq);
        if (ef < nought) f = -f;
    }
}
/* note that e >= nought */
ainv2 = a*inv2;
g = ainv2 - e;
```

```

    gg = ainv2 + e;
    if ( ((b > nought)&&(y > nought))
        || ((b < nought)&&(y < nought)) )
    {
        if (( a > nought) && (e != nought)) g = (b + y)/gg;
        else if (e != nought) gg = (b + y)/g;
    }
    if ((y == nought)&&(f == nought))
    {
        h = nought;
        hh = nought;
    }
    else if ( ((f > nought)&&(y < nought))
        || ((f < nought)&&(y > nought)) )
    {
        hh = -inv2*y + f;
        h = d/hh;
    }
    else
    {
        h = -inv2*y - f;
        hh = d/h;
    }
    n1 = qudrtc(gg,hh,v1, gg*gg - doub4*hh) ;
    n2 = qudrtc(g,h,v2, g*g - doub4*h) ;
    nquar = n1+n2 ;
    rts[0] = v1[0] ;
    rts[1] = v1[1] ;
    rts[n1+0] = v2[0] ;
    rts[n1+1] = v2[1] ;
    return(nquar);
}
}
}

```

```

} /* ferrari */
/*****

```

```

int neumark(a,b,c,d,rts)
    double a,b,c,d,rts[4];
/*

```

solve the quartic equation -

$$x^{**4} + a*x^{**3} + b*x^{**2} + c*x + d = 0$$

called by quartic  
calls cubic, qudrtc.

input parameters -  
a,b,c,e - coeffs of equation.

output parameters -  
nquar - number of real roots.  
rts - array of root values.

method - S. Neumark

Solution of Cubic and Quartic Equations - Pergamon 1965  
translated to C with help of Shawn Neely

```

*/
{
    int nquar,n1,n2 ;

```

```
double y,g,gg,h,hh,gdis,gdisrt,hdis,hdisrt,g1,g2,h1,h2 ;
double bmy,gerr,herr,y4,d4,bmysq ;
double v1[4],v2[4] ;
double asq ;
double p,q,r ;
double hmax,gmax ;
double cubic();
int qudrtc();
```

```
fprintf(stderr,"\nNeumark %g %g %g %g\n",a,b,c,d);
asq = a*a ;
```

```
p = -b*doub2 ;
q = b*b + a*c - doub4*d ;
r = (c - a*b)*c + asq*d ;
y = cubic(p,q,r) ;
```

```
bmy = b - y ;
y4 = y*doub4 ;
d4 = d*doub4 ;
bmysq = bmy*bmy ;
gdis = asq - y4 ;
hdis = bmysq - d4 ;
if ((gdis < nought) || (hdis < nought)) return(0);
else
{
```

```
    g1 = a*inv2 ;
    h1 = bmy*inv2 ;
    gerr = asq + y4 ;
    herr = hdis ;
    if (d > nought) herr = bmysq + d4 ;
    if ((y < nought) || (herr*gdis > gerr*hdis))
    {
        gdisrt = sqrt(gdis) ;
        g2 = gdisrt*inv2 ;
        if (gdisrt != nought) h2 = (a*h1 - c)/gdisrt ;
        else h2 = nought;
    }
    else
    {
        hdisrt = sqrt(hdis) ;
        h2 = hdisrt*inv2 ;
        if (hdisrt != nought) g2 = (a*h1 - c)/hdisrt ;
        else g2 = nought;
    }
}
```

/\*

note that in the following, the tests ensure non-zero  
denominators -

\*/

```
h = h1 - h2 ;
hh = h1 + h2 ;
hmax = hh ;
if (hmax < nought) hmax = -hmax ;
if (hmax < h) hmax = h ;
if (hmax < -h) hmax = -h ;
if ((h1 > nought)&&(h2 > nought)) h = d/hh ;
if ((h1 < nought)&&(h2 < nought)) h = d/hh ;
if ((h1 > nought)&&(h2 < nought)) hh = d/h ;
if ((h1 < nought)&&(h2 > nought)) hh = d/h ;
if (h > hmax) h = hmax ;
if (h < -hmax) h = -hmax ;
```

```
    if (hh > hmax) hh = hmax ;
    if (hh < -hmax) hh = -hmax ;

    g = g1 - g2 ;
    gg = g1 + g2 ;
    gmax = gg ;
    if (gmax < nought) gmax = -gmax ;
    if (gmax < g) gmax = g ;
    if (gmax < -g) gmax = -g ;
    if ((g1 > nought)&&(g2 > nought)) g = y/gg ;
    if ((g1 < nought)&&(g2 < nought)) g = y/gg ;
    if ((g1 > nought)&&(g2 < nought)) gg = y/g ;
    if ((g1 < nought)&&(g2 > nought)) gg = y/g ;
    if (g > gmax) g = gmax ;
    if (g < -gmax) g = -gmax ;
    if (gg > gmax) gg = gmax ;
    if (gg < -gmax) gg = -gmax ;
```

```
    n1 = qudrtc(gg,hh,v1, gg*gg - doub4*hh) ;
    n2 = qudrtc(g,h,v2, g*g - doub4*h) ;
    nquar = n1+n2 ;
    rts[0] = v1[0] ;
    rts[1] = v1[1] ;
    rts[n1+0] = v2[0] ;
    rts[n1+1] = v2[1] ;
```

```
    return(nquar);
```

```
    }
} /* neumark */
/*****
```

```
void errors(a,b,c,d,rts,rterr,nrts)
```

```
double a,b,c,d,rts[4],rterr[4];
```

```
int nrts;
```

```
/*
```

```
    find the errors
```

```
    called by quartic.
```

```
*/
```

```
{
```

```
    int k;
```

```
    double deriv,test;
```

```
    double fabs(),sqrt(),curoot();
```

```
    if (nrts > 0)
```

```
    {
```

```
        for ( k = 0 ; k < nrts ; ++ k )
```

```
        {
```

```
            test = (((rts[k]+a)*rts[k]+b)*rts[k]+c)*rts[k]+d ;
```

```
            if (test == nought) rterr[k] = nought;
```

```
            else
```

```
            {
```

```
                deriv =
```

```
                    ((doub4*rts[k]+doub3*a)*rts[k]+doub2*b)*rts[k]+c ;
```

```
                if (deriv != nought)
```

```
                    rterr[k] = fabs(test/deriv);
```

```
                else
```

```
                {
```

```
                    deriv = (doub12*rts[k]+doub6*a)*rts[k]+doub2*b ;
```

```
                    if (deriv != nought)
```

```
                        rterr[k] = sqrt(fabs(test/deriv)) ;
```

```
        else
        {
            deriv = doub24*rts[k]+doub6*a ;
            if (deriv != nought)
                rterr[k] = curoot(fabs(test/deriv));
            else
                rterr[k] = sqrt(sqrt(fabs(test)/doub24));
        }
    }
    fprintf(stderr,"errorsa  %d %9g %9g\n",
            k,rts[k],rterr[k]);
}
}
else fprintf(stderr,"errors ans: none\n");
} /* errors */
/*****/
```

```
int qudrtc(b,c,rts,dis)
double b,c,rts[4],dis ;
/*
    solve the quadratic equation -

        x**2+b*x+c = 0

    called by  quartic, ferrari, neumark, ellcut
*/
{
    int nquad;
    double rtdis ;

    if (dis >= nought)
    {
        nquad = 2 ;
        rtdis = sqrt(dis) ;
        if (b > nought) rts[0] = ( -b - rtdis)*inv2 ;
        else rts[0] = ( -b + rtdis)*inv2 ;
        if (rts[0] == nought) rts[1] = -b ;
        else rts[1] = c/rts[0] ;
    }
    else
    {
        nquad = 0;
        rts[0] = nought ;
        rts[1] = nought ;
    }
    return(nquad);
} /* qudrtc */
/*****/
```

```
double cubic(p,q,r)
double p,q,r;
/*
    find the lowest real root of the cubic -

        x**3 + p*x**2 + q*x + r = 0

    input parameters -
        p,q,r - coeffs of cubic equation.

    output-
        cubic - a real root.
```



```
global constants -
    rt3 - sqrt(3)
    inv3 - 1/3
    doubmax - square root of largest number held by machine

method -
    see D.E. Littlewood, "A University Algebra" pp.173 - 6

Charles Prineas    April 1981

    called by  neumark.
    calls  acos3
```

```
*/
{
```

```
    int nrts;
    double po3,po3sq,qo3;
    double uo3,u2o3,uo3sq4,uo3cu4 ;
    double v,vsq,wsq ;
    double m,mcube,n;
    double muo3,s,scube,t,cosk,sinsqk ;
    double root;
    double curoot();
    double acos3();
    double sqrt(),fabs();

    m = nought;
    nrts =0;
    if ((p > doubmax) || (p < -doubmax)) root = -p;
    else
    if ((q > doubmax) || (q < -doubmax))
    {
        if (q > nought) root = -r/q;
        else root = -sqrt(-q);
    }
    else
    if ((r > doubmax) || (r < -doubmax)) root = -curoot(r) ;
    else
    {
        po3 = p*inv3 ;
        po3sq = po3*po3 ;
        if (po3sq > doubmax) root = -p ;
        else
        {
            v = r + po3*(po3sq + po3sq - q) ;
            if ((v > doubmax) || (v < -doubmax)) root = -p ;
            else
            {
                vsq = v*v ;
                qo3 = q*inv3 ;
                uo3 = qo3 - po3sq ;
                u2o3 = uo3 + uo3 ;
                if ((u2o3 > doubmax) || (u2o3 < -doubmax))
                {
                    if (p == nought)
                    {
                        if (q > nought) root = -r/q ;
                        else root = -sqrt(-q) ;
                    }
                    else root = -q/p ;
                }
            }
        }
    }
}
```

```
    uo3sq4 = u2o3*u2o3 ;
    if (uo3sq4 > doubmax)
    {
        if (p == nought)
        {
            if (q > nought) root = -r/q ;
            else root = -sqrt(fabs(q)) ;
        }
        else root = -q/p ;
    }
    uo3cu4 = uo3sq4*uo3 ;
    wsq = uo3cu4 + vsq ;
    if (wsq >= nought)
    {
/*
    cubic has one real root
*/
        nrts = 1;
        if (v <= nought) mcube = ( -v + sqrt(wsq))*inv2 ;
        if (v > nought) mcube = ( -v - sqrt(wsq))*inv2 ;
        m = curoot(mcube) ;
        if (m != nought) n = -uo3/m ;
        else n = nought;
        root = m + n - po3 ;
    }
    else
    {
        nrts = 3;
/*
    cubic has three real roots
*/
        if (uo3 < nought)
        {
            muo3 = -uo3;
            s = sqrt(muo3) ;
            scube = s*muo3;
            t = -v/(scube+scube) ;
            cosk = acos3(t) ;
            if (po3 < nought)
                root = (s+s)*cosk - po3;
            else
            {
                sinsqk = doub1 - cosk*cosk ;
                if (sinsqk < nought) sinsqk = nought ;
                root = s*( -cosk - rt3*sqrt(sinsqk)) - po3 ;
            }
        }
        else
/*
    cubic has multiple root -
*/
        root = curoot(v) - po3 ;
    }
}

}

}
fprintf(stderr,"cubic %g %g %g %d %g\n",p,q,r,nrts,root);
return(root);
} /* cubic */
/*****/
```

```
double acos3(x)
    double x ;
/*
    find cos(acos(x)/3)

    Don Herbison-Evans    16/7/81

    called by cubic .
*/
{
    double value;
    double acos(),cos();

    value = cos(acos(x)*inv3);
    return(value);
} /* acos3 */
/*****/

double curoot(x)
    double x ;
/*
    find cube root of x.

    Don Herbison-Evans    30/1/89

    called by cubic .
*/
{
    double exp(),log();
    double value;
    double absx;
    int neg;

    neg = 0;
    absx = x;
    if (x < nought)
    {
        absx = -x;
        neg = 1;
    }
    value = exp( log(absx)*inv3 );
    if (neg == 1) value = -value;
    return(value);
} /* curoot */
/*****/

int simple(a,b,c,d,rts)
    double a,b,c,d,rts[4];
/*
    solve the quartic equation -

    x**4 + a*x**3 + b*x**2 + c*x + d = 0

    called by quartic
    calls      cubic, qudrtc.

    input -
    a,b,c,e - coeffs of equation.

    output -
    nquar - number of real roots.
```

rts - array of root values.

method : unstabilized Ferrari-Lagrange  
Abramowitz, M. & Stegun I.A.  
Handbook of Mathematical Functions  
Dover 1972 (ninth printing), pp. 17-18

calls cubic, qudrtc

```
*/
{
double cubic();
int qudrtc();

int nquar,n1,n2 ;
double asq,y;
double v1[4],v2[4] ;
double p,q,r ;
double e,f,esq,fsq ;
double g,gg,h,hh;

fprintf(stderr,"\nsimple %g %g %g %g\n",a,b,c,d);
asq = a*a;

p = -b ;
q = a*c-doub4*d ;
r = -asq*d - c*c + doub4*b*d ;
y = cubic(p,q,r) ;

esq = inv4*asq - b + y;
fsq = inv4*y*y - d;
if (esq < nought) return(0);
else
if (fsq < nought) return(0);
else
{
e = sqrt(esq);
f = sqrt(fsq);
g = inv2*a - e;
h = inv2*y - f;
gg = inv2*a + e;
hh = inv2*y + f;
n1 = qudrtc(gg,hh,v1, gg*gg - doub4*hh) ;
n2 = qudrtc(g,h,v2, g*g - doub4*h) ;
nquar = n1+n2 ;
rts[0] = v1[0] ;
rts[1] = v1[1] ;
rts[n1+0] = v2[0] ;
rts[n1+1] = v2[1] ;
return(nquar);
}
} /* simple */
/*****/
```

```
int descartes(a,b,c,d,rts)
double a,b,c,d,rts[4];
/*
```

Solve quartic equation using  
Descartes-Euler-Cardano algorithm

Strong, T. "Elementary and Higher Algebra"  
Pratt and Oakley, p. 469 (1859)

29 Jun 1994 Don Herbison-Evans

```
*/
{
    int qudrtc();
    double cubic();

    int nrts;
    int r1,r2;
    double v1[4],v2[4];
    double y;
    double p,q,r;
    double A,B,C;
    double m,n1,n2;
    double d3o8,d3o256;
    double inv8,inv16;
    double asq;
    double Binvm;

    d3o8 = (double)3/(double)8;
    inv8 = doub1/(double)8;
    inv16 = doub1/(double)16;
    d3o256 = (double)3/(double)256;

    fprintf(stderr,"\nDescartes %f %f %f %f\n",a,b,c,d);
    asq = a*a;

    A = b - asq*d3o8;
    B = c + a*(asq*inv8 - b*inv2);
    C = d + asq*(b*inv16 - asq*d3o256) - a*c*inv4;

    p = doub2*A;
    q = A*A - doub4*C;
    r = -B*B;



    /**
    inv64 = doub1/(double)64;
    p = doub2*b - doub3*a*a*inv4 ;
    q = b*b - a*a*b - doub4*d + doub3*a*a*a*a*inv16 + a*c;
    r = -c*c - a*a*a*a*a*a*inv64 - a*a*b*b*inv4
        -a*a*a*c*inv4 + a*b*c + a*a*a*a*b*inv8;
    ***/

    y = cubic(p,q,r) ;
    if (y <= nought)
        nrts = 0;
    else
    {
        m = sqrt(y);
        Binvm = B/m;
        n1 = (y + A + Binvm)*inv2;
        n2 = (y + A - Binvm)*inv2;
        r1 = qudrtc(-m, n1, v1, y-doub4*n1);
        r2 = qudrtc( m, n2, v2, y-doub4*n2);
        rts[0] = v1[0]-a*inv4;
        rts[1] = v1[1]-a*inv4;
        rts[r1] = v2[0]-a*inv4;
        rts[r1+1] = v2[1]-a*inv4;
        nrts = r1+r2;
    }
    return(nrts);
}
```

```
} /* descartes */  
/***** */
```

# Index of

## /pubs/tog/GraphicsGems/gemsv/ch2-6/

Name	Last modified	Size	Description
 <a href="#">_Parent Directory</a>			
 <a href="#">_conmat.c</a>	29-Jun-00 08:22	21K	

```
/* CONMAT.C - Ellipse tranformation and intersection functions */
/* Written by Kenneth J. Hill, June, 24, 1994 */
```

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <values.h>
```

```
#ifndef M_PI
#define M_PI      3.14159265358979323846
#endif
```

```
#ifndef M_PI_2
#define M_PI_2    1.57079632679489661923
#endif
```

```
#ifndef max
#define max(a,b) ((a)>=(b) ? (a) : (b) )
#endif
```

```
/* Type definitions */
```

```
/* Point structure */
typedef struct PointTag
{
    double X,Y;
} Point;
```

```
/* Line structure */
typedef struct LineTag
{
    Point P1,P2;
} Line;
```

```
/* Circle structure */
typedef struct CircleTag
{
    Point Center;
    double Radius;
} Circle;
```

```
/* Ellipse structure */
typedef struct EllipseTag
{
    Point Center;                /* ellipse center      */
    double MaxRad,MinRad;        /* major and minor axis */
    double Phi;                  /* major axis rotation  */
} Ellipse;
```

```
/* Conic coefficients structure */
typedef struct ConicTag
{
    double A,B,C,D,E,F;
} Conic;
```

```
/* transformation matrix type */
typedef struct TMatTag
{
    double a,b,c,d;             /* tranformation coefficients */
    double m,n;                 /* translation coefficients    */
} TMat;
```



```
/* prototypes */

/* Functions with names beginning with a O:
 *
 * 1. Ignore ellipse center coordinates.
 *
 * 2. Ignore any translation components in tranformation matrices.
 *
 * 3. Assume that conic coefficients were generated by "O-name" functions.
 *
 * This keeps the computations relatively simple. The ellipse centers
 * can then be transformed separately as points.
 */

/* OTransformConic - Transform conic coefficients about the origin */
void OTransformConic(Conic *ConicP,TMat *TMatP);

/* OGenEllipseCoefs - Generate conic coefficients of an ellipse */
void OGenEllipseCoefs(Ellipse *EllipseP,Conic *ConicP);

/* OGenEllipseGeom - Generates ellipse geometry from conic coefficients */
void OGenEllipseGeom(Conic *ConicP,Ellipse *EllipseP);

/* TransformPoint - Transform a point using a tranformation matrix */
void TransformPoint(Point *PointP,TMat *TMatP);

/* TransformEllipse - Transform an ellipse using a tranformation matrix */
void TransformEllipse(Ellipse *EllipseP,TMat *TMatP);

/* The following functions are defined in cubicand.c, on
   the Graphic Gems I diskette. */
int SolveCubic(double c[4],double s[3]);
int SolveQuadic(double c[3],double dest[2]);

/* Identity matrix */
static const TMat IdentMat={1.0,0.0,0.0,1.0,0.0,0.0};

/* Transformation matrix routines */

/* Translate a matrix by m,n */
void TranslateMat(TMat *Mat,double m,double n)
{
    Mat->m += m;
    Mat->n += n;
}

/* Rotate a matrix by Phi */
void RotateMat(TMat *Mat,double Phi)
{
    double SinPhi=sin(Phi);
    double CosPhi=cos(Phi);
    TMat temp=*Mat;                /* temporary copy of Mat */

    /* These are just the matrix operations written out long hand */
    Mat->a = temp.a*CosPhi - temp.b*SinPhi;
    Mat->b = temp.b*CosPhi + temp.a*SinPhi;
    Mat->c = temp.c*CosPhi - temp.d*SinPhi;
    Mat->d = temp.d*CosPhi + temp.c*SinPhi;
    Mat->m = temp.m*CosPhi - temp.n*SinPhi;
```

```
    Mat->n = temp.n*CosPhi + temp.m*SinPhi;
}

/* Scale a matrix by sx, sy */
void ScaleMat(TMat *Mat,double sx,double sy)
{
    Mat->a *= sx;
    Mat->b *= sy;
    Mat->c *= sx;
    Mat->d *= sy;
    Mat->m *= sx;
    Mat->n *= sy;
}

/* TransformPoint - Transform a point using a tranformation matrix */
void TransformPoint(Point *PointP,TMat *TMatP)
{
    Point TempPoint;

    TempPoint.X = PointP->X*TMatP->a + PointP->Y*TMatP->c + TMatP->m;
    TempPoint.Y = PointP->X*TMatP->b + PointP->Y*TMatP->d + TMatP->n;
    *PointP=TempPoint;
}

/* Conic routines */

/* near zero test */
#define EPSILON 1e-9
#define IsZero(x) (x > -EPSILON && x < EPSILON)

/* GenEllipseCoefs - Generate conic coefficients of an ellipse */
static void GenEllipseCoefs(Ellipse *elip,Conic *M)
{
    double sqr_r1,sqr_r2;
    double sint, cost,sin2t,sqr_sint,sqr_cost;
    double cenx,ceny,sqr_cenx,sqr_ceny,inv_sqr_r1,inv_sqr_r2;

    /* common coefficients */
    sqr_r1 = elip->MaxRad;
    sqr_r2 = elip->MinRad;
    sqr_r1 *= sqr_r1;
    sqr_r2 *= sqr_r2;
    sint = sin(elip->Phi);
    cost = cos(elip->Phi);
    sin2t = 2.0*sint*cost;
    sqr_sint = sint*sint;
    sqr_cost = cost*cost;
    cenx = elip->Center.X;
    sqr_cenx = cenx*cenx;
    ceny = elip->Center.Y;
    sqr_ceny = ceny*ceny;
    inv_sqr_r1 = 1.0/sqr_r1;
    inv_sqr_r2 = 1.0/sqr_r2;

    /* Compute the coefficients. These formulae are the transformations
       on the unit circle written out long hand */
    M->A = sqr_cost/sqr_r1 + sqr_sint/sqr_r2;
    M->B = (sqr_r2-sqr_r1)*sin2t/(2.0*sqr_r1*sqr_r2);
    M->C = sqr_cost/sqr_r2 + sqr_sint/sqr_r1;
    M->D = -ceny*M->B-cenx*M->A;
    M->E = -cenx*M->B-ceny*M->C;
}
```

```
M->F = -1.0 + (sqr_cenx + sqr_ceny)*(invsqr_r1 + invsqr_r2)/2.0 +
        (sqr_cost - sqr_sint)*(sqr_cenx - sqr_ceny)*(invsqr_r1 - invsqr_r2)/2.0 +
        cenx*ceny*(invsqr_r1 - invsqr_r2)*sin2t;
}
```

```
/* Compute the transformation which turns an ellipse into a circle */
void Elp2Cir(Ellipse *Elp,TMat *CirMat)
```

```
{
    /* Start with identity matrix */
    *CirMat = IdentMat;
    /* Translate to origin */
    TranslateMat(CirMat,-Elp->Center.X,-Elp->Center.Y);
    /* Rotate into standard position */
    RotateMat(CirMat,-Elp->Phi);
    /* Scale into a circle. */
    ScaleMat(CirMat,1.0/Elp->MaxRad,1.0/Elp->MinRad);
}
```

```
/* Compute the inverse of the transformation
   which turns an ellipse into a circle */
```

```
void InvElp2Cir(Ellipse *Elp,TMat *InvMat)
```

```
{
    /* Start with identity matrix */
    *InvMat = IdentMat;
    /* Scale back into an ellipse. */
    ScaleMat(InvMat,Elp->MaxRad,Elp->MinRad);
    /* Rotate */
    RotateMat(InvMat,Elp->Phi);
    /* Translate from origin */
    TranslateMat(InvMat,Elp->Center.X,Elp->Center.Y);
}
```

```
/* OTransformConic - Transform conic coefficients about the origin */
```

```
/* This routine ignores the translation components of *TMatP and
   assumes the conic is "centered" at the origin (i.e. D,E=0, F=-1) */
```

```
/* The computations are just the matrix operations written out long hand */
```

```
/* This code assumes that the transformation is not degenerate */
```

```
void OTransformConic(Conic *ConicP,TMat *TMatP)
```

```
{
    double A,B,C,Denom;

    /* common denominator for transformed coefficients */
    Denom = TMatP->a*TMatP->d - TMatP->b*TMatP->c;
    Denom *= Denom;

    A = (ConicP->C*TMatP->b*TMatP->b - 2.0*ConicP->B*TMatP->b*TMatP->d +
        ConicP->A*TMatP->d*TMatP->d)/Denom;

    B = (-ConicP->C*TMatP->a*TMatP->b + ConicP->B*TMatP->b*TMatP->c +
        ConicP->B*TMatP->a*TMatP->d - ConicP->A*TMatP->c*TMatP->d)/Denom;

    C = (ConicP->C*TMatP->a*TMatP->a - 2.0*ConicP->B*TMatP->a*TMatP->c +
        ConicP->A*TMatP->c*TMatP->c)/Denom;

    ConicP->A=A;
    ConicP->B=B;
    ConicP->C=C;
}
```

```
/* OGenEllipseCoefs - Generate conic coefficients of an ellipse */
```

```
/* The ellipse is assumed to be centered at the origin. */
```

```
void OGenEllipseCoefs(Ellipse *EllipseP, Conic *ConicP)
{
    double SinPhi = sin(EllipseP->Phi); /* sine of ellipse rotation */
    double CosPhi = cos(EllipseP->Phi); /* cosine of ellipse rotation */
    double SqSinPhi = SinPhi*SinPhi;    /* square of sin(phi) */
    double SqCosPhi = CosPhi*CosPhi;    /* square of cos(phi) */
    double SqMaxRad = EllipseP->MaxRad*EllipseP->MaxRad;
    double SqMinRad = EllipseP->MinRad*EllipseP->MinRad;

    /* compute coefficients for the ellipse in standard position */
    ConicP->A = SqCosPhi/SqMaxRad + SqSinPhi/SqMinRad;
    ConicP->B = (1.0/SqMaxRad - 1.0/SqMinRad)*SinPhi*CosPhi;
    ConicP->C = SqCosPhi/SqMinRad + SqSinPhi/SqMaxRad;
    ConicP->D = ConicP->E = 0.0;
    ConicP->F = -1.0;
}

/* OGenEllipseGeom - Generates ellipse geometry from conic coefficients */
/* This routine assumes the conic coefficients D=E=0, F=-1 */
void OGenEllipseGeom(Conic *ConicP, Ellipse *EllipseP)
{
    double Numer, Denom, Temp;
    TMat DiagTransform; /* transform diagonalization */
    Conic ConicT = *ConicP; /* temporary copy of conic coefficients */
    double SinPhi, CosPhi;

    /* compute new ellipse rotation */
    Numer = ConicT.B + ConicT.B;
    Denom = ConicT.A - ConicT.C;
    /* Phi = 1/2 atan(Numer/Denom) = 1/2 (pi/2 - atan(Denom/Numer))
       We use the form that keeps the argument to atan between -1 and 1 */

    EllipseP->Phi = 0.5*(fabs(Numer) < fabs(Denom)?
        atan(Numer/Denom):M_PI_2-atan(Denom/Numer));

    /* diagonalize the conic */
    SinPhi = sin(EllipseP->Phi);
    CosPhi = cos(EllipseP->Phi);
    DiagTransform.a = CosPhi; /* rotate by -Phi */
    DiagTransform.b = -SinPhi;
    DiagTransform.c = SinPhi;
    DiagTransform.d = CosPhi;
    DiagTransform.m = DiagTransform.n = 0.0;
    OTransformConic(&ConicT, &DiagTransform);

    /* compute new radii from diagonalized coefficients */
    EllipseP->MaxRad = 1.0/sqrt(ConicT.A);
    EllipseP->MinRad = 1.0/sqrt(ConicT.C);

    /* be sure MaxRad >= MinRad */
    if (EllipseP->MaxRad < EllipseP->MinRad)
    {
        Temp = EllipseP->MaxRad; /* exchange the radii */
        EllipseP->MaxRad = EllipseP->MinRad;
        EllipseP->MinRad = Temp;
        EllipseP->Phi += M_PI_2; /* adjust the rotation */
    }
}

/* TransformEllipse - Transform an ellipse using a transformation matrix */
void TransformEllipse(Ellipse *EllipseP, TMat *TMatP)
```

```
{
Conic EllipseCoefs;

/* generate the ellipse coefficients (using Center=origin) */
OGenEllipseCoefs(EllipseP,&EllipseCoefs);

/* transform the coefficients */
OTransformConic(&EllipseCoefs,TMatP);

/* turn the transformed coefficients back into geometry */
OGenEllipseGeom(&EllipseCoefs,EllipseP);

/* translate the center */
TransformPoint(&EllipseP->Center,TMatP);
}

/* MultMat3 - Multiply two 3x3 matrices */
void MultMat3(double *Mat1,double *Mat2, double *Result)
{
    int i,j;

    for (i = 0;i < 3;i++)
        for (j = 0;j < 3;j++)
            Result[i*3+j] = Mat1[i*3+0]*Mat2[0*3+j] +
                Mat1[i*3+1]*Mat2[1*3+j] +
                Mat1[i*3+2]*Mat2[2*3+j];
}

/* Transform a conic by a transformation matrix */
void TransformConic(Conic *ConicP,TMat *TMatP)
{
    double InvMat[3][3],ConMat[3][3],TranInvMat[3][3];
    double Result1[3][3],Result2[3][3];
    double D;
    int i,j;

    /* Compute M' = Inv(TMat).M.Transpose(Inv(TMat))

    /* compute the transformation using matrix multiplication */
    ConMat[0][0] = ConicP->A;
    ConMat[0][1] = ConicP->B;
    ConMat[1][0] = ConicP->B;
    ConMat[1][1] = ConicP->C;
    ConMat[0][2] = ConicP->D;
    ConMat[2][0] = ConicP->D;
    ConMat[1][2] = ConicP->E;
    ConMat[2][1] = ConicP->E;
    ConMat[2][2] = ConicP->F;

    /* inverse transformation */
    D = TMatP->a*TMatP->d - TMatP->b*TMatP->c;
    InvMat[0][0] = TMatP->d/D;
    InvMat[0][1] = -TMatP->b/D;
    InvMat[0][2] = 0.0;
    InvMat[1][0] = -TMatP->c/D;
    InvMat[1][1] = TMatP->a/D;
    InvMat[1][2] = 0.0;
    InvMat[2][0] = (TMatP->c*TMatP->n - TMatP->d*TMatP->m)/D;
    InvMat[2][1] = (TMatP->b*TMatP->m - TMatP->a*TMatP->n)/D;
    InvMat[2][2] = 1.0;

    /* compute transpose */
    for (i = 0;i < 3;i++)
        for (j = 0;j < 3;j++)
```

```
TranInvMat[j][i] = InvMat[i][j];
```

```
/* multiply the matrices */
```

```
MultMat3((double *)InvMat,(double *)ConMat,(double *)Result1);
```

```
MultMat3((double *)Result1,(double *)TranInvMat,(double *)Result2);
```

```
ConicP->A = Result2[0][0]; /* return to conic form */
```

```
ConicP->B = Result2[0][1];
```

```
ConicP->C = Result2[1][1];
```

```
ConicP->D = Result2[0][2];
```

```
ConicP->E = Result2[1][2];
```

```
ConicP->F = Result2[2][2];
```

```
}
```

```
/* Compute the intersection of a circle and a line */
```

```
/* See Graphic Gems Volume 1, page 5 for a description of this algorithm */
```

```
int IntCirLine(Point *IntPts,Circle *Cir,Line *Ln)
```

```
{
```

```
Point G,V;
```

```
double a,b,c,d,t,sqrt_d;
```

```
G.X = Ln->P1.X - Cir->Center.X; /* G = Ln->P1 - Cir->Center */
```

```
G.Y = Ln->P1.Y - Cir->Center.Y;
```

```
V.X = Ln->P2.X - Ln->P1.X; /* V = Ln->P2 - Ln->P1 */
```

```
V.Y = Ln->P2.Y - Ln->P1.Y;
```

```
a = V.X*V.X + V.Y*V.Y; /* a = V.V */
```

```
b = V.X*G.X + V.Y*G.Y;b += b; /* b = 2(V.G) */
```

```
c = (G.X*G.X + G.Y*G.Y) - /* c = G.G + Circle->Radius^2 */  
Cir->Radius*Cir->Radius;
```

```
d = b*b - 4.0*a*c; /* discriminant */
```

```
if (d <= 0.0)
```

```
return 0; /* no intersections */
```

```
sqrt_d = sqrt(d);
```

```
t = (-b + sqrt_d)/(a + a); /* t = (-b +/- sqrt(d))/2a */
```

```
IntPts[0].X = Ln->P1.X + t*V.X; /* Pt = Ln->P1 + t V */
```

```
IntPts[0].Y = Ln->P1.Y + t*V.Y;
```

```
t = (-b - sqrt_d)/(a + a);
```

```
IntPts[1].X = Ln->P1.X + t*V.X;
```

```
IntPts[1].Y = Ln->P1.Y + t*V.Y;
```

```
return 2;
```

```
}
```

```
/* compute all intersections of two ellipses */
```

```
/* E1 and E2 are the two ellipses */
```

```
/* IntPts points to an array of twelve points
```

```
(some duplicates may be returned) */
```

```
/* The number of intersections found is returned */
```

```
/* Both ellipses are assumed to have non-zero radii */
```

```
int Int2Elip(Point *IntPts,Ellipse *E1,Ellipse *E2)
```

```
{
```

```
TMat ElpCirMat1,ElpCirMat2,InvMat,TempMat;
```

```
Conic Conic1,Conic2,Conic3,TempConic;
```

```
double Roots[3],qRoots[2];
```

```
static Circle TestCir = {{0.0,0.0},1.0};
```

```
Line TestLine[2];
```

```
Point TestPoint;
```

```
double PolyCoef[4]; /* coefficients of the polynomial */
```

```
double D; /* discriminant: B^2 - AC */
```

```
double Phi; /* ellipse rotation */
```

```
double m,n; /* ellipse translation */
```

```
double Scl;                /* scaling factor */
int NumRoots, NumLines;
int CircleInts;            /* intersections between line and circle */
int IntCount = 0;         /* number of intersections found */
int i, j, k;

/* compute the transformations which turn E1 and E2 into circles */
Elp2Cir(E1, &ElpCirMat1);
Elp2Cir(E2, &ElpCirMat2);

/* compute the inverse transformation of ElpCirMat1 */
InvElp2Cir(E1, &InvMat);

/* Compute the characteristic matrices */
GenEllipseCoefs(E1, &Conic1);
GenEllipseCoefs(E2, &Conic2);

/* Find x such that Det(Conic1 + x Conic2) = 0 */
PolyCoef[0] = -Conic1.C*Conic1.D*Conic1.D + 2.0*Conic1.B*Conic1.D*Conic1.E -
    Conic1.A*Conic1.E*Conic1.E - Conic1.B*Conic1.B*Conic1.F +
    Conic1.A*Conic1.C*Conic1.F;
PolyCoef[1] = -(Conic2.C*Conic1.D*Conic1.D) -
    2.0*Conic1.C*Conic1.D*Conic2.D + 2.0*Conic2.B*Conic1.D*Conic1.E +
    2.0*Conic1.B*Conic2.D*Conic1.E - Conic2.A*Conic1.E*Conic1.E +
    2.0*Conic1.B*Conic1.D*Conic2.E - 2.0*Conic1.A*Conic1.E*Conic2.E -
    2.0*Conic1.B*Conic2.B*Conic1.F + Conic2.A*Conic1.C*Conic1.F +
    Conic1.A*Conic2.C*Conic1.F - Conic1.B*Conic1.B*Conic2.F +
    Conic1.A*Conic1.C*Conic2.F;
PolyCoef[2] = -2.0*Conic2.C*Conic1.D*Conic2.D - Conic1.C*Conic2.D*Conic2.D +
    2.0*Conic2.B*Conic2.D*Conic1.E + 2.0*Conic2.B*Conic1.D*Conic2.E +
    2.0*Conic1.B*Conic2.D*Conic2.E - 2.0*Conic2.A*Conic1.E*Conic2.E -
    Conic1.A*Conic2.E*Conic2.E - Conic2.B*Conic2.B*Conic1.F +
    Conic2.A*Conic2.C*Conic1.F - 2.0*Conic1.B*Conic2.B*Conic2.F +
    Conic2.A*Conic1.C*Conic2.F + Conic1.A*Conic2.C*Conic2.F;
PolyCoef[3] = -Conic2.C*Conic2.D*Conic2.D + 2.0*Conic2.B*Conic2.D*Conic2.E -
    Conic2.A*Conic2.E*Conic2.E - Conic2.B*Conic2.B*Conic2.F +
    Conic2.A*Conic2.C*Conic2.F;
NumRoots = SolveCubic(PolyCoef, Roots);

if (NumRoots == 0)
    return 0;

/* we try all the roots, even though it's redundant, so that we
   avoid some pathological situations */
for (i=0; i<NumRoots; i++)
{
    NumLines = 0;

    /* Conic3 = Conic1 + mu Conic2 */
    Conic3.A = Conic1.A + Roots[i]*Conic2.A;
    Conic3.B = Conic1.B + Roots[i]*Conic2.B;
    Conic3.C = Conic1.C + Roots[i]*Conic2.C;
    Conic3.D = Conic1.D + Roots[i]*Conic2.D;
    Conic3.E = Conic1.E + Roots[i]*Conic2.E;
    Conic3.F = Conic1.F + Roots[i]*Conic2.F;

    D = Conic3.B*Conic3.B - Conic3.A*Conic3.C;
    if (IsZero(Conic3.A) && IsZero(Conic3.B) && IsZero(Conic3.C))
    {
        /* Case 1 - Single line */
        NumLines = 1;
    }
}
```

```
/* compute endpoints of the line, avoiding division by zero */
if (fabs(Conic3.D) > fabs(Conic3.E))
{
    TestLine[0].P1.Y = 0.0;
    TestLine[0].P1.X = -Conic3.F/(Conic3.D + Conic3.D);
    TestLine[0].P2.Y = 1.0;
    TestLine[0].P2.X = -(Conic3.E + Conic3.E + Conic3.F)/
        (Conic3.D + Conic3.D);
}
else
{
    TestLine[0].P1.X = 0.0;
    TestLine[0].P1.Y = -Conic3.F/(Conic3.E + Conic3.E);
    TestLine[0].P2.X = 1.0;
    TestLine[0].P2.X = -(Conic3.D + Conic3.D + Conic3.F)/
        (Conic3.E + Conic3.E);
}
}
else
{
    /* use the espresion for Phi that takes atan of the
       smallest argument */
    Phi = (fabs(Conic3.B + Conic3.B) < fabs(Conic3.A-Conic3.C)?
        atan((Conic3.B + Conic3.B)/(Conic3.A - Conic3.C)):
        M_PI_2 - atan((Conic3.A - Conic3.C)/(Conic3.B + Conic3.B)))/2.0;
    if (IsZero(D))
    {
        /* Case 2 - Parallel lines */
        TempConic = Conic3;
        TempMat = IdentMat;
        RotateMat(&TempMat,-Phi);
        TransformConic(&TempConic,&TempMat);
        if (IsZero(TempConic.C)) /* vertical */
        {
            PolyCoef[0] = TempConic.F;
            PolyCoef[1] = TempConic.D;
            PolyCoef[2] = TempConic.A;
            if ((NumLines=SolveQuadic(PolyCoef,qRoots))!=0)
            {
                TestLine[0].P1.X = qRoots[0];
                TestLine[0].P1.Y = -1.0;
                TestLine[0].P2.X = qRoots[0];
                TestLine[0].P2.Y = 1.0;
                if (NumLines==2)
                {
                    TestLine[1].P1.X = qRoots[1];
                    TestLine[1].P1.Y = -1.0;
                    TestLine[1].P2.X = qRoots[1];
                    TestLine[1].P2.Y = 1.0;
                }
            }
        }
        else /* horizontal */
        {
            PolyCoef[0] = TempConic.F;
            PolyCoef[1] = TempConic.E;
            PolyCoef[2] = TempConic.C;
            if ((NumLines=SolveQuadic(PolyCoef,qRoots))!=0)
            {
                TestLine[0].P1.X = -1.0;
                TestLine[0].P1.Y = qRoots[0];
            }
        }
    }
}
```



```
        TestLine[0].P2.X = 1.0;
        TestLine[0].P2.Y = qRoots[0];
        if (NumLines==2)
        {
            TestLine[1].P1.X = -1.0;
            TestLine[1].P1.Y = qRoots[1];
            TestLine[1].P2.X = 1.0;
            TestLine[1].P2.Y = qRoots[1];
        }
    }
    TempMat = IdentMat;
    RotateMat(&TempMat,Phi);
    TransformPoint(&TestLine[0].P1,&TempMat);
    TransformPoint(&TestLine[0].P2,&TempMat);
    if (NumLines==2)
    {
        TransformPoint(&TestLine[1].P1,&TempMat);
        TransformPoint(&TestLine[1].P2,&TempMat);
    }
}
else
{
    /* Case 3 - Crossing lines */
    NumLines = 2;

    /* translate the system so that the intersection of the lines
       is at the origin */
    TempConic = Conic3;
    m = (Conic3.C*Conic3.D - Conic3.B*Conic3.E)/D;
    n = (Conic3.A*Conic3.E - Conic3.B*Conic3.D)/D;
    TempMat = IdentMat;
    TranslateMat(&TempMat,-m,-n);
    RotateMat(&TempMat,-Phi);
    TransformConic(&TempConic,&TempMat);

    /* Compute the line endpoints */
    TestLine[0].P1.X = sqrt(fabs(1.0/TempConic.A));
    TestLine[0].P1.Y = sqrt(fabs(1.0/TempConic.C));
    Scl = max(TestLine[0].P1.X,TestLine[0].P1.Y); /* adjust range */
    TestLine[0].P1.X /= Scl;
    TestLine[0].P1.Y /= Scl;
    TestLine[0].P2.X = - TestLine[0].P1.X;
    TestLine[0].P2.Y = - TestLine[0].P1.Y;
    TestLine[1].P1.X = TestLine[0].P1.X;
    TestLine[1].P1.Y = - TestLine[0].P1.Y;
    TestLine[1].P2.X = - TestLine[1].P1.X;
    TestLine[1].P2.Y = - TestLine[1].P1.Y;

    /* translate the lines back */
    TempMat = IdentMat;
    RotateMat(&TempMat,Phi);
    TranslateMat(&TempMat,m,n);
    TransformPoint(&TestLine[0].P1,&TempMat);
    TransformPoint(&TestLine[0].P2,&TempMat);
    TransformPoint(&TestLine[1].P1,&TempMat);
    TransformPoint(&TestLine[1].P2,&TempMat);
}
}
```

```
/* find the ellipse line intersections */
```

```
for (j = 0; j < NumLines; j++)
{
    /* transform the line endpts into the circle space of the ellipse */
    TransformPoint(&TestLine[j].P1, &ElpCirMat1);
    TransformPoint(&TestLine[j].P2, &ElpCirMat1);

    /* compute the number of intersections of the transformed line
       and test circle */
    CircleInts = IntCirLine(&IntPts[IntCount], &TestCir, &TestLine[j]);
    if (CircleInts > 0)
    {
        /* transform the intersection points back into ellipse space */
        for (k = 0; k < CircleInts; k++)
            TransformPoint(&IntPts[IntCount+k], &InvMat);
        /* update the number of intersections found */
        IntCount += CircleInts;
    }
}

/* validate the points */
j = IntCount;
IntCount = 0;
for (i = 0; i < j; i++)
{
    TestPoint = IntPts[i];
    TransformPoint(&TestPoint, &ElpCirMat2);
    if (TestPoint.X < 2.0 && TestPoint.Y < 2.0 &&
        IsZero(1.0 - sqrt(TestPoint.X*TestPoint.X +
                          TestPoint.Y*TestPoint.Y)))
        IntPts[IntCount++] = IntPts[i];
}

return IntCount;
}

/* Test routines */

/* Ellipse with center at (1,2), major radius 2, minor radius 1,
   and rotation 0. */
Ellipse TestEllipse = {{2.0, 1.0}, 2.0, 1.0, 0.0};

/* Transform matrix for shear of 45 degrees from vertical */
TMat TestTransform = {1.0, 0.0, 1.0, 1.0, 0.0, 0.0};

/* Display an ellipse. This version lists the structure values. */
void DisplayEllipse(Ellipse *EllipseP)
{
    printf("\tCenter at (%6.3f,%6.3f)\n"
           "\tMajor radius: %6.3f\n"
           "\tMinor radius: %6.3f\n"
           "\tRotation: %6.3f degrees\n\n",
           EllipseP->Center.X, EllipseP->Center.Y,
           EllipseP->MaxRad, EllipseP->MinRad,
           180.0*EllipseP->Phi/M_PI);
}

/* test ellipses for intersection */
Ellipse Elp1 = {{5.0, 4.0}, 1.0, 0.5, M_PI/3.0};
Ellipse Elp2 = {{4.0, 3.0}, 2.0, 1.0, 0.0};
Ellipse Elp3 = {{1.0, 1.0}, 2.0, 1.0, M_PI_2};
Ellipse Elp4 = {{1.0, 1.0}, 2.0, 0.5, 0.0};
```

```
void main(void)
{
    Point IntPts[12];
    int IntCount;
    int i;

    /* find ellipse intersections */
    printf("Intersections of ellipses 1 & 2:\n\n");
    IntCount=Int2Elip(IntPts,&Elp1,&Elp2);
    for (i = 0;i < IntCount;i++)
        printf("    %f, %f\n",IntPts[i].X,IntPts[i].Y);
    printf("Intersections of ellipses 3 & 4:\n");
    IntCount=Int2Elip(IntPts,&Elp3,&Elp4);
    for (i = 0;i < IntCount;i++)
        printf("    %f, %f\n",IntPts[i].X,IntPts[i].Y);

    /* transform ellipses */
    printf("\n\nBefore transformation:\n");
    DisplayEllipse(&TestEllipse);

    TransformEllipse(&TestEllipse,&TestTransform);

    printf("After transformation:\n");
    DisplayEllipse(&TestEllipse);
}
```

```
/* fsqrt.c
 *
 * A fast square root program adapted from the code of
 * Paul Lalonde and Robert Dawson in Graphics Gems I.
 * The format of IEEE double precision floating point numbers is:
 *
 * SEEEEEEEEEEEEEMMM MMMMMMMMMMMMMMMMMM MMMMMMMMMMMMMMMMMM MMMMMMMMMMMMMMMMMM
 *
 * S = Sign bit for whole number
 * E = Exponent bit (exponent in excess 1023 form)
 * M = Mantissa bit
 */

#include <stdio.h>
#include <math.h>

/* MOST_SIG_OFFSET gives the (int *) offset from the address of the double
 * to the part of the number containing the sign and exponent.
 * You will need to find the relevant offset for your architecture.
 */

#define MOST_SIG_OFFSET 1

/* SQRT_TAB_SIZE - the size of the lookup table - must be a power of four.
 */

#define SQRT_TAB_SIZE 16384

/* MANT_SHIFTS is the number of shifts to move mantissa into position.
 * If you quadruple the table size subtract two from this constant,
 * if you quarter the table size then add two.
 * Valid values are: (16384, 7) (4096, 9) (1024, 11) (256, 13)
 */

#define MANT_SHIFTS 7

#define EXP_BIAS 1023 /* Exponents are always positive */
#define EXP_SHIFTS 20 /* Shifts exponent to least sig. bits */
#define EXP_LSB 0x00100000 /* 1 << EXP_SHIFTS */
#define MANT_MASK 0x000FFFFF /* Mask to extract mantissa */

int sqrt_tab[SQRT_TAB_SIZE];

void
init_sqrt_tab()
{
    int i;
    double f;
    unsigned int *fi = (unsigned int *) &f + MOST_SIG_OFFSET;

    for (i = 0; i < SQRT_TAB_SIZE/2; i++)
    {
        f = 0; /* Clears least sig part */
        *fi = (i << MANT_SHIFTS) | (EXP_BIAS << EXP_SHIFTS);
        f = sqrt(f);
        sqrt_tab[i] = *fi & MANT_MASK;

        f = 0; /* Clears least sig part */
        *fi = (i << MANT_SHIFTS) | ((EXP_BIAS + 1) << EXP_SHIFTS);
        f = sqrt(f);
        sqrt_tab[i + SQRT_TAB_SIZE/2] = *fi & MANT_MASK;
    }
}
```

```
    }
}

double
fsqrt(f)
double f;
{
    unsigned int e;
    unsigned int *fi = (unsigned int *) &f + MOST_SIG_OFFSET;





    if (f == 0.0) return(0.0);
    e = (*fi >> EXP_SHIFTS) - EXP_BIAS;
    *fi &= MANT_MASK;
    if (e & 1)
        *fi |= EXP_LSB;
    e >>= 1;
    *fi = (sqrt_tab[*fi >> MANT_SHIFTS]) |
        ((e + EXP_BIAS) << EXP_SHIFTS);
    return(f);
}

void
dump_sqrt_tab()
{
    int i, nl = 0;

    printf("unsigned int sqrt_tab[] = {\n");
    for (i = 0; i < Sqrt_TAB_SIZE-1; i++)
    {
        printf("0x%x,", sqrt_tab[i]);
        nl++;
        if (nl > 8) { nl = 0; putchar('\n'); }
    }
    printf("0x%x\n", sqrt_tab[Sqrt_TAB_SIZE-1]);
    printf("};\n");
}
```

# Index of

## /pubs/tog/GraphicsGems/gemsiii/alloc/

Name	Last modified	Size	Description
 <a href="#">_Parent Directory</a>			
 <a href="#">_Makefile</a>	29-Jun-00 08:16	1K	
 <a href="#">_alloc.c</a>	29-Jun-00 08:16	4K	
 <a href="#">_alloc.h</a>	29-Jun-00 08:16	1K	

```
# -Aa is HPUX's way of invoking ANSI C
CFLAGS = -g -Aa

alloc.o:      alloc.o alloc.h
             cc $(CFLAGS) -c alloc.c -o alloc.o

clean:
             rm -rf alloc.o
```

```
/* alloc.c
 *
 * A simple fast memory allocation package.
 *
 * AllocInit()      - create an alloc pool, returns the old pool handle.
 * Alloc()          - allocate memory.
 * AllocReset()     - reset the current pool.
 * AllocSetPool()   - set the current pool.
 * AllocFree()      - free the memory used by the current pool.
 *
 */

#include <stdio.h>

#include "alloc.h"

/* ALLOC_BLOCK_SIZE - adjust this size to suit your installation - it should
 * be reasonably * large otherwise you will be mallocing a lot.
 */

#define ALLOC_BLOCK_SIZE      (100*1024)

/* alloc_hdr_t - Header for each block of memory
 */

typedef
struct alloc_hdr_s
{
    struct alloc_hdr_s *next;    /* Next Block          */
    char                *block,  /* Start of block      */
                        *free,   /* Next free in block  */
                        *end;    /* block + block size  */
}
alloc_hdr_t;

/* alloc_root_t - Header for the whole pool
 */

typedef
struct alloc_root_s
{
    alloc_hdr_t *first,    /* First header in pool */
                *current; /* Current header        */
}
alloc_root_t;

/* root - Pointer to the the current pool
 */

static alloc_root_t *root;

/* AllocHdr()
 *
 * Private routine to allocate a header and memory block.
 */

static
alloc_hdr_t *
AllocHdr()
{
    alloc_hdr_t    *hdr;
```



```
char          *block;

block = (char *) malloc(ALLOC_BLOCK_SIZE);
hdr    = (alloc_hdr_t *) malloc(sizeof(alloc_hdr_t));

if (hdr == NULL || block == NULL)
{
    fprintf(stderr, "Out of memory\n");
    exit(1);
}
hdr->block = block;
hdr->free  = block;
hdr->next  = NULL;
hdr->end   = block + ALLOC_BLOCK_SIZE;

return(hdr);
}
```

```
/* AllocInit()
 *
 * Create a new memory pool with one block.
 * Returns pointer to the previous pool.
 */
```

```
alloc_handle_t *
AllocInit()
{
    alloc_handle_t      *old = (alloc_handle_t *) root;

    root = (alloc_root_t *) malloc(sizeof(alloc_root_t));
    root->first = AllocHdr();
    root->current = root->first;
    return(old);
}
```

```
/* Alloc()
 *
 * Use as a direct replacement for malloc().  Allocates memory
 * from the current pool.
 */
```

```
char *
Alloc(size)
int    size;
{
    alloc_hdr_t      *hdr = root->current;
    char             *ptr;

    /* Align to 4 byte boundary - should be OK for most machines.
     * Change this if your machine has wierd alignment requirements
     */
    size = (size + 3) & 0xffffffffc;

    ptr = hdr->free;
    hdr->free += size;

    /* Check if the current block is exhausted */

    if (hdr->free >= hdr->end)
    {
        /* Is the next block already allocated? */
```

```
        if (hdr->next != NULL)
        {
            /* re-use block */
            hdr->next->free = hdr->next->block;
            root->current = hdr->next;
        }
        else
        {
            /* extend the pool with a new block */
            hdr->next = AllocHdr();
            root->current = hdr->next;
        }

        /* set ptr to the first location in the next block */
        ptr = root->current->free;
        root->current->free += size;
    }
    /* Return pointer to allocated memory */
    return(ptr);
}

/* AllocSetPool()
 *
 * Change the current pool.  Return the old pool.
 */

alloc_handle_t *
AllocSetPool(new)
alloc_handle_t      *new;
{
    alloc_handle_t *old = (alloc_handle_t *) root;

    root = (alloc_root_t *) new;
    return(old);
}

/* AllocReset()
 *
 * Reset the current pool for re-use.  No memory is freed, so
 * this is very fast.
 */

void
AllocReset()
{
    root->current = root->first;
    root->current->free = root->current->block;
}

/* AllocFreePool()
 *
 * Free the memory used by the current pool.
 * Don't use where AllocReset() could be used.
 */

void
AllocFreePool()
{
    alloc_hdr_t      *hdr = root->first;
```

```
while (hdr != NULL)
{
    free((char *) hdr->block);
    free((char *) hdr);
    hdr = hdr->next;
}
free((char *) root);
root = NULL;
```

```
}
```

```
/* alloc.h
 *
 * Header for alloc.c
 *
 * The type alloc_handle_t provides an opaque reference to the
 * alloc pool - only the alloc routines know its structure.
 */

typedef
struct { int dummy; }
alloc_handle_t;
```

```
/*
 * C code from the article
 * "Tri-linear Interpolation"
 * by Steve Hill, sah@ukc.ac.uk
 * in "Graphics Gems IV", Academic Press, 1994
 *
 * compile with "cc -DMAIN ..." to create a test program
 */
```

```
#include <math.h>
#include "GraphicsGems.h"
```

```
double
trilinear(p, d, xsize, ysize, zsize, def)
Point3 *p;
double *d;
int xsize, ysize, zsize;
double def;
{
#   define DENS(X, Y, Z) d[(X)+xsize*((Y)+ysize*(Z))]

    int      x0, y0, z0,
             x1, y1, z1;
    double    *dp,
             fx, fy, fz,
             d000, d001, d010, d011,
             d100, d101, d110, d111,
             dx00, dx01, dx10, dx11,
             dxy0, dxy1, dxyz;

    x0 = floor(p->x); fx = p->x - x0;
    y0 = floor(p->y); fy = p->y - y0;
    z0 = floor(p->z); fz = p->z - z0;

    x1 = x0 + 1;
    y1 = y0 + 1;
    z1 = z0 + 1;

    if (x0 >= 0 && x1 < xsize &&
        y0 >= 0 && y1 < ysize &&
        z0 >= 0 && z1 < zsize)
    {
        dp = &DENS(x0, y0, z0);
        d000 = dp[0];
        d100 = dp[1];
        dp += xsize;
        d010 = dp[0];
        d110 = dp[1];
        dp += xsize*ysize;
        d011 = dp[0];
        d111 = dp[1];
        dp -= xsize;
        d001 = dp[0];
        d101 = dp[1];
    }
    else
    {
#       define INRANGE(X, Y, Z) \
            ((X) >= 0 && (X) < xsize && \
             (Y) >= 0 && (Y) < ysize && \
             (Z) >= 0 && (Z) < zsize)
```

```
    d000 = INRANGE(x0, y0, z0) ? DENS(x0, y0, z0) : def;
    d001 = INRANGE(x0, y0, z1) ? DENS(x0, y0, z1) : def;
    d010 = INRANGE(x0, y1, z0) ? DENS(x0, y1, z0) : def;
    d011 = INRANGE(x0, y1, z1) ? DENS(x0, y1, z1) : def;

    d100 = INRANGE(x1, y0, z0) ? DENS(x1, y0, z0) : def;
    d101 = INRANGE(x1, y0, z1) ? DENS(x1, y0, z1) : def;
    d110 = INRANGE(x1, y1, z0) ? DENS(x1, y1, z0) : def;
    d111 = INRANGE(x1, y1, z1) ? DENS(x1, y1, z1) : def;
}

dx00 = LERP(fx, d000, d100);
dx01 = LERP(fx, d001, d101);
dx10 = LERP(fx, d010, d110);
dx11 = LERP(fx, d011, d111);

dxy0 = LERP(fy, dx00, dx10);
dxy1 = LERP(fy, dx01, dx11);

dxyz = LERP(fz, dxy0, dxy1);

return dxyz;
}

#ifdef MAIN

/* test program for trilinear interpolation */

#include <stdio.h>

#define XSIZE    2
#define YSIZE    2
#define ZSIZE    2

main()
{
#define TDENS(X,Y,Z) d[(X)+XSIZE*((Y)+YSIZE*(Z))]

    extern double trilinear();
    double  *d, def;
    int      x, y, z;
    Point3   p;

    d = (double *) malloc(sizeof(double) * XSIZE * YSIZE * ZSIZE);

    printf("Test for trilinear interpolation\n");
    printf("Enter the densities for the corners of a cube\n");

    for (x = 0; x < XSIZE; x++)
    for (y = 0; y < YSIZE; y++)
    for (z = 0; z < ZSIZE; z++)
    {
        printf("Point (%d, %d, %d) is: ", x, y, z);
        scanf("%lf", &TDENS(x, y, z));
    }

    printf("Enter the default density: ");
    scanf("%lf", &def);

    printf("Enter point of interest: ");
```

```
while (scanf("%lf %lf %lf", &p.x, &p.y, &p.z) == 3)
{
    double  n;

    n = trilinear(&p, d, XSIZE, YSIZE, ZSIZE, def);
    printf("Density at (%lf, %lf, %lf) is %lf\n", p.x, p.y, p.z, n);
    printf("Enter point of interest: ");
}
```

#endif

```

/*****
*   The following macros handle most vector operations, the exceptions
*   usually being complex equations with four or more vectors.
*
*   An alternate form for the multiple-statement macros is the
*   "if (1) <macro_body> else" form. This allows for temporary variable
*   declaration and control-flow constructs, but cannot be used
*   everywhere a function call could, as with the form used below.
*
*   Note that since the vector arguments are not enclosed in parentheses
*   in the macro body, you can scale the vector arguments in the macro
*   calls, e.g. Vec2Op(vec1,=,scalar*vec2).
*
*   Here are some example uses of the following macros:
*
*       printf ("Vector = <%lg %lg %lg>\n", VecList(vector))
*       vector_dot = VecDot (vec1, vec2)
*       norm = VecNorm (vector)
*       VecScalar (vector, /=, norm)
*       VecScalar (vector, *=, scale)
*       Vec3Scalar (Xaxis, =, 1.0, 0.0, 0.0)
*       Vec3Scalar (vector, *=, Xshear, Yshear, Zshear)
*       Vec2Op (vector, =, Xaxis)
*       Vec2Op (vector, +=, norm * Xaxis)
*       Vec3Op (vec1, =, vec2, =, Xaxis)
*       Vec3Op (vec1, =, vec2, -, vec3)
*       Vec3Op (vec1, +=, scale2 * vec2, -, scale3 * vec3)
*       VecCross (vec1, -=, vec2, X, vec3)
*       VecCrossSafe (vec1, =, vec1, X, Xaxis)
*****/

#include <math.h>          /* Needed for sqrt() definition. */

/* Vector type definition.  If you define colors in the same manner,
** you can also use these macros for color vector operations.  */

typedef long float   Vector[3];
typedef Vector       Point;          /* For readability. */

/* VecList enumerates the vector fields for function calls. */

#define VecList(V)      V[0], V[1], V[2]

/* This macro computes the dot product of two vectors. */

#define VecDot(A,B)     ((A[0]*B[0]) + (A[1]*B[1]) + (A[2]*B[2]))

/* The VecNorm macro computes the norm of the vector. */

#define VecNorm(V)      sqrt(VecDot(V,V))

/* VecScalar provides for scalar operations on a vector. */

#define VecScalar(V,assign_op,k)    \
(   V[0] assign_op k,    \
    V[1] assign_op k,    \
    V[2] assign_op k     \
)

/* Vec3Scalar provides for vector operations that involve three
** distinct scalar factors. */
```



```
#define Vec3Scalar(V,assign_op,a,b,c)    \
(    V[0] assign_op a,    \
    V[1] assign_op b,    \
    V[2] assign_op c    \
)

/* Vec2Op provides for operations with two vectors. */

#define Vec2Op(A,assign_op,B)    \
(    A[0] assign_op B[0],    \
    A[1] assign_op B[1],    \
    A[2] assign_op B[2]    \
)

/* Vec3Op handles vector operations with three vectors. */

#define Vec3Op(A,assign_op,B,op,C)    \
(    A[0] assign_op B[0] op C[0],    \
    A[1] assign_op B[1] op C[1],    \
    A[2] assign_op B[2] op C[2]    \
)

/* The cross product macros come in two flavors. VecCross() requires
** that all three vectors are distinct. With the VecCrossSafe()
** macro, it's OK to do A <- A X B, but this requires a temporary
** vector for storage, which in turn requires the "if (1) ... else"
** form. As an alternative, a global temporary vector could be used.
**/

#define VecCross(A,assign_op,B,dummy_op,C)    \
(    A[0] assign_op (B[1] * C[2]) - (B[2] * C[1]),    \
    A[1] assign_op (B[2] * C[0]) - (B[0] * C[2]),    \
    A[2] assign_op (B[0] * C[1]) - (B[1] * C[0])    \
)

#define VecCrossSafe(A,assign_op,B,dummy_op,C)    \
if (1)    \
{    auto Vector result;    \
    VecCross (result,=,B,X,C);    \
    Vec2Op (A,=,result);    \
} else
```

/\*\*\*\*\*\*  
The following function displays an image using progressive refinement  
via gridded sampling. It assumes the existence of two functions:

```
SetColor (x, y)
    Sets the current color to the color at image location x,y.

Rectangle (x, y, width, height)
    Draws a rectangular region filled with the current color. The
    rectangle ranges from x to (x+width-1), and from y to
    (y+height-1).
```

The parameters given are the subdivision start level (a power of two),  
and the X and Y dimensions of the entire image.

\*\*\*\*\*/

```
void PIR_Display (start_level, Xdim, Ydim)
    int  start_level;      /* Starting Subdivision Level */
    int  Xdim, Ydim;       /* Image Dimensions */
{
    auto int size, size2;  /* Current Region Size & size/2 */
    auto int Ix, Iy;       /* Image Space Indices */

    /* Initialization loop: display the initial coarse image tiling. */

    size = 1 << start_level;

    for (Iy=0; Iy <= Ydim; Iy += size)
    {
        for (Ix=0; Ix <= Xdim; Ix += size)
        {
            SetColor (Ix, Iy);
            Rectangle (Ix, Iy, size, size);
        }
    }

    /* Sampling and Gridding Loop */

    size2 = size / 2;

    while (size > 1)        /* Subdivide down to the pixel level. */
    {
        for (Iy=0; Iy <= Ydim; Iy += size)
        {
            for (Ix=0; Ix <= Xdim; Ix += size)
            {
                /* Draw the three new subpixel regions. */

                SetColor (Ix, Iy + size2);
                Rectangle (Ix, Iy + size2, size2, size2);

                SetColor (Ix + size2, Iy + size2);
                Rectangle (Ix + size2, Iy + size2, size2, size2);

                SetColor (Ix + size2, Iy);
                Rectangle (Ix + size2, Iy, size2, size2);
            }
        }







        /* The new region edge length is half the old edge length. */

        size = size2;
        size2 = size2 / 2;
    }
}
```

```
}  
}
```

# Index of

## /pubs/tog/GraphicsGems/gems/Sturm/

Name	Last modified	Size	Description
 <a href="#">_Parent Directory</a>			
 <a href="#">_Makefile</a>	29-Jun-00 08:12	1K	
 <a href="#">_main.c</a>	29-Jun-00 08:12	2K	
 <a href="#">_solve.h</a>	29-Jun-00 08:12	1K	
 <a href="#">_sturm.c</a>	29-Jun-00 08:12	5K	
 <a href="#">_util.c</a>	29-Jun-00 08:12	1K	

```
#  
# Makefile  
#  
#      command file for make to compile the solver.  
  
solve: main.o sturm.o util.o  
      cc -o solve main.o sturm.o util.o -lm  
  
clean:  
      /bin/rm -f main.o sturm.o util.o solve  
  
main.o sturm.o util.o: solve.h
```

```
/*
Using Sturm Sequences to Bracket Real Roots of Polynomial Equations
by D.G. Hook and P.R. McAree
from "Graphics Gems", Academic Press, 1990
*/

/*
 * main.c
 *
 *      a sample driver program.
 */
#include <stdio.h>
#include <math.h>
#include "solve.h"

/*
 * a driver program for a root solver.
 */
main()
{
    poly    sseq[MAX_ORDER];
    double  min, max, roots[MAX_ORDER];
    int     i, j, order, nroots, nchanges, np, atmin, atmax;

    /*
     * get the details...
     */

    printf("Please enter order of polynomial: ");
    scanf("%d", &order);

    printf("\n");

    for (i = order; i >= 0; i--) {
        printf("Please enter coefficient number %d: ", i);
        scanf("%lf", &sseq[0].coef[i]);
    }

    printf("\n");

    /*
     * build the Sturm sequence
     */
    np = buildsturm(order, sseq);

    printf("Sturm sequence for:\n");

    for (i = order; i >= 0; i--)
        printf("%lf ", sseq[0].coef[i]);

    printf("\n\n");

    for (i = 0; i <= np; i++) {
        for (j = sseq[i].ord; j >= 0; j--)
            printf("%lf ", sseq[i].coef[j]);
        printf("\n");
    }

    printf("\n");
}
```

```
/*
 * get the number of real roots
 */
nroots = numroots(np, sseq, &atmin, &atmax);

if (nroots == 0) {
    printf("solve: no real roots\n");
    exit(0);
}

/*
 * calculate the bracket that the roots live in
 */
min = -1.0;
nchanges = numchanges(np, sseq, min);
for (i = 0; nchanges != atmin && i != MAXPOW; i++) {
    min *= 10.0;
    nchanges = numchanges(np, sseq, min);
}

if (nchanges != atmin) {
    printf("solve: unable to bracket all negative roots\n");
    atmin = nchanges;
}

max = 1.0;
nchanges = numchanges(np, sseq, max);
for (i = 0; nchanges != atmax && i != MAXPOW; i++) {
    max *= 10.0;
    nchanges = numchanges(np, sseq, max);
}

if (nchanges != atmax) {
    printf("solve: unable to bracket all positive roots\n");
    atmax = nchanges;
}

nroots = atmin - atmax;

/*
 * perform the bisection.
 */
sbisect(np, sseq, min, max, atmin, atmax, roots);

/*
 * write out the roots...
 */
if (nroots == 1) {
    printf("\n1 distinct real root at x = %f\n", roots[0]);
} else {
    printf("\n%d distinct real roots for x: ", nroots);

    for (i = 0; i != nroots; i++)
        printf("%f ", roots[i]);
    printf("\n");
}
}
```

```
/*
 * solve.h
 *
 *      some useful constants and types.
 */
#define      MAX_ORDER      12
/* maximum order for a polynomial */

#define      RELERROR      1.0e-14
/* smallest relative error we want */

#define      MAXPOW      32
/* max power of 10 we wish to search to */

#define      MAXIT      800
/* max number of iterations */

/* a coefficient smaller than SMALL_ENOUGH is considered to
   be zero (0.0). */

#define      SMALL_ENOUGH      1.0e-12

/*
 * structure type for representing a polynomial
 */
typedef      struct p {
                int      ord;
                double    coef[MAX_ORDER];
        } poly;

extern int      modrf();
extern int      numroots();
extern int      numchanges();
extern int      buildsturm();

extern double   evalpoly();
```



```
/*
 * sturm.c
 *
 * the functions to build and evaluate the Sturm sequence
 */
#include <math.h>
#include <stdio.h>
#include "solve.h"

/*
 * modp
 *
 * calculates the modulus of u(x) / v(x) leaving it in r, it
 * returns 0 if r(x) is a constant.
 * note: this function assumes the leading coefficient of v
 * is 1 or -1
 */
static int
modp(u, v, r)
    poly *u, *v, *r;
{
    int k, j;
    double *nr, *end, *uc;

    nr = r->coef;
    end = &u->coef[u->ord];

    uc = u->coef;
    while (uc <= end)
        *nr++ = *uc++;

    if (v->coef[v->ord] < 0.0) {

        for (k = u->ord - v->ord - 1; k >= 0; k -= 2)
            r->coef[k] = -r->coef[k];

        for (k = u->ord - v->ord; k >= 0; k--)
            for (j = v->ord + k - 1; j >= k; j--)
                r->coef[j] = -r->coef[j] - r->coef[v->ord + k]
                    * v->coef[j - k];
    } else {
        for (k = u->ord - v->ord; k >= 0; k--)
            for (j = v->ord + k - 1; j >= k; j--)
                r->coef[j] -= r->coef[v->ord + k] * v->coef[j - k];
    }

    k = v->ord - 1;
    while (k >= 0 && fabs(r->coef[k]) < SMALL_ENOUGH) {
        r->coef[k] = 0.0;
        k--;
    }

    r->ord = (k < 0) ? 0 : k;

    return(r->ord);
}

/*
 * buildsturm
```

```
*
*      build up a sturm sequence for a polynomial in smat, returning
*      the number of polynomials in the sequence
*/
int
buildsturm(ord, sseq)
    int      ord;
    poly     *sseq;
{
    int      i;
    double   f, *fp, *fc;
    poly     *sp;

    sseq[0].ord = ord;
    sseq[1].ord = ord - 1;

    /*
     * calculate the derivative and normalise the leading
     * coefficient.
     */
    f = fabs(sseq[0].coef[ord] * ord);
    fp = sseq[1].coef;
    fc = sseq[0].coef + 1;
    for (i = 1; i <= ord; i++)
        *fp++ = *fc++ * i / f;

    /*
     * construct the rest of the Sturm sequence
     */
    for (sp = sseq + 2; modp(sp - 2, sp - 1, sp); sp++) {

        /*
         * reverse the sign and normalise
         */
        f = -fabs(sp->coef[sp->ord]);
        for (fp = &sp->coef[sp->ord]; fp >= sp->coef; fp--)
            *fp /= f;
    }

    sp->coef[0] = -sp->coef[0];      /* reverse the sign */

    return(sp - sseq);
}

/*
 * numroots
 *
 *      return the number of distinct real roots of the polynomial
 *      described in sseq.
 */
int
numroots(np, sseq, atneg, atpos)
    int      np;
    poly     *sseq;
    int      *atneg, *atpos;
{
    int      atposinf, atneginf;
    poly     *s;
    double   f, lf;

```

```
    atposinf = atneginf = 0;
```

```
/*
 * changes at positive infinity
 */
lf = sseq[0].coef[sseq[0].ord];

for (s = sseq + 1; s <= sseq + np; s++) {
    f = s->coef[s->ord];
    if (lf == 0.0 || lf * f < 0)
        atposinf++;
    lf = f;
}

/*
 * changes at negative infinity
 */
if (sseq[0].ord & 1)
    lf = -sseq[0].coef[sseq[0].ord];
else
    lf = sseq[0].coef[sseq[0].ord];

for (s = sseq + 1; s <= sseq + np; s++) {
    if (s->ord & 1)
        f = -s->coef[s->ord];
    else
        f = s->coef[s->ord];
    if (lf == 0.0 || lf * f < 0)
        atneginf++;
    lf = f;
}

*atneg = atneginf;
*atpos = atposinf;

return(atneginf - atposinf);
}
```

```
/*
 * numchanges
 *
 * return the number of sign changes in the Sturm sequence in
 * sseq at the value a.
 */
int
numchanges(np, sseq, a)
    int      np;
    poly     *sseq;
    double   a;
{
    int      changes;
    double   f, lf;
    poly     *s;

    changes = 0;

    lf = evalpoly(sseq[0].ord, sseq[0].coef, a);

    for (s = sseq + 1; s <= sseq + np; s++) {
```

```
        f = evalpoly(s->ord, s->coef, a);
        if (lf == 0.0 || lf * f < 0)
            changes++;
        lf = f;
    }

    return(changes);
}

/*
 * sbisect
 *
 *      uses a bisection based on the sturm sequence for the polynomial
 *      described in sseq to isolate intervals in which roots occur,
 *      the roots are returned in the roots array in order of magnitude.
 */
sbisect(np, sseq, min, max, atmin, atmax, roots)
    int      np;
    poly     *sseq;
    double   min, max;
    int      atmin, atmax;
    double   *roots;
{
    double   mid;
    int      n1 = 0, n2 = 0, its, atmid, nroot;

    if ((nroot = atmin - atmax) == 1) {

        /*
         * first try a less expensive technique.
         */
        if (modrf(sseq->ord, sseq->coef, min, max, &roots[0]))
            return;

        /*
         * if we get here we have to evaluate the root the hard
         * way by using the Sturm sequence.
         */
        for (its = 0; its < MAXIT; its++) {
            mid = (min + max) / 2;

            atmid = numchanges(np, sseq, mid);

            if (fabs(mid) > RELERROR) {
                if (fabs((max - min) / mid) < RELERROR) {
                    roots[0] = mid;
                    return;
                }
            } else if (fabs(max - min) < RELERROR) {
                roots[0] = mid;
                return;
            }

            if ((atmin - atmid) == 0)
                min = mid;
            else
                max = mid;
        }

        if (its == MAXIT) {
```

```
        fprintf(stderr, "sbisect: overflow min %f max %f\
        diff %e nroot %d n1 %d n2 %d\n",
        min, max, max - min, nroot, n1, n2);
    roots[0] = mid;
}

return;
}

/*
 * more than one root in the interval, we have to bisect...
 */
for (its = 0; its < MAXIT; its++) {

    mid = (min + max) / 2;

    atmid = numchanges(np, sseq, mid);

    n1 = atmin - atmid;
    n2 = atmid - atmax;

    if (n1 != 0 && n2 != 0) {
        sbisect(np, sseq, min, mid, atmin, atmid, roots);
        sbisect(np, sseq, mid, max, atmid, atmax, &roots[n1]);
        break;
    }

    if (n1 == 0)
        min = mid;
    else
        max = mid;
}

if (its == MAXIT) {
    fprintf(stderr, "sbisect: roots too close together\n");
    fprintf(stderr, "sbisect: overflow min %f max %f diff %e\
    nroot %d n1 %d n2 %d\n",
    min, max, max - min, nroot, n1, n2);
    for (n1 = atmax; n1 < atmin; n1++)
        roots[n1 - atmax] = mid;
}
}
```

```
/*
 * util.c
 *
 *      some utility functions for root polishing and evaluating
 *      polynomials.
 */
#include <math.h>
#include <stdio.h>
#include "solve.h"

/*
 * evalpoly
 *
 *      evaluate polynomial defined in coef returning its value.
 */
double
evalpoly (ord, coef, x)
    int      ord;
    double   *coef, x;
{
    double   *fp, f;

    fp = &coef[ord];
    f = *fp;

    for (fp--; fp >= coef; fp--)
        f = x * f + *fp;

    return(f);
}

/*
 * modrf
 *
 *      uses the modified regula-falsi method to evaluate the root
 *      in interval [a,b] of the polynomial described in coef. The
 *      root is returned in *val. The routine returns zero
 *      if it can't converge.
 */
int
modrf(ord, coef, a, b, val)
    int      ord;
    double   *coef;
    double   a, b, *val;
{
    int      its;
    double   fa, fb, x, fx, lfx;
    double   *fp, *scoef, *ecof;

    scoef = coef;
    ecof = &coef[ord];

    fb = fa = *ecof;
    for (fp = ecof - 1; fp >= scoef; fp--) {
        fa = a * fa + *fp;
        fb = b * fb + *fp;
    }

    /*
```

```
* if there is no sign difference the method won't work
*/
if (fa * fb > 0.0)
    return(0);

if (fabs(fa) < RELError) {
    *val = a;
    return(1);
}

if (fabs(fb) < RELError) {
    *val = b;
    return(1);
}

lfx = fa;

for (its = 0; its < MAXIT; its++) {

    x = (fb * a - fa * b) / (fb - fa);

    fx = *ecoeff;
    for (fp = ecoef - 1; fp >= scoef; fp--)
        fx = x * fx + *fp;

    if (fabs(x) > RELError) {
        if (fabs(fx / x) < RELError) {
            *val = x;
            return(1);
        }
    } else if (fabs(fx) < RELError) {
        *val = x;
        return(1);
    }

    if ((fa * fx) < 0) {
        b = x;
        fb = fx;
        if ((lfx * fx) > 0)
            fa /= 2;
    } else {
        a = x;
        fa = fx;
        if ((lfx * fx) > 0)
            fb /= 2;
    }



    lfx = fx;
}

fprintf(stderr, "modrf overflow %f %f %f\n", a, b, fx);

return(0);
}
```

# Index of

## /pubs/tog/GraphicsGems/gemsv/ch6-2/

Name	Last modified	Size	Description
 <a href="#">_Parent Directory</a>			
 <a href="#">_halfadap.c</a>	29-Jun-00 08:24	6K	



```
/*=====*
 * Halftoning using Space Filling Curve with adaptive clustering and *
 * selective precipitation *
 * *
 * Limitation: *
 * Only process image with size 2^n x 2^n where n is positive integer. *
 *=====*/
#include <stdio.h>
#include <stdlib.h>

unsigned char **path;      /* space filling curve path */
/*
 * path[] is a global array storing the information to move along
 * the space filling curve.
 * genspacefill() is a function to generate the information in path[].
 * This function is implemented based on a gem in Graphics Gems II,
 * Ken Musgrave, "A Peano Curve Generation Algorithm".
 * move() is a macro to move along the space filling curve using the
 * the information stored in path[].
 */
extern genspacefill();

#define TRUE      1
#define FALSE    0
#define BLACK    255
#define WHITE     0
#define LEFT     0
#define RIGHT    1
#define UP       2
#define DOWN     3
#define END      255
#define move(x,y) switch (path[x][y])          \
{                                                \
    case UP:    y++; break;                    \
    case DOWN:  y--; break;                    \
    case LEFT:  x--; break;                    \
    case RIGHT: x++; break;                    \
}

/*
 * Description of parameters:
 * picture,      2D array holding the grayscale image.
 * out,          2D array holding the dithered image.
 * maxclustersize, Max cluster size, N.
 * thresh,       Edge detection threshold T.
 * do_sp,        Flag to switch on/off selective precipitation.
 *              To switch off the selective precipitation,
 *              set do_sp = FALSE.
 * do_ac,        Flag to switch on/off adaptive clustering.
 *              To switch off the adaptive clustering, set do_ac=FALSE
 */
void spacefilterwindow(int **picture, int **out, int maxclustersize,
                      int thresh, char do_sp, char do_ac)
{
    char edge;          /* Flag indicate sudden change detected */
    char ending;        /* flag indicates end of space filling curve */
    int accumulator;    /* Accumulate gray value */
    int currclustersize; /* Record size of current cluster */
    int frontx, fronty; /* Pointer to the front of the cluster */
    int windowx, windowy; /* Pointer to first pixel applied with filter */
    int clusterx, clustery; /* Pointer to first pixel in current cluster */
}
```

```

if ((cluster=(int*)malloc(sizeof(int)*maxclustersize))==NULL)
{
    fprintf(stderr,"not enough memory for cluster\n");
    return;
}
genspacefill();      /* generates the spacefilling path */

convolution=0;
currclustersize=0;
accumulator=0;
for (frontx=0, fronty=0, i=0 ; i<7 ; i++)
{
    if (i<3)
    {
        cluster[currclustersize] = picture[frontx][fronty];
        accumulator += cluster[currclustersize];
        currclustersize++;
    }
    if (i==3)
    {   currx = frontx;   curry = fronty;   }
    convolution += filter[i]*(long)(picture[frontx][fronty]);
    move(frontx,fronty); /* assume the image at least has 7 pixels */
}
lastconvolution = convolution;
clusterx=0;   clustery=0;
windowx=0;   windowy=0;
edge=FALSE;
ending=FALSE;

while (TRUE)
{
    if (do_ac) /* switch on/off adaptive clustering */
    {

```

```
/* do convolution */
convolution = 0;
for (tempx=windowx, tempy=windowy, i=0 ; i<7 ; i++)
{
    convolution += filter[i]*picture[tempx][tempy];
    move(tempx,tempy);
}

/* detect sudden change */
if ( (convolution >= 0 && lastconvolution <=0
      && abs(convolution-lastconvolution)>thresh)
    ||(convolution <= 0 && lastconvolution >=0
      && abs(convolution-lastconvolution)>thresh))
    edge=TRUE; /* force output dots */
}

/* Output dots if necessary */
if (edge || currclustersize >= maxclustersize || ending)
{
    edge=FALSE;

    /* Search the best position within cluster to precipitate */
    rightplace = 0;
    if (do_sp) /* switch on/off selective precipitation */
    {
        windowlen = accumulator/BLACK;
        winsum = 0;
        for (i=0; i<windowlen; i++)
            winsum += cluster[i];
        for (maxsum=winsum, last=0; i<currclustersize; i++, last++)
        {
            winsum+= cluster[i] - cluster[last];
            if (winsum > maxsum)
            {
                rightplace=last+1;
                maxsum=winsum;
            }
        }
    }

    /* Output dots */
    for (i=0 ; currclustersize!=0 ; currclustersize--, i++)
    {
        if (accumulator>=BLACK && i>=rightplace) /* precipitates */
        {
            out[clusterx][clustery]=BLACK;
            accumulator-=BLACK;
        }
        else
            out[clusterx][clustery]=WHITE;
        move(clusterx,clustery)
    } /* for */




    if (ending)
        break;
} /* if */

cluster[currclustersize] = picture[currx][curry];
accumulator += cluster[currclustersize];
currclustersize++;
if (path[currx][curry]==END)
```

```
        ending = TRUE;
    move(currx,curry);
    move(windowx>windowy);
    move(frontx,fronty);
} /* while */
}
```

# Index of

## /pubs/tog/GraphicsGems/gemsv/ch6-5/

Name	Last modified	Size	Description
 <a href="#">_Parent Directory</a>			
 <a href="#">_revfit.c</a>	29-Jun-00 08:24	23K	
 <a href="#">_revfit.h</a>	29-Jun-00 08:24	1K	

```
/*
 * revfit.c : edge reconstruction and the inverse process.
 */
#include <stdio.h>
#include "revfit.h"

#ifndef abs
#define abs(a) ((a)>=0 ? (a) : -(a))
#endif

#define HalfSUBPIXRES (SUBPIXRES/2)
#define ESTABLISHED 127
#define MAXRUN 2000 /* max no of pixel edges in a line */

extern DrawPixelEdge(int x, int y, int V_H); /* a user supplied function */
/* for drawing a PixelEdge */

/*****\
 * typedef's for sub-pixel resolution pixel edges and gradient bounds *
 \*****/
typedef struct {
    int x1,y1; /* from (coordinates multiplied by sub-pixel resolution) */
    int x2,y2; /* to (coordinates multiplied by sub-pixel resolution) */
} Pedge;

typedef struct {
    int ly,lx; /* lower limit */
    int uy,ux; /* upper limit */
} Bound;

#define MidX(e) (((e).x1+(e).x2)/2) /* midpt coordinates of a Pedge */
#define MidY(e) (((e).y1+(e).y2)/2)
#define Is_Horizontal(d) (abs(d)==HRZ) /* a horizontal direction? (1, -1) */
#define Is_Vertical(d) (abs(d)==VRT) /* a vertical direction? (2, -2) */
#define against(a,b) (!(a)+(b)) /* whether two directions are opp. */
#define Bound_OK(b) (slopecmp((b).uy,(b).ux,(b).ly,(b).lx))
#define WithinBound(dy,dx,b) (slopecmp((dy),(dx),(b).ly,(b).lx) &&\
    slopecmp((b).uy,(b).ux,(dy),(dx)))

/*****\
 * Get_Pedge(): Returns a pointer to the current Pedge from the list el. *
 * The position of the cursor of list is not modified. *
 * Returns NULL if no more edges in the list. *
 * Coordinates multiplied by sub-pixel resolution. *
 \*****/
static Pedge *Get_Pedge(Edgelist el) {
    static Pedge e;
    int dir;
    if (el.current>=el.Nedges) return NULL;
    if (Is_Horizontal(dir=(el.list[el.current].dir))) {
        e.y1=e.y2=el.list[el.current].y*SUBPIXRES + HalfSUBPIXRES;
        e.x1=el.list[el.current].x*SUBPIXRES
            - (dir>0 ? HalfSUBPIXRES : -HalfSUBPIXRES);
        e.x2=e.x1 + (dir>0 ? SUBPIXRES : -SUBPIXRES);
    }
    else {
        e.x1=e.x2=el.list[el.current].x*SUBPIXRES + HalfSUBPIXRES;
        e.y1=el.list[el.current].y*SUBPIXRES
            - (dir>0 ? HalfSUBPIXRES : -HalfSUBPIXRES);
        e.y2=e.y1 + (dir>0 ? SUBPIXRES : -SUBPIXRES);
    }
}
```

```
return &e;
} /* Get_Pedge() */

/*****
 *      forward(): Update the cursor of the list to the next edge.      *
 *****/
#define forward(el) (((el).current)++)

/*****
 *      backward(): Move back the cursor of the list one place so that  *
 *                  the previous edge can be visited again.            *
 *****/
#define backward(el) (((el).current)--)

/*****\
 *      wayof(): return a direction.                                     *
 \*****/
/* the directions no.s are chosen s.t. d1== -d2 if d1,d2 are opp. */
static int wayof(Pedge e) {
    int d=e.x2-e.x1;
    return d ? d/SUBPIXRES /* 1 or -1 for horizontal edge */
            : (e.y2 - e.y1)/HalfSUBPIXRES; /* 2 or -2 for vertical edge */
} /* wayof() */

/*****
 * slopecmp(): True if grad vector of the 1st is on the counter-clockwise *
 *              side of the 2nd one                                         *
 *****/
static int slopecmp(int dy1,int dx1, int dy2,int dx2) {
    return (long)dx2*dy1 > (long)dx1*dy2;
} /* slopecmp() */

/*****\
 * calcbound(): calc the bounds (the pair of gradient limits) for the Pedge *
 \*****/
void calcbound(int dominantdir, Pedge e, int Sx, int Sy,
               Bound* b, IntPoint2 *gradU, IntPoint2 *gradL) {
/* gradU and gradL shall be filled with the gradients just within the limits */
    int dy,dx;
    if (Is_Horizontal(dominantdir)) { /* horizontal dominant direction */
        b->uy = (e.y1+e.y2+SUBPIXRES)/2-Sy;
        b->ux = (e.x1+e.x2)/2 -Sx;
        b->ly = (e.y1+e.y2-SUBPIXRES)/2-Sy;
        gradU->x = gradL->x = b->lx = b->ux;
        gradU->y = b->uy-1; gradL->y = b->ly+1;
    }
    else { /* up or down dominant direction */
        b->uy = (e.y1+e.y2)/2 -Sy;
        b->ux = (e.x1+e.x2+SUBPIXRES)/2-Sx;
        gradU->y = gradL->y = b->ly = b->uy;
        b->lx = (e.x1+e.x2-SUBPIXRES)/2-Sx;
        gradU->x = b->ux-1; gradL->x = b->lx+1;
    }
    if (!Bound_OK(*b)) { /* swaps the bounds if necessary */
        IntPoint2 p;
        dx=b->ux;    dy=b->uy;
        b->ux=b->lx; b->uy=b->ly;
        b->lx=dx;    b->ly=dy;
        p=*gradU; *gradU=*gradL; *gradL=p;
    }
} /* calcbound() */
```

```

/*****
 * fitlines() : The reversible straight line edge reconstruction routine
 *****/
int fitlines(Edgelist el, boolean Pretest, boolean TryAllEndPts,
             IntPoint2 *lines, int MaxNLine) {
/*-----*/
 * el          : The supplied list of PixelEdges.
 * Pretest     : 1=perform pre-test on each pixel edge, i.e., stop as soon as
 *               a valid end pt cannot be found on a pixel edge.
 *               0=Allows stepping back.
 * TryAllEndPts: 1=Try all possible end-pts, 0=Use the one closest to mid-pt.
 * lines[]     : A preallocated array to be filled with end pts of fitted lines
 *               Note: Coordinates of the end pts are multiplied by SUBPIXRES.
 * MaxNLine    : The size of the lines[] array.
 *-----*/
int i,linescount,startpt,Nendpt,Nstartpt,NPedges,Nbound;          /* counters */
int Sx,Sy,Ex,Ey,  Ux,Uy,Lx,Ly,  maindir,trnsvrse,dnow,  ndir,dir[3];
flag breaktrace, starttrace;                                     /* flags */
int currentsave, bestpt, maxlen, bestpt_currentsave, bestpt_Nendpt;
IntPoint2 startpts[SUBPIXRES],endlist[SUBPIXRES],bestpt_endlist[SUBPIXRES];
Pedge Pedgehistory[MAXRUN],e,last,*nextp,estartsave,bestpt_last;
Bound bound[MAXRUN];

el.current=0;                                                    /* set cursor to the first edge */
e = *Get_Pedge(el);                                              /* first edge */
Sx = MidX(e);
Sy = MidY(e);

if (!TryAllEndPts) {
    lines[0].x = Sx;                                              /* record the 1st starting pt. */
    lines[0].y = Sy;
    linescount=1;
}
else {
    flag hori = Is_Horizontal(wayof(e));
    Nstartpt=0;
    startpts[0].x = Sx;
    startpts[0].y = Sy;
    for (i=1;i<HalfSUBPIXRES;i++) { /* the list of possible init. starting pts */
        startpts[Nstartpt ].x = hori ? Sx-i : Sx;
        startpts[Nstartpt++].y = !hori ? Sy+i : Sy;
        startpts[Nstartpt ].x = hori ? Sx-i : Sx;
        startpts[Nstartpt++].y = !hori ? Sy+i : Sy;
    }
    startpt=0; /* counter for the list of possible starting pts (startpts[]) */
    bestpt_currentsave=currentsave=el.current; /* save these for rewinding */
    estartsave=e;
    maxlen=bestpt=-1;                                           /* no best starting pt (bestpt) yet */
    linescount=0;
} /* if (!TryAllEndPts) .. else .. */

for (starttrace=TRUE;;) { /* loop for all PixelEdges */
    if (starttrace) { /* beginning of a new line segment */
        dir[0]=wayof(e);  ndir=1; /* no.of distinct directions so far */
        starttrace=0; breaktrace=0;
        Pedgehistory[0]=e; /* the first Pedge traced */
        NPedges=1; /* reset the counters */
        Nbound=0;
    } /* if (starttrace) */

```



```

last=e;
forward(el);
if ((nextp=Get_Pedge(el))!=NULL) { /* go on to the next PixelEdge */
    Pedgehistory[NPedges++]=*nextp; /* get a new Pedge */
    e=*nextp;
    dnow=wayof(e); /* direction of the current edge */
}

if (nextp==NULL || ndir==ESTABLISHED){ /* maindir and trnsvrse established */
    Bound b;
    IntPoint2 gradU,gradL;
    flag lowerupdated, upperupdated;

    if (nextp!=NULL) {
        calcbound(maindir,e,Sx,Sy,&b,&gradU,&gradL);

        bound[Nbound]=bound[Nbound-1];

        lowerupdated=upperupdated=FALSE;
        if (slopecmp(bound[Nbound-1].uy,bound[Nbound-1].ux,
                    b.uy,b.ux)) { /* update the upper limit */
            bound[Nbound].uy=b.uy;
            bound[Nbound].ux=b.ux;
            upperupdated=TRUE;
        }
        if (slopecmp(b.ly,b.lx,
                    bound[Nbound-1].ly,
                    bound[Nbound-1].lx)) { /* update the lower limit */
            bound[Nbound].ly=b.ly;
            bound[Nbound].lx=b.lx;
            lowerupdated=TRUE;
        }
    }
} /* if (nextp!=NULL) */

if (nextp==NULL || /* no more PixelEdge */
    (dnow!=trnsvrse && dnow!=maindir) || /* U-turn */
    (dnow==trnsvrse && dnow==wayof(last)) || /* 2 trnsvrse edges */
    !Bound_OK(bound[Nbound]) || /* not within limits */
    (Pretest && /* if Pretest, check if there is any pt within limits */
     ((lowerupdated && !WithinBound(gradU.y,gradU.x,bound[Nbound])) ||
      (upperupdated && !WithinBound(gradL.y,gradL.x,bound[Nbound])))) {
    /* now we shall calculate the starting pt for the next trace */
    for (;;) { /* loop until the end-point lies within the gradient limits */
        int dx,dy,tmp; /* flag exact,EndptOK;

        Ex=MidX(last); Ey=MidY(last);
        if (Nbound==0) { /* i.e. first few PixelEdges. therefore mid-pt is ok */
            if (TryAllEndPts){
                endlst[0].x=Ex; endlst[0].y=Ey;
                Nendpt=1;
            }
            break; /* end pt found */
        }

        b = bound[Nbound-1];

        dx= Ex - Sx; /* the slope of the mid-pt of the last Pedge */
        dy= Ey - Sy;

        if (TryAllEndPts && el.current-currentsave>maxlen) {
            /* find all possible end pts only if length longer than maxlen so far */

```

```
int h, addy, addx;

if (abs(maindir)==1) { addy=1; addx=0; } else { addy=0; addx=1; }
if (WithinBound(dy,dx,b)) { /* check mid-pt first */
    endlist[0].x=Ex; endlist[0].y=Ey; Nendpt=1;
}
else Nendpt=0;
for (h=1; h<SUBPIXRES/2; h++) { /* offset from mid-pt */
    if (WithinBound(dy+addy*h,dx+addx*h,b)) {
        endlist[Nendpt].x = Ex + addx*h;
        endlist[Nendpt++].y = Ey + addy*h;
    }
    else if (WithinBound(dy-addy*h,dx-addx*h,b)) {
        endlist[Nendpt].x = Ex - addx*h;
        endlist[Nendpt++].y = Ey - addy*h;
    }
} /* for (h) */
Ex=endlist[0].x; Ey=endlist[0].y;
EndptOK = Nendpt>0;
}
else { /* TryAllEndPts==FALSE. just calc the pt closest to the mid-pt */
    if (!slopecmp(dy,dx,b.ly,b.lx)) {
        /*
        * dy dx is equal or below the lower limit.
        * i.e. the slope just above the lower limit should be taken.
        * if the lower gradient limit hits exactly on a sub-pixel res point,
        * the truncation of the integer division has done part of the job.
        */
        if (Is_Horizontal(maindir)) {
            tmp= dx*b.ly; exact= (dx==0 || tmp%b.lx==0);
            Ey = tmp/b.lx + Sy + (b.lx>0 ? (b.ly>0 ? 1 : exact)
                                   : (b.ly>0 ? -exact : -1 ));
        }
        else {
            tmp= dy*b.lx; exact= (dy==0 || tmp%b.ly==0);
            Ex = tmp/b.ly + Sx + (b.ly>0 ? (b.lx>0 ? -exact : -1 )
                                   : (b.lx>0 ? 1 : exact));
        }
        EndptOK = Pretest || WithinBound(Ey-Sy,Ex-Sx,b);
    }
    else if (!slopecmp(b.uy,b.ux,dy,dx)) {
        /*
        * dy dx is equal or above the upper limit.
        * i.e. the slope just below the upper limit should be taken.
        * if the upper gradient limit hits exactly on a sub-pixel res point,
        * the truncation of the integer division has done part of the job.
        */
        if (Is_Horizontal(maindir)) {
            tmp= dx*b.uy; exact= (tmp%b.ux==0);
            Ey = tmp/b.ux + Sy + (b.ux>0 ? (b.uy>0 ? -exact :-1 )
                                   : (b.uy>0 ? 1 : exact));
        }
        else {
            tmp= dy*b.ux; exact= (tmp%b.uy==0);
            Ex = tmp/b.uy + Sx + (b.uy>0 ? (b.ux>0 ? 1 : exact)
                                   : (b.ux>0 ? -exact :-1 ));
        }
        EndptOK = Pretest || WithinBound(Ey-Sy,Ex-Sx,b);
    }
    else /* dy,dx is within the limits. i.e. mid-point is taken. */
        EndptOK=1;
}
```

```

    } /* if (TryAllEndPts)..else.. */

    if (EndptOK) break; /* if Pretest is TRUE, EndptOK always TRUE */
    else { /* no valid end-point can be found, step back one edge */
        backward(e1);
        last = Pedgehistory[--NPedges-2];
        Nbound--;
    }
} /* for (;;) */ /* until a valid end pt is found */
breaktrace=TRUE; /* one line segment found. */
}
else { /* limits not crossed over yet */
    Nbound++; /* one more new valid bound */
    continue; /* continue to get another Pedge */
} /* if (various trace breaking conditions) */
} /* if (nextp==NULL || ndir==ESTABLISHED) */
else { /* i.e. dominant and trnsvrse direction not yet established */
    breaktrace = FALSE;
    if (ndir<3) {
        for (i=0;i<ndir;i++) { /* compare with previous dir's */
            if (against(dnow,dir[i])) { /* there is a `U' turn ... */
                breaktrace = TRUE; /* therefore an early stop */
                Ex=MidX(last); Ey=MidY(last);
                if (TryAllEndPts) {
                    endlst[0].x=Ex; endlst[0].y=Ey;
                    Nendpt=1;
                } /* if (TryAllEndPts) */
            } /* for () */
        }
        if (ndir<2 || dnow!=dir[1] || dir[0]!=dir[1]) {
            dir[ndir]=dnow;
            ndir++;
        }
    }

    if (ndir==3) /* now we can establish the directions... */
    {
        /*      -   |   */
        if (dir[0]!=dir[1]) { /* -| or -| */
            maindir=dir[2]; /*      |   */
            if (dir[1]==dir[2]) { /*      -|   */
                trnsvrse=dir[0]; /* the 1st dir is the trnsvrse dir */
                if (Is_Horizontal(maindir)) {
                    Ux = Lx = MidX(e) - Sx;
                    Uy = (Ly = e.y1-Sy-HalfSUBPIXRES) +SUBPIXRES;
                }
                else {
                    Uy = Ly = MidY(e) - Sy;
                    Ux = (Lx = e.x1-Sx-HalfSUBPIXRES) +SUBPIXRES;
                }
            }
            else {
                /*      -   |   */
                /* -|   */
                trnsvrse=dir[1];
                if (Is_Horizontal(maindir)) {
                    Lx = Ux = MidX(e)-Sx;
                    Ly = (Uy = MidY(e)+HalfSUBPIXRES-Sy) -SUBPIXRES;
                }
                else {
                    Ly = Uy = MidY(e)-Sy;
                    Lx = (Ux = MidX(e)+HalfSUBPIXRES-Sx) -SUBPIXRES;
                }
            }
        }
    }
}

```

```

    }
    else {
        maindir=dir[0];
        trnsvrse=dir[2];
        if (Is_Horizontal(maindir)) {
            Lx = e.x1 + (maindir>0 ? -HalfSUBPIXRES : HalfSUBPIXRES) - Sx;
            Ux = Lx + (maindir>0 ? SUBPIXRES : -SUBPIXRES);
            Uy = Ly = MidY(e) - Sy;
        }
        else {
            Ly = e.y1 + (maindir>0 ? -HalfSUBPIXRES : HalfSUBPIXRES) - Sy ;
            Uy = Ly + (maindir>0 ? SUBPIXRES : -SUBPIXRES);
            Ux = Lx = MidX(e) - Sx;
        }
    }
    if (slopecmp(Ly,Lx,Uy,Ux)) { /* swap the grad limits if necessary */
        bound[0].uy=Ly; bound[0].ux=Lx; /* Ly Lx larger */
        bound[0].ly=Uy; bound[0].lx=Ux;
    }
    else {
        bound[0].uy=Uy; bound[0].ux=Ux; /* Uy Ux larger */
        bound[0].ly=Ly; bound[0].lx=Lx;
    }
    Nbound=1; /* first bound established */
    ndir = ESTABLISHED;
} /* if (ndir==3) */
} /* if (ndir==ESTABLISHED)...else... */

/*-----*/
if (breaktrace) { /* one line ended */
    /*-----*/

    backward(el); /* last pixel edge shall be the start of another line. */

    if (TryAllEndPts) {
        if (maxlen < (el.current-currentsave)) { /* longer than the longest */
            maxlen = el.current-currentsave; /* longest distance so far */
            bestpt_last=last; /* save the last edge */
            bestpt=startpt; /* update the best pt so far*/
            bestpt_currentsave=el.current; /* save the cursor for el */
            for (i=0; i<Nendpt; i++) bestpt_endlist[i]=endlist[i]; /* save end pts */
            bestpt_Nendpt=Nendpt; /* save the no. of end pts */
        }
        startpt++; /* next starting pt in startpts[] */
        if (startpt >= Nstartpt) { /* all starting pts have been tried */
            currentsave=el.current=bestpt_currentsave; /* save the ending pos */
            estartsave=e=bestpt_last; /* save the ending Pedge */
            lines[linescount++] = startpts[bestpt]; /* record the best pt */
            if (linescount>=MaxNLine) return -1; /* too many lines */
            if (bestpt_currentsave>=el.Nedges-1) { /* no more Pixel edges ? */
                lines[linescount++]=bestpt_endlist[0]; /* record end pt as well */
                return linescount>=MaxNLine ? -1 : linescount; /* done */
            }
        }

        Nstartpt=bestpt_Nendpt; /* use the list of end pts as starting pts */
        for (i=0; i<bestpt_Nendpt; i++) startpts[i]=bestpt_endlist[i];

        startpt=0; /* consider the first one in the new list */
        Sx=startpts[0].x; Sy=startpts[0].y;
        maxlen=bestpt=-1; /* reset maxlen and bestpt to undefined */
    }
    else { /* i.e. startpt<Nstartpt. try next starting point */
        Sx=startpts[startpt].x; Sy=startpts[startpt].y; /* next starting pt */
    }
}

```

```

/* rewind and start again */
    el.current=currentsave;
    e=last=estartsave;
} /* if (startp>=Nstartpt) ... else ... */
}
else { /* i.e. TryAllEndPts==FALSE. simply start at the end pt again */
    Sx=Ex; Sy=Ey; e=last;
    lines[linescount].x=Ex; lines[linescount++].y=Ey;
    if (linescount>=MaxNLine) return -1; /* too many lines */
    if (el.current>=el.Nedges-1) return linescount; /* no more Pedges, done */
}
starttrace=TRUE; /* start again */
} /* if (breaktrace) */
} /* for (starttrace=TRUE;;) infinite loop */
} /* fitlines() */

/*****
* THE INVERSE PROCESS
*****/
#define divisible(a,b) ((a)%(b)==0)
#define ishori(x,y) (divisible(x,SUBPIXRES)||\
                    divisible(y+HalfSUBPIXRES,SUBPIXRES))
#define isvert(x,y) (divisible(y,SUBPIXRES)||\
                    divisible(x+HalfSUBPIXRES,SUBPIXRES))
#define sign(x) ((x)>=0 ? 1 : -1)
#define Trunc(n) ((n)/SUBPIXRES*SUBPIXRES)
static int lastx,lasty,lastdir; /* to avoid duplicated pixel edges */

static void drawHPedge(int x, int y) { /* draw a horizontal pixel edge */
    if (lastx==x && lasty==y && lastdir==HRZ) /* starting edge==last ending edge */
        return;
    lastx=x; lasty=y; lastdir=HRZ;
    DrawPixelEdge(x/SUBPIXRES, y/SUBPIXRES, HRZ); /* call the user function */
} /* drawHPedge() */

static void drawVPedge(int x, int y) { /* draw a vertical pixel edge */
    if (lastx==x && lasty==y && lastdir==VRT) /* starting edge==last ending edge */
        return;
    lastx=x; lasty=y; lastdir=VRT;
    DrawPixelEdge(x/SUBPIXRES, y/SUBPIXRES, VRT); /* call the user function */
} /* drawVPedge() */

/*****
* makejaggedline(): A modified Bresenham's mid-point algorithm. Based on
* the code from the original Graphics Gem. Neither the starting pt
* nor the ending pt need to be at the mid-pt of a pixel edge.
* The decision variable has been scaled by SUBPIXRES and preloaded
* with the offset from a 'proper' starting pt, i.e. the mid-pt of the
* first pixel edge pointing to the dominant direction.
*****/
static makejaggedline(int x1, int y1, int x2, int y2) {
    int d, x, y, ax, ay, sx, sy, dx, dy, finaltrnsvrse;

    dx = x2-x1; ax = abs(dx)*SUBPIXRES; sx = sign(dx)*SUBPIXRES;
    dy = y2-y1; ay = abs(dy)*SUBPIXRES; sy = sign(dy)*SUBPIXRES;

    /*=====*/
    if (ax>ay) /* x dominant */
    { /*=====*/
        if (isvert(x1,y1)) /* 1st edge is trnsvrse. skip to the mid-pt */
        { /* of the next dominant dir edge. */
            y=Trunc(y1 + HalfSUBPIXRES) + sy/2;

```

```

    x=Trunc(x1) + HalfSUBPIXRES + sx/2;
    drawVPedge(x-sx/2,y-sy/2);          /* draw the skipped edge */
}
else { /* 1st edge is dominant. shift to the mid-pt */
    x=Trunc(x1 + HalfSUBPIXRES);
    y=Trunc(y1) + HalfSUBPIXRES;
}
/* preload decision var `d' with offset x-x1, y-y1. (if any) */
d = ay - (ax>>1) + ay*(x-x1)/sx - ax*(y-y1)/sy;
for (;;) {
    drawHPedge(x,y);
    if (abs(x-x2) < HalfSUBPIXRES) return; /* final edge is a dominant one */
    x += sx;
    finaltrnsvrse = dx>0 ? x>x2: x<x2;
    if (d>0 || finaltrnsvrse) {          /* if the final edge is a trnsvrse */
        drawVPedge(x-sx/2,y+sy/2);      /* one, draw it before stopping */
        y += sy;
        d -= ax;
    }
    if (finaltrnsvrse) return;
    d += ay;
} /* for (;;) */
}
else
{
    if (ishori(x1,y1)) /* 1st edge trnsvrse. skip to the mid-pt */
    {
        /* of the next dominant dir edge */
        x=Trunc(x1 + HalfSUBPIXRES) + sx/2;
        y=Trunc(y1) + HalfSUBPIXRES + sy/2;
        drawHPedge(x-sx/2, y-sy/2);      /* draw the skipped edge */
    }
    else { /* 1st edge is dominant. shift to the mid-pt */
        x=Trunc(x1) + HalfSUBPIXRES;
        y=Trunc(y1 + HalfSUBPIXRES);
    }
    /* preload decision var `d' with offset x-x1, y-y1 (if any) */
    d = ax - (ay>>1) + ax*(y-y1)/sy - ay*(x-x1)/sx;
    for (;;) {
        drawVPedge(x,y);
        if (abs(y-y2) < HalfSUBPIXRES) return; /* final edge is a dominant one */
        y += sy;
        finaltrnsvrse = dy>0 ? y>y2 : y<y2;
        if (d>0 || finaltrnsvrse) {      /* if the final one is a trnsvrse */
            drawHPedge(x+sx/2, y-sy/2);  /* one, draw it before stopping. */
            x += sx;
            d -= ay;
        }
        if (finaltrnsvrse) return;
        d += ax;
    } /* for (;;) */
} /* if (ax>ay)... else ...*/
} /* makejaggedline() */

```

```

/*****
 * linestojagged(): reconstruct a sequence of pixel edges from given lines *
 *                  by calling the makejaggedline() function.                *
 *****/

```

```

void linestojagged(int Nlines, IntPoint2 *lines) {
    int from_x, from_y, i;
    lastdir=0;
    for (from_x=lines[0].x, from_y=lines[0].y, i=1; i<Nlines; i++) {

```

```
    makejaggedline(from_x,from_y,lines[i].x,lines[i].y);  
    from_x=lines[i].x;    from_y=lines[i].y;  
}  
} /* linetojagged() */
```

```
/* revfit.h: definitions for reversible straight line reconstruction routines */
#include "../ch7-7/GG4D/GGems.h"
#define HRZ 1
#define VRT 2
/* Watch out for the precision of `int' type. Make sure that the max */
/* coordinate value * SUBPIXRES can be stored in an `int'. */
#define SUBPIXRES 32

/*****
/* typedef for Edgelist: the list of edges where lines are to be fitted */
/*****/
typedef struct {
    int x,y; /* in bitmap resolution +ve */
    int dir; /* --- <-- H edge ^ */
} PixelEdge; /* (x,y)--> * | <-- V edge | --> +ve */











typedef struct {
    int Nedges; /* number of edges in the list */
    int current; /* current edge being visited */
    PixelEdge *list; /* the list of edges found from the pixmap */
} Edgelist;

int fitlines(Edgelist el, boolean Pretest, boolean TryAllEndPts,
             IntPoint2 *lines, int MaxLines);
void linestojagged(int Nlines, IntPoint2 *lines);
```



# Index of

## /pubs/tog/GraphicsGems/gemsv/ch7-6/

Name	Last modified	Size	Description
 <a href="#">_Parent Directory</a>			
 <a href="#">_Makefile</a>	29-Jun-00 08:25	1K	
 <a href="#">_NFF</a>	29-Jun-00 08:25	4K	
 <a href="#">_ZRendv10/</a>	29-Jun-00 08:25	1K	
 <a href="#">_ZRendv10.c</a>	29-Jun-00 08:25	37K	
 <a href="#">_ZRendv10.h</a>	29-Jun-00 08:25	2K	
 <a href="#">_ZRendv10Announce</a>	29-Jun-00 08:25	3K	
 <a href="#">_sx11/</a>	29-Jun-00 08:25	1K	
 <a href="#">_tga/</a>	29-Jun-00 08:25	1K	
 <a href="#">_tpot11.nff</a>	29-Jun-00 08:25	30K	

```
CC = gcc

PROG=ZRendv10

GCCFLAGS = -fpcc-struct-return
CFLAGS = -ansi -O2

INCLUDE = -I/usr/X11R5/include -I./tga

LDLIBS = -lX11 -lm -ltga

LDFLAGS = -L/usr/X11R5/lib -L./tga

$(PROG): $(PROG).c $(PROG).h
    (cd tga; make)
    (cd sx11; make)
    $(CC) $(GCCFLAGS) $(CFLAGS) $(INCLUDE) $(PROG).c \
    $(LDFLAGS) $(LDLIBS) -o $(PROG)
```

## Neutral File Format

Presently the following entities are supported by the modified NFF:

A simple perspective frustum

A background color description

A positional (vs. directional) light source description

A surface properties description

Triangle descriptions

NFF files contain lines of text. For each entity, the first field defines its type. The rest of the line and possibly other lines contain further information about the entity.

Entities include:

"v" - viewing vectors and angles

"b" - background color

"l" - positional diffuse light location

"s" - positional specular light location

"f" - object material properties

"pp" - triangular patch primitive

These are explained below.

Viewpoint location

Template:

"v"

"from" Fx Fy Fz

"at" Ax Ay Az

"up" Ux Uy Uz

"angle" angle

"hither" hither

"yon" yon

"resolution" xres yres

Type Specification:

v

from float float float

at float float float

up float float float

angle float

hither float

yon float

resolution integer integer

Description:

From: the eye location in XYZ.

At: a position to be at the center of the image, in XYZ world coordinates. A.k.a. "lookat".

Up: a vector defining which direction is up, as an XYZ vector.

Angle: in degrees, defined as from the center of top pixel row to bottom pixel row and left column to right column.

Hither: distance of the hither plane from the eye.

Yon: distance of the yon plane from the eye.

Resolution: in pixels, in x and in y (currently ignored).

Note that no assumptions are made about normalizing the data (e.g. the from-at distance does not have to be 1). Also, vectors are not required to be perpendicular to each other.

For all databases, the aspect ratio is 1.0.

A view entity must be defined before any objects are defined.

Background color

Template:

"b" R G B

Type Specification:

b float float float

Description:

Background color is given R,G and B values between 0 and 1.

Positional Diffuse light

Template:

"l" X Y Z IpR IpG IpB

Type Specification:

l float float float float float float

Description:

A diffuse light is defined by its XYZ position. All light entities must be defined before any objects are defined. Light source position and its Red, Green and Blue components, in the range 0.0 to 1.0 are specified. Up to NUM\_LIGHT\_SOURCES light sources can be specified.

Positional Specular light

Template:

"s" X Y Z IpR IpG IpB ks n

Type Specification:

1 float float float float float float float integer

Description:

A specular light is defined by its XYZ position. All light entities must be defined before any objects are defined. Light source position and its Red, Green and Blue components, in the range 0.0 to 1.0 are specified. At most one specular light source can be specified. The specular reflection coefficient (ks) in the range 0.0 to 1.0 and the specular exponent (n) are also specified.

Fill color and shading parameters

Template:

"f" OdR OdG OdB Kd Ka c1 c2

Format:

f float float float float float float float float

Description:

OdR, OdG and OdB specify the object's diffuse color (RGB triple) in the range 0.0 to 1.0. Kd is the diffuse reflection coefficient, Ka the ambient reflection coefficient, c1 and c2 are the parameters for the attenuation function  $fatt(d) = 1/(1 + c1*d + c2 * d * d)$  used for the light source where "d" is the distance of a point from the light source (See the textbook by Foley, van Dam, Feiner, Hughes and Phillips, Chapter 14). Usually,  $0 \leq Kd \leq 1$  and  $0 \leq Ka \leq 1$ , and Kd and Ka are independent.

Triangular patch

Template:

"pp" total\_vertices

vert1.x vert1.y vert1.z norm1.x norm1.y norm1.z

[etc. for total\_vertices vertices]

Format:


pp integer

```
[float float float float float float] <-- for total_vertices vertices
```

Description:

All objects are considered one-sided. A patch is defined by a set of vertices and their normals. With these databases, a patch is defined to have all points coplanar. A patch has only one side, with the order of the vertices being counterclockwise as you face the patch (right-handed coordinate system). The first two edges must form a non-zero convex angle, so that the normal and side visibility can be determined. Note that, since only triangular patches are supported total\_vertices shown must always be 3.

# Index of /pubs/tog/GraphicsGems/gemsv/ch7-6/ZRendv10/

Name	Last modified	Size	Description
<hr/>			
 <a href="#">_Parent Directory</a>			

```
/*
    Z Buffer Rendering Program Version 10, 02/12/96
    Copyright Raghu Karinithi, West Virginia University
*/
#include "ZRendv10.h"

ZBuffer zb;
OrigTriangle localSet[NUM_TRIANGLES];
FrameBuffer fb;

int16 BFCULL;
int16 TRIVIAL_REJECT;
int16 CLIP;
int16 PHONG;
MAT3fvec Hvector;
SpecLightPoint SpecSource;
float out_xmin, out_xmax, out_ymin, out_ymax, out_zmin, out_zmax;

/* ----- */

/*  Generic Utilities */

#define TSWAP(_a_, _b_, _c_) ((_c_) = (_a_), (_a_) = (_b_), (_b_) = (_c_))
#define min(_a_, _b_) (((_a_) < (_b_)) ? (_a_) : (_b_))
#define max(_a_, _b_) (((_a_) > (_b_)) ? (_a_) : (_b_))
#define ZFABS(x) (((x) < 0.0) ? -(x) : (x))

/* ----- */

/*      Matrix and Vector utilities */

#define MAT3_SCALE_VEC(RESULT_V,V,SCALE) \
    ((RESULT_V)[0]=(V)[0]*(SCALE), (RESULT_V)[1]=(V)[1]*(SCALE), \
     (RESULT_V)[2]=(V)[2]*(SCALE))

#define MAT3_DOT_PRODUCT(V1,V2) \
    ((V1)[0]*(V2)[0] + (V1)[1]*(V2)[1] + (V1)[2]*(V2)[2])

#define MAT3_SUB_VEC(RESULT_V,V1,V2) \
    ((RESULT_V)[0] = (V1)[0]-(V2)[0], (RESULT_V)[1] = (V1)[1]-(V2)[1], \
     (RESULT_V)[2] = (V1)[2]-(V2)[2])

#define MAT3_NORMALIZE_VEC(V,TEMP) \
    if ((TEMP = sqrt(MAT3_DOT_PRODUCT(V,V))) > MAT3_EPSILON) { \
        TEMP = 1.0 / TEMP; \
        MAT3_SCALE_VEC(V,V,TEMP); \
    } else TEMP = 0.0

#define MAT3_CROSS_PRODUCT(RES,V1,V2) \
    ((RES)[0] = (V1)[1] * (V2)[2] - (V1)[2] * (V2)[1], \
     (RES)[1] = (V1)[2] * (V2)[0] - (V1)[0] * (V2)[2], \
     (RES)[2] = (V1)[0] * (V2)[1] - (V1)[1] * (V2)[0])

void MAT3identity(MAT3mat m)
{
    int16 i, j;
```



```
for (i=0; i < 4; i++)
    for (j=0; j < 4; j++)
        if (i==j)
            m[i][j] = 1.0;
        else
            m[i][j] = 0.0;
}

void MAT3mult(MAT3mat result_mat, MAT3mat mat1, MAT3mat mat2)
/* result_mat = mat1 * mat2 */
{
    int16 i, j, k;
    double tmp;
    MAT3mat tmp_mat;

    for (i=0; i < 4; i++)
        for (j=0; j < 4; j++) {
            tmp = 0.0;
            for (k=0; k < 4; k++)
                tmp += mat1[i][k]*mat2[k][j];
            tmp_mat[i][j] = tmp;
        }
    for (i=0; i < 4; i++)
        for (j=0; j < 4; j++)
            result_mat[i][j] = tmp_mat[i][j];
}

#define Det2D(a,b,c,d)((a)*(d)-(b)*(c))

/* ----- */

/* Fixed Point Utilities */
#define fp_floor(x) \
    (((x) < 0) ? \
        (!((x) & LOMASK)) ? ((x) >> LOBITS) : (((x) >> LOBITS)+1) : \
        ((x)>> LOBITS) \
    )

/*
 * Works only if x +ve.
 */

#define fp_floor_pos(x) ((x) >> LOBITS)

/*
 * We use the following approximation.
 */

#define fp_div(x,y) \
    (((x) < 0) ? (((x) / (y) - 1) << LOBITS) : (((x) / (y)) << LOBITS))

/*
 * Works only if xlo = 0.
 */

#define fp_mult1(x,y) \
    (((x)<0) ? \
        -(((-(x) & HIMASK)>> LOBITS)*(y)) : \
```

```
((x)>> LOBITS)*(y))

/*
 * Works only if xhi = 0
 */

#define fp_mult2(x,y) \
(((y)<0)?(-((((-(y))&HIMASK)>>LOBITS)*(x))+((((-(y))&LOMASK)*(x))>>LOBITS)))\
: (((y)>> LOBITS) * (x)) + (((y)&LOMASK)*(x))>>LOBITS))

/*
 * Fractional Part if x +ve
 */

#define fp_fraction(x) ((x) & LOMASK)

/*
 * float to fix conversion
 */

#define sh_float_to_fix(x) ( \
(((x) >> FLOATSZM1) ? \
(-( \
((int32) (((x) & EXPMASK) >> FFRACSZ) - FEXPOFFSET) > FFRACSZMLOBITS)? \
((((x) & FFRACMASK) | (0x00800000)) << \
(((x) & EXPMASK) >> FFRACSZ) - FEXPOFFSET - FFRACSZ + LOBITS)) \
: \
((((x) & FFRACMASK) | (0x00800000)) >> \
(FFRACSZ - LOBITS -(((x) & EXPMASK) >> FFRACSZ) + FEXPOFFSET )) \
)) \
:\
(((int32) (( (x) & EXPMASK) >> FFRACSZ) - FEXPOFFSET) > FFRACSZMLOBITS)? \
((((x) & FFRACMASK) | (0x00800000)) << \
(((x) & EXPMASK) >> FFRACSZ) - FEXPOFFSET - FFRACSZ + LOBITS)) \
: \
((((x) & FFRACMASK) | (0x00800000)) >> \
(FFRACSZ - LOBITS -(((x) & EXPMASK) >> FFRACSZ) + FEXPOFFSET )) \
)) \
)

/*
 * float to fix conversion for Z
 */

#define shzfloat_to_fix(x) ( \
(((x) >> FLOATSZM1) ? \
(-( \
((int32) (((x) & EXPMASK) >> FFRACSZ) - FEXPOFFSET) > (FFRACSZ-ZLOBITS)))? \
((((x) & FFRACMASK) | (0x00800000)) << \
(((x) & EXPMASK) >> FFRACSZ) - FEXPOFFSET - FFRACSZ + ZLOBITS)) \
: \
((((x) & FFRACMASK) | (0x00800000)) >> \
(FFRACSZ - ZLOBITS -(((x) & EXPMASK) >> FFRACSZ) + FEXPOFFSET )) \
)) \
:\
(((int32) (( (x) & EXPMASK) >> FFRACSZ) - FEXPOFFSET) > (FFRACSZ-ZLOBITS)))? \
((((x) & FFRACMASK) | (0x00800000)) << \
(((x) & EXPMASK) >> FFRACSZ) - FEXPOFFSET - FFRACSZ + ZLOBITS)) \
: \
((((x) & FFRACMASK) | (0x00800000)) >> \
(FFRACSZ - ZLOBITS -(((x) & EXPMASK) >> FFRACSZ) + FEXPOFFSET )) \
)
```

```
) ) \
)

/* ----- */

/* Output / Display */

/*
 * This function writes the framebuffer to a TARGA file with
 * 24 bit color.
 */

void write_tga_buffer (FrameBuffer fb, char *filename)
{
    bitmap_hdr    output;
    unsigned char *rptr, *gptr, *bptr;
    int16         i, j;

    allocatebitmap (&output, WW, WH, COLOR_DEPTH, 1<<COLOR_DEPTH);
    rptr = output.r;
    gptr = output.g;
    bptr = output.b;

    /*
     * Top to bottom row, left to right within a row.
     */

    for (j = (WH - 1); j >= 0; j --) {
        for (i = 0; i < WW; i ++ ) {
            *rptr++ = ((fb[j][i]) & RMASK);
            *gptr++ = ((fb[j][i]) & GMASK) >> 8;
            *bptr++ = ((fb[j][i]) & BMASK) >> 16;
        }
    }

    write_tga_file (filename,&output);
} /* write_tga_buffer */

/* ----- */

/* Rasterization */

/*
 * The following : edge data structure, and functions EdgeScan,
 * EdgeSetup, and rasterize_sorted_triangle, are modified from the code
 * obtained in Graphics Gems III subdirectory accurate_scan due to Kurt
 * Fleischer. Also Refer Lathrop, Kirk, Voorhies, IEEE CG&A 10(5),
 * 1990 for a discussion
 */

#define EdgeScan(e) \
    if ((e).E<0) { \
        (e).Ix += (e).AStep; (e).E += (e).DEA; \
    } \
    else { \
        (e).Ix += (e).BStep; (e).E += (e).DEB; \
    }
```

```
#define EdgeSetup(e,xs,ys,dx,dy) \
    if ((dy) >= FIX1) { \
        si = fp_mult2((FIX1 - (fp_fraction((ys)))),(dx)); \
        xi = (xs) + fp_div(si,(dy)); \
        si = fp_div((dx), (dy)); \
        (e).AStep = fp_floor(si); \
        (e).BStep = (e).AStep+1; \
        (e).DEA= (dx) - fp_mult1(si, (dy)); \
        (e).DEB = (e).DEA - (dy); \
        (e).E = fp_mult2(fp_fraction(xi),(dy)) + (e).DEB; \
        (e).Ix = fp_floor_pos(xi); \
    } /* EdgeSetup */

/*
 * This function does the actual rasterization.
 */

void rasterize_sorted_triangle (int16          minyv, int16          midyv,
                               int16          maxyv, ColorTriangleP tp)
{
    int16          xleft[WH], xright[WH];
    int16          xmin, xmax, ymin, ymax;
    fixpoint       x0, y0, x1, y1, x2, y2;
    edge           left, right;
    int16          temp;
    int16          ccw;
    fixpoint       xi, si, dx, dy;
    int16          y;

    float fdx, fdy;
    float dr_by_dy, dg_by_dy, db_by_dy, dr_by_dx, dg_by_dx, db_by_dx, dz_by_dy,
          dz_by_dx;
    int16          x;
    float fx0, fy0, z0, r0, g0, b0; /* fx0, fy0 distinguish them from x0 and y0 */
    float rxy0,gxy0,bxy0,rx0ymin,gx0ymin,bx0ymin,zx0ymin;
    int32 col, rgboffset[WW];
    fixpoint rx0yminf, gx0yminf, bx0yminf, zx0yminf, dr_by_dy_f, dg_by_dy_f,
          db_by_dy_f, dz_by_dy_f, dz_by_dx_f, dz_by_dx_x0f, z;
    float yly0, y2y0, xlx0, x2x0, det, zlz0, z2z0;
    float rlr0, r2r0, glg0, g2g0, blb0, b2b0;
    float ndoth, ndothpown;
    int16 specr, specg, specb;
    int32 specterm;

    xlx0 = (tp->vertices[midyv]).vertex[X] - (tp->vertices[minyv]).vertex[X];
    x2x0 = (tp->vertices[maxyv]).vertex[X] - (tp->vertices[minyv]).vertex[X];
    yly0 = (tp->vertices[midyv]).vertex[Y] - (tp->vertices[minyv]).vertex[Y];
    y2y0 = (tp->vertices[maxyv]).vertex[Y] - (tp->vertices[minyv]).vertex[Y];

    det = xlx0 * y2y0 - x2x0 * yly0;

    /*
     * Convert the vertices of the triangle and their color value
     * to 'fixpoint' representation.
     */

    x0 = sh_float_to_fix (*((fixpoint *)&((tp->vertices[minyv]).vertex[X])));
    y0 = sh_float_to_fix (*((fixpoint *)&((tp->vertices[minyv]).vertex[Y])));
```

```
x1 = sh_float_to_fix (*(fixpoint *)&((tp->vertices[midyv]).vertex[X]));
y1 = sh_float_to_fix (*(fixpoint *)&((tp->vertices[midyv]).vertex[Y]));

x2 = sh_float_to_fix (*(fixpoint *)&((tp->vertices[maxyv]).vertex[X]));
y2 = sh_float_to_fix (*(fixpoint *)&((tp->vertices[maxyv]).vertex[Y]));

if (((tp->vertices[minyv]).vertex[X]) == 0.0) x0 = 0;
if (((tp->vertices[minyv]).vertex[Y]) == 0.0) y0 = 0;

if (((tp->vertices[midyv]).vertex[X]) == 0.0) x1 = 0;
if (((tp->vertices[midyv]).vertex[Y]) == 0.0) y1 = 0;

if (((tp->vertices[maxyv]).vertex[X]) == 0.0) x2 = 0;
if (((tp->vertices[maxyv]).vertex[Y]) == 0.0) y2 = 0;

xmin = max ((fp_floor_pos(min(min(x0,x1),x2))-1), 0);
xmax = min ((fp_floor_pos(max(max(x0,x1),x2))+1), (WW-1));

/*
 * Find out whether the triangle edges are oriented in clockwise
 * or anti-clockwise direction.
 */

ccw = det > 0.0;

/*
 * Setup first pair of edges.
 */

if (ccw) {
    EdgeSetup (left,  x0, y0, x2-x0, y2-y0)
    EdgeSetup (right, x0, y0, x1-x0, y1-y0)
} /* if ccw orientation of edges */
else {
    EdgeSetup (left,  x0, y0, x1-x0, y1-y0)
    EdgeSetup (right, x0, y0, x2-x0, y2-y0)
} /* else clockwise orientation of edges */

ymin = fp_floor_pos(y0)+1;
temp = fp_floor_pos(y1);
for (y = ymin; y <= temp; y++) {
    xleft[y] = left.Ix;    EdgeScan (left)
    xright[y] = right.Ix; EdgeScan (right)
} /* for every scan line upto the lower 'y' of the two edges */

/*
 * Setup the third edge.
 */

if (ccw) {
    EdgeSetup (right, x1, y1, x2-x1, y2-y1)
}
else {
    EdgeSetup (left,  x1, y1, x2-x1, y2-y1)
}

/*
 * Now scan convert the rest of the triangle.
 */
```

```
temp = fp_floor_pos(y1) + 1;
ymax = fp_floor_pos (y2);
for (y = temp; y <= ymax; y++) {
    xleft[y] = left.Ix;    EdgeScan (left)
    xright[y] = right.Ix;  EdgeScan (right)
} /* for every scan line till the uppermost vertex of the triangle */

if (PHONG) {
    ndoth = tp->normal[X] * Hvector[X] + tp->normal[Y] * Hvector[Y] +
        tp->normal[Z] * Hvector[Z];
    ndothpown = pow(ndoth, SpecSource.spec_exp);
    specr = SpecSource.red * ndothpown;
    specg = SpecSource.green * ndothpown;
    specb = SpecSource.blue * ndothpown;
    specterm = specr + (specg << 8) + (specb << 16);
}
else
    specterm = 0;

z1z0 = (tp->vertices[midyv]).vertex[Z] - (tp->vertices[minyv]).vertex[Z];
z2z0 = (tp->vertices[maxyv]).vertex[Z] - (tp->vertices[minyv]).vertex[Z];

dz_by_dx = (y2y0 * z1z0 - y1y0 * z2z0)/det;
dz_by_dy = (x1x0 * z2z0 - x2x0 * z1z0)/det;

r1r0 = (tp->vertices[midyv]).red - (tp->vertices[minyv]).red;
r2r0 = (tp->vertices[maxyv]).red - (tp->vertices[minyv]).red;

dr_by_dx = (y2y0 * r1r0 - y1y0 * r2r0)/det;
dr_by_dy = (x1x0 * r2r0 - x2x0 * r1r0)/det;

g1g0 = (tp->vertices[midyv]).green - (tp->vertices[minyv]).green;
g2g0 = (tp->vertices[maxyv]).green - (tp->vertices[minyv]).green;

dg_by_dx = (y2y0 * g1g0 - y1y0 * g2g0)/det;
dg_by_dy = (x1x0 * g2g0 - x2x0 * g1g0)/det;

b1b0 = (tp->vertices[midyv]).blue - (tp->vertices[minyv]).blue;
b2b0 = (tp->vertices[maxyv]).blue - (tp->vertices[minyv]).blue;

db_by_dx = (y2y0 * b1b0 - y1y0 * b2b0)/det;
db_by_dy = (x1x0 * b2b0 - x2x0 * b1b0)/det;

fx0 = (tp->vertices[minyv]).vertex[X];
fy0 = (tp->vertices[minyv]).vertex[Y];
z0 = (tp->vertices[minyv]).vertex[Z];
r0 = (tp->vertices[minyv]).red;
g0 = (tp->vertices[minyv]).green;
b0 = (tp->vertices[minyv]).blue;

fdy = ymin - fy0;
rx0ymin = r0 + dr_by_dy * fdy;
gx0ymin = g0 + dg_by_dy * fdy;
bx0ymin = b0 + db_by_dy * fdy;
zx0ymin = z0 + dz_by_dy * fdy;

fdx = xmin - fx0;
rxy0 = dr_by_dx * fdx;
gxy0 = dg_by_dx * fdx;
bxy0 = db_by_dx * fdx;
```

```
for (x=xmin; x <= xmax; x++) {
    rgboffset[x] = (((int32) rxy0) + (((int32) gxy0) << 8) +
        (((int32) bxy0) << 16) + specterm;
    rxy0 += dr_by_dx;
    gxy0 += dg_by_dx;
    bxy0 += db_by_dx;
}

rx0yminf = sh_float_to_fix (*((fixpoint *)&rx0yminf));
gx0yminf = sh_float_to_fix (*((fixpoint *)&gx0yminf));
bx0yminf = sh_float_to_fix (*((fixpoint *)&bx0yminf));
zx0yminf = shzfloat_to_fix (*((fixpoint *)&zx0yminf));
dr_by_dy_f = sh_float_to_fix (*((fixpoint *)&dr_by_dy));
dg_by_dy_f = sh_float_to_fix (*((fixpoint *)&dg_by_dy));
db_by_dy_f = sh_float_to_fix (*((fixpoint *)&db_by_dy));
dz_by_dy_f = shzfloat_to_fix (*((fixpoint *)&dz_by_dy));
dz_by_dx_f = shzfloat_to_fix (*((fixpoint *)&dz_by_dx));
fdx = dz_by_dx*fx0;
dz_by_dx_x0f = shzfloat_to_fix (*((fixpoint *)&fdx));

for (y=ymin; y <= ymax; y++) {
    col = (fp_floor_pos(rx0yminf) + (fp_floor_pos(gx0yminf)<<8) +
        (fp_floor_pos(bx0yminf)<< 16));
    z = zx0yminf + xleft[y]*dz_by_dx_f - dz_by_dx_x0f;
    for (x = xleft[y]; x <= xright[y]; x++) {
        if (zb[y][x] < z) {
            zb[y][x] = z;
            fb[y][x] = col + rgboffset[x];
        }
        z += dz_by_dx_f;
    }
    rx0yminf += dr_by_dy_f;
    gx0yminf += dg_by_dy_f;
    bx0yminf += db_by_dy_f;
    zx0yminf += dz_by_dy_f;
}
}/* rasterize_sorted_triangle */

void sort_and_rasterize_triangle (ColorTriangleP tp)
{
    int16 minyv = 0, midyv = 1, maxyv = 2, tmp = 0;
    int16 minxv = 0, midxv = 1, maxxv = 2;

    if ((tp->vertices[minyv]).vertex[Y] > (tp->vertices[midyv]).vertex[Y])
        TSWAP (minyv, midyv, tmp);

    if ((tp->vertices[minyv]).vertex[Y] > (tp->vertices[maxyv]).vertex[Y])
        TSWAP (minyv, maxyv, tmp);

    if ((tp->vertices[midyv]).vertex[Y] > (tp->vertices[maxyv]).vertex[Y])
        TSWAP (midyv, maxyv, tmp);

    if ((tp->vertices[minxv]).vertex[X] > (tp->vertices[midxv]).vertex[X])
        TSWAP (minxv, midxv, tmp);

    if ((tp->vertices[minxv]).vertex[X] > (tp->vertices[maxxv]).vertex[X])
        TSWAP (minxv, maxxv, tmp);

    if ((tp->vertices[midxv]).vertex[X] > (tp->vertices[maxxv]).vertex[X])
        TSWAP (midxv, maxxv, tmp);
```

```
if ((ZFABS((tp->vertices[maxxv]).vertex[X] - (tp->vertices[minxv]).vertex[X])
    > MAT3_EPSILON) &&
    (ZFABS((tp->vertices[maxyv]).vertex[Y] - (tp->vertices[minyv]).vertex[Y])
    > MAT3_EPSILON))
    rasterize_sorted_triangle (minyv, midyv, maxyv, tp);
}

/* ----- */

/* Clipping */

int16 ClassifyVertex (MAT3fvec v, int16 plane, int16 val)
{
    switch (plane)
    {
        case X : switch (val)
        {
            case 1 : return ( (v[W] >= 0.0) ? (v[X] > v[W]) : (v[X] < v[W]) );
            case -1 : return ( (v[W] >= 0.0) ? (v[X] < -v[W]) : (v[X] > -v[W]) );
        }
        case Y : switch (val)
        {
            case 1 : return ( (v[W] >= 0.0) ? (v[Y] > v[W]) : (v[Y] < v[W]) );
            case -1 : return ( (v[W] >= 0.0) ? (v[Y] < -v[W]) : (v[Y] > -v[W]) );
        }
        case Z : switch (val)
        {
            case 1 : return ( (v[W] >= 0.0) ? (v[Z] > v[W]) : (v[Z] < v[W]) );
            case 0 : return ( (v[W] >= 0.0) ? (v[Z] < 0.0) : (v[Z] > 0.0) );
        }
    }
    return (0);
}

int16 ClassifyTriangle (ColorTriangleP tp, int16 plane, int16 val)
{
    return (
        (ClassifyVertex (tp->vertices[0].vertex, plane, val) << 2) |
        (ClassifyVertex (tp->vertices[1].vertex, plane, val) << 1) |
        (ClassifyVertex (tp->vertices[2].vertex, plane, val))
    );
}

void CopyVertex (Color_VertexP source, Color_VertexP dest)
{
    int i;

    for (i=0; i < 4; i++)
        dest->vertex[i] = source->vertex[i];
    dest->red = source->red;
    dest->green = source->green;
    dest->blue = source->blue;
}

void CopyNormal (ColorTriangleP source, ColorTriangleP dest)
{
    int i;
```



```
for (i=0; i < 3; i++)
    dest->normal[i] = source->normal[i];
}

void InterpolateVertex (Color_VertexP start, Color_VertexP end, float t,
                       Color_VertexP dest)
{
    int i;

    for (i=0; i < 4; i++)
        dest->vertex[i] = start->vertex[i] + t*(end->vertex[i] - start->vertex[i]);

    dest->red = start->red + t * (end->red - start->red);
    dest->green = start->green + t * (end->green - start->green);
    dest->blue = start->blue + t * (end->blue - start->blue);
}

float ComputeT (MAT3fvec vstart, MAT3fvec vend, int16 plane, int16 val)
{
    switch (plane)
    {
        case X:
            case Y: switch (val)
                {
                    case 1 : return ((vstart[plane] - vstart[W]) /
                                     (vend[W] - vstart[W] - vend[plane] + vstart[plane]));
                    case -1: return ((vstart[plane] + vstart[W]) /
                                     (vstart[plane] - vend[plane] + vstart[W] - vend[W]));
                }
            case Z: switch(val)
                {
                    case 1 : return ((vstart[plane] - vstart[W]) /
                                     (vend[W] - vstart[W] - vend[plane] + vstart[plane]));
                    case 0 : return (vstart[plane] / (vstart[plane] - vend[plane] ));
                }
    }
    return (0.0);
}

void Clip_Process_Rasterize_Triangle(ColorTriangleP tp)
{
    ColorTriangle cur_t[7];
    int16 k, plane, val, l, maxtriangles, curmax, clipcode, inout[7], j;
    float t1, t2;

    CopyNormal(tp, &(cur_t[0]));

    for (k=0; k < 3; k++)
        CopyVertex ((Color_VertexP) (&(tp->vertices[k])),
                    (Color_VertexP) (&(cur_t[0].vertices[k])));

    curmax = maxtriangles = 1;
    inout[0] = 0;

    for (plane = 0; plane < 3; plane++) {
        for (k = -1; k < 2; k += 2) {
            val = k;
            if ((plane == 2) && (val == -1)) val = 0;
            for (l = 0; l < maxtriangles; l++) {
                if (inout[l] == 0) {
                    clipcode = ClassifyTriangle ((ColorTriangleP) (&cur_t[l]), plane, val);
```

```
switch (clipcode) {
  case 0: break;
  case 1: t1 = ComputeT (cur_t[l].vertices[2].vertex,
                        cur_t[l].vertices[0].vertex, plane, val);
        t2 = ComputeT (cur_t[l].vertices[1].vertex,
                        cur_t[l].vertices[2].vertex, plane, val);
        CopyVertex((Color_VertexP) (&(cur_t[l].vertices[0])),
                   (Color_VertexP) (&(cur_t[curmax].vertices[0])));
        InterpolateVertex((Color_VertexP) (&(cur_t[l].vertices[2])),
                          (Color_VertexP) (&(cur_t[l].vertices[0])), t1,
                          (Color_VertexP) (&(cur_t[curmax].vertices[2])));
        InterpolateVertex((Color_VertexP) (&(cur_t[l].vertices[1])),
                          (Color_VertexP) (&(cur_t[l].vertices[2])), t2,
                          (Color_VertexP) (&(cur_t[curmax].vertices[1])));
        CopyVertex((Color_VertexP) (&(cur_t[curmax].vertices[1])),
                   (Color_VertexP) (&(cur_t[l].vertices[2])));
        CopyNormal(&(cur_t[l]), &(cur_t[curmax]));
        inout[curmax] = 0;
        curmax++;
        break;
  case 2: t1 = ComputeT (cur_t[l].vertices[0].vertex,
                        cur_t[l].vertices[1].vertex, plane, val);
        t2 = ComputeT (cur_t[l].vertices[1].vertex,
                        cur_t[l].vertices[2].vertex, plane, val);
        CopyVertex((Color_VertexP) (&(cur_t[l].vertices[0])),
                   (Color_VertexP) (&(cur_t[curmax].vertices[0])));
        InterpolateVertex((Color_VertexP) (&(cur_t[l].vertices[0])),
                          (Color_VertexP) (&(cur_t[l].vertices[1])), t1,
                          (Color_VertexP) (&(cur_t[curmax].vertices[1])));
        InterpolateVertex((Color_VertexP) (&(cur_t[l].vertices[1])),
                          (Color_VertexP) (&(cur_t[l].vertices[2])), t2,
                          (Color_VertexP) (&(cur_t[curmax].vertices[2])));
        CopyVertex((Color_VertexP) (&(cur_t[curmax].vertices[2])),
                   (Color_VertexP) (&(cur_t[l].vertices[1])));
        CopyNormal(&(cur_t[l]), &(cur_t[curmax]));
        inout[curmax] = 0;
        curmax++;
        break;
  case 3: t1 = ComputeT (cur_t[l].vertices[0].vertex,
                        cur_t[l].vertices[1].vertex, plane, val);
        t2 = ComputeT (cur_t[l].vertices[2].vertex,
                        cur_t[l].vertices[0].vertex, plane, val);
        InterpolateVertex((Color_VertexP) (&(cur_t[l].vertices[0])),
                          (Color_VertexP) (&(cur_t[l].vertices[1])), t1,
                          (Color_VertexP) (&(cur_t[l].vertices[1])));
        InterpolateVertex((Color_VertexP) (&(cur_t[l].vertices[2])),
                          (Color_VertexP) (&(cur_t[l].vertices[0])), t2,
                          (Color_VertexP) (&(cur_t[l].vertices[2])));
        break;
  case 4: t1 = ComputeT (cur_t[l].vertices[2].vertex,
                        cur_t[l].vertices[0].vertex, plane, val);
        t2 = ComputeT (cur_t[l].vertices[0].vertex,
                        cur_t[l].vertices[1].vertex, plane, val);
        CopyVertex((Color_VertexP) (&(cur_t[l].vertices[1])),
                   (Color_VertexP) (&(cur_t[curmax].vertices[0])));
        InterpolateVertex((Color_VertexP) (&(cur_t[l].vertices[2])),
                          (Color_VertexP) (&(cur_t[l].vertices[0])), t1,
                          (Color_VertexP) (&(cur_t[curmax].vertices[1])));
        InterpolateVertex((Color_VertexP) (&(cur_t[l].vertices[0])),
                          (Color_VertexP) (&(cur_t[l].vertices[1])), t2,
                          (Color_VertexP) (&(cur_t[curmax].vertices[2])));
```

```
        CopyVertex((Color_VertexP) (&(cur_t[curmax].vertices[1])),
                  (Color_VertexP) (&(cur_t[1].vertices[0])));
        CopyNormal(&(cur_t[1]), &(cur_t[curmax]));
        inout[curmax] = 0;
        curmax++;
        break;
    case 5: t1 = ComputeT (cur_t[1].vertices[0].vertex,
                        cur_t[1].vertices[1].vertex, plane, val);
        t2 = ComputeT (cur_t[1].vertices[1].vertex,
                        cur_t[1].vertices[2].vertex, plane, val);
        InterpolateVertex((Color_VertexP) (&(cur_t[1].vertices[0])),
                          (Color_VertexP) (&(cur_t[1].vertices[1])), t1,
                          (Color_VertexP) (&(cur_t[1].vertices[0])));
        InterpolateVertex((Color_VertexP) (&(cur_t[1].vertices[1])),
                          (Color_VertexP) (&(cur_t[1].vertices[2])), t2,
                          (Color_VertexP) (&(cur_t[1].vertices[2])));
        break;
    case 6: t1 = ComputeT (cur_t[1].vertices[2].vertex,
                        cur_t[1].vertices[0].vertex, plane, val);
        t2 = ComputeT (cur_t[1].vertices[1].vertex,
                        cur_t[1].vertices[2].vertex, plane, val);
        InterpolateVertex((Color_VertexP) (&(cur_t[1].vertices[2])),
                          (Color_VertexP) (&(cur_t[1].vertices[0])), t1,
                          (Color_VertexP) (&(cur_t[1].vertices[0])));
        InterpolateVertex((Color_VertexP) (&(cur_t[1].vertices[1])),
                          (Color_VertexP) (&(cur_t[1].vertices[2])), t2,
                          (Color_VertexP) (&(cur_t[1].vertices[1])));
        break;
    case 7: inout[1] = 1;
        break;
}
}
}
maxtriangles = curmax;
}
}
for (k=0; k < maxtriangles; k++)
    if (inout[k] == 0) {
        for (j=0; j < 3; j++) {
            cur_t[k].vertices[j].vertex[X] = (cur_t[k].vertices[j].vertex[X] +
                                                cur_t[k].vertices[j].vertex[W]) * (WW - 1)/2.0;
            cur_t[k].vertices[j].vertex[Y] = (cur_t[k].vertices[j].vertex[Y] +
                                                cur_t[k].vertices[j].vertex[W]) * (WH - 1)/2.0;
        }
        for (j=0; j < 3; j++) {
            cur_t[k].vertices[j].vertex[X] = cur_t[k].vertices[j].vertex[X] /
                                                cur_t[k].vertices[j].vertex[W];
            cur_t[k].vertices[j].vertex[Y] = cur_t[k].vertices[j].vertex[Y] /
                                                cur_t[k].vertices[j].vertex[W];
            cur_t[k].vertices[j].vertex[Z] = cur_t[k].vertices[j].vertex[Z] /
                                                cur_t[k].vertices[j].vertex[W];
            cur_t[k].vertices[j].vertex[X] = ZFABS(cur_t[k].vertices[j].vertex[X]);
            cur_t[k].vertices[j].vertex[Y] = ZFABS(cur_t[k].vertices[j].vertex[Y]);
            cur_t[k].vertices[j].vertex[Z] = ZFABS(cur_t[k].vertices[j].vertex[Z]);
        }

        sort_and_rasterize_triangle ((ColorTriangleP) (&(cur_t[k])));
    }
}
```

```
/* ----- */

/* Viewing Transformation */

#define mathpointmult(x,y,w,vec,mat) {\
    (x) = (vec)[0] * (mat)[0][0] + \
          (vec)[1] * (mat)[0][1] + \
          (vec)[2] * (mat)[0][2] + \
          (mat)[0][3]; \
    \
    (y) = (vec)[0] * (mat)[1][0] + \
          (vec)[1] * (mat)[1][1] + \
          (vec)[2] * (mat)[1][2] + \
          (mat)[1][3]; \
    \
    (w) = (vec)[0] * (mat)[3][0] + \
          (vec)[1] * (mat)[3][1] + \
          (vec)[2] * (mat)[3][2] + \
          (mat)[3][3]; \
} /* mathpointmult */

void EvaluateViewTransformationMatrix (MAT3vec view_ref_point,
                                       MAT3vec view_plane_normal,
                                       MAT3vec view_up_vector,
                                       double kay, double ell,
                                       double Hither, double Yon,
                                       MAT3fmat vo_inverse,
                                       MAT3mat transform, float *b)
{
    MAT3vec    uvec, vvec;
    MAT3mat    trans, rot, vo_matrix, transinv, rotinv, vo_inverse_d, vm_matrix;
    double     det, zmin, tmp1;
    int16      i, j;

    MAT3_NORMALIZE_VEC(view_plane_normal, tmp1);
    MAT3_CROSS_PRODUCT(uvec, view_up_vector, view_plane_normal);
    MAT3_NORMALIZE_VEC(uvec, tmp1);
    MAT3_CROSS_PRODUCT(vvec, view_plane_normal, uvec);

    MAT3identity (trans);
    for (i=0; i < 3; i++)
        trans[i][3] = -view_ref_point[i];

    MAT3identity (rot);

    for(i=0; i < 3; i++)
        rot[0][i] = uvec[i];
    for(i=0; i < 3; i++)
        rot[1][i] = vvec[i];
    for(i=0; i < 3; i++)
        rot[2][i] = view_plane_normal[i];

    MAT3mult (vo_matrix, rot, trans);
    MAT3identity (transinv);
    for (i=0; i < 3; i++)
        transinv[i][3] = view_ref_point[i];

    MAT3identity (rotinv);
    det = rot[0][0] * Det2D(rot[1][1],rot[1][2],rot[2][1],rot[2][2])
        -rot[0][1] * Det2D(rot[1][0],rot[1][2],rot[2][0],rot[2][2])
```

```
+rot[0][2] * Det2D(rot[1][0],rot[1][1],rot[2][0],rot[2][1]);

rotinv[0][0] = Det2D(rot[1][1],rot[1][2],rot[2][1],rot[2][2])/det;
rotinv[1][0] = -Det2D(rot[1][0],rot[1][2],rot[2][0],rot[2][2])/det;
rotinv[2][0] = Det2D(rot[1][0],rot[1][1],rot[2][0],rot[2][1])/det;

rotinv[0][1] = -Det2D(rot[0][1],rot[0][2],rot[2][1],rot[2][2])/det;
rotinv[1][1] = Det2D(rot[0][0],rot[0][2],rot[2][0],rot[2][2])/det;
rotinv[2][1] = -Det2D(rot[0][0],rot[0][1],rot[2][0],rot[2][1])/det;

rotinv[0][2] = Det2D(rot[0][1],rot[0][2],rot[1][1],rot[1][2])/det;
rotinv[1][2] = -Det2D(rot[0][0],rot[0][2],rot[1][0],rot[1][2])/det;
rotinv[2][2] = Det2D(rot[0][0],rot[0][1],rot[1][0],rot[1][1])/det;

MAT3mult(vo_inverse_d,transinv,rotinv);

for (i=0; i < 4; i++)
    for (j=0; j < 4; j++)
        vo_inverse[i][j] = vo_inverse_d[i][j];

MAT3identity (vm_matrix);
vm_matrix[0][0] = vm_matrix[1][1] =
    ell/ (kay * Yon);
vm_matrix[2][2] = 1/Yon;
vm_matrix[2][3] = -ell/Yon;

zmin = - Hither/Yon;
MAT3mult (transform, vm_matrix, vo_matrix);
for (j=0; j < 4; j++)
    transform[3][j] = -transform[2][j];
*b = -zmin/(1.0 + zmin);

for (j=0; j < 3; j++)
    transform[2][j] *= (1.0+*b);

transform[2][3] = (1.0 + *b) * transform[2][3] + *b;
}

/* ----- */

/* Trivial Accept and Reject */

void MAT3fmult_hvec(MAT3fvec result_vec, float a, float b, float c,
                   MAT3fmat mat)
{
    result_vec[0] = a*mat[0][0] + b*mat[0][1] + c*mat[0][2] + mat[0][3];
    result_vec[1] = a*mat[1][0] + b*mat[1][1] + c*mat[1][2] + mat[1][3];
    result_vec[2] = a*mat[2][0] + b*mat[2][1] + c*mat[2][2] + mat[2][3];
    result_vec[3] = 1.0;
}

void set_bounds(MAT3fvec curpt, float xval, float yval, float zval,
               MAT3fmat vo_inverse)
{
    MAT3fmult_hvec(curpt,xval,yval,zval,vo_inverse);
    out_xmin = out_xmax = curpt[X];
    out_ymin = out_ymax = curpt[Y];
    out_zmin = out_zmax = curpt[Z];
}
```

```
void update_bounds(MAT3fvec curpt, float xval, float yval, float zval,
                  MAT3fmat vo_inverse)
{
    MAT3fmult_hvec(curpt,xval,yval,zval,vo_inverse);
    out_xmin = min(curpt[X],out_xmin);
    out_xmax = max(curpt[X],out_xmax);
    out_ymin = min(curpt[Y],out_ymin);
    out_ymax = max(curpt[Y],out_ymax);
    out_zmin = min(curpt[Z],out_zmin);
    out_zmax = max(curpt[Z],out_zmax);
}

/* ----- */

/* Reading Input */

void readNFFFFile (char *filename, MAT3fvec from, float *frustumf,
                  float *frustumr, MAT3vec vrp, MAT3vec vpn, MAT3vec vupv,
                  double *ellp, double *Hitherp, double *Yonp, double *kayp,
                  Lights lights, BackGroundColor *backgndcolor, int16 *datasizep,
                  MtlProp *activeProp, int16 *currentlight, SpecLightP spec_lightp)
{
    FILE *infile;
    char string[80];
    int16 i, count, state = WAITING, currentTriangle=-1;
    double angle, delta[3], lx, ly, lz;

    *currentlight = 0;
    infile = fopen (filename,"r");
    while (fgets (&string[0],80,infile) != NULL)
    {
        if (state == WAITING)
        {
            switch (string[0])
            {
                case 'v':
                    state = VIEWING; /* Read View Specification */
                    break;
                case 'b': /* Background Color */
                    sscanf (&string[1]," %lf %lf %lf\n", &lx, &ly, &lz);
                    backgndcolor->red    = MAX_INTENSITY * lx;
                    backgndcolor->green  = MAX_INTENSITY * ly;
                    backgndcolor->blue   = MAX_INTENSITY * lz;
                    break;

                case 'l': /* Light Source */
                    sscanf (&string[1], " %f %f %f %f %f %f",
                        &lights[*currentlight].location[0],
                        &lights[*currentlight].location[1],
                        &lights[*currentlight].location[2],
                        &lights[*currentlight].red,
                        &lights[*currentlight].green,
                        &lights[*currentlight].blue);
                    (*currentlight)++;
                    break;
                case 's': /* PHONG Light Source */
                    sscanf (&string[1], " %f %f %f %f %f %f %f %d",
```

```
        &spec_lightp->location[0],
        &spec_lightp->location[1],
        &spec_lightp->location[2],
        &spec_lightp->red,
        &spec_lightp->green,
        &spec_lightp->blue,
        &spec_lightp->ks,
        &spec_lightp->spec_exp);
spec_lightp->location[3] = 1.0;
break;

case 'f': /* Lighting Constants */
    sscanf (&string[1], " %f %f %f %f %f %f %f",
            &(*activeProp).red,
            &(*activeProp).green,
            &(*activeProp).blue,
            &(*activeProp).diffuseK,
            &(*activeProp).ambientK,
            &(*activeProp).c1,
            &(*activeProp).c2);
    break;

case 'p': /* Triangle to Follow */
    state = TRIDATA;
    count = 0;
    currentTriangle++;
    break;
} /* switch on first char of line */
} /* if in waiting state */
else if (state == VIEWING)
{
    switch(string[0])
    {
        case 'f': /* View point location */
            sscanf (&string[4], " %f %f %f\n", &from[0], &from[1], &from[2]);
            from[W] = 1.0;
            break;

        case 'a':
            if (string[1] == 't') /* Look at */
                sscanf(&string[2], " %lf %lf %lf\n", &vrp[0], &vrp[1], &vrp[2]);
            else if (string[1] == 'n') /* Angle */
                sscanf(&string[5], "%lf\n", &angle);
            break;

        case 'u': /* Up Vector */
            sscanf (&string[2], " %lf %lf %lf\n", &vupv[0], &vupv[1], &vupv[2]);
            break;

        case 'h': /* Front or Hither Clipping Plane */
            sscanf (&string[6], " %lf\n", Hitherp);
            break;

        case 'y': /* Back or Yon Clipping Plane */
            sscanf (&string[3], " %lf\n", Yonp);
            break;

        case 'r': /* resolution is currently ignored */
            for (i=0; i < 3; i++)
                delta[i] = from[i] - vrp[i];
            *ellp = sqrt(delta[0]*delta[0]+delta[1]*delta[1]+delta[2]*delta[2]);
    }
}
```

```
    for (i=0; i < 3; i++)
        vpn[i] = (from[i] - vrp[i]) / (*ellp);
    *kayp = (*ellp) * tan(angle*M_PI/360.0);
    *frustumf = (*kayp) * (*Hitherp) / (*ellp);
    *frustumr = (*kayp) * (*Yonp) / (*ellp);
    state = WAITING;
    break;
} /* switch first char in line */
} /* else if in viewing state */
else if (state == TRIDATA)
{
    sscanf (string, "%f%f%f%f%f\n",
            &(localSet[currentTriangle].vertices[count].vertex[0]),
            &(localSet[currentTriangle].vertices[count].vertex[1]),
            &(localSet[currentTriangle].vertices[count].vertex[2]),
            &(localSet[currentTriangle].vertices[count].normal[0]),
            &(localSet[currentTriangle].vertices[count].normal[1]),
            &(localSet[currentTriangle].vertices[count].normal[2]));
    if (count == 2)
        state = WAITING;
    else
        count++;
} /* else if just read in triangle data */
} /* while not eof */
fclose(infile);
*datasizep = currentTriangle + 1;
} /* readNFFFFile */

/* ----- */

/* Backface Culling, Trivial Accept and Reject, Lighting,
   Viewing Transformation and Clip Check */

/*
 * Processes a triangle. It also orients the triangle edges so that
 * y(vertex 0) <= y(vertex 1) <= y(vertex 2).
 */

void ProcessTriangle (OrigTriangleP tp, MAT3fvec from,
                     MAT3fmat transform, float b,
                     float Iar, float Iag, float Iab,
                     float c1, float c2, int16 num_lights, Lights lights)
{
    ColorTriangle cur_t;
    int16 j, k;
    MAT3fvec temp1;
    float dist, tmp3;

    if (BFCULL)
        if ((from[X]*tp->normal[X]+from[Y]*tp->normal[Y]+from[Z]*tp->normal[Z])
            < tp->v0dotn)
            return;

    if (TRIVIAL_REJECT) {
        if ((tp->vertices[0].vertex[X] < out_xmin) &&
            (tp->vertices[1].vertex[X] < out_xmin) &&
            (tp->vertices[2].vertex[X] < out_xmin)) {
            return;
        }
    }
}
```



```
if ((tp->vertices[0].vertex[X] > out_xmax) &&
    (tp->vertices[1].vertex[X] > out_xmax) &&
    (tp->vertices[2].vertex[X] > out_xmax)) {
    return;
}
if ((tp->vertices[0].vertex[Y] < out_ymin) &&
    (tp->vertices[1].vertex[Y] < out_ymin) &&
    (tp->vertices[2].vertex[Y] < out_ymin)) {
    return;
}
if ((tp->vertices[0].vertex[Y] > out_ymax) &&
    (tp->vertices[1].vertex[Y] > out_ymax) &&
    (tp->vertices[2].vertex[Y] > out_ymax)) {
    return;
}
if ((tp->vertices[0].vertex[Z] < out_zmin) &&
    (tp->vertices[1].vertex[Z] < out_zmin) &&
    (tp->vertices[2].vertex[Z] < out_zmin)) {
    return;
}
if ((tp->vertices[0].vertex[Z] > out_zmax) &&
    (tp->vertices[1].vertex[Z] > out_zmax) &&
    (tp->vertices[2].vertex[Z] > out_zmax)) {
    return;
}
}

for (j=0; j < 3; j++)
    cur_t.normal[j] = tp->normal[j];

for (j = 0; j < 3; j++) {
    cur_t.vertices[j].red = Iar;
    cur_t.vertices[j].green = Iag;
    cur_t.vertices[j].blue = Iab;
    for (k = 0; k < num_lights; k++) {
        MAT3_SUB_VEC(temp1,lights[k].location,tp->vertices[j].vertex);

        dist = sqrt (temp1[X]*temp1[X] + temp1[Y]*temp1[Y] + temp1[Z]*temp1[Z]);
        dist = 1 / (dist * (1 + dist * (c1 + c2*dist)));
        tmp3 = dist * ZFABS(MAT3_DOT_PRODUCT(temp1,tp->vertices[j].normal));
        cur_t.vertices[j].red += lights[k].red * tmp3;
        cur_t.vertices[j].green += lights[k].green * tmp3;
        cur_t.vertices[j].blue += lights[k].blue * tmp3;
    } /* for k */
} /* for j */
{
float x[3], y[3], z[3], w[3];

for( j = 0; j < 3; j++) {
    mathpointmult (x[j],y[j],w[j],tp->vertices[j].vertex,transform)
    z[j] = b - w[j]*b;
} /* for j */

if ((ZFABS(x[0]) <= ZFABS(w[0])) && (ZFABS(y[0]) <= ZFABS(w[0])) &&
    (ZFABS(z[0])<=ZFABS(w[0])) && ((w[0] >= 0.0)?(z[0] >= 0.0):(z[0]<0.0)) &&
    (ZFABS(x[1]) <= ZFABS(w[1])) && (ZFABS(y[1]) <= ZFABS(w[1])) &&
    (ZFABS(z[1])<=ZFABS(w[1])) && ((w[1] >= 0.0)?(z[1] >= 0.0):(z[1]<0.0)) &&
    (ZFABS(x[2]) <= ZFABS(w[2])) && (ZFABS(y[2]) <= ZFABS(w[2])) &&
    (ZFABS(z[2])<=ZFABS(w[2])) && ((w[2] >= 0.0)?(z[2] >= 0.0):(z[2]<0.0))
    ) {
    for (j=0; j < 3; j++) {
```

```
    x[j] = (x[j] + w[j]) * (WW - 1)/2.0;
    y[j] = (y[j] + w[j]) * (WH - 1)/2.0;
}
for (j=0; j < 3; j++) {
    cur_t.vertices[j].vertex[X] = x[j]/w[j];
    cur_t.vertices[j].vertex[Y] = y[j]/w[j];
    cur_t.vertices[j].vertex[Z] = z[j]/w[j];
}
for (j=0; j < 3; j++) {
    cur_t.vertices[j].vertex[X] = ZFABS(cur_t.vertices[j].vertex[X]);
    cur_t.vertices[j].vertex[Y] = ZFABS(cur_t.vertices[j].vertex[Y]);
    cur_t.vertices[j].vertex[Z] = ZFABS(cur_t.vertices[j].vertex[Z]);
}
sort_and_rasterize_triangle ((ColorTriangleP) (&cur_t));
}
else if (CLIP) {
    for (j=0; j < 3; j++) {
        cur_t.vertices[j].vertex[X] = x[j];
        cur_t.vertices[j].vertex[Y] = y[j];
        cur_t.vertices[j].vertex[Z] = z[j];
        cur_t.vertices[j].vertex[W] = w[j];
    }
    Clip_Process_Rasterize_Triangle((ColorTriangleP) (&cur_t));
}
}
}
} /* ProcessTriangle */

/*
 * The function processes a set of triangles one at a time.
 */

void PipelineCompute (int16 count, MAT3fvec from,
                     MAT3fmat transform, float b,
                     float Iar, float Iag, float Iab, float c1, float c2,
                     int16 num_lights, Lights lights)
{
    int16 i;

    for (i = 0; i < count; i++)
        ProcessTriangle (&localSet[i], from, transform, b,
                        Iar, Iag, Iab, c1, c2, num_lights, lights);
} /* Pipeline Compute */

/* ----- */

void main(int argc, char **argv)
{
    int16 i, j, tmp;
    int16 datasize, num_lights, scale_factor;
    MAT3vec vrp, vpn, vupv;
    MAT3fvec from, temp1, temp2, curpt, Lvector, Vvector, SpecLocNPC;
    double ell, Hither, Yon, kay;
    MAT3mat transform;
    float mag, b, frustumf, frustumr, ffplane, fbplane;
    MAT3fmat vo_inverse, ftransform;
    MtlProp activeProp;
    BackGroundColor backgndcolor;
    color fbackgndcolor;
```

```
LightPoint lights[NUM_LIGHT_SOURCES];

if (argc < 7) {
    printf ("Usage: ZRndv10 <bf> <tr> <clip> <phong> <datafile> <outputfile>\n");
    return;
}
if (strcmp(argv[1],"-bf"))
    BFCULL = 0;
else
    BFCULL = 1;

if (strcmp(argv[2],"-tr"))
    TRIVIAL_REJECT = 0;
else
    TRIVIAL_REJECT = 1;

if (strcmp(argv[3],"-clip"))
    CLIP = 0;
else
    CLIP = 1;
if (strcmp(argv[4],"-phong"))
    PHONG = 0;
else
    PHONG = 1;

readNFFFFile (argv[5], from, &frustumf, &frustumr, vrp, vpn, vupv,
               &ell, &Hither, &Yon, &kay, lights, &backgndcolor,
               &datasize, &activeProp, &num_lights, &SpecSource);

for (i=0; i < datasize; i++) {
    MAT3_SUB_VEC(temp1,localSet[i].vertices[1].vertex,
                 localSet[i].vertices[0].vertex);
    MAT3_SUB_VEC(temp2,localSet[i].vertices[2].vertex,
                 localSet[i].vertices[0].vertex);
    MAT3_CROSS_PRODUCT(localSet[i].normal,temp1,temp2);
    MAT3_NORMALIZE_VEC(localSet[i].normal,mag);

    localSet[i].v0dotn=localSet[i].vertices[0].vertex[X]*localSet[i].normal[X]+
        localSet[i].vertices[0].vertex[Y]*localSet[i].normal[Y]+
        localSet[i].vertices[0].vertex[Z]*localSet[i].normal[Z];
}

if (PHONG)
    scale_factor = num_lights + 2;
else
    scale_factor = num_lights + 1;

activeProp.red = activeProp.red * activeProp.ambientK * MAX_INTENSITY /
    scale_factor;
activeProp.green = activeProp.green * activeProp.ambientK * MAX_INTENSITY /
    scale_factor;
activeProp.blue = activeProp.blue * activeProp.ambientK * MAX_INTENSITY /
    scale_factor;

for (j=0; j < num_lights; j++) {
    lights[j].red *= activeProp.diffuseK * MAX_INTENSITY / scale_factor;
    lights[j].green *= activeProp.diffuseK * MAX_INTENSITY / scale_factor;
    lights[j].blue *= activeProp.diffuseK * MAX_INTENSITY / scale_factor;
}

fbackgndcolor = ((int32) backgndcolor.red) +
```

```
((int32) backgndcolor.green) << 8) + (((int32) backgndcolor.blue) << 16);

EvaluateViewTransformationMatrix (vrp, vpn, vupv, kay, ell, Hither, Yon,
                                  vo_inverse, transform, &b);

for (i=0; i < 4; i++)
    for (j=0; j < 4; j++)
        ftransform[i][j] = transform[i][j];

if (PHONG) {
    SpecSource.red *= SpecSource.ks * MAX_INTENSITY/scale_factor;
    SpecSource.green *= SpecSource.ks * MAX_INTENSITY/scale_factor;
    SpecSource.blue *= SpecSource.ks * MAX_INTENSITY/scale_factor;
    mathpointmult (SpecLocNPC[X],SpecLocNPC[Y],SpecLocNPC[W],
                  SpecSource.location,ftransform)
    SpecLocNPC[Z] = b - SpecLocNPC[W]*b;

    SpecLocNPC[X] = (SpecLocNPC[X] + SpecLocNPC[W]);
    SpecLocNPC[Y] = (SpecLocNPC[Y] + SpecLocNPC[W]);

    for (i=0; i < 3; i++)
        SpecLocNPC[i] /= SpecLocNPC[W];

    MAT3_SUB_VEC(Lvector,SpecSource.location,vrp);
    MAT3_SUB_VEC(Vvector,from,vrp);
    for(i=0; i < 3; i++)
        Hvector[i] = Lvector[i] + Vvector[i];
    MAT3_NORMALIZE_VEC(Hvector,mag);
}

ffplane = ell - Hither; fbplane = ell - Yon;
set_bounds (curpt,frustumf,frustumf,ffplane,vo_inverse);
update_bounds (curpt,-frustumf,-frustumf,ffplane,vo_inverse);
update_bounds (curpt,-frustumf,frustumf,ffplane,vo_inverse);
update_bounds (curpt,frustumf,-frustumf,ffplane,vo_inverse);
update_bounds (curpt,frustumr,frustumr,fbplane,vo_inverse);
update_bounds (curpt,-frustumr,-frustumr,fbplane,vo_inverse);
update_bounds (curpt,-frustumr,frustumr,fbplane,vo_inverse);
update_bounds (curpt,frustumr,-frustumr,fbplane,vo_inverse);

for (i=0; i < WH; i++)
    for (j=0; j < WW; j++)
        fb[i][j] = fbackgndcolor;

for (i=0; i < WH; i++)
    for (j=0; j < WW; j++)
        zb[i][j] = 0;

PipelineCompute (datasize, from, ftransform, b,
                activeProp.red, activeProp.green, activeProp.blue,
                activeProp.cl, activeProp.c2, num_lights, lights);

write_tga_buffer (fb, argv[6]);
}
```

```
/*
    Z Buffer Rendering Program Version 10, 02/12/96
    Copyright Raghu Karinithi, West Virginia University
*/

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>
#include "../tga/lug.h"
#include "../tga/lugfnts.h"

#define MAT3_EPSILON    1e-12                /* Close enough to zero */
#define MAT3_PI        M_PI                 /* Pi */

typedef double MAT3mat[4][4];               /* 4x4 matrix */
typedef double MAT3vec[3];                  /* Vector */

#define HIBITS          15
#define LOBITS          14
#define ZLOBITS         22

#define FLOATSZM1 31
#define FFRACSZ 23
#define FEXPOFFSET 127
#define FFRACMASK 0x007FFFFFFF
#define EXPMASK 0x7F800000
#define FFRACSZMLOBITS (FFRACSZ - LOBITS)

#ifndef TRUE
#define TRUE          1
#endif

#ifndef FALSE
#define FALSE         0
#endif

#define LOMASK        (~(0xffffffff << LOBITS))
#define HIMASK        (~(0xffffffff << HIBITS)) << LOBITS
#define FIX1          (1 << LOBITS)

#define RMASK 0x000000FF
#define GMASK 0x0000FF00
#define BMASK 0x00FF0000

#define WAITING        0
#define VIEWING        1
#define TRIDATA        2

#define NUM_TRIANGLES  10000
#define X              0
#define Y              1
#define Z              2
#define W              3

#define WW            500
#define WH            480

#define MAX_INTENSITY  256
#define NUM_LIGHT_SOURCES 4
#define COLOR_DEPTH    24
```

```
typedef long int32;

typedef int int16;

typedef int32 fixpoint;

typedef int32 color;  /* Framebuffer Color */

typedef float MAT3fvec[4];

typedef float MAT3fmat[4][4];

typedef struct Normal_Vertex_struct {
    MAT3fvec vertex;
    MAT3fvec normal;
} Normal_Vertex;

typedef struct Color_Vertex_struct {
    MAT3fvec vertex;
    float red, green, blue;
} Color_Vertex;

typedef Color_Vertex *Color_VertexP;

typedef color FrameBuffer[WH][WW];

typedef fixpoint ZBuffer[WH][WW];

typedef struct _origtriangle {
    Normal_Vertex vertices[3];
    MAT3fvec normal;
    float v0dotn;
} OrigTriangle;

typedef OrigTriangle *OrigTriangleP;

typedef struct color_triangle_struct {
    MAT3fvec normal;
    Color_Vertex vertices[3];
} ColorTriangle;

typedef ColorTriangle *ColorTriangleP;

typedef struct BackgroundColor_struct {
    unsigned char red, green, blue;
} BackGroundColor;

typedef struct LightPoint_struct {
    MAT3fvec location;
    float red, green, blue;
} LightPoint;

typedef LightPoint *Lights;

typedef struct MtlProperties_struct {
    float red, green, blue, diffuseK, ambientK, c1, c2;
} MtlProp;

/*
 * The following edge data structure is modified from the code
```

```
*   obtained in Graphics Gems III subdirectory accurate_scan due to Kurt
*   Fleischer.  Also Refer Lathrop, Kirk, Voorhies, IEEE CG&A 10(5),
*   1990 for a discussion.
*/
```

```
typedef struct edge_struct {
    fixpoint E, DEA, DEB;
    short Ix, AStep, BStep;} edge;
```

```
typedef struct SpecLightPoint_struct {
    MAT3fvec location;
    float red, green, blue;
    float ks;
    int16 spec_exp;
} SpecLightPoint;
```

```
typedef SpecLightPoint *SpecLightP;
```

This document describes the Z buffer based rendering program developed at the West Virginia University. The key features of this program are:

- (a) Reads a variant of the standard Neutral File Format (NFF) as input
- (b) Outputs a 24 bit color TARGA file
- (c) Uses accurate fixpoint arithmetic in all its calculations
- (d) Works on X windows environment. We have tested it on Sun Sparc machines.
- (e) High Performance. The renderer has almost realtime performance.

This software is available through the Web and via anonymous ftp. The software includes sample files, including the teapot. We would appreciate any feedback you can give us on this software. We are quite impressed by the quality of the images produced. Enjoy the use of this utility!

Raghu Karinithi  
Department of Statistics and Computer Science  
Concurrent Engineering Research Center  
West Virginia University  
Morgantown, WV 26506-6506  
Voice: (304) 293-7226  
Fax: (304) 293-7541  
Email: [raghu@cs.wvu.edu](mailto:raghu@cs.wvu.edu) OR [raghu@cerc.wvu.edu](mailto:raghu@cerc.wvu.edu)

-----  
This describes briefly our implementation of the Z buffer rendering algorithm. The rendering pipeline has the following steps:

1. Reading the input file.
2. Backface Culling
3. Trivial Accept/Reject of Triangles
4. Computing the light intensity at the vertices. Later, Gouraud shading is used for interpolating the light intensity. Several diffuse light sources and one specular light source are supported but only Gouraud shading is supported.
5. Applying the view transformation to all the vertices
6. Clipping
7. Division by W, Map to 3DViewport
8. Rasterization, and writing to framebuffer.
9. Writing framebuffer to a TARGA file.

Subsequently, one can display the TARGA file, by invoking a separate utility (included with this package).

We compute 24 bit color (R,G,B) at each pixel and write it to a TARGA file. The functions to do write in TARGA file format are from the "LUG" library due to Raul Rivero. The program "sx11" to display a TARGA file is also from the same library. We have extracted these two utilities from the LUG library and provided them in the subdirectories "tga" and "sx11".

Input File Format: The input is a modified NFF file format described in the file called NFF.

URL  
[http://www.cerc.wvu.edu/public\\_domain\\_sw.html](http://www.cerc.wvu.edu/public_domain_sw.html)

FTP INSTRUCTIONS



```
ftp ftp.cerc.wvu.edu
```

```
Enter name: anonymous
```

```
Enter Password: type in your email address
```

```
cd pub/sources
```

```
binary
```

```
get ZRendv10.tar.Z
```

```
quit
```

Then, uncompress and untar the files. Example (not guaranteed!):

```
uncompress < ZRendv10.tar.Z | tar xf -
```

Installation: type INSTALL in the main directory.

Usage: ZRendv10 <backface cull> <trivial accept/reject> <clip> <phong>

<NFFfilename> <TARGAfilename>.

<backface cull> must be -bf to enable backface culling, anything else to disable it.

<trivial accept/reject> must be -tr to enable trivial accept/reject of triangles, anything else to disable it.

<clip> must be -clip to enable triangle clipping, anything else to disable it.

<phong> must be -phong to enable the use of the Phong model (the data file must have a Phong source) and anything else to use the diffuse model.

<NFFfilename> is input NFF file name.

<TARGAfilename> is output TARGA file name.

Example:

```
ZRendv10 -bf -tr -clip -phong tpot11.nff tpot11.tga
```




The display program is "sx11" in the "sx11" subdirectory. It is invoked by typing: sx11 <TARGAfilename>. The <TARGAfilename> must have a ".tga" suffix.

Example:

```
sx11 tpot11.tga
```























# Index of

## /pubs/tog/GraphicsGems/gemsv/ch7-6/sx11/

Name	Last modified	Size	Description
 <a href="#">_Parent Directory</a>			
 <a href="#">_Makefile</a>	29-Jun-00 08:25	1K	
 <a href="#">_sx11.c</a>	29-Jun-00 08:25	2K	

# Index of

## /pubs/tog/GraphicsGems/gemsv/ch7-6/tga/

Name	Last modified	Size	Description
 <a href="#">Parent Directory</a>			
 <a href="#">Makefile</a>	29-Jun-00 08:25	1K	
 <a href="#">README</a>	29-Jun-00 08:25	1K	
 <a href="#">alias.h</a>	29-Jun-00 08:25	1K	
 <a href="#">bitmap.c</a>	29-Jun-00 08:25	3K	
 <a href="#">cnv.c</a>	29-Jun-00 08:25	2K	
 <a href="#">dither.c</a>	29-Jun-00 08:25	4K	
 <a href="#">encodgif.c</a>	29-Jun-00 08:25	10K	
 <a href="#">error.c</a>	29-Jun-00 08:25	3K	
 <a href="#">general.c</a>	29-Jun-00 08:25	4K	
 <a href="#">gif.c</a>	29-Jun-00 08:25	12K	
 <a href="#">hsl.c</a>	29-Jun-00 08:25	3K	
 <a href="#">in_out.c</a>	29-Jun-00 08:25	5K	
 <a href="#">lug.h</a>	29-Jun-00 08:25	6K	
 <a href="#">lugconf.h</a>	29-Jun-00 08:25	2K	
 <a href="#">lugfnts.h</a>	29-Jun-00 08:25	20K	
 <a href="#">memory.c</a>	29-Jun-00 08:25	1K	
 <a href="#">rla.h</a>	29-Jun-00 08:25	1K	
 <a href="#">targa.h</a>	29-Jun-00 08:25	1K	
 <a href="#">tga.c</a>	29-Jun-00 08:25	13K	
 <a href="#">tobw.c</a>	29-Jun-00 08:25	5K	
 <a href="#">x11.c</a>	29-Jun-00 08:25	10K	

```
b 0.5 0.5 0.5
v
from 4.86 7.2 5.4
at 0.03 0.03 0.03
up 0 0 1
angle 30
hither 7
yon 11.5
resolution 512 512
l 4.0 2.5 3.5 0.95 0.2 0.1
s 5.0 5.0 5.0 0.05 0.1 0.9 0.85 4
f 0.9 0.9 0.9 1.0 1.0 0.01 0.001 0
pp 3
1.4 0 2.4 -0.902861 -0 -0.429934
0.994 -0.994 2.4 -0.637936 0.637936 -0.431366
0.996219 -0.996219 2.49844 1.4538e-15 -1.67359e-15 1
pp 3
0.996219 -0.996219 2.49844 1.4538e-15 -1.67359e-15 1
1.40312 0 2.49844 2.22045e-15 0 1
1.4 0 2.4 -0.902861 -0 -0.429934
pp 3
0.994 -0.994 2.4 -0.637936 0.637936 -0.431366
0 -1.4 2.4 7.30595e-17 0.902861 -0.429934
0 -1.40312 2.49844 -5.65179e-16 -2.22045e-15 1
pp 3
0 -1.40312 2.49844 -5.65179e-16 -2.22045e-15 1
0.996219 -0.996219 2.49844 1.4538e-15 -1.67359e-15 1
0.994 -0.994 2.4 -0.637936 0.637936 -0.431366
pp 3
1.40312 0 2.49844 2.22045e-15 0 1
0.996219 -0.996219 2.49844 1.4538e-15 -1.67359e-15 1
1.065 -1.065 2.4 0.637936 -0.637936 0.431366
pp 3
1.065 -1.065 2.4 0.637936 -0.637936 0.431366
1.5 0 2.4 0.902861 0 0.429934
1.40312 0 2.49844 2.22045e-15 0 1
pp 3
0.996219 -0.996219 2.49844 1.4538e-15 -1.67359e-15 1
0 -1.40312 2.49844 -5.65179e-16 -2.22045e-15 1
0 -1.5 2.4 -6.81889e-17 -0.902861 0.429934
pp 3
0 -1.5 2.4 -6.81889e-17 -0.902861 0.429934
1.065 -1.065 2.4 0.637936 -0.637936 0.431366
0.996219 -0.996219 2.49844 1.4538e-15 -1.67359e-15 1
pp 3
0 -1.4 2.4 0 0.902861 -0.429934
-0.994 -0.994 2.4 0.637936 0.637936 -0.431366
-0.996219 -0.996219 2.49844 -1.67359e-15 -1.4538e-15 1
pp 3
-0.996219 -0.996219 2.49844 -1.67359e-15 -1.4538e-15 1
0 -1.40312 2.49844 0 -2.22045e-15 1
0 -1.4 2.4 0 0.902861 -0.429934
pp 3
-0.994 -0.994 2.4 0.637936 0.637936 -0.431366
-1.4 0 2.4 0.902861 -7.30595e-17 -0.429934
-1.40312 0 2.49844 -2.22045e-15 5.65179e-16 1
pp 3
-1.40312 0 2.49844 -2.22045e-15 5.65179e-16 1
-0.996219 -0.996219 2.49844 -1.67359e-15 -1.4538e-15 1
-0.994 -0.994 2.4 0.637936 0.637936 -0.431366
pp 3
```

```
0 -1.40312 2.49844 0 -2.22045e-15 1
-0.996219 -0.996219 2.49844 -1.67359e-15 -1.4538e-15 1
-1.065 -1.065 2.4 -0.637936 -0.637936 0.431366
pp 3
-1.065 -1.065 2.4 -0.637936 -0.637936 0.431366
0 -1.5 2.4 0 -0.902861 0.429934
0 -1.40312 2.49844 0 -2.22045e-15 1
pp 3
-0.996219 -0.996219 2.49844 -1.67359e-15 -1.4538e-15 1
-1.40312 0 2.49844 -2.22045e-15 5.65179e-16 1
-1.5 0 2.4 -0.902861 6.81889e-17 0.429934
pp 3
-1.5 0 2.4 -0.902861 6.81889e-17 0.429934
-1.065 -1.065 2.4 -0.637936 -0.637936 0.431366
-0.996219 -0.996219 2.49844 -1.67359e-15 -1.4538e-15 1
pp 3
-1.4 0 2.4 0.902861 0 -0.429934
-0.994 0.994 2.4 0.637936 -0.637936 -0.431366
-0.996219 0.996219 2.49844 -1.4538e-15 1.67359e-15 1
pp 3
-0.996219 0.996219 2.49844 -1.4538e-15 1.67359e-15 1
-1.40312 0 2.49844 -2.22045e-15 0 1
-1.4 0 2.4 0.902861 0 -0.429934
pp 3
-0.994 0.994 2.4 0.637936 -0.637936 -0.431366
0 1.4 2.4 -7.30595e-17 -0.902861 -0.429934
0 1.40312 2.49844 5.65179e-16 2.22045e-15 1
pp 3
0 1.40312 2.49844 5.65179e-16 2.22045e-15 1
-0.996219 0.996219 2.49844 -1.4538e-15 1.67359e-15 1
-0.994 0.994 2.4 0.637936 -0.637936 -0.431366
pp 3
-1.40312 0 2.49844 -2.22045e-15 0 1
-0.996219 0.996219 2.49844 -1.4538e-15 1.67359e-15 1
-1.065 1.065 2.4 -0.637936 0.637936 0.431366
pp 3
-1.065 1.065 2.4 -0.637936 0.637936 0.431366
-1.5 0 2.4 -0.902861 0 0.429934
-1.40312 0 2.49844 -2.22045e-15 0 1
pp 3
-0.996219 0.996219 2.49844 -1.4538e-15 1.67359e-15 1
0 1.40312 2.49844 5.65179e-16 2.22045e-15 1
0 1.5 2.4 6.81889e-17 0.902861 0.429934
pp 3
0 1.5 2.4 6.81889e-17 0.902861 0.429934
-1.065 1.065 2.4 -0.637936 0.637936 0.431366
-0.996219 0.996219 2.49844 -1.4538e-15 1.67359e-15 1
pp 3
0 1.4 2.4 0 -0.902861 -0.429934
0.994 0.994 2.4 -0.637936 -0.637936 -0.431366
0.996219 0.996219 2.49844 1.67359e-15 1.4538e-15 1
pp 3
0.996219 0.996219 2.49844 1.67359e-15 1.4538e-15 1
0 1.40312 2.49844 -0 2.22045e-15 1
0 1.4 2.4 0 -0.902861 -0.429934
pp 3
0.994 0.994 2.4 -0.637936 -0.637936 -0.431366
1.4 0 2.4 -0.902861 7.30595e-17 -0.429934
1.40312 0 2.49844 2.22045e-15 -5.65179e-16 1
pp 3
1.40312 0 2.49844 2.22045e-15 -5.65179e-16 1
```

```
0.996219 0.996219 2.49844 1.67359e-15 1.4538e-15 1
0.994 0.994 2.4 -0.637936 -0.637936 -0.431366
pp 3
0 1.40312 2.49844 -0 2.22045e-15 1
0.996219 0.996219 2.49844 1.67359e-15 1.4538e-15 1
1.065 1.065 2.4 0.637936 0.637936 0.431366
pp 3
1.065 1.065 2.4 0.637936 0.637936 0.431366
0 1.5 2.4 -0 0.902861 0.429934
0 1.40312 2.49844 -0 2.22045e-15 1
pp 3
0.996219 0.996219 2.49844 1.67359e-15 1.4538e-15 1
1.40312 0 2.49844 2.22045e-15 -5.65179e-16 1
1.5 0 2.4 0.902861 -6.81889e-17 0.429934
pp 3
1.5 0 2.4 0.902861 -6.81889e-17 0.429934
1.065 1.065 2.4 0.637936 0.637936 0.431366
0.996219 0.996219 2.49844 1.67359e-15 1.4538e-15 1
pp 3
1.5 0 2.4 0.902861 0 0.429934
1.065 -1.065 2.4 0.637936 -0.637936 0.431366
1.30906 -1.30906 1.62188 0.662761 -0.662761 0.348563
pp 3
1.30906 -1.30906 1.62188 0.662761 -0.662761 0.348563
1.84375 0 1.62188 0.937749 0 0.347314
1.5 0 2.4 0.902861 0 0.429934
pp 3
1.065 -1.065 2.4 0.637936 -0.637936 0.431366
0 -1.5 2.4 -2.27296e-16 -0.902861 0.429934
0 -1.84375 1.62187 -1.49384e-16 -0.937749 0.347314
pp 3
0 -1.84375 1.62187 -1.49384e-16 -0.937749 0.347314
1.30906 -1.30906 1.62188 0.662761 -0.662761 0.348563
1.065 -1.065 2.4 0.637936 -0.637936 0.431366
pp 3
1.84375 0 1.62188 0.937749 0 0.347314
1.30906 -1.30906 1.62188 0.662761 -0.662761 0.348563
1.42 -1.42 0.9 0.707107 -0.707107 1.74455e-16
pp 3
1.42 -1.42 0.9 0.707107 -0.707107 1.74455e-16
2 0 0.9 1 0 0
1.84375 0 1.62188 0.937749 0 0.347314
pp 3
1.30906 -1.30906 1.62188 0.662761 -0.662761 0.348563
0 -1.84375 1.62187 -1.49384e-16 -0.937749 0.347314
0 -2 0.9 0 -1 -0
pp 3
0 -2 0.9 0 -1 -0
1.42 -1.42 0.9 0.707107 -0.707107 1.74455e-16
1.30906 -1.30906 1.62188 0.662761 -0.662761 0.348563
pp 3
0 -1.5 2.4 0 -0.902861 0.429934
-1.065 -1.065 2.4 -0.637936 -0.637936 0.431366
-1.30906 -1.30906 1.62188 -0.662761 -0.662761 0.348563
pp 3
-1.30906 -1.30906 1.62188 -0.662761 -0.662761 0.348563
0 -1.84375 1.62188 0 -0.937749 0.347314
0 -1.5 2.4 0 -0.902861 0.429934
pp 3
-1.065 -1.065 2.4 -0.637936 -0.637936 0.431366
-1.5 0 2.4 -0.902861 2.27296e-16 0.429934
```

```
-1.84375 0 1.62187 -0.937749 1.49384e-16 0.347314
pp 3
-1.84375 0 1.62187 -0.937749 1.49384e-16 0.347314
-1.30906 -1.30906 1.62188 -0.662761 -0.662761 0.348563
-1.065 -1.065 2.4 -0.637936 -0.637936 0.431366
pp 3
0 -1.84375 1.62188 0 -0.937749 0.347314
-1.30906 -1.30906 1.62188 -0.662761 -0.662761 0.348563
-1.42 -1.42 0.9 -0.707107 -0.707107 1.74455e-16
pp 3
-1.42 -1.42 0.9 -0.707107 -0.707107 1.74455e-16
0 -2 0.9 -0 -1 -0
0 -1.84375 1.62188 0 -0.937749 0.347314
pp 3
-1.30906 -1.30906 1.62188 -0.662761 -0.662761 0.348563
-1.84375 0 1.62187 -0.937749 1.49384e-16 0.347314
-2 0 0.9 -1 0 0
pp 3
-2 0 0.9 -1 0 0
-1.42 -1.42 0.9 -0.707107 -0.707107 1.74455e-16
-1.30906 -1.30906 1.62188 -0.662761 -0.662761 0.348563
pp 3
-1.5 0 2.4 -0.902861 0 0.429934
-1.065 1.065 2.4 -0.637936 0.637936 0.431366
-1.30906 1.30906 1.62188 -0.662761 0.662761 0.348563
pp 3
-1.30906 1.30906 1.62188 -0.662761 0.662761 0.348563
-1.84375 0 1.62188 -0.937749 0 0.347314
-1.5 0 2.4 -0.902861 0 0.429934
pp 3
-1.065 1.065 2.4 -0.637936 0.637936 0.431366
0 1.5 2.4 2.27296e-16 0.902861 0.429934
0 1.84375 1.62187 1.49384e-16 0.937749 0.347314
pp 3
0 1.84375 1.62187 1.49384e-16 0.937749 0.347314
-1.30906 1.30906 1.62188 -0.662761 0.662761 0.348563
-1.065 1.065 2.4 -0.637936 0.637936 0.431366
pp 3
-1.84375 0 1.62188 -0.937749 0 0.347314
-1.30906 1.30906 1.62188 -0.662761 0.662761 0.348563
-1.42 1.42 0.9 -0.707107 0.707107 1.74455e-16
pp 3
-1.42 1.42 0.9 -0.707107 0.707107 1.74455e-16
-2 0 0.9 -1 0 0
-1.84375 0 1.62188 -0.937749 0 0.347314
pp 3
-1.30906 1.30906 1.62188 -0.662761 0.662761 0.348563
0 1.84375 1.62187 1.49384e-16 0.937749 0.347314
0 2 0.9 0 1 0
pp 3
0 2 0.9 0 1 0
-1.42 1.42 0.9 -0.707107 0.707107 1.74455e-16
-1.30906 1.30906 1.62188 -0.662761 0.662761 0.348563
pp 3
0 1.5 2.4 -0 0.902861 0.429934
1.065 1.065 2.4 0.637936 0.637936 0.431366
1.30906 1.30906 1.62188 0.662761 0.662761 0.348563
pp 3
1.30906 1.30906 1.62188 0.662761 0.662761 0.348563
0 1.84375 1.62188 -0 0.937749 0.347314
0 1.5 2.4 -0 0.902861 0.429934
```

```
pp 3
1.065 1.065 2.4 0.637936 0.637936 0.431366
1.5 0 2.4 0.902861 -2.27296e-16 0.429934
1.84375 0 1.62187 0.937749 -1.49384e-16 0.347314
pp 3
1.84375 0 1.62187 0.937749 -1.49384e-16 0.347314
1.30906 1.30906 1.62188 0.662761 0.662761 0.348563
1.065 1.065 2.4 0.637936 0.637936 0.431366
pp 3
0 1.84375 1.62188 -0 0.937749 0.347314
1.30906 1.30906 1.62188 0.662761 0.662761 0.348563
1.42 1.42 0.9 0.707107 0.707107 1.74455e-16
pp 3
1.42 1.42 0.9 0.707107 0.707107 1.74455e-16
0 2 0.9 -0 1 0
0 1.84375 1.62188 -0 0.937749 0.347314
pp 3
1.30906 1.30906 1.62188 0.662761 0.662761 0.348563
1.84375 0 1.62187 0.937749 -1.49384e-16 0.347314
2 0 0.9 1 0 0
pp 3
2 0 0.9 1 0 0
1.42 1.42 0.9 0.707107 0.707107 1.74455e-16
1.30906 1.30906 1.62188 0.662761 0.662761 0.348563
pp 3
2 0 0.9 1 0 0
1.42 -1.42 0.9 0.707107 -0.707107 0
1.2425 -1.2425 0.384375 0.492597 -0.492597 -0.717423
pp 3
1.2425 -1.2425 0.384375 0.492597 -0.492597 -0.717423
1.75 0 0.384375 0.6981 0 -0.716
2 0 0.9 1 0 0
pp 3
1.42 -1.42 0.9 0.707107 -0.707107 0
0 -2 0.9 -0 -1 -0
2.22045e-16 -1.75 0.384375 -8.11143e-17 -0.6981 -0.716
pp 3
2.22045e-16 -1.75 0.384375 -8.11143e-17 -0.6981 -0.716
1.2425 -1.2425 0.384375 0.492597 -0.492597 -0.717423
1.42 -1.42 0.9 0.707107 -0.707107 0
pp 3
1.75 0 0.384375 0.6981 0 -0.716
1.2425 -1.2425 0.384375 0.492597 -0.492597 -0.717423
1.065 -1.065 0.15 0.707107 -0.707107 0
pp 3
1.065 -1.065 0.15 0.707107 -0.707107 0
1.5 0 0.15 1 0 0
1.75 0 0.384375 0.6981 0 -0.716
pp 3
1.2425 -1.2425 0.384375 0.492597 -0.492597 -0.717423
2.22045e-16 -1.75 0.384375 -8.11143e-17 -0.6981 -0.716
0 -1.5 0.15 -0 -1 -0
pp 3
0 -1.5 0.15 -0 -1 -0
1.065 -1.065 0.15 0.707107 -0.707107 0
1.2425 -1.2425 0.384375 0.492597 -0.492597 -0.717423
pp 3
0 -2 0.9 -0 -1 -0
-1.42 -1.42 0.9 -0.707107 -0.707107 -0
-1.2425 -1.2425 0.384375 -0.492597 -0.492597 -0.717423
pp 3
```



```
-1.2425 -1.2425 0.384375 -0.492597 -0.492597 -0.717423
0 -1.75 0.384375 -0 -0.6981 -0.716
0 -2 0.9 -0 -1 -0
pp 3
-1.42 -1.42 0.9 -0.707107 -0.707107 -0
-2 0 0.9 -1 0 0
-1.75 -2.22045e-16 0.384375 -0.6981 8.11143e-17 -0.716
pp 3
-1.75 -2.22045e-16 0.384375 -0.6981 8.11143e-17 -0.716
-1.2425 -1.2425 0.384375 -0.492597 -0.492597 -0.717423
-1.42 -1.42 0.9 -0.707107 -0.707107 -0
pp 3
0 -1.75 0.384375 -0 -0.6981 -0.716
-1.2425 -1.2425 0.384375 -0.492597 -0.492597 -0.717423
-1.065 -1.065 0.15 -0.707107 -0.707107 -0
pp 3
-1.065 -1.065 0.15 -0.707107 -0.707107 -0
0 -1.5 0.15 -0 -1 -0
0 -1.75 0.384375 -0 -0.6981 -0.716
pp 3
-1.2425 -1.2425 0.384375 -0.492597 -0.492597 -0.717423
-1.75 -2.22045e-16 0.384375 -0.6981 8.11143e-17 -0.716
-1.5 0 0.15 -1 0 0
pp 3
-1.5 0 0.15 -1 0 0
-1.065 -1.065 0.15 -0.707107 -0.707107 -0
-1.2425 -1.2425 0.384375 -0.492597 -0.492597 -0.717423
pp 3
-2 0 0.9 -1 0 0
-1.42 1.42 0.9 -0.707107 0.707107 0
-1.2425 1.2425 0.384375 -0.492597 0.492597 -0.717423
pp 3
-1.2425 1.2425 0.384375 -0.492597 0.492597 -0.717423
-1.75 0 0.384375 -0.6981 0 -0.716
-2 0 0.9 -1 0 0
pp 3
-1.42 1.42 0.9 -0.707107 0.707107 0
0 2 0.9 -0 1 0
-2.22045e-16 1.75 0.384375 8.11143e-17 0.6981 -0.716
pp 3
-2.22045e-16 1.75 0.384375 8.11143e-17 0.6981 -0.716
-1.2425 1.2425 0.384375 -0.492597 0.492597 -0.717423
-1.42 1.42 0.9 -0.707107 0.707107 0
pp 3
-1.75 0 0.384375 -0.6981 0 -0.716
-1.2425 1.2425 0.384375 -0.492597 0.492597 -0.717423
-1.065 1.065 0.15 -0.707107 0.707107 0
pp 3
-1.065 1.065 0.15 -0.707107 0.707107 0
-1.5 0 0.15 -1 0 0
-1.75 0 0.384375 -0.6981 0 -0.716
pp 3
-1.2425 1.2425 0.384375 -0.492597 0.492597 -0.717423
-2.22045e-16 1.75 0.384375 8.11143e-17 0.6981 -0.716
0 1.5 0.15 -0 1 0
pp 3
0 1.5 0.15 -0 1 0
-1.065 1.065 0.15 -0.707107 0.707107 0
-1.2425 1.2425 0.384375 -0.492597 0.492597 -0.717423
pp 3
0 2 0.9 -0 1 0
```

```
1.42 1.42 0.9 0.707107 0.707107 0
1.2425 1.2425 0.384375 0.492597 0.492597 -0.717423
pp 3
1.2425 1.2425 0.384375 0.492597 0.492597 -0.717423
0 1.75 0.384375 0 0.6981 -0.716
0 2 0.9 -0 1 0
pp 3
1.42 1.42 0.9 0.707107 0.707107 0
2 0 0.9 1 0 0
1.75 2.22045e-16 0.384375 0.6981 -8.11143e-17 -0.716
pp 3
1.75 2.22045e-16 0.384375 0.6981 -8.11143e-17 -0.716
1.2425 1.2425 0.384375 0.492597 0.492597 -0.717423
1.42 1.42 0.9 0.707107 0.707107 0
pp 3
0 1.75 0.384375 0 0.6981 -0.716
1.2425 1.2425 0.384375 0.492597 0.492597 -0.717423
1.065 1.065 0.15 0.707107 0.707107 0
pp 3
1.065 1.065 0.15 0.707107 0.707107 0
0 1.5 0.15 -0 1 0
0 1.75 0.384375 0 0.6981 -0.716
pp 3
1.2425 1.2425 0.384375 0.492597 0.492597 -0.717423
1.75 2.22045e-16 0.384375 0.6981 -8.11143e-17 -0.716
1.5 0 0.15 1 0 0
pp 3
1.5 0 0.15 1 0 0
1.065 1.065 0.15 0.707107 0.707107 0
1.2425 1.2425 0.384375 0.492597 0.492597 -0.717423
pp 3
-1.6 0 2.025 -0 -0 -1
-1.55 -0.225 2.1375 0 -1 0
-2.51875 -0.225 2.09531 -1.96915e-17 -1 -2.12543e-17
pp 3
-2.51875 -0.225 2.09531 -1.96915e-17 -1 -2.12543e-17
-2.4125 0 1.99687 0.14834 0 -0.988936
-1.6 0 2.025 -0 -0 -1
pp 3
-1.55 -0.225 2.1375 0 -1 0
-1.5 0 2.25 0 1.97373e-15 1
-2.625 0 2.19375 -0.219512 1.81728e-15 0.97561
pp 3
-2.625 0 2.19375 -0.219512 1.81728e-15 0.97561
-2.51875 -0.225 2.09531 -1.96915e-17 -1 -2.12543e-17
-1.55 -0.225 2.1375 0 -1 0
pp 3
-2.4125 0 1.99687 0.14834 0 -0.988936
-2.51875 -0.225 2.09531 -1.96915e-17 -1 -2.12543e-17
-2.85 -0.225 1.8 -1.21738e-31 -1 -1.23358e-16
pp 3
-2.85 -0.225 1.8 -1.21738e-31 -1 -1.23358e-16
-2.7 0 1.8 1 0 2.63164e-15
-2.4125 0 1.99687 0.14834 0 -0.988936
pp 3
-2.51875 -0.225 2.09531 -1.96915e-17 -1 -2.12543e-17
-2.625 0 2.19375 -0.219512 1.81728e-15 0.97561
-3 0 1.8 -1 -4.93432e-16 -0
pp 3
-3 0 1.8 -1 -4.93432e-16 -0
-2.85 -0.225 1.8 -1.21738e-31 -1 -1.23358e-16
```

-2.51875 -0.225 2.09531 -1.96915e-17 -1 -2.12543e-17  
pp 3  
-1.5 0 2.25 0 -0 1  
-1.55 0.225 2.1375 0 1 0  
-2.51875 0.225 2.09531 -1.96915e-17 1 -2.12543e-17  
pp 3  
-2.51875 0.225 2.09531 -1.96915e-17 1 -2.12543e-17  
-2.625 0 2.19375 -0.219512 0 0.97561  
-1.5 0 2.25 0 -0 1  
pp 3  
-1.55 0.225 2.1375 0 1 0  
-1.6 0 2.025 -0 -4.93432e-16 -1  
-2.4125 0 1.99687 0.14834 -4.14777e-16 -0.988936  
pp 3  
-2.4125 0 1.99687 0.14834 -4.14777e-16 -0.988936  
-2.51875 0.225 2.09531 -1.96915e-17 1 -2.12543e-17  
-1.55 0.225 2.1375 0 1 0  
pp 3  
-2.625 0 2.19375 -0.219512 0 0.97561  
-2.51875 0.225 2.09531 -1.96915e-17 1 -2.12543e-17  
-2.85 0.225 1.8 -2.13041e-31 1 -1.23358e-16  
pp 3  
-2.85 0.225 1.8 -2.13041e-31 1 -1.23358e-16  
-3 0 1.8 -1 0 0  
-2.625 0 2.19375 -0.219512 0 0.97561  
pp 3  
-2.51875 0.225 2.09531 -1.96915e-17 1 -2.12543e-17  
-2.4125 0 1.99687 0.14834 -4.14777e-16 -0.988936  
-2.7 0 1.8 1 4.93432e-16 2.63164e-15  
pp 3  
-2.7 0 1.8 1 4.93432e-16 2.63164e-15  
-2.85 0.225 1.8 -2.13041e-31 1 -1.23358e-16  
-2.51875 0.225 2.09531 -1.96915e-17 1 -2.12543e-17  
pp 3  
-2.7 0 1.8 1 0 0  
-2.85 -0.225 1.8 -0 -1 -0  
-2.63437 -0.225 1.25391 1.5603e-17 -1 -1.57299e-17  
pp 3  
-2.63437 -0.225 1.25391 1.5603e-17 -1 -1.57299e-17  
-2.5375 0 1.35 0.83205 0 0.5547  
-2.7 0 1.8 1 0 0  
pp 3  
-2.85 -0.225 1.8 -0 -1 -0  
-3 0 1.8 -1 -9.86865e-16 -0  
-2.73125 0 1.15781 -0.743581 -2.38916e-16 -0.668646  
pp 3  
-2.73125 0 1.15781 -0.743581 -2.38916e-16 -0.668646  
-2.63437 -0.225 1.25391 1.5603e-17 -1 -1.57299e-17  
-2.85 -0.225 1.8 -0 -1 -0  
pp 3  
-2.5375 0 1.35 0.83205 0 0.5547  
-2.63437 -0.225 1.25391 1.5603e-17 -1 -1.57299e-17  
-1.95 -0.225 0.75 7.83687e-17 -1 2.61229e-17  
pp 3  
-1.95 -0.225 0.75 7.83687e-17 -1 2.61229e-17  
-2 0 0.9 0.410365 0 0.911922  
-2.5375 0 1.35 0.83205 0 0.5547  
pp 3  
-2.63437 -0.225 1.25391 1.5603e-17 -1 -1.57299e-17  
-2.73125 0 1.15781 -0.743581 -2.38916e-16 -0.668646  
-1.9 0 0.6 -0.410365 4.49972e-17 -0.911922

```
pp 3
-1.9 0 0.6 -0.410365 4.49972e-17 -0.911922
-1.95 -0.225 0.75 7.83687e-17 -1 2.61229e-17
-2.63437 -0.225 1.25391 1.5603e-17 -1 -1.57299e-17
pp 3
-3 0 1.8 -1 0 0
-2.85 0.225 1.8 -0 1 0
-2.63437 0.225 1.25391 1.5603e-17 1 -1.57299e-17
pp 3
-2.63437 0.225 1.25391 1.5603e-17 1 -1.57299e-17
-2.73125 0 1.15781 -0.743581 0 -0.668646
-3 0 1.8 -1 0 0
pp 3
-2.85 0.225 1.8 -0 1 0
-2.7 0 1.8 1 9.86865e-16 0
-2.5375 0 1.35 0.83205 1.23168e-15 0.5547
pp 3
-2.5375 0 1.35 0.83205 1.23168e-15 0.5547
-2.63437 0.225 1.25391 1.5603e-17 1 -1.57299e-17
-2.85 0.225 1.8 -0 1 0
pp 3
-2.73125 0 1.15781 -0.743581 0 -0.668646
-2.63437 0.225 1.25391 1.5603e-17 1 -1.57299e-17
-1.95 0.225 0.75 7.83687e-17 1 2.61229e-17
pp 3
-1.95 0.225 0.75 7.83687e-17 1 2.61229e-17
-1.9 0 0.6 -0.410365 0 -0.911922
-2.73125 0 1.15781 -0.743581 0 -0.668646
pp 3
-2.63437 0.225 1.25391 1.5603e-17 1 -1.57299e-17
-2.5375 0 1.35 0.83205 1.23168e-15 0.5547
-2 0 0.9 0.410365 1.30492e-15 0.911922
pp 3
-2 0 0.9 0.410365 1.30492e-15 0.911922
-1.95 0.225 0.75 7.83687e-17 1 2.61229e-17
-2.63437 0.225 1.25391 1.5603e-17 1 -1.57299e-17
pp 3
1.7 0 1.425 -0 0 1
1.7 -0.495 1.0125 0 -1 -0
2.5375 -0.34125 1.62187 0.214084 -0.960035 0.180281
pp 3
2.5375 -0.34125 1.62187 0.214084 -0.960035 0.180281
2.3875 0 1.8 -0.920582 0 0.39055
1.7 0 1.425 -0 0 1
pp 3
1.7 -0.495 1.0125 0 -1 -0
1.7 0 0.6 0.158678 9.39168e-16 -0.98733
2.6875 0 1.44375 0.957826 1.83855e-15 -0.287348
pp 3
2.6875 0 1.44375 0.957826 1.83855e-15 -0.287348
2.5375 -0.34125 1.62187 0.214084 -0.960035 0.180281
1.7 -0.495 1.0125 0 -1 -0
pp 3
2.3875 0 1.8 -0.920582 0 0.39055
2.5375 -0.34125 1.62187 0.214084 -0.960035 0.180281
3 -0.1875 2.4 0 -1 2.19303e-16
pp 3
3 -0.1875 2.4 0 -1 2.19303e-16
2.7 0 2.4 -0.6 0 0.8
2.3875 0 1.8 -0.920582 0 0.39055
pp 3
```

2.5375 -0.34125 1.62187 0.214084 -0.960035 0.180281  
2.6875 0 1.44375 0.957826 1.83855e-15 -0.287348  
3.3 0 2.4 0.384615 6.46776e-15 -0.923077  
pp 3  
3.3 0 2.4 0.384615 6.46776e-15 -0.923077  
3 -0.1875 2.4 0 -1 2.19303e-16  
2.5375 -0.34125 1.62187 0.214084 -0.960035 0.180281  
pp 3  
1.7 0 0.6 0.158678 0 -0.98733  
1.7 0.495 1.0125 0 1 -0  
2.5375 0.34125 1.62188 0.214084 0.960035 0.180281  
pp 3  
2.5375 0.34125 1.62188 0.214084 0.960035 0.180281  
2.6875 0 1.44375 0.957826 0 -0.287348  
1.7 0 0.6 0.158678 0 -0.98733  
pp 3  
1.7 0.495 1.0125 0 1 -0  
1.7 0 1.425 0 -4.48575e-16 1  
2.3875 0 1.8 -0.920582 -2.76813e-16 0.39055  
pp 3  
2.3875 0 1.8 -0.920582 -2.76813e-16 0.39055  
2.5375 0.34125 1.62188 0.214084 0.960035 0.180281  
1.7 0.495 1.0125 0 1 -0  
pp 3  
2.6875 0 1.44375 0.957826 0 -0.287348  
2.5375 0.34125 1.62188 0.214084 0.960035 0.180281  
3 0.1875 2.4 4.05793e-32 1 2.19303e-16  
pp 3  
3 0.1875 2.4 4.05793e-32 1 2.19303e-16  
3.3 0 2.4 0.384615 0 -0.923077  
2.6875 0 1.44375 0.957826 0 -0.287348  
pp 3  
2.5375 0.34125 1.62188 0.214084 0.960035 0.180281  
2.3875 0 1.8 -0.920582 -2.76813e-16 0.39055  
2.7 0 2.4 -0.6 -3.55271e-15 0.8  
pp 3  
2.7 0 2.4 -0.6 -3.55271e-15 0.8  
3 0.1875 2.4 4.05793e-32 1 2.19303e-16  
2.5375 0.34125 1.62188 0.214084 0.960035 0.180281  
pp 3  
2.7 0 2.4 -0.6 0 0.8  
3 -0.1875 2.4 0 -1 0  
3.12656 -0.15 2.4668 -0.0348909 -0.0594438 0.997622  
pp 3  
3.12656 -0.15 2.4668 -0.0348909 -0.0594438 0.997622  
2.825 0 2.45625 2.22045e-15 0 1  
2.7 0 2.4 -0.6 0 0.8  
pp 3  
3 -0.1875 2.4 0 -1 0  
3.3 0 2.4 0.384615 -2.55067e-15 -0.923077  
3.42813 0 2.47734 0.106533 1.89241e-15 0.994309  
pp 3  
3.42813 0 2.47734 0.106533 1.89241e-15 0.994309  
3.12656 -0.15 2.4668 -0.0348909 -0.0594438 0.997622  
3 -0.1875 2.4 0 -1 0  
pp 3  
2.825 0 2.45625 2.22045e-15 0 1  
3.12656 -0.15 2.4668 -0.0348909 -0.0594438 0.997622  
3 -0.1125 2.4 0 1 0  
pp 3  
3 -0.1125 2.4 0 1 0

```
2.8 0 2.4 0.6 0 -0.8
2.825 0 2.45625 2.22045e-15 0 1
pp 3
3.12656 -0.15 2.4668 -0.0348909 -0.0594438 0.997622
3.42813 0 2.47734 0.106533 1.89241e-15 0.994309
3.2 0 2.4 -0.410365 4.31973e-15 0.911922
pp 3
3.2 0 2.4 -0.410365 4.31973e-15 0.911922
3 -0.1125 2.4 0 1 0
3.12656 -0.15 2.4668 -0.0348909 -0.0594438 0.997622
pp 3
3.3 0 2.4 0.384615 0 -0.923077
3 0.1875 2.4 0 1 -0
3.12656 0.15 2.4668 -0.0348909 0.0594438 0.997622
pp 3
3.12656 0.15 2.4668 -0.0348909 0.0594438 0.997622
3.42813 0 2.47734 0.106533 -0 0.994309
3.3 0 2.4 0.384615 0 -0.923077
pp 3
3 0.1875 2.4 0 1 -0
2.7 0 2.4 -0.6 2.24387e-29 0.8
2.825 0 2.45625 2.22045e-15 -2.22045e-15 1
pp 3
2.825 0 2.45625 2.22045e-15 -2.22045e-15 1
3.12656 0.15 2.4668 -0.0348909 0.0594438 0.997622
3 0.1875 2.4 0 1 -0
pp 3
3.42813 0 2.47734 0.106533 -0 0.994309
3.12656 0.15 2.4668 -0.0348909 0.0594438 0.997622
3 0.1125 2.4 0 -1 0
pp 3
3 0.1125 2.4 0 -1 0
3.2 0 2.4 -0.410365 0 0.911922
3.42813 0 2.47734 0.106533 -0 0.994309
pp 3
3.12656 0.15 2.4668 -0.0348909 0.0594438 0.997622
2.825 0 2.45625 2.22045e-15 -2.22045e-15 1
2.8 0 2.4 0.6 -3.73979e-29 -0.8
pp 3
2.8 0 2.4 0.6 -3.73979e-29 -0.8
3 0.1125 2.4 0 -1 0
3.12656 0.15 2.4668 -0.0348909 0.0594438 0.997622
pp 3
0.231031 -0.231031 2.98125 0.550896 -0.550896 -0.626919
0.325 0 2.98125 0.780869 0 -0.624695
0 0 3.15 0 0 1
pp 3
-1.11022e-16 -0.325 2.98125 1.51803e-15 -0.780869 -0.624695
0.231031 -0.231031 2.98125 0.550896 -0.550896 -0.626919
0 0 3.15 0 0 1
pp 3
0.325 0 2.98125 0.780869 0 -0.624695
0.231031 -0.231031 2.98125 0.550896 -0.550896 -0.626919
0.142 -0.142 2.7 0.423155 -0.423155 0.801174
pp 3
0.142 -0.142 2.7 0.423155 -0.423155 0.801174
0.2 0 2.7 0.6 0 0.8
0.325 0 2.98125 0.780869 0 -0.624695
pp 3
0.231031 -0.231031 2.98125 0.550896 -0.550896 -0.626919
-1.11022e-16 -0.325 2.98125 1.51803e-15 -0.780869 -0.624695
```

```
0 -0.2 2.7 -3.17207e-15 -0.6 0.8
pp 3
0 -0.2 2.7 -3.17207e-15 -0.6 0.8
0.142 -0.142 2.7 0.423155 -0.423155 0.801174
0.231031 -0.231031 2.98125 0.550896 -0.550896 -0.626919
pp 3
-0.231031 -0.231031 2.98125 -0.550896 -0.550896 -0.626919
0 -0.325 2.98125 -0 -0.780869 -0.624695
0 0 3.15 0 0 1
pp 3
-0.325 1.11022e-16 2.98125 -0.780869 -1.51803e-15 -0.624695
-0.231031 -0.231031 2.98125 -0.550896 -0.550896 -0.626919
0 0 3.15 0 0 1
pp 3
0 -0.325 2.98125 -0 -0.780869 -0.624695
-0.231031 -0.231031 2.98125 -0.550896 -0.550896 -0.626919
-0.142 -0.142 2.7 -0.423155 -0.423155 0.801174
pp 3
-0.142 -0.142 2.7 -0.423155 -0.423155 0.801174
0 -0.2 2.7 0 -0.6 0.8
0 -0.325 2.98125 -0 -0.780869 -0.624695
pp 3
-0.231031 -0.231031 2.98125 -0.550896 -0.550896 -0.626919
-0.325 1.11022e-16 2.98125 -0.780869 -1.51803e-15 -0.624695
-0.2 0 2.7 -0.6 3.17207e-15 0.8
pp 3
-0.2 0 2.7 -0.6 3.17207e-15 0.8
-0.142 -0.142 2.7 -0.423155 -0.423155 0.801174
-0.231031 -0.231031 2.98125 -0.550896 -0.550896 -0.626919
pp 3
-0.231031 0.231031 2.98125 -0.550896 0.550896 -0.626919
-0.325 0 2.98125 -0.780869 0 -0.624695
0 0 3.15 0 0 1
pp 3
1.11022e-16 0.325 2.98125 -1.51803e-15 0.780869 -0.624695
-0.231031 0.231031 2.98125 -0.550896 0.550896 -0.626919
0 0 3.15 0 0 1
pp 3
-0.325 0 2.98125 -0.780869 0 -0.624695
-0.231031 0.231031 2.98125 -0.550896 0.550896 -0.626919
-0.142 0.142 2.7 -0.423155 0.423155 0.801174
pp 3
-0.142 0.142 2.7 -0.423155 0.423155 0.801174
-0.2 0 2.7 -0.6 0 0.8
-0.325 0 2.98125 -0.780869 0 -0.624695
pp 3
-0.231031 0.231031 2.98125 -0.550896 0.550896 -0.626919
1.11022e-16 0.325 2.98125 -1.51803e-15 0.780869 -0.624695
0 0.2 2.7 3.17207e-15 0.6 0.8
pp 3
0 0.2 2.7 3.17207e-15 0.6 0.8
-0.142 0.142 2.7 -0.423155 0.423155 0.801174
-0.231031 0.231031 2.98125 -0.550896 0.550896 -0.626919
pp 3
0.231031 0.231031 2.98125 0.550896 0.550896 -0.626919
0 0.325 2.98125 0 0.780869 -0.624695
0 0 3.15 0 0 1
pp 3
0.325 -1.11022e-16 2.98125 0.780869 1.51803e-15 -0.624695
0.231031 0.231031 2.98125 0.550896 0.550896 -0.626919
0 0 3.15 0 0 1
```

```
pp 3
0 0.325 2.98125 0 0.780869 -0.624695
0.231031 0.231031 2.98125 0.550896 0.550896 -0.626919
0.142 0.142 2.7 0.423155 0.423155 0.801174
pp 3
0.142 0.142 2.7 0.423155 0.423155 0.801174
0 0.2 2.7 -0 0.6 0.8
0 0.325 2.98125 0 0.780869 -0.624695
pp 3
0.231031 0.231031 2.98125 0.550896 0.550896 -0.626919
0.325 -1.11022e-16 2.98125 0.780869 1.51803e-15 -0.624695
0.2 0 2.7 0.6 -3.17207e-15 0.8
pp 3
0.2 0 2.7 0.6 -3.17207e-15 0.8
0.142 0.142 2.7 0.423155 0.423155 0.801174
0.231031 0.231031 2.98125 0.550896 0.550896 -0.626919
pp 3
0.2 0 2.7 0.6 0 0.8
0.142 -0.142 2.7 0.423155 -0.423155 0.801174
0.58575 -0.58575 2.55 0.104474 -0.104474 0.989025
pp 3
0.58575 -0.58575 2.55 0.104474 -0.104474 0.989025
0.825 0 2.55 0.14834 0 0.988936
0.2 0 2.7 0.6 0 0.8
pp 3
0.142 -0.142 2.7 0.423155 -0.423155 0.801174
0 -0.2 2.7 -9.91271e-17 -0.6 0.8
1.11022e-16 -0.825 2.55 -2.37649e-17 -0.14834 0.988936
pp 3
1.11022e-16 -0.825 2.55 -2.37649e-17 -0.14834 0.988936
0.58575 -0.58575 2.55 0.104474 -0.104474 0.989025
0.142 -0.142 2.7 0.423155 -0.423155 0.801174
pp 3
0.825 0 2.55 0.14834 0 0.988936
0.58575 -0.58575 2.55 0.104474 -0.104474 0.989025
0.923 -0.923 2.4 0.707107 -0.707107 -7.41433e-16
pp 3
0.923 -0.923 2.4 0.707107 -0.707107 -7.41433e-16
1.3 0 2.4 1 0 -9.86865e-16
0.825 0 2.55 0.14834 0 0.988936
pp 3
0.58575 -0.58575 2.55 0.104474 -0.104474 0.989025
1.11022e-16 -0.825 2.55 -2.37649e-17 -0.14834 0.988936
6.66134e-16 -1.3 2.4 -4.06675e-16 -1 -1.4803e-15
pp 3
6.66134e-16 -1.3 2.4 -4.06675e-16 -1 -1.4803e-15
0.923 -0.923 2.4 0.707107 -0.707107 -7.41433e-16
0.58575 -0.58575 2.55 0.104474 -0.104474 0.989025
pp 3
0 -0.2 2.7 0 -0.6 0.8
-0.142 -0.142 2.7 -0.423155 -0.423155 0.801174
-0.58575 -0.58575 2.55 -0.104474 -0.104474 0.989025
pp 3
-0.58575 -0.58575 2.55 -0.104474 -0.104474 0.989025
0 -0.825 2.55 0 -0.14834 0.988936
0 -0.2 2.7 0 -0.6 0.8
pp 3
-0.142 -0.142 2.7 -0.423155 -0.423155 0.801174
-0.2 0 2.7 -0.6 9.91271e-17 0.8
-0.825 -1.11022e-16 2.55 -0.14834 2.37649e-17 0.988936
pp 3
```











```
-0.825 -1.11022e-16 2.55 -0.14834 2.37649e-17 0.988936
-0.58575 -0.58575 2.55 -0.104474 -0.104474 0.989025
-0.142 -0.142 2.7 -0.423155 -0.423155 0.801174
pp 3
0 -0.825 2.55 0 -0.14834 0.988936
-0.58575 -0.58575 2.55 -0.104474 -0.104474 0.989025
-0.923 -0.923 2.4 -0.707107 -0.707107 -7.41433e-16
pp 3
-0.923 -0.923 2.4 -0.707107 -0.707107 -7.41433e-16
0 -1.3 2.4 -0 -1 -9.86865e-16
0 -0.825 2.55 0 -0.14834 0.988936
pp 3
-0.58575 -0.58575 2.55 -0.104474 -0.104474 0.989025
-0.825 -1.11022e-16 2.55 -0.14834 2.37649e-17 0.988936
-1.3 -6.66134e-16 2.4 -1 4.06675e-16 -1.4803e-15
pp 3
-1.3 -6.66134e-16 2.4 -1 4.06675e-16 -1.4803e-15
-0.923 -0.923 2.4 -0.707107 -0.707107 -7.41433e-16
-0.58575 -0.58575 2.55 -0.104474 -0.104474 0.989025
pp 3
-0.2 0 2.7 -0.6 0 0.8
-0.142 0.142 2.7 -0.423155 0.423155 0.801174
-0.58575 0.58575 2.55 -0.104474 0.104474 0.989025
pp 3
-0.58575 0.58575 2.55 -0.104474 0.104474 0.989025
-0.825 0 2.55 -0.14834 0 0.988936
-0.2 0 2.7 -0.6 0 0.8
pp 3
-0.142 0.142 2.7 -0.423155 0.423155 0.801174
0 0.2 2.7 9.91271e-17 0.6 0.8
-1.11022e-16 0.825 2.55 2.37649e-17 0.14834 0.988936
pp 3
-1.11022e-16 0.825 2.55 2.37649e-17 0.14834 0.988936
-0.58575 0.58575 2.55 -0.104474 0.104474 0.989025
-0.142 0.142 2.7 -0.423155 0.423155 0.801174
pp 3
-0.825 0 2.55 -0.14834 0 0.988936
-0.58575 0.58575 2.55 -0.104474 0.104474 0.989025
-0.923 0.923 2.4 -0.707107 0.707107 -7.41433e-16
pp 3
-0.923 0.923 2.4 -0.707107 0.707107 -7.41433e-16
-1.3 0 2.4 -1 0 -9.86865e-16
-0.825 0 2.55 -0.14834 0 0.988936
pp 3
-0.58575 0.58575 2.55 -0.104474 0.104474 0.989025
-1.11022e-16 0.825 2.55 2.37649e-17 0.14834 0.988936
-6.66134e-16 1.3 2.4 4.06675e-16 1 -1.4803e-15
pp 3
-6.66134e-16 1.3 2.4 4.06675e-16 1 -1.4803e-15
-0.923 0.923 2.4 -0.707107 0.707107 -7.41433e-16
-0.58575 0.58575 2.55 -0.104474 0.104474 0.989025
pp 3
0 0.2 2.7 -0 0.6 0.8
0.142 0.142 2.7 0.423155 0.423155 0.801174
0.58575 0.58575 2.55 0.104474 0.104474 0.989025
pp 3
0.58575 0.58575 2.55 0.104474 0.104474 0.989025
0 0.825 2.55 -0 0.14834 0.988936
0 0.2 2.7 -0 0.6 0.8
pp 3
0.142 0.142 2.7 0.423155 0.423155 0.801174
```

```
0.2 0 2.7 0.6 -9.91271e-17 0.8
0.825 1.11022e-16 2.55 0.14834 -2.37649e-17 0.988936
pp 3
0.825 1.11022e-16 2.55 0.14834 -2.37649e-17 0.988936
0.58575 0.58575 2.55 0.104474 0.104474 0.989025
0.142 0.142 2.7 0.423155 0.423155 0.801174
pp 3
0 0.825 2.55 -0 0.14834 0.988936
0.58575 0.58575 2.55 0.104474 0.104474 0.989025
0.923 0.923 2.4 0.707107 0.707107 -7.41433e-16
pp 3
0.923 0.923 2.4 0.707107 0.707107 -7.41433e-16
0 1.3 2.4 0 1 -9.86865e-16
0 0.825 2.55 -0 0.14834 0.988936
pp 3
0.58575 0.58575 2.55 0.104474 0.104474 0.989025
0.825 1.11022e-16 2.55 0.14834 -2.37649e-17 0.988936
1.3 6.66134e-16 2.4 1 -4.06675e-16 -1.4803e-15
pp 3
1.3 6.66134e-16 2.4 1 -4.06675e-16 -1.4803e-15
0.923 0.923 2.4 0.707107 0.707107 -7.41433e-16
0.58575 0.58575 2.55 0.104474 0.104474 0.989025
pp 3
0.911906 0.911906 0.046875 0.0996006 0.0996006 -0.99003
1.28438 0 0.046875 0.141421 0 -0.989949
0 0 0 0 0 -1
pp 3
2.22045e-16 1.28437 0.046875 0 0.141421 -0.989949
0.911906 0.911906 0.046875 0.0996006 0.0996006 -0.99003
0 0 0 0 0 -1
pp 3
1.28438 0 0.046875 0.141421 0 -0.989949
0.911906 0.911906 0.046875 0.0996006 0.0996006 -0.99003
1.065 1.065 0.15 0.707107 0.707107 -6.97819e-16
pp 3
1.065 1.065 0.15 0.707107 0.707107 -6.97819e-16
1.5 0 0.15 1 0 0
1.28438 0 0.046875 0.141421 0 -0.989949
pp 3
0.911906 0.911906 0.046875 0.0996006 0.0996006 -0.99003
2.22045e-16 1.28437 0.046875 0 0.141421 -0.989949
0 1.5 0.15 0 1 -0
pp 3
0 1.5 0.15 0 1 -0
1.065 1.065 0.15 0.707107 0.707107 -6.97819e-16
0.911906 0.911906 0.046875 0.0996006 0.0996006 -0.99003
pp 3
-0.911906 0.911906 0.046875 -0.0996006 0.0996006 -0.99003
0 1.28438 0.046875 0 0.141421 -0.989949
0 0 0 0 0 -1
pp 3
-1.28437 2.22045e-16 0.046875 -0.141421 -0 -0.989949
-0.911906 0.911906 0.046875 -0.0996006 0.0996006 -0.99003
0 0 0 0 0 -1
pp 3
0 1.28438 0.046875 0 0.141421 -0.989949
-0.911906 0.911906 0.046875 -0.0996006 0.0996006 -0.99003
-1.065 1.065 0.15 -0.707107 0.707107 -6.97819e-16
pp 3
-1.065 1.065 0.15 -0.707107 0.707107 -6.97819e-16
0 1.5 0.15 0 1 -0
```

```
0 1.28438 0.046875 0 0.141421 -0.989949
pp 3
-0.911906 0.911906 0.046875 -0.0996006 0.0996006 -0.99003
-1.28437 2.22045e-16 0.046875 -0.141421 -0 -0.989949
-1.5 0 0.15 -1 0 0
pp 3
-1.5 0 0.15 -1 0 0
-1.065 1.065 0.15 -0.707107 0.707107 -6.97819e-16
-0.911906 0.911906 0.046875 -0.0996006 0.0996006 -0.99003
pp 3
-0.911906 -0.911906 0.046875 -0.0996006 -0.0996006 -0.99003
-1.28438 0 0.046875 -0.141421 -0 -0.989949
0 0 0 0 0 -1
pp 3
-2.22045e-16 -1.28437 0.046875 0 -0.141421 -0.989949
-0.911906 -0.911906 0.046875 -0.0996006 -0.0996006 -0.99003
0 0 0 0 0 -1
pp 3
-1.28438 0 0.046875 -0.141421 -0 -0.989949
-0.911906 -0.911906 0.046875 -0.0996006 -0.0996006 -0.99003
-1.065 -1.065 0.15 -0.707107 -0.707107 -6.97819e-16
pp 3
-1.065 -1.065 0.15 -0.707107 -0.707107 -6.97819e-16
-1.5 0 0.15 -1 0 0
-1.28438 0 0.046875 -0.141421 -0 -0.989949
pp 3
-0.911906 -0.911906 0.046875 -0.0996006 -0.0996006 -0.99003
-2.22045e-16 -1.28437 0.046875 0 -0.141421 -0.989949
0 -1.5 0.15 0 -1 0
pp 3
0 -1.5 0.15 0 -1 0
-1.065 -1.065 0.15 -0.707107 -0.707107 -6.97819e-16
-0.911906 -0.911906 0.046875 -0.0996006 -0.0996006 -0.99003
pp 3
0.911906 -0.911906 0.046875 0.0996006 -0.0996006 -0.99003
0 -1.28438 0.046875 0 -0.141421 -0.989949
0 0 0 0 0 -1
pp 3
1.28437 -2.22045e-16 0.046875 0.141421 0 -0.989949
0.911906 -0.911906 0.046875 0.0996006 -0.0996006 -0.99003
0 0 0 0 0 -1
pp 3
0 -1.28438 0.046875 0 -0.141421 -0.989949
0.911906 -0.911906 0.046875 0.0996006 -0.0996006 -0.99003
1.065 -1.065 0.15 0.707107 -0.707107 -6.97819e-16
pp 3
1.065 -1.065 0.15 0.707107 -0.707107 -6.97819e-16
0 -1.5 0.15 0 -1 0
0 -1.28438 0.046875 0 -0.141421 -0.989949
pp 3
0.911906 -0.911906 0.046875 0.0996006 -0.0996006 -0.99003
1.28437 -2.22045e-16 0.046875 0.141421 0 -0.989949
1.5 0 0.15 1 0 0
pp 3
1.5 0 0.15 1 0 0
1.065 -1.065 0.15 0.707107 -0.707107 -6.97819e-16
0.911906 -0.911906 0.046875 0.0996006 -0.0996006 -0.99003
```

# Index of

## /pubs/tog/GraphicsGems/gemsiv/curve\_isect/

Name	Last modified	Size	Description
 <a href="#">_</a> <a href="#">Parent Directory</a>			
 <a href="#">Bezier.cc</a>	29-Jun-00 08:19	12K	
 <a href="#">Bezier.h</a>	29-Jun-00 08:19	1K	
 <a href="#">README</a>	29-Jun-00 08:19	1K	
 <a href="#">makefile</a>	29-Jun-00 08:19	1K	
 <a href="#">test.cc</a>	29-Jun-00 08:19	2K	
 <a href="#">testout.ps</a>	29-Jun-00 08:19	3K	
 <a href="#">vector.h</a>	29-Jun-00 08:19	2K	

```
#include <stddef.h> // for size_t
// #include <sys/types.h> // for size_t, on some systems
#include "Bezier.h"
#include <math.h>
/* The value of 1.0 / (1L<<23) is float machine epsilon. */
#ifdef FLOAT_ACCURACY
#define INV_EPS (1L<<23)
#else
/* The value of 1.0 / (1L<<14) is enough for most applications */
#define INV_EPS (1L<<14)
#endif
#define log2(x) (log(x)/log(2.))

extern "C" void qsort( char *base, int nel, size_t width, int (*compar)(void *, void *));

int compare_doubles( void *a, void *b )
{
    register double *A = (double *)a, *B = (double *)b;
    return( *A > *B )?1:(*A < *B ? -1 : 0 );
}

void Sort( double *array, int length )
{
    qsort( (char *)array, length, sizeof( double ), compare_doubles );
}

/*
 * Split the curve at the midpoint, returning an array with the two parts
 * Temporary storage is minimized by using part of the storage for the result
 * to hold an intermediate value until it is no longer needed.
 */
#define left r[0]
#define right r[1]
Bezier *Bezier::Split()
{
    {
        Bezier *r = new Bezier[2];
        (left.p0 = p0)->refcount++;
        (right.p3 = p3)->refcount++;
        left.p1 = new point( p0, p1, 1);
        right.p2 = new point( p2, p3, 1);
        right.p1 = new point( p1, p2, 1); // temporary holding spot
        left.p2 = new point ( left.p1, right.p1, 1);
        *right.p1 = mid( right.p1, right.p2 ); // Real value this time
        left.p3 = right.p0 = new point( left.p2, right.p1, 2 );
        return r;
    }
}

#undef left
#undef right

/*
 * Test the bounding boxes of two Bezier curves for interference.
 * Several observations:
 *     First, it is cheaper to compute the bounding box of the second curve
 *     and test its bounding box for interference than to use a more direct
 *     approach of comparing all control points of the second curve with
 *     the various edges of the bounding box of the first curve to test
 *     for interference.
 *     Second, after a few subdivisions it is highly probable that two corners
 *     of the bounding box of a given Bezier curve are the first and last
 *     control point. Once this happens once, it happens for all subsequent
```

```
*      subcurves.  It might be worth putting in a test and then short-circuit
*      code for further subdivision levels.
*      Third, in the final comparison (the interference test) the comparisons
*      should both permit equality.  We want to find intersections even if they
*      occur at the ends of segments.
*      Finally, there are tighter bounding boxes that can be derived.  It isn't
*      clear whether the higher probability of rejection (and hence fewer
*      subdivisions and tests) is worth the extra work.
*/
```

```
int IntersectBB( Bezier a, Bezier b )
{
    // Compute bounding box for a
    double minax, maxax, minay, maxay;
    if( a.p0->x > a.p3->x )      // These are the most likely to be extremal
        minax = a.p3->x, maxax = a.p0->x;
    else
        minax = a.p0->x, maxax = a.p3->x;
    if( a.p2->x < minax )
        minax = a.p2->x;
    else if( a.p2->x > maxax )
        maxax = a.p2->x;
    if( a.p1->x < minax )
        minax = a.p1->x;
    else if( a.p1->x > maxax )
        maxax = a.p1->x;
    if( a.p0->y > a.p3->y )
        minay = a.p3->y, maxay = a.p0->y;
    else
        minay = a.p0->y, maxay = a.p3->y;
    if( a.p2->y < minay )
        minay = a.p2->y;
    else if( a.p2->y > maxay )
        maxay = a.p2->y;
    if( a.p1->y < minay )
        minay = a.p1->y;
    else if( a.p1->y > maxay )
        maxay = a.p1->y;
    // Compute bounding box for b
    double minbx, maxbx, minby, maxby;
    if( b.p0->x > b.p3->x )
        minbx = b.p3->x, maxbx = b.p0->x;
    else
        minbx = b.p0->x, maxbx = b.p3->x;
    if( b.p2->x < minbx )
        minbx = b.p2->x;
    else if( b.p2->x > maxbx )
        maxbx = b.p2->x;
    if( b.p1->x < minbx )
        minbx = b.p1->x;
    else if( b.p1->x > maxbx )
        maxbx = b.p1->x;
    if( b.p0->y > b.p3->y )
        minby = b.p3->y, maxby = b.p0->y;
    else
        minby = b.p0->y, maxby = b.p3->y;
    if( b.p2->y < minby )
        minby = b.p2->y;
    else if( b.p2->y > maxby )
        maxby = b.p2->y;
    if( b.p1->y < minby )
        minby = b.p1->y;
```

```
else if( b.pl->y > maxby )
    maxby = b.pl->y;
// Test bounding box of b against bounding box of a
if( ( minax > maxbx ) || ( minay > maxby ) // Not >= : need boundary case
    || ( minbx > maxax ) || ( minby > maxay ) )
    return 0; // they don't intersect
else
    return 1; // they intersect
}
```

```
/*
* Recursively intersect two curves keeping track of their real parameters
* and depths of intersection.
* The results are returned in a 2-D array of doubles indicating the parameters
* for which intersections are found. The parameters are in the order the
* intersections were found, which is probably not in sorted order.
* When an intersection is found, the parameter value for each of the two
* is stored in the index elements array, and the index is incremented.
*
* If either of the curves has subdivisions left before it is straight
* (depth > 0)
* that curve (possibly both) is (are) subdivided at its (their) midpoint(s).
* the depth(s) is (are) decremented, and the parameter value(s) corresponding
* to the midpoints(s) is (are) computed.
* Then each of the subcurves of one curve is intersected with each of the
* subcurves of the other curve, first by testing the bounding boxes for
* interference. If there is any bounding box interference, the corresponding
* subcurves are recursively intersected.
*
* If neither curve has subdivisions left, the line segments from the first
* to last control point of each segment are intersected. (Actually the
* only the parameter value corresponding to the intersection point is found).
*
* The apriori flatness test is probably more efficient than testing at each
* level of recursion, although a test after three or four levels would
* probably be worthwhile, since many curves become flat faster than their
* asymptotic rate for the first few levels of recursion.
*
* The bounding box test fails much more frequently than it succeeds, providing
* substantial pruning of the search space.
*
* Each (sub)curve is subdivided only once, hence it is not possible that for
* one final line intersection test the subdivision was at one level, while
* for another final line intersection test the subdivision (of the same curve)
* was at another. Since the line segments share endpoints, the intersection
* is robust: a near-tangential intersection will yield zero or two
* intersections.
*/
```

```
void RecursivelyIntersect( Bezier a, double t0, double t1, int deptha,
                          Bezier b, double u0, double u1, int depthb,
                          double **parameters, int &index )
{
    if( deptha > 0 )
    {
        Bezier *A = a.Split();
        double tmid = (t0+t1)*0.5;
        deptha--;
        if( depthb > 0 )
        {
            Bezier *B = b.Split();
            double umid = (u0+u1)*0.5;
```

```

    depthb--;
    if( IntersectBB( A[0], B[0] ) )
        RecursivelyIntersect( A[0], t0, tmid, deptha,
                               B[0], u0, umid, depthb,
                               parameters, index );
    if( IntersectBB( A[1], B[0] ) )
        RecursivelyIntersect( A[1], tmid, t1, deptha,
                               B[0], u0, umid, depthb,
                               parameters, index );
    if( IntersectBB( A[0], B[1] ) )
        RecursivelyIntersect( A[0], t0, tmid, deptha,
                               B[1], umid, u1, depthb,
                               parameters, index );
    if( IntersectBB( A[1], B[1] ) )
        RecursivelyIntersect( A[1], tmid, t1, deptha,
                               B[1], umid, u1, depthb,
                               parameters, index );
}
else
{
    if( IntersectBB( A[0], b ) )
        RecursivelyIntersect( A[0], t0, tmid, deptha,
                               b, u0, u1, depthb,
                               parameters, index );
    if( IntersectBB( A[1], b ) )
        RecursivelyIntersect( A[1], tmid, t1, deptha,
                               b, u0, u1, depthb,
                               parameters, index );
}
}
else
    if( depthb > 0 )
    {
        Bezier *B = b.Split();
        double umid = (u0 + u1)*0.5;
        depthb--;
        if( IntersectBB( a, B[0] ) )
            RecursivelyIntersect( a, t0, t1, deptha,
                                   B[0], u0, umid, depthb,
                                   parameters, index );
        if( IntersectBB( a, B[1] ) )
            RecursivelyIntersect( a, t0, t1, deptha,
                                   B[0], umid, u1, depthb,
                                   parameters, index );
    }
else // Both segments are fully subdivided; now do line segments
{
    double xlk = a.p3->x - a.p0->x;
    double ylk = a.p3->y - a.p0->y;
    double xnm = b.p3->x - b.p0->x;
    double ynm = b.p3->y - b.p0->y;
    double xmk = b.p0->x - a.p0->x;
    double ymk = b.p0->y - a.p0->y;
    double det = xnm * ylk - ynm * xlk;
    if( 1.0 + det == 1.0 )
        return;
    else
    {
        double detinv = 1.0 / det;
        double s = ( xnm * ymk - ynm * xmk ) * detinv;
        double t = ( xlk * ymk - ylk * xmk ) * detinv;
    }
}

```



```
        if( ( s < 0.0 ) || ( s > 1.0 ) || ( t < 0.0 ) || ( t > 1.0 ) )
            return;
        parameters[0][index] = t0 + s * ( t1 - t0 );
        parameters[1][index] = u0 + t * ( u1 - u0 );
        index++;
    }
}
```

```
inline double log4( double x ) { return 0.5 * log2( x ); }
```

```
/*
 * Wang's theorem is used to estimate the level of subdivision required,
 * but only if the bounding boxes interfere at the top level.
 * Assuming there is a possible intersection, RecursivelyIntersect is
 * used to find all the parameters corresponding to intersection points.
 * these are then sorted and returned in an array.
 */
```

```
double ** FindIntersections( Bezier a, Bezier b )
{
    double **parameters = new double *[2];
    parameters[0] = new double[9];
    parameters[1] = new double[9];
    int index = 0;
    if( IntersectBB( a, b ) )
    {
        vector la1 = vabs( ( *(a.p2) - *(a.p1) ) - ( *(a.p1) - *(a.p0) ) );
        vector la2 = vabs( ( *(a.p3) - *(a.p2) ) - ( *(a.p2) - *(a.p1) ) );
        vector la;
        if( la1.x > la2.x ) la.x = la1.x; else la.x = la2.x;
        if( la1.y > la2.y ) la.y = la1.y; else la.y = la2.y;
        vector lb1 = vabs( ( *(b.p2) - *(b.p1) ) - ( *(b.p1) - *(b.p0) ) );
        vector lb2 = vabs( ( *(b.p3) - *(b.p2) ) - ( *(b.p2) - *(b.p1) ) );
        vector lb;
        if( lb1.x > lb2.x ) lb.x = lb1.x; else lb.x = lb2.x;
        if( lb1.y > lb2.y ) lb.y = lb1.y; else lb.y = lb2.y;
        double l0;
        if( la.x > la.y )
            l0 = la.x;
        else
            l0 = la.y;
        int ra;
        if( l0 * 0.75 * M_SQRT2 + 1.0 == 1.0 )
            ra = 0;
        else
            ra = (int)ceil( log4( M_SQRT2 * 6.0 / 8.0 * INV_EPS * l0 ) );
        if( lb.x > lb.y )
            l0 = lb.x;
        else
            l0 = lb.y;
        int rb;
        if( l0 * 0.75 * M_SQRT2 + 1.0 == 1.0 )
            rb = 0;
        else
            rb = (int)ceil( log4( M_SQRT2 * 6.0 / 8.0 * INV_EPS * l0 ) );
        RecursivelyIntersect( a, 0., 1., ra, b, 0., 1., rb, parameters, index );
    }
    if( index < 9 )
        parameters[0][index] = parameters[1][index] = -1.0;
    Sort( parameters[0], index );
}
```

```
Sort( parameters[1], index );
return parameters;
}
```

```
void
Bezier::ParameterSplitLeft( double t, Bezier &left )
{
    left.p0 = p0;
    left.p1 = new point( *p0 + t * ( *p1 - *p0 ) );
    left.p2 = new point( *p1 + t * ( *p2 - *p1 ) ); // temporary holding spot
    p2->refcount--;
    p2 = new point( *p2 + t * ( *p3 - *p2 ) );
    p1->refcount--;
    p1 = new point( *(left.p2) + t * ( *p2 - *(left.p2) ) );
    *(left.p2) = ( *(left.p1) + t * ( *(left.p2) - *(left.p1) ) );
    (left.p3 = p0 = new point(*(left.p2) + t * (*(p1)-*(left.p2))))->refcount++;
    left.p0->refcount++; left.p1->refcount++;
    left.p2->refcount++; left.p3->refcount++;
}
```

```
/*
 * Intersect two curves, returning an array of two arrays of curves.
 * The first array of curves corresponds to 'this' curve, the second
 * corresponds to curve B, passed in.
 * The intersection parameter values are computed by FindIntersections,
 * and they come back in the range 0..1, using the original parameterization.
 * Once one segment has been removed, ie the curve is split at splitT, the
 * parameterization of the second half is from 0..1, so the parameter for
 * the next split point, if any, must be adjusted.
 * If we split at t[i], the split point at t[i+1] is
 * ( t[i+1] - t[i] ) / ( t - t[i] ) of the way to the end from the new
 * start point.
 */
```

```
Bezier **Bezier::Intersect( Bezier B )
{
    // The return from FindIntersections will decrement all refcounts.
    // (a c++-ism)
    B.p0->refcount++; B.p1->refcount++; B.p2->refcount++; B.p3->refcount++;
    Bezier **rvalue = new Bezier *[2];
    rvalue[0] = new Bezier[10];
    rvalue[1] = new Bezier[10];
    double **t = FindIntersections( *this, B );
    int index = 0;
    if( t[0][0] > -0.5 )
    {
        ParameterSplitLeft( t[0][0], rvalue[0][0] );
        B.ParameterSplitLeft( t[1][0], rvalue[1][0] );
        index++;
        while( t[0][index] > -0.5 && index < 9 )
        {
            double splitT = (t[0][index] - t[0][index-1])/(1.0 - t[0][index-1]);
            ParameterSplitLeft( splitT, rvalue[0][index] );
            splitT = (t[1][index] - t[1][index-1])/(1.0 - t[1][index-1]);
            B.ParameterSplitLeft( splitT, rvalue[1][index] );
            index++;
        }
    }
    rvalue[0][index] = *this;
    rvalue[1][index] = B;
    return rvalue;
}
```

}

```
#ifndef _BEZIER_INCLUDED_
#include "vector.h"
class Bezier {
public:
    point *p0, *p1, *p2, *p3;
    Bezier()
    {
        p0 = 0; p1 = 0; p2 = 0; p3 = 0;
    }
    Bezier( point *_p0, point *_p1, point *_p2, point *_p3 )
    {
        p0 = _p0; p1 = _p1; p2 = _p2; p3 = _p3;
    }
    Bezier * Split( );
    void ParameterSplitLeft( double t, Bezier &result );

    // Intersect with another curve.  Return two 10-elt arrays. Array 0
    // contains fragments of self. Array 1 contains fragments of other curve.
    // Fragments continue until one with nil pointers pointing at point data.
    Bezier **Intersect( Bezier B );
    ~Bezier()
    {
        if( --p0->refcount <= 0 ) delete p0;
        if( --p1->refcount <= 0 ) delete p1;
        if( --p2->refcount <= 0 ) delete p2;
        if( --p3->refcount <= 0 ) delete p3;
    }
};
#define _BEZIER_INCLUDED_
#endif
```

C++ code from the article

"Intersecting Parametric Cubic Curves by Midpoint Subdivision"

by R. Victor Klassen, [klassen.wbst128@xerox.com](mailto:klassen.wbst128@xerox.com)

in "Graphics Gems IV", Academic Press, 1994

files:

Bezier.cc	- C++ source for cubic Bezier curve intersection
Bezier.h	- header file for "
makefile	
test.cc	- test program that prints out Postscript of curve intersection
testout.ps	- color Postscript output of test program
vector.h	- floating point vector library

[http://www.acm.org/pubs/tog/GraphicsGems/gemsiv/curve\\_isect/makefile](http://www.acm.org/pubs/tog/GraphicsGems/gemsiv/curve_isect/makefile)

```
test: Bezier.cc Bezier.h vector.h test.cc  
      CC Bezier.cc test.cc -o test -lm  
      ./test | diff - testout.ps
```

```
#include "Bezier.h"
#include <stdio.h>

#define H 1200.0
#define STEPS 3
main()
{
    point Origin = point( 0, 0 );
    point p0 = Origin;
    point p1 = Origin + vector( H, H/2 );
    point p2 = Origin + vector( -7*H/10, 2*H/10 );
    point p3 = Origin + vector( 4*H/10, 0 );
    point q0 = Origin + vector( 5*H/100, 0 );
    point q1 = Origin + vector( 35*H/100, H );
    point q2 = Origin + vector( 25*H/100, -7*H/10 );
    point q3 = Origin + vector( 0, 4*H/10 );
    puts( "!\n0 setlinewidth\n" );
    puts( "0 1 0 setrgbcolor\n" );
    int rg = 0;
    Bezier A = Bezier( &p0, &p1, &p2, &p3 );
    Bezier B = Bezier( &q0, &q1, &q2, &q3 );
    printf( "%g %g scale\n", 72 * 8.0 / H, 72 * 10.0/H );
    printf( "%g %g translate\n", H*0.1, H*0.1 );
    printf( "/rad 5 def\n" );
    Bezier **curves = B.Intersect( A );
    for( int i = 0; i < 10; i++ )
    {
        if( curves[0][i].p0 == NULL )
        {
            break;
        }
        printf( "%d %d 0 setrgbcolor\n", rg, 1-rg );
        rg = 1-rg;
        printf( "%15.9f %15.9f moveto\n",
            curves[0][i].p0->x, curves[0][i].p0->y );
        printf( "%15.9f %15.9f %15.9f %15.9f %15.9f %15.9f curveto stroke\n",
            curves[0][i].p1->x, curves[0][i].p1->y,
            curves[0][i].p2->x, curves[0][i].p2->y,
            curves[0][i].p3->x, curves[0][i].p3->y );
        printf( "%15.9f %15.9f moveto\n",
            curves[1][i].p0->x, curves[1][i].p0->y );
        printf( "%15.9f %15.9f %15.9f %15.9f %15.9f %15.9f curveto stroke\n",
            curves[1][i].p1->x, curves[1][i].p1->y,
            curves[1][i].p2->x, curves[1][i].p2->y,
            curves[1][i].p3->x, curves[1][i].p3->y );
    }
    puts( "showpage\n" );

#   ifdef PROFILE
    for( i = 0; i < STEPS; i++ )
    {
        p0 = p0 + vector( H/STEPS, H/STEPS );
        for( int j = 0; j < STEPS; j++ )
        {
            p1 = p1 + vector( -5*H/STEPS, H/(3*STEPS) );
            for( int k = 0; k < STEPS; k++ )
            {
                p2 = p2 + vector( 5*H/STEPS, -H/(3*STEPS) );
                for( int l = 0; l < STEPS; l++ )
                {
                    p3 = p3 + vector( -H/STEPS, -H/STEPS );
                }
            }
        }
    }
#   endif
}
```

```

        for( int m = 0; m < STEPS; m++ )
        {
            q0 = q0 + vector( H/STEPS, H/STEPS );
            for( int n = 0; n < STEPS; n++ )
            {
                q1 = q1 + vector( H/(3*STEPS), -2*H/STEPS );
                for( int o = 0; o < STEPS; o++ )
                {
                    q2 = q2 + vector( -H/(3*STEPS), -2*H/STEPS );
                    curves = A.Intersect( B );
                }
            }
        }
    }
}

# endif
}

```



```
// Floating point vector library
#ifndef VECTORS_INCLUDED__
#include <math.h>
class point
{
private:
int refcount;
public:
double x, y;
point()
{
refcount = 0;
}
point( const double _x, const double _y )
{
x = _x, y = _y;
refcount = 1;
}
point( const point &p )
{
x = p.x, y = p.y; refcount = 1;
}
point( const point &p, const int count )
{
x = p.x, y = p.y; refcount = count;
}
point( const point *a, const point *b, const int count )
{
x = ( a->x + b->x ) * 0.5;
y = ( a->y + b->y ) * 0.5;
refcount = count;
}
friend class Bezier;
}
;

class vector
{
public:
double x, y;
vector()
{
;
}
vector( const vector &v )
{
x = v.x, y = v.y;
}
vector( const double _x, const double _y )
{
x = _x, y = _y;
}
}
;

inline vector operator-(const point a, const point b ) // p - p = v
{
return vector( a.x - b.x, a.y - b.y );
}

inline vector operator-(const vector a, const vector b ) // v - v = v
```

```
{
return vector( a.x - b.x, a.y - b.y );
}

inline point operator+(const point *a, const vector b ) // p + v = p
{
return point( a->x + b.x, a->y + b.y );
}

inline point operator+(const point a, const vector b ) // p + v = p
{
return point( a.x + b.x, a.y + b.y );
}

inline vector operator+(const vector a, const vector b ) // v + v = v
{
return vector( a.x + b.x, a.y + b.y );
}

inline vector operator*(const double s, const vector v ) // sv = v
{
return vector( s * v.x, s * v.y );
}

inline vector operator*(const vector v, const double s) // v s = v
{
return vector( s * v.x, s * v.y );
}

inline double operator*(const vector a, const vector b) // v * v = s (dot product)
{
return a.x * b.x + a.y * b.y;
}





inline point mid( const point *a, const point *b )
{
return point( ( a->x + b->x ) * 0.5, ( a->y + b->y ) * 0.5 );
}

inline vector vabs( const vector a )
{
return vector( fabs( a.x ), fabs( a.y ) );
}

;
#define VECTORS_INCLUDED__
#endif
```

# Index of

## /pubs/tog/GraphicsGems/gemsv/ch5-4/

Name	Last modified	Size	Description
 <a href="#">_Parent Directory</a>			
 <a href="#">_poly.cpp</a>	29-Jun-00 08:23	3K	
 <a href="#">_poly.h</a>	29-Jun-00 08:23	1K	
 <a href="#">_sweep.cpp</a>	29-Jun-00 08:23	3K	

```

/*****
 * POLY.CPP
 * Andreas Leipelt, "Ray Tracing a Swept Sphere"
 * from "Graphics Gems", Academic Press
 *
 * Implementation of the polynomial class. The code is
 * not complete ! You need to insert a root solver in
 * the method 'root_between' .
 */

#include <math.h>
#include "poly.h"

// constructor of the polynomial class
polynomial::polynomial()
{
    deg = 0;
    for (double *fp = &coef[MAX_DEGREE]; fp >= coef; fp--) *fp = 0.0;
}

// evaluates the polynomial with Horner's scheme.
double polynomial::eval(double x)
{
    double *fp = &coef[deg], val;
    for (val = *fp--; fp >= coef; fp--) val = val*x + *fp;
    return val;
}

// returns the first derivative of the polynomial.
polynomial polynomial::derivative()
{
    polynomial ret;

    if (!deg) return ret;
    ret.deg = deg-1;
    for (int i=0; i <= ret.deg; i++) ret.coef[i] = (i+1)*coef[i+1];
    return ret;
}

// returns the absolute minimum of the given polynomial in the
// interval [a , b]
double polynomial::min(double a, double b)
{
    double roots[MAX_DEGREE], tmp, Min = eval(a);

    int n = derivative().roots_between(a, b, roots);
    roots[n] = b;
    for (int i=0; i <= n; i++) {
        tmp = eval(roots[i]);
        if (tmp < Min) Min = tmp;
    }
    return Min;
}

// returns the absolute maximum of the given polynomial in the
// interval [a ; b]
double polynomial::max(double a, double b)
{
    double roots[MAX_DEGREE], tmp, Max = eval(a);

    int n = derivative().roots_between(a, b, roots);
```

```
    roots[n] = b;
    for (int i=0; i <= n; i++) {
        tmp = eval(roots[i]);
        if (tmp > Max) Max = tmp;
    }
    return Max;
}

int polynomial::roots_between(double a, double b, double *roots)
{
    // This function should return the number of roots between
    // a and b and the array 'roots' should contain these roots.
    // Refer to Hook and McAree, "Using Sturm Sequences to Bracket
    // Real Roots of Polynomial Equations" in "Graphics Gems I"
    return 0;
}

polynomial operator+(polynomial& p, polynomial& q)
{
    polynomial sum;

    if (p.deg < q.deg) sum.deg = q.deg;
    else sum.deg = p.deg;
    for (int i=0; i <= sum.deg; i++)
        sum.coef[i] = p.coef[i] + q.coef[i];
    if (p.deg == q.deg) {
        while (sum.deg > -1 && fabs(sum.coef[sum.deg]) < polyeps)
            sum.coef[sum.deg--] = 0.0;
        if (sum.deg < 0) sum.deg = 0;
    }
    return sum;
}

polynomial operator-(polynomial& p, polynomial& q)
{
    polynomial dif;

    if (p.deg < q.deg) dif.deg = q.deg;
    else dif.deg = p.deg;
    for (int i=0; i <= dif.deg; i++)
        dif.coef[i] = p.coef[i] - q.coef[i];
    if (p.deg == q.deg) {
        while (dif.deg > -1 && fabs(dif.coef[dif.deg]) < polyeps)
            dif.coef[dif.deg--] = 0.0;
        if (dif.deg < 0) dif.deg = 0;
    }
    return dif;
}

polynomial operator*(polynomial& p, polynomial& q)
{
    polynomial prod;

    prod.deg = p.deg + q.deg;
    for (int i=0; i <= p.deg; i++)
        for (int j=0; j <= q.deg; j++)
            prod.coef[i+j] += p.coef[i]*q.coef[j];
    return prod;
}

polynomial operator*(double s, polynomial& p)
```

```
{
    polynomial scale;

    if (s == 0.0) return scale;
    scale.deg = p.deg;
    for (int i=0; i <= p.deg; i++) scale.coef[i] = s*p.coef[i];
    return scale;
}
```

```

/*****
 *   POLY.H
 *   Andreas Leipelt, "Ray Tracing a Swept Sphere"
 *   from "Graphics Gems", Academic Press
 *
 */

#ifndef POLY_CLASS
#define POLY_CLASS

#define MAX_DEGREE 10
#define polyeps 1E-10  // tolerance for polynomial coefficients

class polynomial {
public:

    int deg;
    double coef[MAX_DEGREE+1];

    polynomial();
    double    eval(double);
    int      roots_between(double,double,double*);
    double    min(double,double);
    double    max(double,double);
    polynomial derivative();
};

polynomial operator+(polynomial&, polynomial&);
polynomial operator-(polynomial&, polynomial&);
polynomial operator*(polynomial&, polynomial&);
polynomial operator*(double, polynomial&);

#endif
```

```

/*****
 *   SWEEP.CPP
 *   Andreas Leipelt, "Ray Tracing a Swept Sphere"
 *   from "Graphics Gems", Academic Press
 *
 *   This file contains the code to handle a swept sphere in
 *   ray tracing
 */

#include <math.h>
#include "poly.h"

#define rayeps    1E-8    // tolerance for intersection test

// refer to Andrew Woo, "Fast Ray-Box Intersection",
// "Graphics Gems I"
extern char HitBoundingBox(double*,double*,double*,double*);

// class of the swept sphere primitive
class swept_sph {
    polynomial m[3]; // center of the sphere
    polynomial r;    // radius of the sphere
    polynomial r2;   // r2 = r*r
    double a, b;     // the interval [a;b], where m and r live
    double minB[3], // lower left corner of the bounding box
           maxB[3]; // upper right corner of the bounding box
    double param;    // parameter of last intersection, used for member
                   // 'normal'

public:

    swept_sph() {}
    swept_sph(polynomial*,polynomial,double,double);
    int intersect(double*,double*,double*);
    void normal(double*,double*);
    int inside(double*);
};

// constructor of the swept_sph-class
swept_sph::swept_sph(polynomial *M, polynomial R, double A, double B)
// M : trajectory of the center of the moving sphere.
//     An array of polynomials, which is interpreted as a
//     vector valued polynomial.
// R : varying radius of the moving sphere. The radius is assumed
//     to be non-negative.
{
    for (int i=0; i < 3; i++) m[i] = M[i];
    r = R;
    r2 = r*r;
    a = A; b = B;
    // Calculate the axis aligned bounding box
    for (i=0; i < 3; i++) {
        minB[i] = (m[i] - r).min(a, b);
        maxB[i] = (m[i] + r).max(a, b);
    }
}

int swept_sph::intersect(double *origin, double *dir, double *l)
// origin : origin of the ray
// dir     : unit direction of the ray
// t       : intersection parameter of the ray
{

```



```
polynomial p, q, dp, dq, s;
double save[3];
double roots[MAX_DEGREE];
double p_val, q_val, D, test;

if (!HitBoundingBox(minB, maxB, origin, dir)) return 0;
// save the constant term of the trajectory
for (int i=0; i < 3; i++) {
    save[i] = m[i].coef[0];
    m[i].coef[0] -= origin[i];
}
p = dir[0]*m[0] + dir[1]*m[1] + dir[2]*m[2];
q = m[0]*m[0] + m[1]*m[1] + m[2]*m[2] - r2;
dp = p.derivative();
dq = q.derivative();
s = dq*dq + 4.0*dp*(dp*q - p*dq);
int n = s.roots_between(a, b, roots);
roots[n++] = a;
roots[n] = b;
*1 = 1E20;
// test all possible values
for (i=0; i <= n; i++) {
    // calculate the real solutions of the equation
    // 1 = p_val +/- sqrt(p_val*p_val - q_val)
    p_val = p.eval(roots[i]);
    q_val = q.eval(roots[i]);
    D = p_val*p_val - q_val;
    if (D >= 0.0) {
        // check, if the candidate roots[i] leads to a better
        // intersection value 1
        D = sqrt(D);
        test = p_val - D;
        if (test < rayeps) test = p_val + D;
        if ((test >= rayeps) && (test < *1)) {
            param = roots[i];
            *1 = test;
        }
    }
}
// restore the constant term of the trajectory
for (i=0; i < 3; i++) m[i].coef[0] = save[i];
if (*1 < 1E20) return 1;
else return 0;
}

void swept_sph::normal(double *IP, double* Nrm)
// IP : intersection point
// Nrm : normal at IP
{
    double R = r.eval(param);
    // if the radius is zero, return an arbitrary normal.
    if (R < polyeps) {
        Nrm[0] = Nrm[1] = 0.0;
        Nrm[2] = 1.0;
        return;
    }
    for (int i=0; i < 3; i++) Nrm[i] = (IP[i] - m[i].eval(param))/R;
}

// returns 1, if the point P lies inside.
int swept_sph::inside(double *P)
```

```
{
    double save[3];
    int is_inside;

    for (int i=0; i < 3; i++) {
        save[i] = m[i].coef[0];
        m[i].coef[0] -= P[i];
    };
    is_inside =
        ((m[0]*m[0]+m[1]*m[1]+m[2]*m[2]-r2).min(a, b) < rayeps);
    for (i=0; i < 3; i++) m[i].coef[0] = save[i];
    return is_inside;
}
```

/\*\*\*\*\*

This is C++ code. Given here are skeleton class definitions for the control points (ControlPoint), quadratic and cubic Bezier triangles (BezierTri2 and BezierTri3, respectively), and for the bi-quadratic and bi-cubic Bezier patches (BezierRect2 and BezierRect3). The conversion described in the gem takes place in the constructors provided for BezierRect2 and BezierRect3, which each take a Bezier triangle (of appropriate degree) as an argument.

Note that the ControlPoint does not have to be to be an (x,y,z) triplet. For instance, it can be a scalar, an RGB triplet, etc., as long as the operators +, \*, /, and = (assignment) are provided. If you have a class that you wish to use instead of the one given in the code, all you have to do is to remove the definitions of the ControlPoint class and its operators, and insert instead something like:

```
#include <my_class.h>
typedef MyClass ControlPoint;
```

\*\*\*\*\*/

```
class ControlPoint {
    friend ControlPoint operator+(const ControlPoint&, const ControlPoint&);
    friend ControlPoint operator*(float, const ControlPoint&);
    friend ControlPoint operator/(const ControlPoint&, float);
private:
    float x, y, z;
public:
    ControlPoint() {}
    ControlPoint(float a, float b, float c)    { x = a; y = b; z = c; }
};
```

```
ControlPoint operator+(const ControlPoint& a, const ControlPoint& b)
{
    return ControlPoint(a.x + b.x, a.y + b.y, a.z + b.z);
}
```

```
ControlPoint operator*(float c, const ControlPoint& a)
{
    return ControlPoint(c * a.x, c * a.y, c * a.z);
}
```

```
ControlPoint operator/(const ControlPoint& a, float c)
{
    return ControlPoint(a.x / c, a.y / c, a.z / c);
}
```

```
class BezierTri2 {
private:
    ControlPoint cp[6];
public:
    const ControlPoint& b(int, int, int) const;
};
```

```
const ControlPoint& BezierTri2::b(int i, int j, int /* k */) const
{
    static int row_start[3] = {0, 3, 5};
    return cp[row_start[j] + i];
}
```

```
class BezierTri3 {
private:
    ControlPoint cp[10];
public:
    const ControlPoint& b(int, int, int) const;
};

const ControlPoint& BezierTri3::b(int i, int j, int /* k */) const
{
    static int row_start[4] = {0, 4, 7, 9};
    return cp[row_start[j] + i];
}








class BezierRect2 {
private:
    ControlPoint cp[3][3];
public:
    BezierRect2(const BezierTri2&);
    ControlPoint& p(int i, int j)      { return cp[i][j]; }
};

BezierRect2::BezierRect2(const BezierTri2& bt)
// convert a quadratic triangle into a bi-quadratic patch
{
    p(0,0) = bt.b(0,0,2);
    p(0,1) = bt.b(1,0,1);
    p(0,2) = bt.b(2,0,0);
    p(1,0) = bt.b(0,1,1);
    p(1,1) = (bt.b(0,1,1) + bt.b(1,1,0)) / 2;
    p(1,2) = bt.b(1,1,0);
    p(2,0) = p(2,1) = p(2,2) = bt.b(0,2,0);
}

class BezierRect3 {
private:
    ControlPoint cp[4][4];
public:
    BezierRect3(const BezierTri3&);
    ControlPoint& p(int i, int j)      { return cp[i][j]; }
};

BezierRect3::BezierRect3(const BezierTri3& bt)
// convert a cubic triangle into a bi-cubic patch
{
    p(0,0) = bt.b(0,0,3);
    p(0,1) = bt.b(1,0,2);
    p(0,2) = bt.b(2,0,1);
    p(0,3) = bt.b(3,0,0);
    p(1,0) = bt.b(0,1,2);
    p(1,1) = (bt.b(0,1,2) + 2*bt.b(1,1,1)) / 3;
    p(1,2) = (bt.b(2,1,0) + 2*bt.b(1,1,1)) / 3;
    p(1,3) = bt.b(2,1,0);
    p(2,0) = bt.b(0,2,1);
    p(2,1) = (bt.b(1,2,0) + 2*bt.b(0,2,1)) / 3;
    p(2,2) = (bt.b(0,2,1) + 2*bt.b(1,2,0)) / 3;
    p(2,3) = bt.b(1,2,0);
    p(3,0) = p(3,1) = p(3,2) = p(3,3) = bt.b(0,3,0);
}
```

# Index of /pubs/tog/GraphicsGems/gemsiv/delaunay/

Name	Last modified	Size	Description
 <a href="#">Parent Directory</a>			
 <a href="#">Makefile</a>	29-Jun-00 08:19	1K	
 <a href="#">README</a>	29-Jun-00 08:19	1K	
 <a href="#">geom2d.h</a>	29-Jun-00 08:19	3K	
 <a href="#">quadedge.C</a>	29-Jun-00 08:19	6K	
 <a href="#">quadedge.h</a>	29-Jun-00 08:19	3K	
 <a href="#">test.C</a>	29-Jun-00 08:19	3K	

```
delaunay: test.o quadedge.o
        CC -o delaunay test.o quadedge.o -lgl_s -lX11 -lm

test.o: geom2d.h quadedge.h test.C
        CC -I. -c test.C

quadedge.o: geom2d.h quadedge.h quadedge.C
        CC -I. -c quadedge.C
```

C++ code from the article  
"Incremental Delaunay Triangulation"  
by Dani Lischinski, [danix@graphics.cornell.edu](mailto:danix@graphics.cornell.edu)  
in "Graphics Gems IV", Academic Press, 1994

test.C contains an interactive test program for Delaunay triangulation.  
The program should compile and run on SGI workstations.  
Use left-mouse to add new points.

The rest of the code is more portable (not limited to SGIs).

```
#ifndef GEOM2D_H
#define GEOM2D_H

#include <math.h>
#include <iostream.h>

#ifndef ABS
#define ABS(a) ((a) >= 0 ? (a) : -(a))
#endif

#ifndef MAX
#define MAX(a, b) ((a) >= (b) ? (a) : (b))
#define MIN(a, b) ((a) <= (b) ? (a) : (b))
#endif

#ifndef TRUE
#define FALSE 0
#define TRUE 1
#endif

#define EPS 1e-6

typedef double Real;

class Vector2d {
public:
    Real x, y;
    Vector2d() { x = 0; y = 0; }
    Vector2d(Real a, Real b) { x = a; y = b; }
    Real norm() const;
    void normalize();
    Vector2d operator+(const Vector2d&) const;
    Vector2d operator-(const Vector2d&) const;
    friend Vector2d operator*(Real, const Vector2d&);
    friend Real dot(const Vector2d&, const Vector2d&);
    friend istream& operator>>(istream&, Vector2d&);
    friend ostream& operator<<(ostream&, const Vector2d&);
};

class Point2d {
public:
    Real x, y;
    Point2d() { x = 0; y = 0; }
    Point2d(Real a, Real b) { x = a; y = b; }
    Point2d(const Point2d& p) { *this = p; }
    Point2d operator+(const Vector2d&) const;
    Vector2d operator-(const Point2d&) const;
    int operator==(const Point2d&) const;
    friend istream& operator>>(istream&, Point2d&);
    friend ostream& operator<<(ostream&, const Point2d&);
};

class Line {
public:
    Line() {}
    Line(const Point2d&, const Point2d&);
    Real eval(const Point2d&) const;
    int classify(const Point2d&) const;
private:
    Real a, b, c;
};
```



```
// Vector2d:

inline Real Vector2d::norm() const
{
    return sqrt(x * x + y * y);
}

inline void Vector2d::normalize()
{
    Real len;

    if ((len = sqrt(x * x + y * y)) == 0.0)
        cerr << "Vector2d::normalize: Division by 0\n";
    else {
        x /= len;
        y /= len;
    }
}

inline Vector2d Vector2d::operator+(const Vector2d& v) const
{
    return Vector2d(x + v.x, y + v.y);
}

inline Vector2d Vector2d::operator-(const Vector2d& v) const
{
    return Vector2d(x - v.x, y - v.y);
}

inline Vector2d operator*(Real c, const Vector2d& v)
{
    return Vector2d(c * v.x, c * v.y);
}

inline Real dot(const Vector2d& u, const Vector2d& v)
{
    return u.x * v.x + u.y * v.y;
}

inline ostream& operator<<(ostream& os, const Vector2d& v)
{
    os << '(' << v.x << ", " << v.y << ')';
    return os;
}

inline istream& operator>>(istream& is, Vector2d& v)
{
    is >> v.x >> v.y;
    return is;
}

// Point2d:

inline Point2d Point2d::operator+(const Vector2d& v) const
{
    return Point2d(x + v.x, y + v.y);
}

inline Vector2d Point2d::operator-(const Point2d& p) const
{

```

```
        return Vector2d(x - p.x, y - p.y);
    }

inline int Point2d::operator==(const Point2d& p) const
{
    return ((*this - p).norm() < EPS);
}

inline istream& operator>>(istream& is, Point2d& p)
{
    is >> p.x >> p.y;
    return is;
}

inline ostream& operator<<(ostream& os, const Point2d& p)
{
    os << '(' << p.x << ", " << p.y << ')';
    return os;
}

// Line:

inline Line::Line(const Point2d& p, const Point2d& q)
// Computes the normalized line equation through the
// points p and q.
{
    Vector2d t = q - p;
    Real len = t.norm();
    a = t.y / len;
    b = - t.x / len;
    c = -(a*p.x + b*p.y);
}

inline Real Line::eval(const Point2d& p) const
// Plugs point p into the line equation.
{
    return (a * p.x + b* p.y + c);
}

inline int Line::classify(const Point2d& p) const
// Returns -1, 0, or 1, if p is to the left of, on,
// or right of the line, respectively.
{
    Real d = eval(p);
    return (d < -EPS) ? -1 : (d > EPS ? 1 : 0);
}

#endif
```

```
#include <quadedge.h>

/***** Basic Topological Operators *****/

Edge* MakeEdge()
{
    QuadEdge *ql = new QuadEdge;
    return ql->e;
}

void Splice(Edge* a, Edge* b)
// This operator affects the two edge rings around the origins of a and b,
// and, independently, the two edge rings around the left faces of a and b.
// In each case, (i) if the two rings are distinct, Splice will combine
// them into one; (ii) if the two are the same ring, Splice will break it
// into two separate pieces.
// Thus, Splice can be used both to attach the two edges together, and
// to break them apart. See Guibas and Stolfi (1985) p.96 for more details
// and illustrations.
{
    Edge* alpha = a->Onext()->Rot();
    Edge* beta  = b->Onext()->Rot();

    Edge* t1 = b->Onext();
    Edge* t2 = a->Onext();
    Edge* t3 = beta->Onext();
    Edge* t4 = alpha->Onext();

    a->next = t1;
    b->next = t2;
    alpha->next = t3;
    beta->next = t4;
}

void DeleteEdge(Edge* e)
{
    Splice(e, e->Oprev());
    Splice(e->Sym(), e->Sym()->Oprev());
    delete e->Qedge();
}

/***** Topological Operations for Delaunay Diagrams *****/

Subdivision::Subdivision(const Point2d& a, const Point2d& b, const Point2d& c)
// Initialize a subdivision to the triangle defined by the points a, b, c.
{
    Point2d *da, *db, *dc;
    da = new Point2d(a), db = new Point2d(b), dc = new Point2d(c);
    Edge* ea = MakeEdge();
    ea->Endpoints(da, db);
    Edge* eb = MakeEdge();
    Splice(ea->Sym(), eb);
    eb->Endpoints(db, dc);
    Edge* ec = MakeEdge();
    Splice(eb->Sym(), ec);
    ec->Endpoints(dc, da);
    Splice(ec->Sym(), ea);
    startingEdge = ea;
}

Edge* Connect(Edge* a, Edge* b)
```

```
// Add a new edge e connecting the destination of a to the
// origin of b, in such a way that all three have the same
// left face after the connection is complete.
// Additionally, the data pointers of the new edge are set.
{
    Edge* e = MakeEdge();
    Splice(e, a->Lnext());
    Splice(e->Sym(), b);
    e->EndPoints(a->Dest(), b->Org());
    return e;
}

void Swap(Edge* e)
// Essentially turns edge e counterclockwise inside its enclosing
// quadrilateral. The data pointers are modified accordingly.
{
    Edge* a = e->Oprev();
    Edge* b = e->Sym()->Oprev();
    Splice(e, a);
    Splice(e->Sym(), b);
    Splice(e, a->Lnext());
    Splice(e->Sym(), b->Lnext());
    e->EndPoints(a->Dest(), b->Dest());
}

/***** Geometric Predicates for Delaunay Diagrams *****/

inline Real TriArea(const Point2d& a, const Point2d& b, const Point2d& c)
// Returns twice the area of the oriented triangle (a, b, c), i.e., the
// area is positive if the triangle is oriented counterclockwise.
{
    return (b.x - a.x)*(c.y - a.y) - (b.y - a.y)*(c.x - a.x);
}

int InCircle(const Point2d& a, const Point2d& b,
              const Point2d& c, const Point2d& d)
// Returns TRUE if the point d is inside the circle defined by the
// points a, b, c. See Guibas and Stolfi (1985) p.107.
{
    return (a.x*a.x + a.y*a.y) * TriArea(b, c, d) -
           (b.x*b.x + b.y*b.y) * TriArea(a, c, d) +
           (c.x*c.x + c.y*c.y) * TriArea(a, b, d) -
           (d.x*d.x + d.y*d.y) * TriArea(a, b, c) > 0;
}

int ccw(const Point2d& a, const Point2d& b, const Point2d& c)
// Returns TRUE if the points a, b, c are in a counterclockwise order
{
    return (TriArea(a, b, c) > 0);
}

int RightOf(const Point2d& x, Edge* e)
{
    return ccw(x, e->Dest2d(), e->Org2d());
}

int LeftOf(const Point2d& x, Edge* e)
{
    return ccw(x, e->Org2d(), e->Dest2d());
}
```

```
int OnEdge(const Point2d& x, Edge* e)
// A predicate that determines if the point x is on the edge e.
// The point is considered on if it is in the EPS-neighborhood
// of the edge.
{
    Real t1, t2, t3;
    t1 = (x - e->Org2d()).norm();
    t2 = (x - e->Dest2d()).norm();
    if (t1 < EPS || t2 < EPS)
        return TRUE;
    t3 = (e->Org2d() - e->Dest2d()).norm();
    if (t1 > t3 || t2 > t3)
        return FALSE;
    Line line(e->Org2d(), e->Dest2d());
    return (fabs(line.eval(x)) < EPS);
}

/***** An Incremental Algorithm for the Construction of *****/
/***** Delaunay Diagrams *****/

Edge* Subdivision::Locate(const Point2d& x)
// Returns an edge e, s.t. either x is on e, or e is an edge of
// a triangle containing x. The search starts from startingEdge
// and proceeds in the general direction of x. Based on the
// pseudocode in Guibas and Stolfi (1985) p.121.
{
    Edge* e = startingEdge;

    while (TRUE) {
        if (x == e->Org2d() || x == e->Dest2d())
            return e;
        else if (RightOf(x, e))
            e = e->Sym();
        else if (!RightOf(x, e->Onext()))
            e = e->Onext();
        else if (!RightOf(x, e->Dprev()))
            e = e->Dprev();
        else
            return e;
    }
}

void Subdivision::InsertSite(const Point2d& x)
// Inserts a new point into a subdivision representing a Delaunay
// triangulation, and fixes the affected edges so that the result
// is still a Delaunay triangulation. This is based on the
// pseudocode from Guibas and Stolfi (1985) p.120, with slight
// modifications and a bug fix.
{
    Edge* e = Locate(x);
    if ((x == e->Org2d()) || (x == e->Dest2d())) // point is already in
        return;
    else if (OnEdge(x, e)) {
        e = e->Oprev();
        DeleteEdge(e->Onext());
    }

    // Connect the new point to the vertices of the containing
    // triangle (or quadrilateral, if the new point fell on an
    // existing edge.)
    Edge* base = MakeEdge();
```

```
base->EndPoints(e->Org(), new Point2d(x));
Splice(base, e);
startingEdge = base;
do {
    base = Connect(e, base->Sym());
    e = base->Oprev();
} while (e->Lnext() != startingEdge);

// Examine suspect edges to ensure that the Delaunay condition
// is satisfied.
do {
    Edge* t = e->Oprev();
    if (RightOf(t->Dest2d(), e) &&
        InCircle(e->Org2d(), t->Dest2d(), e->Dest2d(), x)) {
        Swap(e);
        e = e->Oprev();
    }
    else if (e->Onext() == startingEdge) // no more suspect edges
        return;
    else // pop a suspect edge
        e = e->Onext()->Lprev();
} while (TRUE);
}
```

/\*\*\*\*\*\*

```
#include <gl.h>
```

```
static unsigned int timestamp = 0;
```

```
void Subdivision::Draw()
```

```
{
    if (++timestamp == 0)
        timestamp = 1;
    startingEdge->Draw(timestamp);
}
```

```
void Edge::Draw(unsigned int stamp)
```

```
// This is a recursive drawing routine that uses time stamps to
// determine if the edge has already been drawn. This is given
// here for testing purposes only: it is not efficient, and for
// large triangulations the stack might overflow. A better way
// of doing this (and other traversals of the edges) is to maintain
// a list of edges in the corresponding Subdivision object. This
// list should be updated every time an edge is created or destroyed.
```

```
{
    if (Qedge()->TimeStamp(stamp)) {

        // Draw the edge
        Point2d a = Org2d();
        Point2d b = Dest2d();
        bgnline();
        v2d((double*)&a);
        v2d((double*)&b);
        endlne();

        // visit neighbors
        Onext()->Draw(stamp);
        Oprev()->Draw(stamp);
        Dnext()->Draw(stamp);
        Dprev()->Draw(stamp);
    }
}
```

}

```
#ifndef QUADEGE_H
#define QUADEGE_H

#include <geom2d.h>

class QuadEdge;

class Edge {
    friend QuadEdge;
    friend void Splice(Edge*, Edge*);
private:
    int num;
    Edge *next;
    Point2d *data;
public:
    Edge() { data = 0; }
    Edge* Rot();
    Edge* invRot();
    Edge* Sym();
    Edge* Onext();
    Edge* Oprev();
    Edge* Dnext();
    Edge* Dprev();
    Edge* Lnext();
    Edge* Lprev();
    Edge* Rnext();
    Edge* Rprev();
    Point2d* Org();
    Point2d* Dest();
    const Point2d& Org2d() const;
    const Point2d& Dest2d() const;
    void EndPoints(Point2d*, Point2d*);
    QuadEdge* Qedge() { return (QuadEdge*)(this - num); }
    void Draw(unsigned int);
};

class QuadEdge {
    friend Edge *MakeEdge();
private:
    Edge e[4];
    unsigned int ts;
public:
    QuadEdge();
    int TimeStamp(unsigned int);
};

class Subdivision {
private:
    Edge *startingEdge;
    Edge *Locate(const Point2d&);
public:
    Subdivision(const Point2d&, const Point2d&, const Point2d&);
    void InsertSite(const Point2d&);
    void Draw();
};

inline QuadEdge::QuadEdge()
{
    e[0].num = 0, e[1].num = 1, e[2].num = 2, e[3].num = 3;
    e[0].next = &(e[0]); e[1].next = &(e[3]);
    e[2].next = &(e[2]); e[3].next = &(e[1]);
}
```



```
    ts = 0;
}

inline int QuadEdge::TimeStamp(unsigned int stamp)
{
    if (ts != stamp) {
        ts = stamp;
        return TRUE;
    } else
        return FALSE;
}

/***** Edge Algebra *****/

inline Edge* Edge::Rot()
// Return the dual of the current edge, directed from its right to its left.
{
    return (num < 3) ? this + 1 : this - 3;
}

inline Edge* Edge::invRot()
// Return the dual of the current edge, directed from its left to its right.
{
    return (num > 0) ? this - 1 : this + 3;
}

inline Edge* Edge::Sym()
// Return the edge from the destination to the origin of the current edge.
{
    return (num < 2) ? this + 2 : this - 2;
}

inline Edge* Edge::Onext()
// Return the next ccw edge around (from) the origin of the current edge.
{
    return next;
}

inline Edge* Edge::Oprev()
// Return the next cw edge around (from) the origin of the current edge.
{
    return Rot()->Onext()->Rot();
}

inline Edge* Edge::Dnext()
// Return the next ccw edge around (into) the destination of the current edge.
{
    return Sym()->Onext()->Sym();
}

inline Edge* Edge::Dprev()
// Return the next cw edge around (into) the destination of the current edge.
{
    return invRot()->Onext()->invRot();
}

inline Edge* Edge::Lnext()
// Return the ccw edge around the left face following the current edge.
{
    return invRot()->Onext()->Rot();
}
```

```
inline Edge* Edge::Lprev()
// Return the ccw edge around the left face before the current edge.
{
    return Onext()->Sym();
}

inline Edge* Edge::Rnext()
// Return the edge around the right face ccw following the current edge.
{
    return Rot()->Onext()->invRot();
}

inline Edge* Edge::Rprev()
// Return the edge around the right face ccw before the current edge.
{
    return Sym()->Onext();
}

/***** Access to data pointers *****/

inline Point2d* Edge::Org()
{
    return data;
}

inline Point2d* Edge::Dest()
{
    return Sym()->data;
}

inline const Point2d& Edge::Org2d() const
{
    return *data;
}

inline const Point2d& Edge::Dest2d() const
{
    return (num < 2) ? *((this + 2)->data) : *((this - 2)->data);
}

inline void Edge::Endpoints(Point2d* or, Point2d* de)
{
    data = or;
    Sym()->data = de;
}

#endif /* QUADEDGE_H */
```

```
/* TEST PROGRAM FOR DELAUNAY */
```

```
#include <stdlib.h>
#include <stream.h>
#include <string.h>
#include <gl.h>
#include <device.h>
#include <quadedge.h>
```

```
void getArguments(int, char**);
void InsertPoints(Subdivision&);
void display(Subdivision&, Real, Real, Real, Real);
```

```
char *program;
int num = 20;
```

```
main(int argc, char** argv)
{
    getArguments(argc, argv);

    // Construct a triangle containing the unit square:
    Point2d p1(-1,-1), p2(2,-1), p3(0.5,3);
    Subdivision mesh(p1, p2, p3);

    InsertPoints(mesh);
    display(mesh, -.1, -.1, 1.1, 1.1);

    exit(0);
}
```

```
static void usage()
{
    cerr << "usage: " << program << " [ -n number_of_points ]\n";
}
```

```
static void errmsg(char *msg)
{
    cerr << program << ": " << msg << endl;
    usage();
    exit(1);
}
```

```
void getArguments(int argc, char** argv)
{
    program = argv[0];

    if(argc == 2 && strcmp(argv[1], "-h") == 0) {
        usage();
        exit(0);
    }

    for (int i = 1; i < argc && argv[i][0] == '-'; i++)
        if (strcmp(argv[i], "-n") == 0) {
            if(++i < argc)
                num = atoi(argv[i]);
            else
                errmsg("option ``-n'': missing parameter");
        } else
            errmsg("unknown option");

    if(argc - i > 0)
```

```
        errmsg("too many parameters");
    }

void InsertPoints(Subdivision& mesh)
{
    for (int i = 0; i < num; i++) {
        double u = drand48();
        double v = drand48();
        mesh.InsertSite(Point2d(u,v));
    }
}

long createWindow()
{
    prefsize(512, 512);
    long w = winopen("Delaunay Triangulation");
    doublebuffer();
    RGBmode();
    gconfig();
    return w;
}

void draw(Subdivision& mesh)
{
    cpack(0);
    clear();
    cpack(0xffffffff);

    /* draw your stuff here */
    mesh.Draw();

    swapbuffers();
}

void setView(float cx, float cy, float zoom, float xsize, float ysize)
{
    float dx = 0.5 * xsize / zoom;
    float dy = 0.5 * ysize / zoom;

    ortho2(cx - dx, cx + dx, cy - dy, cy + dy);
}

void printHelp()
{
    cerr << "<left>    insert a new point\n"
          << "    <h>    print help message\n"
          << "    <q>    quit\n";
}

void getMouse (long,
               float cx, float cy, float zoom,
               float xsize, float ysize,
               float& x, float& y)
{
    long sx, sy, ox, oy, sxsize, sysize;
    Coord p[3], v[3];

    sx = getvaluator(MOUSEX);
    sy = getvaluator(MOUSEY);
    getorigin(&ox, &oy);
    sx -= ox;
```

```
    sy -= oy;
    getsize(&sxsize, &sysize);
    xsize /= zoom;
    ysize /= zoom;
    x = (float(sx) / float(sxsize)) * xsize - 0.5 * xsize + cx;
    y = (float(sy) / float(sysize)) * ysize - 0.5 * ysize + cy;
}

void display(Subdivision& mesh, Real left, Real bottom, Real right, Real top)
{
    long wid;
    float cx, cy, zoom, xsize, ysize;

    wid = createWindow();

    cx = 0.5 * (left + right);
    cy = 0.5 * (bottom + top);
    zoom = 1;
    xsize = 1.1 * (right - left);
    ysize = 1.1 * (top - bottom);

    setView(cx, cy, zoom, xsize, ysize);

    qdevice(LEFTMOUSE);
    qdevice(HKEY);
    qdevice(QKEY);

    qreset();
    qenter(REDRAW, (short)wid);

    for (int done = FALSE; !done; ) {
        short val;

        switch(qread(&val)) {
            case REDRAW:
                winset(wid);
                reshapeviewport();
                setView(cx, cy, zoom, xsize, ysize);
                draw(mesh);
                break;
            case LEFTMOUSE:
                if(val) {
                    float x, y;
                    getMouse(wid, cx, cy, zoom, xsize, ysize, x, y);
                    cerr << "Mouse at (" << x << ", " << y << ")\n";
                    x = (x < 0) ? 0 : (x > 1) ? 1 : x;
                    y = (y < 0) ? 0 : (y > 1) ? 1 : y;
                    mesh.InsertSite(Point2d(x, y));
                    qreset();
                    draw(mesh);
                }
                break;
            case HKEY:
                if(val)
                    printHelp();
                break;
            case QKEY:
                if(val)
                    done = TRUE;
                break;
        }
    }
}
```

```
}
```

```
winclose(wid);
```

```
gexit();
```

```
}
```

```
/*
 * C++ code from the article
 * "Converting Rectangular Patches into Bezier Triangles"
 * by Dani Lischinski, danix@graphics.cornell.edu
 * in "Graphics Gems IV", Academic Press, 1994
 */

/*****

This is C++ code. Given here are class definitions and code for the
control point (ControlPoint) class, quadratic and quartic Bezier
triangle classes (BezierTri2 and BezierTri4, respectively), and for the
bilinear and biquadratic Bezier rectangular patch classes (BezierRect1
and BezierRect2.)

The conversion described in the gem takes place in the Convert
member functions of the BezierRect1 and BezierRect2 classes,
which each take references to two Bezier triangles (of appropriate
degree) as an argument.

Note that control points do not have to be (x,y,z) triplets.
For instance, they can be scalars, RGB triplets, etc., as long as the
operators +, *, /, and = (assignment) are provided. If you have a
class that you wish to use instead of the one given in the code, all
you have to do is to remove the definitions of the ControlPoint class
and its operators, and insert instead something like:

#include <my_class.h>
typedef MyClass ControlPoint;

*****/

#include <stream.h>
#include <stdlib.h>
#include <math.h>

#define FRAND() (random()/2147483648.) /* uniform rand# in [0,1) */

/***** Control Point Class *****/

class ControlPoint {
public:
    ControlPoint() { x = 0; y = 0; z = 0; }
    ControlPoint(float a, float b, float c) { x = a; y = b; z = c; }
    ControlPoint operator+(const ControlPoint&);
    ControlPoint operator-(const ControlPoint&);
    friend ControlPoint operator*(float, const ControlPoint&);
    friend ControlPoint operator/(const ControlPoint&, float);
    friend ostream& operator<<(ostream&, const ControlPoint&);
private:
    float x, y, z;
};

inline ControlPoint ControlPoint::operator+(const ControlPoint& p)
{
    return ControlPoint(x + p.x, y + p.y, z + p.z);
}

inline ControlPoint ControlPoint::operator-(const ControlPoint& p)
{
    return ControlPoint(x - p.x, y - p.y, z - p.z);
}
```

```
inline ControlPoint operator*(float c, const ControlPoint& a)
{
    return ControlPoint(c * a.x, c * a.y, c * a.z);
}

inline ControlPoint operator/(const ControlPoint& a, float c)
{
    return ControlPoint(a.x / c, a.y / c, a.z / c);
}

inline ostream& operator<<(ostream& os, const ControlPoint& p)
{
    return os << '(' << p.x << ',' << p.y << ',' << p.z << ") ";
}

/***** Quadratic Bezier Triangle Class *****/

class BezierTri2 {
private:
    ControlPoint cp[6];
public:
    ControlPoint& b(int, int, int);
    ControlPoint operator()(float, float);
};

ControlPoint& BezierTri2::b(int i, int j, int /* k */)
// Returns the (i,j,k) control point.
{
    static int row_start[3] = {0, 3, 5};
    return cp[row_start[j] + i];
}

ControlPoint BezierTri2::operator()(float u, float v)
// Evaluates the Bezier triangle at (u,v).
{
    float w = 1 - u - v;
    float u2 = u * u;
    float v2 = v * v;
    float w2 = w * w;
    return (w2*b(0,0,2) + (2*u*w)*b(1,0,1) + u2*b(2,0,0) +
            (2*v*w)*b(0,1,1) + (2*u*v)*b(1,1,0) + v2*b(0,2,0));
}

/***** Quartic Bezier Triangle Class *****/

class BezierTri4 {
private:
    ControlPoint cp[15];
public:
    ControlPoint& b(int, int, int);
    ControlPoint operator()(float, float);
};

ControlPoint& BezierTri4::b(int i, int j, int /* k */)
// Returns the (i,j,k) control point.
{
    static int row_start[5] = {0, 5, 9, 12, 14};
    return cp[row_start[j] + i];
}
```



```
ControlPoint BezierTri4::operator()(float u, float v)
// Evaluates the Bezier triangle at (u,v).
{
    float w = 1 - u - v;
    float u2 = u * u, u3 = u2 * u, u4 = u3 * u;
    float v2 = v * v, v3 = v2 * v, v4 = v3 * v;
    float w2 = w * w, w3 = w2 * w, w4 = w3 * w;
    return (w4*b(0,0,4) + (4*u*w3)*b(1,0,3) + (6*u2*w2)*b(2,0,2) +
            (4*u3*w)*b(3,0,1) + u4*b(4,0,0) + (4*v*w3)*b(0,1,3) +
            (12*u*v*w2)*b(1,1,2) + (12*u2*v*w)*b(2,1,1) + (4*u3*v)*b(3,1,0) +
            (6*v2*w2)*b(0,2,2) + (12*u*v2*w)*b(1,2,1) + (6*u2*v2)*b(2,2,0) +
            (4*v3*w)*b(0,3,1) + (4*u*v3)*b(1,3,0) + v4*b(0,4,0));
}

/***** Bilinear Bezier Rectangle Class *****/

class BezierRect1 {
private:
    ControlPoint cp[2][2];
public:
    ControlPoint& p(int i, int j) { return cp[i][j]; }
    ControlPoint operator()(float, float);
    void Convert(BezierTri2&, BezierTri2&);
};

ControlPoint BezierRect1::operator()(float s, float t)
// Evaluates the Bezier rectangle at (s,t).
{
    float s1 = 1 - s;

    return ((1-t) * (s1*p(0,0) + s*p(0,1)) +
            t * (s1*p(1,0) + s*p(1,1)));
}

void BezierRect1::Convert(BezierTri2& t1, BezierTri2& t2)
// Converts a bilinear Bezier rectangle into two quadratic Bezier
// triangles t1 and t2, such that the value of the bilinear
// at (s,t) is equal to t1(s,t) if (s + t <= 1), and t2(1-t,1-s)
// otherwise.
{
    // lower left triangle:
    t1.b(0,0,2) = p(0,0);
    t1.b(1,0,1) = 0.5 * (p(0,0) + p(0,1));
    t1.b(2,0,0) = p(0,1);

    t1.b(0,1,1) = 0.5 * (p(0,0) + p(1,0));
    t1.b(1,1,0) = 0.5 * (p(0,0) + p(1,1));

    t1.b(0,2,0) = p(1,0);

    // upper right triangle:
    t2.b(0,0,2) = p(1,1);
    t2.b(1,0,1) = 0.5 * (p(1,1) + p(0,1));
    t2.b(2,0,0) = p(0,1);

    t2.b(0,1,1) = 0.5 * (p(1,1) + p(1,0));
    t2.b(1,1,0) = 0.5 * (p(0,0) + p(1,1));

    t2.b(0,2,0) = p(1,0);
}
```

/\*\*\*\*\*\* Biquadratic Bezier Rectangle Class \*\*\*\*\*/

```
class BezierRect2 {
private:
    ControlPoint cp[3][3];
public:
    ControlPoint& p(int i, int j)    { return cp[i][j]; }
    ControlPoint operator()(float, float);
    void Convert(BezierTri4&, BezierTri4&);
};

ControlPoint BezierRect2::operator()(float s, float t)
// Evaluates the Bezier rectangle at (s,t).
{
    float s1 = 1 - s, ss1 = 2*s*s1, s2 = s*s, s12 = s1*s1;
    float t1 = 1 - t, tt1 = 2*t*t1, t2 = t*t, t12 = t1*t1;

    return (t12 * (s12*p(0,0) + ss1*p(0,1) + s2*p(0,2)) +
            tt1 * (s12*p(1,0) + ss1*p(1,1) + s2*p(1,2)) +
            t2   * (s12*p(2,0) + ss1*p(2,1) + s2*p(2,2)));
}

void BezierRect2::Convert(BezierTri4& t1, BezierTri4& t2)
// Converts a biquadratic Bezier rectangle into two quartic Bezier
// triangles t1 and t2, such that the value of the biquadratic
// at (s,t) is equal to t1(s,t) if (s + t <= 1), and t2(1-t,1-s)
// otherwise.
{
    // lower left triangle:
    t1.b(0,0,4) = p(0,0);
    t1.b(1,0,3) = 0.5 * (p(0,0) + p(0,1));
    t1.b(2,0,2) = (p(0,0) + 4 * p(0,1) + p(0,2)) / 6;
    t1.b(3,0,1) = 0.5 * (p(0,1) + p(0,2));
    t1.b(4,0,0) = p(0,2);

    t1.b(0,1,3) = 0.5 * (p(0,0) + p(1,0));
    t1.b(1,1,2) = (p(0,0) + p(1,1)) / 3 + (p(0,1) + p(1,0)) / 6;
    t1.b(2,1,1) = (p(0,0) + p(1,2)) / 6 + (p(0,1) + p(1,1)) / 3;
    t1.b(3,1,0) = 0.5 * (p(0,1) + p(1,2));

    t1.b(0,2,2) = (p(0,0) + 4 * p(1,0) + p(2,0)) / 6;
    t1.b(1,2,1) = (p(0,0) + p(2,1)) / 6 + (p(1,0) + p(1,1)) / 3;
    t1.b(2,2,0) = (p(0,0) + 4 * p(1,1) + p(2,2)) / 6;

    t1.b(0,3,1) = 0.5 * (p(1,0) + p(2,0));
    t1.b(1,3,0) = 0.5 * (p(1,0) + p(2,1));

    t1.b(0,4,0) = p(2,0);

    // upper right triangle:
    t2.b(0,0,4) = p(2,2);
    t2.b(1,0,3) = 0.5 * (p(2,2) + p(1,2));
    t2.b(2,0,2) = (p(2,2) + 4 * p(1,2) + p(0,2)) / 6;
    t2.b(3,0,1) = 0.5 * (p(1,2) + p(0,2));
    t2.b(4,0,0) = p(0,2);

    t2.b(0,1,3) = 0.5 * (p(2,2) + p(2,1));
    t2.b(1,1,2) = (p(2,2) + p(1,1)) / 3 + (p(1,2) + p(2,1)) / 6;
    t2.b(2,1,1) = (p(2,2) + p(0,1)) / 6 + (p(1,2) + p(1,1)) / 3;
    t2.b(3,1,0) = 0.5 * (p(0,1) + p(1,2));
}
```

```
t2.b(0,2,2) = (p(2,2) + 4 * p(2,1) + p(2,0)) / 6;  
t2.b(1,2,1) = (p(2,2) + p(1,0)) / 6 + (p(2,1) + p(1,1)) / 3;  
t2.b(2,2,0) = (p(2,2) + 4 * p(1,1) + p(0,0)) / 6;  
  
t2.b(0,3,1) = 0.5 * (p(2,1) + p(2,0));  
t2.b(1,3,0) = 0.5 * (p(1,0) + p(2,1));  
  
t2.b(0,4,0) = p(2,0);
```

```
}
```

```
/****** A Test Program *****/
```

```
void main()  
{  
    // init biquadratic Bezier rectangle:  
    BezierRect2 brect;  
    brect.p(0,0) = ControlPoint(FRAND(), FRAND(), FRAND());  
    brect.p(0,1) = ControlPoint(FRAND(), FRAND(), FRAND());  
    brect.p(0,2) = ControlPoint(FRAND(), FRAND(), FRAND());  
    brect.p(1,0) = ControlPoint(FRAND(), FRAND(), FRAND());  
    brect.p(1,1) = ControlPoint(FRAND(), FRAND(), FRAND());  
    brect.p(1,2) = ControlPoint(FRAND(), FRAND(), FRAND());  
    brect.p(2,0) = ControlPoint(FRAND(), FRAND(), FRAND());  
    brect.p(2,1) = ControlPoint(FRAND(), FRAND(), FRAND());  
    brect.p(2,2) = ControlPoint(FRAND(), FRAND(), FRAND());  
  
    // convert to two quartic Bezier triangles:  
    BezierTri4 btri1, btri2;  
    brect.Convert(btri1, btri2);  
  
    float s, t;  
    ControlPoint pt1, pt2;  
  
    while (cin >> s >> t) {  
        pt1 = brect(s, t);  
        if (s + t <= 1.0) {  
            pt2 = btri1(s, t);  
        } else {  
            pt2 = btri2(1-t, 1-s);  
        }  
        cerr << pt1 - pt2 << endl;  
    }  
}
```

```
/*
Using Quaternions for Coding 3D Transformations
Patrick-Gilles Mailliot
from "Graphics Gems", Academic Press, 1990
*/

extern double P[3], Q[4], M[4][4];

set_obs_position(x,y,z)
float    x, y, z;
{
    int    i;

/*
 * Set the values of the eye's position.
 * The position here represents the position of the orthonormal base
 * in respect to the observer.
 */
    P[0] = -x;
    P[1] = -y;
    P[2] = -z;

/*
 * Set the visualization to be in the decreasing x axis
 */
    Q[0] = 1.;
    for (i = 1; i < 4; i++) Q[i] = 0.;
}

translate_quaternion(x,i,w)
float    x;
int      i, w;
{
    int    j, k;
    float  A, B, D, E, F;

    if (w < 0) {
/*
 * The observer moves in respect to the scene.
 */
        P[i - 1] -= x;
    } else {
/*
 * The scene moves in respect to the observer.
 * Compute the successor axis of i [1,2,3];
 * and then the successor axis of j [1,2,3];
 */
        if ((j = i + 1) > 3) j = 1;
        if ((k = j + 1) > 3) k = 1;
        A = Q[j]; B = Q[k]; F = Q[0]; E = Q[i];
        P[i - 1] += x * (E * E + F * F - A * A - B * B);
        D = x + x;
        P[j - 1] += D * (E * A + F * B);
        P[k - 1] += D * (E * B + F * A);
    }
}










rotate_quaternion(x,y,i,w)
float    x, y;
int      i, w;
{
```

```
int      j, k;
float    E, F, R1;
/*
 * Compute the successor axis of i [1,2,3] and j [1,2,3];
 */
    if ((j = i + 1) > 3) j = 1;
    if ((k = j + 1) > 3) k = 1;
    E = Q[i];
    Q[i] = E * x + w * y * Q[0];
    Q[0] = Q[0] * x - w * y * E;
    E = Q[j];
    Q[j] = E * x + y * Q[k];
    Q[k] = Q[k] * x - y * E;
    if (w < 0) {
/* Compute a new position if the observer moves in respect to the scene. */
        j -= 1; k -= 1;
        R1 = x * x - y * y;
        F = 2. * x * y;
        E = P[j];
        P[j] = E * R1 + F * P[k];
        P[k] = P[k] * R1 - F * E;
    }
}
```

```
Evaluate_matrix()
{
float    e, f, r[4];
int      i, j, k;
/*
 * We will need some square values!
 */
    for (i = 0; i < 4; i++) r[i] = Q[i] * Q[i];
/*
 * Compute each element of the matrix.
 * j is the successor of i (in 1,2,3), while k is the successor of j.
 */
    for (i = 1; i < 4; i++) {
        if ((j = i + 1) > 3) j = 1;
        if ((k = j + 1) > 3) k = 1;
        e = 2. * Q[i] * Q[j];
        f = 2. * Q[k] * Q[0];
        M[j][i] = e - f;
        M[i][j] = e + f;
        M[i][i] = r[i] + r[0] - r[j] - r[k];
        M[0][i] = P[i - 1];
        M[i][0] = 0.;
    }
    M[0][0] = 1.;
}
```

# Index of

## /pubs/tog/GraphicsGems/gemsv/ch7-5/

Name	Last modified	Size	Description
 <a href="#">_Parent Directory</a>			
 <a href="#">_README</a>	29-Jun-00 08:24	2K	
 <a href="#">_basic.h</a>	29-Jun-00 08:24	2K	
 <a href="#">_construct.c</a>	29-Jun-00 08:24	24K	
 <a href="#">_data_1</a>	29-Jun-00 08:24	1K	
 <a href="#">_makefile</a>	29-Jun-00 08:24	1K	
 <a href="#">_misc.c</a>	29-Jun-00 08:24	2K	
 <a href="#">_monotone.c</a>	29-Jun-00 08:24	17K	
 <a href="#">_tri.c</a>	29-Jun-00 08:24	3K	

This program is an implementation of a fast polygon triangulation algorithm based on the paper "A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons" by Raimund Seidel.

The input is specified as a list of points (x, y). If the polygon has n distinct vertices (the n+1 th vertex being the same as the 1st vertex), then the file should contain only the n distinct vertices in ANTI-CLOCKWISE order. Also, the polygon should be simple (i.e. non self-intersecting) with NO HOLES. A sample data file is included in the package (Notice the no point is repeated).

Triangulation should produce (n - 2) triangles as the output. (i.e. no additional vertices are introduced). The result is available in the array triangles[][3], where each of triangles[0] to triangles[n-3] contain the 3 vertices forming the triangle (the vertices are numbered 1..n according to the input). The vertices of each triangle are output in anti-clockwise order.

Use gmake to create the executable. (There should not be any compilation problem. If log2() is not defined in your math library, you will have to supply the definition)

#### USAGE:

triangulate <filename>

#### Bibliography:

(Seidel 1991) R. Seidel. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions for triangulating polygons. Computational Geometry: Theory and applications, 1(1991):51-64.

(Fournier et al. 84) A. Fournier and D.Y. Montuno, Triangulating simple polygons and equivalent problems, ACM Trans. on Graphics 3 (1984) 153-174.

Implementation report: Narkhede A. and Manocha D., Fast polygon triangulation algorithm based on Seidel's Algorithm, UNC-CH, 1994.

-----  
This code is in the public domain. Specifically, we give to the public domain all rights for future licensing of the source code, all resale rights, and all publishing rights.

UNC-CH GIVES NO WARRANTY, EXPRESSED OR IMPLIED, FOR THE SOFTWARE AND/OR DOCUMENTATION PROVIDED, INCLUDING, WITHOUT LIMITATION, WARRANTY OF MERCHANTABILITY AND WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE.

- Atul Narkhede (narkhede@cs.unc.edu)

```
#include <sys/types.h>
#include <stdlib.h>
#include <stdio.h>

typedef struct {
    double x, y;
} point_t, vector_t;

typedef struct {
    point_t v0, v1;
    int is_inserted;
    int root0, root1;
} segment_t;

typedef struct {
    int lseg, rseg;
    point_t hi, lo;
    int u0, u1;
    int d0, d1;
    int sink;
    int usave, uside;
    int state;
} trap_t;

typedef struct {
    int nodetype;
    int segnum;
    point_t yval;
    int trnum;
    int parent;
    int left, right;
} node_t;

typedef struct {
    int vnum;
    int next;
    int prev;
} monchain_t;

typedef struct {
    point_t pt;
    int vnext[4];           /* next vertices for the 4 chains */
    int vpos[4];           /* position of v in the 4 chains */
    int nextfree;
} vertexchain_t;

struct global_s {
    int nseg;
};

#define T_X      1
#define T_Y      2
#define T_SINK   3

#define QSIZE     800      /* maximum table sizes */
#define TRSIZE    400      /* max# trapezoids */
#define SEGSIZE   100      /* max# of segments */

#define TRUE      1
#define FALSE     0
```



```
#define FIRSTPT 1                /* checking whether pt. is inserted */
#define LASTPT 2

#define EPS 0.000005

#define INFINITY 1<<30

#define C_EPS 1.0e-7

#define S_LEFT 1                 /* for merge-direction */
#define S_RIGHT 2

#define ST_VALID 1               /* for trapezium table */
#define ST_INVALID 2

#define SP_SIMPLE_LRUP 1         /* for splitting trapezoids */
#define SP_SIMPLE_LRDN 2
#define SP_2UP_2DN 3
#define SP_2UP_LEFT 4
#define SP_2UP_RIGHT 5
#define SP_2DN_LEFT 6
#define SP_2DN_RIGHT 7
#define SP_NOSPLIT -1

#define TR_FROM_UP 1             /* for traverse-direction */
#define TR_FROM_DN 2

#define TRI_LHS 1
#define TRI_RHS 2

#define MAX(a, b) (((a) > (b)) ? (a) : (b))
#define MIN(a, b) (((a) < (b)) ? (a) : (b))

#define CROSS(v0, v1, v2) (((v1).x - (v0).x)*((v2).y - (v0).y) - \
                           ((v1).y - (v0).y)*((v2).x - (v0).x))

#define DOT(v0, v1) ((v0).x * (v1).x + (v0).y * (v1).y)

#define MODULO_NEXT(v0, n) (((v0) - 1) % (n) + 1)

#define FP_EQUAL(s, t) (fabs(s - t) <= C_EPS)

/* Global variables */

node_t qs[QSIZE];                /* Query structure */
trap_t tr[TRSIZE];              /* Trapezoid structure */
segment_t seg[SEGSIZE];         /* Segment table */

struct global_s global;
```

```
#include "basic.h"
#include <math.h>
#include <string.h>      /* for memset() */

static int q_idx;
static int tr_idx;

/* Return a new node to be added into the query tree */
static int newnode()
{
    if (q_idx < QSIZE)
        return q_idx++;
    else
    {
        fprintf(stderr, "newnode: Query-table overflow\n");
        return -1;
    }
}

/* Return a free trapezoid */
static int newtrap()
{
    if (tr_idx < TRSIZE)
    {
        tr[tr_idx].lseg = -1;
        tr[tr_idx].rseg = -1;
        tr[tr_idx].state = ST_VALID;
        return tr_idx++;
    }
    else
    {
        fprintf(stderr, "newtrap: Trapezoid-table overflow\n");
        return -1;
    }
}

/* Return the maximum of the two points into the yval structure */
static int _max(yval, v0, v1)
    point_t *yval;
    point_t *v0;
    point_t *v1;
{
    if (v0->y > v1->y + C_EPS)
        *yval = *v0;
    else if (FP_EQUAL(v0->y, v1->y))
    {
        if (v0->x > v1->x + C_EPS)
            *yval = *v0;
        else
            *yval = *v1;
    }
    else
        *yval = *v1;

    return 0;
}

/* Return the minimum of the two points into the yval structure */
static int _min(yval, v0, v1)
```

```
    point_t *yval;
    point_t *v0;
    point_t *v1;
{
    if (v0->y < v1->y - C_EPS)
        *yval = *v0;
    else if (FP_EQUAL(v0->y, v1->y))
    {
        if (v0->x < v1->x)
            *yval = *v0;
        else
            *yval = *v1;
    }
    else
        *yval = *v1;

    return 0;
}
```

```
int _greater_than(v0, v1)
    point_t *v0;
    point_t *v1;
{
    if (v0->y > v1->y + C_EPS)
        return TRUE;
    else if (v0->y < v1->y - C_EPS)
        return FALSE;
    else
        return (v0->x > v1->x);
}
```

```
int _equal_to(v0, v1)
    point_t *v0;
    point_t *v1;
{
    return (FP_EQUAL(v0->y, v1->y) && FP_EQUAL(v0->x, v1->x));
}
```

```
int _greater_than_equal_to(v0, v1)
    point_t *v0;
    point_t *v1;
{
    if (v0->y > v1->y + C_EPS)
        return TRUE;
    else if (v0->y < v1->y - C_EPS)
        return FALSE;
    else
        return (v0->x >= v1->x);
}
```

```
int _less_than(v0, v1)
    point_t *v0;
    point_t *v1;
{
    if (v0->y < v1->y - C_EPS)
        return TRUE;
    else if (v0->y > v1->y + C_EPS)
        return FALSE;
    else
```

```
        return (v0->x < v1->x);
    }

/* Initilialise the query structure (Q) and the trapezoid table (T)
 * when the first segment is added to start the trapezoidation
 */
int init_query_structure(segnum)
    int segnum;
{
    int i1, i2, i3, i4, i5, i6, i7, root;
    int t1, t2, t3, t4;
    segment_t *s = &seg[segnum];

    memset((void *)tr, 0, sizeof(tr));
    memset((void *)qs, 0, sizeof(qs));

    i1 = newnode();
    qs[i1].nodetype = T_Y;
    _max(&qs[i1].yval, &s->v0, &s->v1); /* root */
    root = i1;

    qs[i1].right = i2 = newnode();
    qs[i2].nodetype = T_SINK;
    qs[i2].parent = i1;

    qs[i1].left = i3 = newnode();
    qs[i3].nodetype = T_Y;
    _min(&qs[i3].yval, &s->v0, &s->v1); /* root */
    qs[i3].parent = i1;

    qs[i3].left = i4 = newnode();
    qs[i4].nodetype = T_SINK;
    qs[i4].parent = i3;

    qs[i3].right = i5 = newnode();
    qs[i5].nodetype = T_X;
    qs[i5].segnum = segnum;
    qs[i5].parent = i3;

    qs[i5].left = i6 = newnode();
    qs[i6].nodetype = T_SINK;
    qs[i6].parent = i5;

    qs[i5].right = i7 = newnode();
    qs[i7].nodetype = T_SINK;
    qs[i7].parent = i5;

    t1 = newtrap();          /* middle left */
    t2 = newtrap();          /* middle right */
    t3 = newtrap();          /* bottom-most */
    t4 = newtrap();          /* topmost */

    tr[t1].hi = tr[t2].hi = tr[t4].lo = qs[i1].yval;
    tr[t1].lo = tr[t2].lo = tr[t3].hi = qs[i3].yval;
    tr[t4].hi.y = (double) (INFINITY);
    tr[t4].hi.x = (double) (INFINITY);
    tr[t3].lo.y = (double) -1* (INFINITY);
    tr[t3].lo.x = (double) -1* (INFINITY);
    tr[t1].rseg = tr[t2].lseg = segnum;
    tr[t1].u0 = tr[t2].u0 = t4;
}
```

```
tr[t1].d0 = tr[t2].d0 = t3;
tr[t4].d0 = tr[t3].u0 = t1;
tr[t4].d1 = tr[t3].u1 = t2;

tr[t1].sink = i6;
tr[t2].sink = i7;
tr[t3].sink = i4;
tr[t4].sink = i2;

tr[t1].state = tr[t2].state = ST_VALID;
tr[t3].state = tr[t4].state = ST_VALID;

qs[i2].trnum = t4;
qs[i4].trnum = t3;
qs[i6].trnum = t1;
qs[i7].trnum = t2;

s->is_inserted = TRUE;
return root;
}
```

```
/* Return TRUE if the vertex v is to the left of line segment no.
 * segnum
 */
```

```
static int is_left_of(segnum, v)
    int segnum;
    point_t *v;
{
    segment_t *s = &seg[segnum];
    double area;

    if (_greater_than(&s->v1, &s->v0)) /* seg. going upwards */
    {
        if (FP_EQUAL(s->v1.y, v->y))
        {
            if (v->x < s->v1.x)
                area = 1.0;
            else
                area = -1.0;
        }
        else if (FP_EQUAL(s->v0.y, v->y))
        {
            if (v->x < s->v0.x)
                area = 1.0;
            else
                area = -1.0;
        }
        else
            area = CROSS(s->v0, s->v1, (*v));
    }
    else /* v0 > v1 */
    {
        if (FP_EQUAL(s->v1.y, v->y))
        {
            if (v->x < s->v1.x)
                area = 1.0;
            else
                area = -1.0;
        }
    }
}
```

```
    else if (FP_EQUAL(s->v0.y, v->y))
    {
        if (v->x < s->v0.x)
            area = 1.0;
        else
            area = -1.0;
    }
    else
        area = CROSS(s->v1, s->v0, (*v));
}

if (area > 0.0)
    return TRUE;
else
    return FALSE;
}

int is_collinear(segnum, v, is_swapped)
    int segnum;
    point_t *v;
    int is_swapped;
{
    int n;

    /* First check if the endpoint is already inserted */
    if (!is_swapped)
        n = MODULO_NEXT(segnum + 1, global.nseg);
    else
        if ((n = segnum - 1) == 0)
            n = 1;

    return seg[n].is_inserted;
}

/* This is query routine which determines which trapezoid does the
 * point v lie in. The return value is the trapezoid number
 */

int locate_endpoint(v, vo, r)
    point_t *v;
    point_t *vo;
    int r;
{
    node_t *rptr = &qs[r];

    switch (rptr->nodetype)
    {
        case T_SINK:
            return rptr->trnum;

        case T_Y:
            if (_greater_than(v, &rptr->yval)) /* above */
                return locate_endpoint(v, vo, rptr->right);
            else if (_equal_to(v, &rptr->yval)) /* the point is already */
                /* inserted. */
            {
                if (_greater_than(vo, &rptr->yval)) /* above */
                    return locate_endpoint(v, vo, rptr->right);
                else
                    return locate_endpoint(v, vo, rptr->left); /* below */
            }
    }
}
```

```
    }
    else
        return locate_endpoint(v, vo, rptr->left); /* below */

case T_X:
    if (_equal_to(v, &seg[rptr->segnum].v0) ||
        _equal_to(v, &seg[rptr->segnum].v1))
    {
        if (FP_EQUAL(v->y, vo->y)) /* horizontal segment */
        {
            if (vo->x < v->x)
                return locate_endpoint(v, vo, rptr->left); /* left */
            else
                return locate_endpoint(v, vo, rptr->right); /* right */
        }

        else if (is_left_of(rptr->segnum, vo))
            return locate_endpoint(v, vo, rptr->left); /* left */
        else
            return locate_endpoint(v, vo, rptr->right); /* right */
    }
    else if (is_left_of(rptr->segnum, v))
        return locate_endpoint(v, vo, rptr->left); /* left */
    else
        return locate_endpoint(v, vo, rptr->right); /* right */

default:
    fprintf(stderr, "Haggu !!!!!\n");
    break;
}
}

/* Thread in the segment into the existing trapezoidation. The
 * limiting trapezoids are given by tfirst and tlast (which are the
 * trapezoids containing the two endpoints of the segment
 */

int merge_trapezoids(segnum, tfirst, tlast, side)
    int segnum;
    int tfirst;
    int tlast;
    int side;
{
    int t, tnext, cond;
    int ptnext;

    /* First merge polys on the LHS */
    t = tfirst;
    while ((t > 0) && _greater_than_equal_to(&tr[t].lo, &tr[tlast].lo))
    {
        if (side == S_LEFT)
            cond = (((tnext = tr[t].d0) > 0) && (tr[tnext].rseg == segnum)) ||
                  (((tnext = tr[t].d1) > 0) && (tr[tnext].rseg == segnum));
        else
            cond = (((tnext = tr[t].d0) > 0) && (tr[tnext].lseg == segnum)) ||
                  (((tnext = tr[t].d1) > 0) && (tr[tnext].lseg == segnum));

        if (cond)
        {
            if ((tr[t].lseg == tr[tnext].lseg) &&
```

```
(tr[t].rseg == tr[tnext].rseg)) /* good neighbours */
{
    /* merge them */
    /* Use the upper node as the new node i.e. t */

    ptnext = qs[tr[tnext].sink].parent;

    if (qs[ptnext].left == tr[tnext].sink)
        qs[ptnext].left = tr[t].sink;
    else
        qs[ptnext].right = tr[t].sink; /* redirect parent */

    /* Change the upper neighbours of the lower trapezoids */

    if ((tr[t].d0 = tr[tnext].d0) > 0)
        if (tr[tr[t].d0].u0 == tnext)
            tr[tr[t].d0].u0 = t;
        else if (tr[tr[t].d0].u1 == tnext)
            tr[tr[t].d0].u1 = t;

    if ((tr[t].d1 = tr[tnext].d1) > 0)
        if (tr[tr[t].d1].u0 == tnext)
            tr[tr[t].d1].u0 = t;
        else if (tr[tr[t].d1].u1 == tnext)
            tr[tr[t].d1].u1 = t;

    tr[t].lo = tr[tnext].lo;
    tr[tnext].state = ST_INVALID; /* invalidate the lower */
                                /* trapezium */
}
else /* not good neighbours */
    t = tnext;
}
else /* do not satisfy the outer if */
    t = tnext;

} /* end-while */

return 0;
}

/* Add in the new segment into the trapezoidation and update Q and T
 * structures
 */
int add_segment(segnum)
    int segnum;
{
    segment_t s;
    int tu, tl, sk, tfirst, tlast, tnext;
    int tfirstr, tlastr, tfirstl, tlastl;
    int i1, i2, t, tn;
    point_t vper, tpt;
    int tritop = 0, tribot = 0, is_swapped = 0;
    int tmptriseg;

    s = seg[segnum];
    if (_greater_than(&s.v1, &s.v0)) /* Get higher vertex in v0 */
    {
        int tmp;
        tpt = s.v0;
    }
}
```



```
s.v0 = s.v1;
s.v1 = tpt;
tmp = s.root0;
s.root0 = s.root1;
s.root1 = tmp;
is_swapped = TRUE;
}

if ((is_swapped) ? !inserted(segnum, LASTPT) :
    !inserted(segnum, FIRSTPT))    /* insert v0 in the tree */
{
    int tmp_d;

    tu = locate_endpoint(&s.v0, &s.v1, s.root0);
    tl = newtrap();                /* tl is the new lower trapezoid */
    tr[tl].state = ST_VALID;
    tr[tl] = tr[tu];
    tr[tu].lo.y = tr[tl].hi.y = s.v0.y;
    tr[tu].lo.x = tr[tl].hi.x = s.v0.x;
    tr[tu].d0 = tl;
    tr[tu].d1 = 0;
    tr[tl].u0 = tu;
    tr[tl].u1 = 0;

    if (((tmp_d = tr[tl].d0) > 0) && (tr[tmp_d].u0 == tu))
        tr[tmp_d].u0 = tl;
    if (((tmp_d = tr[tl].d0) > 0) && (tr[tmp_d].u1 == tu))
        tr[tmp_d].u1 = tl;

    if (((tmp_d = tr[tl].d1) > 0) && (tr[tmp_d].u0 == tu))
        tr[tmp_d].u0 = tl;
    if (((tmp_d = tr[tl].d1) > 0) && (tr[tmp_d].u1 == tu))
        tr[tmp_d].u1 = tl;

    /* Now update the query structure and obtain the sinks for the */
    /* two trapezoids */

    i1 = newnode();                /* Upper trapezoid sink */
    i2 = newnode();                /* Lower trapezoid sink */
    sk = tr[tu].sink;

    qs[sk].nodetype = T_Y;
    qs[sk].yval = s.v0;
    qs[sk].segnum = segnum;        /* not really reqd ... maybe later */
    qs[sk].left = i2;
    qs[sk].right = i1;

    qs[i1].nodetype = T_SINK;
    qs[i1].trnum = tu;
    qs[i1].parent = sk;

    qs[i2].nodetype = T_SINK;
    qs[i2].trnum = tl;
    qs[i2].parent = sk;

    tr[tu].sink = i1;
    tr[tl].sink = i2;
    tfirst = tl;
}
else                                /* v0 already present */
{
    /* Get the topmost intersecting trapezoid */
```

```
    vper.x = s.v0.x + EPS * (s.v1.x - s.v0.x);
    vper.y = s.v0.y + EPS * (s.v1.y - s.v0.y);
    tfirst = locate_endpoint(&s.v0, &s.v1, s.root0);
    tritop = 1;
}

if ((is_swapped) ? !inserted(segnum, FIRSTPT) :
    !inserted(segnum, LASTPT)) /* insert v1 in the tree */
{
    int tmp_d;

    tu = locate_endpoint(&s.v1, &s.v0, s.root1);

    tl = newtrap(); /* tl is the new lower trapezoid */
    tr[tl].state = ST_VALID;
    tr[tl] = tr[tu];
    tr[tu].lo.y = tr[tl].hi.y = s.v1.y;
    tr[tu].lo.x = tr[tl].hi.x = s.v1.x;
    tr[tu].d0 = tl;
    tr[tu].d1 = 0;
    tr[tl].u0 = tu;
    tr[tl].u1 = 0;

    if (((tmp_d = tr[tl].d0) > 0) && (tr[tmp_d].u0 == tu))
        tr[tmp_d].u0 = tl;
    if (((tmp_d = tr[tl].d0) > 0) && (tr[tmp_d].u1 == tu))
        tr[tmp_d].u1 = tl;

    if (((tmp_d = tr[tl].d1) > 0) && (tr[tmp_d].u0 == tu))
        tr[tmp_d].u0 = tl;
    if (((tmp_d = tr[tl].d1) > 0) && (tr[tmp_d].u1 == tu))
        tr[tmp_d].u1 = tl;

    /* Now update the query structure and obtain the sinks for the */
    /* two trapezoids */

    i1 = newnode(); /* Upper trapezoid sink */
    i2 = newnode(); /* Lower trapezoid sink */
    sk = tr[tu].sink;

    qs[sk].nodetype = T_Y;
    qs[sk].yval = s.v1;
    qs[sk].segnum = segnum; /* not really reqd ... maybe later */
    qs[sk].left = i2;
    qs[sk].right = i1;

    qs[i1].nodetype = T_SINK;
    qs[i1].trnum = tu;
    qs[i1].parent = sk;

    qs[i2].nodetype = T_SINK;
    qs[i2].trnum = tl;
    qs[i2].parent = sk;

    tr[tu].sink = i1;
    tr[tl].sink = i2;
    tlast = tu;
}
else /* v1 already present */
{
    /* Get the lowermost intersecting trapezoid */

```

```
    vper.x = s.v1.x + EPS * (s.v0.x - s.v1.x);
    vper.y = s.v1.y + EPS * (s.v0.y - s.v1.y);
    tlast = locate_endpoint(&s.v1, &s.v0, s.root1);
    tribot = 1;
}

/* Thread the segment into the query tree creating a new X-node */
/* First, split all the trapezoids which are intersected by s into */
/* two */

t = tfirst;                                /* topmost trapezoid */

while ((t > 0) &&
       _greater_than_equal_to(&tr[t].lo, &tr[tlast].lo))
    /* traverse from top to bot */
    {
        int t_sav, tn_sav;
        sk = tr[t].sink;
        i1 = newnode();                     /* left trapezoid sink */
        i2 = newnode();                     /* right trapezoid sink */

        qs[sk].nodetype = T_X;
        qs[sk].segnum = segnum;
        qs[sk].left = i1;
        qs[sk].right = i2;

        qs[i1].nodetype = T_SINK; /* left trapezoid (use existing one) */
        qs[i1].trnum = t;
        qs[i1].parent = sk;

        qs[i2].nodetype = T_SINK; /* right trapezoid (allocate new) */
        qs[i2].trnum = tn = newtrap();
        tr[tn].state = ST_VALID;
        qs[i2].parent = sk;

        if (t == tfirst)
            tfirst = tn;
        if (_equal_to(&tr[t].lo, &tr[tlast].lo))
            tlast = tn;

        tr[tn] = tr[t];
        tr[t].sink = i1;
        tr[tn].sink = i2;
        t_sav = t;
        tn_sav = tn;

        /* error */

        if ((tr[t].d0 <= 0) && (tr[t].d1 <= 0)) /* case cannot arise */
        {
            fprintf(stderr, "add_segment: error\n");
            break;
        }

        /* only one trapezoid below. partition t into two and make the */
        /* two resulting trapezoids t and tn as the upper neighbours of */
        /* the sole lower trapezoid */

        else if ((tr[t].d0 > 0) && (tr[t].d1 <= 0))
        {
            /* Only one trapezoid below */
            if ((tr[t].u0 > 0) && (tr[t].u1 > 0))
```

```
{
    /* continuation of a chain from abv. */
    if (tr[t].usave > 0) /* three upper neighbours */
    {
        if (tr[t].uside == S_LEFT)
        {
            tr[tn].u0 = tr[t].u1;
            tr[t].u1 = -1;
            tr[tn].u1 = tr[t].usave;

            tr[tr[t].u0].d0 = t;
            tr[tr[tn].u0].d0 = tn;
            tr[tr[tn].u1].d0 = tn;
        }
        else /* intersects in the right */
        {
            tr[tn].u1 = -1;
            tr[tn].u0 = tr[t].u1;
            tr[t].u1 = tr[t].u0;
            tr[t].u0 = tr[t].usave;

            tr[tr[t].u0].d0 = t;
            tr[tr[t].u1].d0 = t;
            tr[tr[tn].u0].d0 = tn;
        }

        tr[t].usave = tr[tn].usave = 0;
    }
    else /* No usave.... simple case */
    {
        tr[tn].u0 = tr[t].u1;
        tr[t].u1 = tr[tn].u1 = -1;
        tr[tr[tn].u0].d0 = tn;
    }
}
else
{
    /* fresh seg. or upward cusp */
    int tmp_u = tr[t].u0;
    int td0, td1;
    if (((td0 = tr[tmp_u].d0) > 0) &&
        ((td1 = tr[tmp_u].d1) > 0))
    {
        /* upward cusp */
        if ((tr[td0].rseg > 0) &&
            !is_left_of(tr[td0].rseg, &s.v1))
        {
            tr[t].u0 = tr[t].u1 = tr[tn].u1 = -1;
            tr[tr[tn].u0].d1 = tn;
        }
        else /* cusp going leftwards */
        {
            tr[tn].u0 = tr[tn].u1 = tr[t].u1 = -1;
            tr[tr[t].u0].d0 = t;
        }
    }
    else /* fresh segment */
    {
        tr[tr[t].u0].d0 = t;
        tr[tr[t].u0].d1 = tn;
    }
}

if (FP_EQUAL(tr[t].lo.y, tr[tlast].lo.y) &&
```

```
    FP_EQUAL(tr[t].lo.x, tr[tlast].lo.x) && tribot)
    {
        /* bottom forms a triangle */

        if (is_swapped)
        {
            tmptriseq = segnum - 1;
            if (tmptriseq == 0)
                tmptriseq = global.nseg;
        }
        else
            tmptriseq = MODULO_NEXT(segnum + 1, global.nseg);

        if ((tmptriseq > 0) && is_left_of(tmptriseq, &s.v0))
        {
            /* L-R downward cusp */
            tr[tr[t].d0].u0 = t;
            tr[tn].d0 = tr[tn].d1 = -1;
        }
        else
        {
            /* R-L downward cusp */
            tr[tr[tn].d0].u1 = tn;
            tr[t].d0 = tr[t].d1 = -1;
        }
    }
else
    {
        if ((tr[tr[t].d0].u0 > 0) && (tr[tr[t].d0].u1 > 0))
        {
            if (tr[tr[t].d0].u0 == t) /* passes thru LHS */
            {
                tr[tr[t].d0].usave = tr[tr[t].d0].u1;
                tr[tr[t].d0].uside = S_LEFT;
            }
            else
            {
                tr[tr[t].d0].usave = tr[tr[t].d0].u0;
                tr[tr[t].d0].uside = S_RIGHT;
            }
        }
        tr[tr[t].d0].u0 = t;
        tr[tr[t].d0].u1 = tn;
    }

    t = tr[t].d0;
}

else if ((tr[t].d0 <= 0) && (tr[t].d1 > 0))
{
    /* Only one trapezoid below */
    if ((tr[t].u0 > 0) && (tr[t].u1 > 0))
    {
        /* continuation of a chain from abv. */
        if (tr[t].usave > 0) /* three upper neighbours */
        {
            if (tr[t].uside == S_LEFT)
            {
                tr[tn].u0 = tr[t].u1;
                tr[t].u1 = -1;
                tr[tn].u1 = tr[t].usave;

                tr[tr[t].u0].d0 = t;
            }
        }
    }
}
```

```
        tr[tr[tn].u0].d0 = tn;
        tr[tr[tn].u1].d0 = tn;
    }
    else /* intersects in the right */
    {
        tr[tn].u1 = -1;
        tr[tn].u0 = tr[t].u1;
        tr[t].u1 = tr[t].u0;
        tr[t].u0 = tr[t].usave;

        tr[tr[t].u0].d0 = t;
        tr[tr[t].u1].d0 = t;
        tr[tr[tn].u0].d0 = tn;
    }

    tr[t].usave = tr[tn].usave = 0;
}
else /* No usave.... simple case */
{
    tr[tn].u0 = tr[t].u1;
    tr[t].u1 = tr[tn].u1 = -1;
    tr[tr[tn].u0].d0 = tn;
}
}
else
{
    /* fresh seg. or upward cusp */
    int tmp_u = tr[t].u0;
    int td0, td1;
    if (((td0 = tr[tmp_u].d0) > 0) &&
        ((td1 = tr[tmp_u].d1) > 0))
    {
        /* upward cusp */
        if ((tr[td0].rseg > 0) &&
            !is_left_of(tr[td0].rseg, &s.v1))
        {
            tr[t].u0 = tr[t].u1 = tr[tn].u1 = -1;
            tr[tr[tn].u0].d1 = tn;
        }
        else
        {
            tr[tn].u0 = tr[tn].u1 = tr[t].u1 = -1;
            tr[tr[t].u0].d0 = t;
        }
    }
    else /* fresh segment */
    {
        tr[tr[t].u0].d0 = t;
        tr[tr[t].u0].d1 = tn;
    }
}

if (FP_EQUAL(tr[t].lo.y, tr[tlast].lo.y) &&
    FP_EQUAL(tr[t].lo.x, tr[tlast].lo.x) && tribot)
{
    /* bottom forms a triangle */
    int tmpseg;
    if (is_swapped)
    {
        tmpseg = segnum - 1;
        if (tmpseg == 0)
            tmpseg = global.nseg;
    }
    else
```

```
        tmpseg = MODULO_NEXT(segnum + 1, global.nseg);

        if ((tmpseg > 0) && is_left_of(tmpseg, &s.v0))
        {
            /* L-R downward cusp */
            tr[tr[t].d1].u0 = t;
            tr[tn].d0 = tr[tn].d1 = -1;
        }
        else
        {
            /* R-L downward cusp */
            tr[tr[tn].d1].u1 = tn;
            tr[t].d0 = tr[t].d1 = -1;
        }
    }
    else
    {
        if ((tr[tr[t].d1].u0 > 0) && (tr[tr[t].d1].u1 > 0))
        {
            if (tr[tr[t].d1].u0 == t) /* passes thru LHS */
            {
                tr[tr[t].d1].usave = tr[tr[t].d1].u1;
                tr[tr[t].d1].uside = S_LEFT;
            }
            else
            {
                tr[tr[t].d1].usave = tr[tr[t].d1].u0;
                tr[tr[t].d1].uside = S_RIGHT;
            }
        }
        tr[tr[t].d1].u0 = t;
        tr[tr[t].d1].u1 = tn;
    }

    t = tr[t].d1;
}

/* two trapezoids below. Find out which one is intersected by */
/* this segment and proceed down that one */

else
{
    double y0, yt;
    point_t tmppt;
    int i_d0;

    i_d0 = FALSE;
    if (FP_EQUAL(tr[t].lo.y, s.v0.y))
    {
        if (tr[t].lo.x > s.v0.x)
            i_d0 = TRUE;
    }
    else
    {
        tmppt.y = y0 = tr[t].lo.y;
        yt = (y0 - s.v0.y)/(s.v1.y - s.v0.y);
        tmppt.x = s.v0.x + yt * (s.v1.x - s.v0.x);

        if (_less_than(&tmppt, &tr[t].lo))
            i_d0 = TRUE;
    }
}
```

```
/* check continuity from the top so that the lower-neighbour */
/* values are properly filled for the upper trapezoid */

if ((tr[t].u0 > 0) && (tr[t].u1 > 0))
{
    /* continuation of a chain from abv. */
    if (tr[t].usave > 0) /* three upper neighbours */
    {
        if (tr[t].uside == S_LEFT)
        {
            tr[tn].u0 = tr[t].u1;
            tr[t].u1 = -1;
            tr[tn].u1 = tr[t].usave;

            tr[tr[t].u0].d0 = t;
            tr[tr[tn].u0].d0 = tn;
            tr[tr[tn].u1].d0 = tn;
        }
        else /* intersects in the right */
        {
            tr[tn].u1 = -1;
            tr[tn].u0 = tr[t].u1;
            tr[t].u1 = tr[t].u0;
            tr[t].u0 = tr[t].usave;

            tr[tr[t].u0].d0 = t;
            tr[tr[t].u1].d0 = t;
            tr[tr[tn].u0].d0 = tn;
        }

        tr[t].usave = tr[tn].usave = 0;
    }
    else /* No usave.... simple case */
    {
        tr[tn].u0 = tr[t].u1;
        tr[tn].u1 = -1;
        tr[t].u1 = -1;
        tr[tr[tn].u0].d0 = tn;
    }
}
else
{
    /* fresh seg. or upward cusp */
    int tmp_u = tr[t].u0;
    int td0, td1;
    if (((td0 = tr[tmp_u].d0) > 0) &&
        ((td1 = tr[tmp_u].d1) > 0))
    {
        /* upward cusp */
        if ((tr[td0].rseg > 0) &&
            !is_left_of(tr[td0].rseg, &s.v1))
        {
            tr[t].u0 = tr[t].u1 = tr[tn].u1 = -1;
            tr[tr[tn].u0].d1 = tn;
        }
        else
        {
            tr[tn].u0 = tr[tn].u1 = tr[t].u1 = -1;
            tr[tr[t].u0].d0 = t;
        }
    }
    else /* fresh segment */
    {

```



```
        tr[tr[t].u0].d0 = t;
        tr[tr[t].u0].d1 = tn;
    }
}

if (FP_EQUAL(tr[t].lo.y, tr[tlast].lo.y) &&
    FP_EQUAL(tr[t].lo.x, tr[tlast].lo.x) && tribot)
{
    /* this case arises only at the lowest trapezoid.. i.e.
       tlast, if the lower endpoint of the segment is
       already inserted in the structure */

    tr[tr[t].d0].u0 = t;
    tr[tr[t].d0].u1 = -1;
    tr[tr[t].d1].u0 = tn;
    tr[tr[t].d1].u1 = -1;

    tr[tn].d0 = tr[t].d1;
    tr[t].d1 = tr[tn].d1 = -1;

    tnext = tr[t].d1;
}
else if (i_d0)
    /* intersecting d0 */
    {
        tr[tr[t].d0].u0 = t;
        tr[tr[t].d0].u1 = tn;
        tr[tr[t].d1].u0 = tn;
        tr[tr[t].d1].u1 = -1;

        /* new code to determine the bottom neighbours of the */
        /* newly partitioned trapezoid */

        tr[t].d1 = -1;

        tnext = tr[t].d0;
    }
else
    /* intersecting d1 */
    {
        tr[tr[t].d0].u0 = t;
        tr[tr[t].d0].u1 = -1;
        tr[tr[t].d1].u0 = t;
        tr[tr[t].d1].u1 = tn;

        /* new code to determine the bottom neighbours of the */
        /* newly partitioned trapezoid */

        tr[tn].d0 = tr[t].d1;
        tr[tn].d1 = -1;

        tnext = tr[t].d1;
    }

t = tnext;
}

tr[t_sav].rseg = tr[tn_sav].lseg = segnum;
} /* end-while */
```

/\* Now combine those trapezoids which share common segments. We can \*/  
/\* use the pointers to the parent to connect these together. This \*/

```
/* works only because all these new trapezoids have been formed */
/* due to splitting by the segment, and hence have only one parent */

tfirstl = tfirst;
tlastl = tlast;
merge_trapezoids(segnum, tfirstl, tlastl, S_LEFT);
merge_trapezoids(segnum, tfistr, tlastr, S_RIGHT);

seg[segnum].is_inserted = TRUE;
return 0;
}

/* Update the roots stored for each of the endpoints of the segment.
 * This is done to speed up the location-query for the endpoint when
 * the segment is inserted into the trapezoidation subsequently
 */
static int find_new_roots(segnum)
    int segnum;
{
    segment_t *s = &seg[segnum];
    point_t vper;

    if (s->is_inserted)
        return 0;

    vper.x = s->v0.x + EPS * (s->v1.x - s->v0.x);
    vper.y = s->v0.y + EPS * (s->v1.y - s->v0.y);
    s->root0 = locate_endpoint(&s->v0, &s->v1, s->root0);
    s->root0 = tr[s->root0].sink;

    vper.x = s->v1.x + EPS * (s->v0.x - s->v1.x);
    vper.y = s->v1.y + EPS * (s->v0.y - s->v1.y);
    s->root1 = locate_endpoint(&s->v1, &s->v0, s->root1);
    s->root1 = tr[s->root1].sink;
    return 0;
}

/* Main routine to perform trapezoidation */
int construct_trapezoids(nseg, seg)
    int nseg;
    segment_t *seg;
{
    register int i;
    int root, h;

    q_idx = tr_idx = 1;
    /* Add the first segment and get the query structure and trapezoid */
    /* list initialised */
    root = init_query_structure(choose_segment());

#ifdef SIMPLE
    /* no randomization */

    for (i = 1; i <= nseg; i++)
        seg[i].root0 = seg[i].root1 = root;

    for (i = 2; i <= nseg; i++)
        add_segment(choose_segment());
#else

```

```
for (i = 1; i <= nseg; i++)
    seg[i].root0 = seg[i].root1 = root;

for (h = 1; h <= math_logstar_n(nseg); h++)
{
    for (i = math_N(nseg, h - 1) + 1; i <= math_N(nseg, h); i++)
        add_segment(choose_segment());

    /* Find a new root for each of the segment endpoints */
    for (i = 1; i <= nseg; i++)
        find_new_roots(i);
}

for (i = math_N(nseg, math_logstar_n(nseg)) + 1; i <= nseg; i++)
    add_segment(choose_segment());

#endif

}
```

0.0 0.0  
10.0 0.0  
20.0 0.0  
20 5.0  
20.0 20.0  
0.0 20.0  
5.0 10.0

```
inclpath = .

CC=gcc

CFLAGS= -UCHOOSE_MANUAL -UDEBUG -USIMPLE -DSTANDALONE -UCLOCK\
        -I$(inclpath) -L/lib/pa1.1 -g

# DEBUG: turn on debugging output
# CLOCK: time the triangulation process
#
# STANDALONE: run as a separate program. read data from file.
#             If this flag is False, then use the interface procedure
#             triangulate_polygon() instead.
#
# SIMPLE: if defined, turn off randomization
#

LD_FLAGS= -lm

objects= construct.o misc.o monotone.o tri.o
executable = triangulate

$(executable): $(objects)
    rm -f $(executable)
    $(CC) $(CFLAGS) $(objects) $(LD_FLAGS) -o $(executable)

$(objects): $(inclpath)/basic.h

clean:
    rm -f $(objects)
```

```
#include "basic.h"
#include <sys/time.h>
#include <math.h>

/* If you do not have log2() on your machine, use this macro:
#define log2(x) (log((double)x)/log(2.0))
*/

#ifdef __STDC__
extern double log2(double);
#else
extern double log2();
#endif

static int choose_idx;
static int permute[SEGSIZE];

/* Generate a random permutation of the segments 1..n */
int generate_random_ordering(n)
    int n;
{
    struct timeval tval;
    struct timezone tzone;
    register int i;
    int m, st[SEGSIZE], *p;

    choose_idx = 1;
    gettimeofday(&tval, &tzone);
    srand48(tval.tv_sec);

    for (i = 0; i <= n; i++)
        st[i] = i;

    p = st;
    for (i = 1; i <= n; i++, p++)
    {
        m = lrand48() % (n + 1 - i) + 1;
        permute[i] = p[m];
        if (m != 1)
            p[m] = p[1];
    }
    return 0;
}

/* Return the next segment in the generated random ordering of all the */
/* segments in S */
int choose_segment()
{
    int i;
    /*
#ifdef DEBUG
    fprintf(stderr, "choose_segment: %d\n", permute[choose_idx]);
#endif
    return permute[choose_idx++];
*/
}

#ifdef CHOOSE_MANUAL
printf("Enter seg: ");
scanf("%d", &i);
return i;
#endif
```

```
#else

#ifdef DEBUG
    fprintf(stderr, "choose_segment: %d\n", permute[choose_idx]);
#endif
    return permute[choose_idx++];
#endif
}

int inserted(segnum, whichpt)
    int segnum;
    int whichpt;
{
    int n1, n2;

    n1 = segnum % global.nseg + 1; /* next seg. */
    n2 = (segnum - 1 + global.nseg - 1) % global.nseg + 1; /* prev. */

    if (whichpt == FIRSTPT)
        return seg[n2].is_inserted;
    else
        return seg[n1].is_inserted;
}

#ifdef STANDALONE

/* Read in the list of vertices from infile */
int read_segments(infile)
    FILE *infile;
{
    int nseg;
    register int i;

    memset((void *)seg, 0, sizeof(seg));
    i = 1;
    nseg = 0;
    while (fscanf(infile, "%lf%lf", &seg[i].v0.x, &seg[i].v0.y) == 2)
    {
        seg[i - 1].v1.x = seg[i].v0.x;
        seg[i - 1].v1.y = seg[i].v0.y;
        seg[i].is_inserted = FALSE;
        i++;
        nseg++;
    }
    seg[nseg].v1.x = seg[1].v0.x;
    seg[nseg].v1.y = seg[1].v0.y;

    global.nseg = nseg;
    return nseg;
}

#endif

/* Get log*n for given n */
int math_logstar_n(n)
    int n;
{
    register int i;
```

```
double v;

for (i = 0, v = (double) n; v >= 1; i++)
    v = log2(v);

return (i - 1);
}

int math_N(n, h)
    int n;
    int h;
{
    register int i;
    double v;

    for (i = 0, v = (int) n; i < h; i++)
        v = log2(v);

    return (int) ceil((double) 1.0*n/v);
}
```



```
#include "basic.h"
#include <math.h>

#define CROSS_SINE(v0, v1) ((v0).x * (v1).y - (v1).x * (v0).y)
#define LENGTH(v0) (sqrt((v0).x * (v0).x + (v0).y * (v0).y))

static monchain_t mchain[TRSIZE]; /* Table to hold all the monotone */
                                   /* polygons . Each monotone polygon */
                                   /* is a circularly linked list */

static vertexchain_t vert[SEGSIZE]; /* chain init. information. This */
                                     /* is used to decide which */
                                     /* monotone polygon to split if */
                                     /* there are several other */
                                     /* polygons touching at the same */
                                     /* vertex */

static int mon[SEGSIZE]; /* contains position of any vertex in */
                         /* the monotone chain for the polygon */

static int visited[TRSIZE];
static int chain_idx, op_idx, mon_idx;

/* Function returns TRUE if the trapezoid lies inside the polygon */
static int inside_polygon(t)
    trap_t *t;
{
    int rseg = t->rseg;

    if (t->state == ST_INVALID)
        return 0;

    if ((t->lseg <= 0) || (t->rseg <= 0))
        return 0;

    if (((t->u0 <= 0) && (t->u1 <= 0)) ||
        ((t->d0 <= 0) && (t->d1 <= 0))) /* triangle */
        return (_greater_than(&seg[rseg].v1, &seg[rseg].v0));

    return 0;
}

/* return a new mon structure from the table */
static int newmon()
{
    return ++mon_idx;
}

/* return a new chain element from the table */
static int new_chain_element()
{
    return ++chain_idx;
}

static double get_angle(vp0, vpnext, vp1)
    point_t *vp0;
    point_t *vpnext;
    point_t *vp1;
{

```

```
point_t v0, v1;

v0.x = vpnext->x - vp0->x;
v0.y = vpnext->y - vp0->y;

v1.x = vp1->x - vp0->x;
v1.y = vp1->y - vp0->y;

if (CROSS_SINE(v0, v1) >= 0) /* sine is positive */
    return DOT(v0, v1)/LENGTH(v0)/LENGTH(v1);
else
    return (-1.0 * DOT(v0, v1)/LENGTH(v0)/LENGTH(v1) - 2);
}

/* (v0, v1) is the new diagonal to be added to the polygon. Find which */
/* chain to use and return the positions of v0 and v1 in p and q */
static int get_vertex_positions(v0, v1, ip, iq)
    int v0;
    int v1;
    int *ip;
    int *iq;
{
    vertexchain_t *vp0, *vp1;
    register int i;
    double angle, temp;
    int tp, tq;

    vp0 = &vert[v0];
    vp1 = &vert[v1];

    /* p is identified as follows. Scan from (v0, v1) rightwards till */
    /* you hit the first segment starting from v0. That chain is the */
    /* chain of our interest */

    angle = -4.0;
    for (i = 0; i < 4; i++)
    {
        if (vp0->vnext[i] <= 0)
            continue;
        if ((temp = get_angle(&vp0->pt, &(vert[vp0->vnext[i]].pt),
                             &vp1->pt)) > angle)
        {
            angle = temp;
            tp = i;
        }
    }

    *ip = tp;

    /* Do similar actions for q */

    angle = -4.0;
    for (i = 0; i < 4; i++)
    {
        if (vp1->vnext[i] <= 0)
            continue;
        if ((temp = get_angle(&vp1->pt, &(vert[vp1->vnext[i]].pt),
                             &vp0->pt)) > angle)
        {
            angle = temp;
        }
    }
}
```

```
        tq = i;
    }
}

*iq = tq;

return 0;
}

/* v0 and v1 are specified in anti-clockwise order with respect to
 * the current monotone polygon mcur. Split the current polygon into
 * two polygons using the diagonal (v0, v1)
 */
static int make_new_monotone_poly(mcur, v0, v1)
    int mcur;
    int v0;
    int v1;
{
    int p, q, ip, iq;
    int mnew = newmon();
    int i, j, nf0, nf1;
    vertexchain_t *vp0, *vp1;

    vp0 = &vert[v0];
    vp1 = &vert[v1];

    get_vertex_positions(v0, v1, &ip, &iq);

    p = vp0->vpos[ip];
    q = vp1->vpos[iq];

    /* At this stage, we have got the positions of v0 and v1 in the */
    /* desired chain. Now modify the linked lists */

    i = new_chain_element();      /* for the new list */
    j = new_chain_element();

    mchain[i].vnum = v0;
    mchain[j].vnum = v1;

    mchain[i].next = mchain[p].next;
    mchain[mchain[p].next].prev = i;
    mchain[i].prev = j;
    mchain[j].next = i;
    mchain[j].prev = mchain[q].prev;
    mchain[mchain[q].prev].next = j;

    mchain[p].next = q;
    mchain[q].prev = p;

    nf0 = vp0->nextfree;
    nf1 = vp1->nextfree;

    vp0->vnext[ip] = v1;

    vp0->vpos[nf0] = i;
    vp0->vnext[nf0] = mchain[mchain[i].next].vnum;
    vp1->vpos[nf1] = j;
    vp1->vnext[nf1] = v0;
}
```

```
    vp0->nextfree++;
    vp1->nextfree++;

#ifdef DEBUG
    fprintf(stderr, "make_poly: mcur = %d, (v0, v1) = (%d, %d)\n",
               mcur, v0, v1);
    fprintf(stderr, "next posns = (p, q) = (%d, %d)\n", p, q);
#endif

    mon[mcur] = p;
    mon[mnew] = i;
    return mnew;
}

/* Main routine to get monotone polygons from the trapezoidation of
 * the polygon.
 */

int monotone_trapezoids(n)
    int n;
{
    register int i;
    int tr_start;

    memset((void *)vert, 0, sizeof(vert));
    memset((void *)visited, 0, sizeof(visited));
    memset((void *)mchain, 0, sizeof(mchain));
    memset((void *)mon, 0, sizeof(mon));

    /* First locate a trapezoid which lies inside the polygon */
    /* and which is triangular */
    for (i = 0; i < TRSIZE; i++)
        if (inside_polygon(&tr[i]))
            break;
    tr_start = i;

    /* Initialise the mon data-structure and start spanning all the */
    /* trapezoids within the polygon */

    for (i = 1; i <= n; i++)
    {
        mchain[i].prev = i - 1;
        mchain[i].next = i + 1;
        mchain[i].vnum = i;
        vert[i].pt = seg[i].v0;
        vert[i].vnext[0] = i + 1; /* next vertex */
        vert[i].vpos[0] = i;      /* locn. of next vertex */
        vert[i].nextfree = 1;
    }
    mchain[1].prev = n;
    mchain[n].next = 1;
    vert[n].vnext[0] = 1;
    vert[n].vpos[0] = n;
    chain_idx = n;
    mon_idx = 0;
    mon[0] = 1; /* position of any vertex in the first */
               /* chain */

    /* traverse the polygon */
    if (tr[tr_start].u0 > 0)
        traverse_polygon(0, tr_start, tr[tr_start].u0, TR_FROM_UP);
}
```

```
    else if (tr[tr_start].d0 > 0)
        traverse_polygon(0, tr_start, tr[tr_start].d0, TR_FROM_DN);

    /* return the number of polygons created */
    return newmon();
}

/* recursively visit all the trapezoids */
int traverse_polygon(mcur, trnum, from, dir)
    int mcur;
    int trnum;
    int from;
    int dir;
{
    trap_t *t = &tr[trnum];
    int mnew;
    int v0, v1;
    int retval;
    int do_switch = FALSE;

    if ((trnum <= 0) || visited[trnum])
        return 0;

    visited[trnum] = TRUE;

    /* We have much more information available here. */
    /* rseg: goes upwards */
    /* lseg: goes downwards */

    /* Initially assume that dir = TR_FROM_DN (from the left) */
    /* Switch v0 and v1 if necessary afterwards */

    /* special cases for triangles with cusps at the opposite ends. */
    /* take care of this first */
    if ((t->u0 <= 0) && (t->u1 <= 0))
    {
        if ((t->d0 > 0) && (t->d1 > 0)) /* downward opening triangle */
        {
            v0 = tr[t->d1].lseg;
            v1 = t->lseg;
            if (from == t->d1)
            {
                do_switch = TRUE;
                mnew = make_new_monotone_poly(mcur, v1, v0);
                traverse_polygon(mcur, t->d1, trnum, TR_FROM_UP);
                traverse_polygon(mnew, t->d0, trnum, TR_FROM_UP);
            }
            else
            {
                mnew = make_new_monotone_poly(mcur, v0, v1);
                traverse_polygon(mcur, t->d0, trnum, TR_FROM_UP);
                traverse_polygon(mnew, t->d1, trnum, TR_FROM_UP);
            }
        }
        else
        {
            retval = SP_NOSPLIT; /* Just traverse all neighbours */
            traverse_polygon(mcur, t->u0, trnum, TR_FROM_DN);
            traverse_polygon(mcur, t->u1, trnum, TR_FROM_DN);
        }
    }
}
```

```
        traverse_polygon(mcur, t->d0, trnum, TR_FROM_UP);
        traverse_polygon(mcur, t->d1, trnum, TR_FROM_UP);
    }
}

else if ((t->d0 <= 0) && (t->d1 <= 0))
{
    if ((t->u0 > 0) && (t->u1 > 0)) /* upward opening triangle */
    {
        v0 = t->rseg;
        v1 = tr[t->u0].rseg;
        if (from == t->u1)
        {
            do_switch = TRUE;
            mnew = make_new_monotone_poly(mcur, v1, v0);
            traverse_polygon(mcur, t->u1, trnum, TR_FROM_DN);
            traverse_polygon(mnew, t->u0, trnum, TR_FROM_DN);
        }
        else
        {
            mnew = make_new_monotone_poly(mcur, v0, v1);
            traverse_polygon(mcur, t->u0, trnum, TR_FROM_DN);
            traverse_polygon(mnew, t->u1, trnum, TR_FROM_DN);
        }
    }
    else
    {
        retval = SP_NOSPLIT; /* Just traverse all neighbours */
        traverse_polygon(mcur, t->u0, trnum, TR_FROM_DN);
        traverse_polygon(mcur, t->u1, trnum, TR_FROM_DN);
        traverse_polygon(mcur, t->d0, trnum, TR_FROM_UP);
        traverse_polygon(mcur, t->d1, trnum, TR_FROM_UP);
    }
}

else if ((t->u0 > 0) && (t->u1 > 0))
{
    if ((t->d0 > 0) && (t->d1 > 0)) /* downward + upward cusps */
    {
        v0 = tr[t->d1].lseg;
        v1 = tr[t->u0].rseg;
        retval = SP_2UP_2DN;
        if (((dir == TR_FROM_DN) && (t->d1 == from)) ||
            ((dir == TR_FROM_UP) && (t->u1 == from)))
        {
            do_switch = TRUE;
            mnew = make_new_monotone_poly(mcur, v1, v0);
            traverse_polygon(mcur, t->u1, trnum, TR_FROM_DN);
            traverse_polygon(mcur, t->d1, trnum, TR_FROM_UP);
            traverse_polygon(mnew, t->u0, trnum, TR_FROM_DN);
            traverse_polygon(mnew, t->d0, trnum, TR_FROM_UP);
        }
        else
        {
            mnew = make_new_monotone_poly(mcur, v0, v1);
            traverse_polygon(mcur, t->u0, trnum, TR_FROM_DN);
            traverse_polygon(mcur, t->d0, trnum, TR_FROM_UP);
            traverse_polygon(mnew, t->u1, trnum, TR_FROM_DN);
            traverse_polygon(mnew, t->d1, trnum, TR_FROM_UP);
        }
    }
}
```

```
else /* only downward cusp */
{
    if (_equal_to(&t->lo, &seg[t->lseg].v1))
    {
        v0 = tr[t->u0].rseg;
        v1 = MODULO_NEXT(t->lseg + 1, global.nseg);
        retval = SP_2UP_LEFT;
        if ((dir == TR_FROM_UP) && (t->u0 == from))
        {
            do_switch = TRUE;
            mnew = make_new_monotone_poly(mcur, v1, v0);
            traverse_polygon(mcur, t->u0, trnum, TR_FROM_DN);
            traverse_polygon(mnew, t->d0, trnum, TR_FROM_UP);
            traverse_polygon(mnew, t->u1, trnum, TR_FROM_DN);
            traverse_polygon(mnew, t->d1, trnum, TR_FROM_UP);
        }
        else
        {
            mnew = make_new_monotone_poly(mcur, v0, v1);
            traverse_polygon(mcur, t->u1, trnum, TR_FROM_DN);
            traverse_polygon(mcur, t->d0, trnum, TR_FROM_UP);
            traverse_polygon(mcur, t->d1, trnum, TR_FROM_UP);
            traverse_polygon(mnew, t->u0, trnum, TR_FROM_DN);
        }
    }
    else
    {
        v0 = t->rseg;
        v1 = tr[t->u0].rseg;
        retval = SP_2UP_RIGHT;
        if ((dir == TR_FROM_UP) && (t->u1 == from))
        {
            do_switch = TRUE;
            mnew = make_new_monotone_poly(mcur, v1, v0);
            traverse_polygon(mcur, t->u1, trnum, TR_FROM_DN);
            traverse_polygon(mnew, t->d1, trnum, TR_FROM_UP);
            traverse_polygon(mnew, t->d0, trnum, TR_FROM_UP);
            traverse_polygon(mnew, t->u0, trnum, TR_FROM_DN);
        }
        else
        {
            mnew = make_new_monotone_poly(mcur, v0, v1);
            traverse_polygon(mcur, t->u0, trnum, TR_FROM_DN);
            traverse_polygon(mcur, t->d0, trnum, TR_FROM_UP);
            traverse_polygon(mcur, t->d1, trnum, TR_FROM_UP);
            traverse_polygon(mnew, t->u1, trnum, TR_FROM_DN);
        }
    }
}
}
else if ((t->u0 > 0) || (t->u1 > 0)) /* no downward cusp */
{
    if ((t->d0 > 0) && (t->d1 > 0)) /* only upward cusp */
    {
        if (_equal_to(&t->hi, &seg[t->lseg].v0))
        {
            v0 = tr[t->d1].lseg;
            v1 = t->lseg;
            retval = SP_2DN_LEFT;
            if (!(dir == TR_FROM_DN) && (t->d0 == from))
            {
```

```
        do_switch = TRUE;
        mnew = make_new_monotone_poly(mcur, v1, v0);
        traverse_polygon(mcur, t->u1, trnum, TR_FROM_DN);
        traverse_polygon(mcur, t->d1, trnum, TR_FROM_UP);
        traverse_polygon(mcur, t->u0, trnum, TR_FROM_DN);
        traverse_polygon(mnew, t->d0, trnum, TR_FROM_UP);
    }
else
{
    mnew = make_new_monotone_poly(mcur, v0, v1);
    traverse_polygon(mcur, t->d0, trnum, TR_FROM_UP);
    traverse_polygon(mnew, t->u0, trnum, TR_FROM_DN);
    traverse_polygon(mnew, t->u1, trnum, TR_FROM_DN);
    traverse_polygon(mnew, t->d1, trnum, TR_FROM_UP);
}
}
else
{
    v0 = tr[t->d1].lseg;
    v1 = MODULO_NEXT(t->rseg + 1, global.nseg);
    retval = SP_2DN_RIGHT;
    if ((dir == TR_FROM_DN) && (t->d1 == from))
    {
        do_switch = TRUE;
        mnew = make_new_monotone_poly(mcur, v1, v0);
        traverse_polygon(mcur, t->d1, trnum, TR_FROM_UP);
        traverse_polygon(mnew, t->u1, trnum, TR_FROM_DN);
        traverse_polygon(mnew, t->u0, trnum, TR_FROM_DN);
        traverse_polygon(mnew, t->d0, trnum, TR_FROM_UP);
    }
else
{
    mnew = make_new_monotone_poly(mcur, v0, v1);
    traverse_polygon(mcur, t->u0, trnum, TR_FROM_DN);
    traverse_polygon(mcur, t->d0, trnum, TR_FROM_UP);
    traverse_polygon(mcur, t->u1, trnum, TR_FROM_DN);
    traverse_polygon(mnew, t->d1, trnum, TR_FROM_UP);
}
}
}
else
/* no cusp */
{
    if (_equal_to(&t->hi, &seg[t->lseg].v0) &&
        _equal_to(&t->lo, &seg[t->rseg].v0))
    {
        v0 = t->rseg;
        v1 = t->lseg;
        retval = SP_SIMPLE_LRDN;
        if (dir == TR_FROM_UP)
        {
            do_switch = TRUE;
            mnew = make_new_monotone_poly(mcur, v1, v0);
            traverse_polygon(mcur, t->u0, trnum, TR_FROM_DN);
            traverse_polygon(mcur, t->u1, trnum, TR_FROM_DN);
            traverse_polygon(mnew, t->d1, trnum, TR_FROM_UP);
            traverse_polygon(mnew, t->d0, trnum, TR_FROM_UP);
        }
else
{
            mnew = make_new_monotone_poly(mcur, v0, v1);
            traverse_polygon(mcur, t->d1, trnum, TR_FROM_UP);
        }
    }
}
```



```
        traverse_polygon(mcur, t->d0, trnum, TR_FROM_UP);
        traverse_polygon(mnew, t->u0, trnum, TR_FROM_DN);
        traverse_polygon(mnew, t->u1, trnum, TR_FROM_DN);
    }
}
else if (_equal_to(&t->hi, &seg[t->rseg].v1) &&
        _equal_to(&t->lo, &seg[t->lseg].v1))
{
    v0 = MODULO_NEXT(t->rseg + 1, global.nseg);
    v1 = MODULO_NEXT(t->lseg + 1, global.nseg);
    retval = SP_SIMPLE_LRUP;
    if (dir == TR_FROM_UP)
    {
        do_switch = TRUE;
        mnew = make_new_monotone_poly(mcur, v1, v0);
        traverse_polygon(mcur, t->u0, trnum, TR_FROM_DN);
        traverse_polygon(mcur, t->u1, trnum, TR_FROM_DN);
        traverse_polygon(mnew, t->d1, trnum, TR_FROM_UP);
        traverse_polygon(mnew, t->d0, trnum, TR_FROM_UP);
    }
    else
    {
        mnew = make_new_monotone_poly(mcur, v0, v1);
        traverse_polygon(mcur, t->d1, trnum, TR_FROM_UP);
        traverse_polygon(mcur, t->d0, trnum, TR_FROM_UP);
        traverse_polygon(mnew, t->u0, trnum, TR_FROM_DN);
        traverse_polygon(mnew, t->u1, trnum, TR_FROM_DN);
    }
}
else /* no split possible */
{
    retval = SP_NOSPLIT;
    traverse_polygon(mcur, t->u0, trnum, TR_FROM_DN);
    traverse_polygon(mcur, t->d0, trnum, TR_FROM_UP);
    traverse_polygon(mcur, t->u1, trnum, TR_FROM_DN);
    traverse_polygon(mcur, t->d1, trnum, TR_FROM_UP);
}
}
}
```

```
    return retval;
}
```

```
int triangulate_monotone_polygons(nmonpoly, op)
```

```
    int nmonpoly;
```

```
    int op[][3];
```

```
{
    register int i;
    point_t ymax, ymin;
    int p, vfirst, posmax, posmin, v;
    int vcount;
```

```
#ifdef DEBUG
```

```
    for (i = 0; i < nmonpoly; i++)
```

```
    {
        fprintf(stderr, "\n\nPolygon %d: ", i);
        vfirst = mchain[mon[i]].vnum;
        p = mchain[mon[i]].next;
        fprintf(stderr, "%d ", mchain[mon[i]].vnum);
        while (mchain[p].vnum != vfirst)
```

```
{
    fprintf(stderr, "%d ", mchain[p].vnum);
    p = mchain[p].next;
}
}
fprintf(stderr, "\n");
#endif

op_idx = 0;
for (i = 0; i < nmonpoly; i++)
{
    vcount = 1;
    vfirst = mchain[mon[i]].vnum;
    ymax = ymin = vert[vfirst].pt;
    posmax = posmin = mon[i];
    p = mchain[mon[i]].next;
    while ((v = mchain[p].vnum) != vfirst)
    {
        if (_greater_than(&vert[v].pt, &ymax))
        {
            ymax = vert[v].pt;
            posmax = p;
        }
        if (_less_than(&vert[v].pt, &ymin))
        {
            ymin = vert[v].pt;
            posmin = p;
        }
        p = mchain[p].next;
        vcount++;
    }

    if (vcount == 3)          /* already a triangle */
    {
        op[op_idx][0] = mchain[p].vnum;
        op[op_idx][1] = mchain[mchain[p].next].vnum;
        op[op_idx][2] = mchain[mchain[p].prev].vnum;
        op_idx++;
    }
    else                      /* triangulate the polygon */
    {
        v = mchain[mchain[posmax].next].vnum;
        if (_equal_to(&vert[v].pt, &ymin))
        {
            /* LHS is a single line */
            triangulate_single_polygon(posmax, TRI_LHS, op);
        }
        else
            triangulate_single_polygon(posmax, TRI_RHS, op);
    }
}

#ifdef DEBUG
    for (i = 0; i < (global.nseg - 2); i++)
        fprintf(stderr, "tri #%d: (%d, %d, %d)\n", i, op[i][0], op[i][1],
            op[i][2]);
#endif
}

/* A greedy corner-cutting algorithm to triangulate a y-monotone
 * polygon in O(n) time.
```

\* Joseph O'Rourke, Computational Geometry in C.

\*/

```
int triangulate_single_polygon(posmax, side, op)
    int posmax;
    int side;
    int op[][3];
{
    register int v;
    int rc[SEGSIZE], ri = 0;          /* reflex chain */
    int endv, tmp, vpos;

    if (side == TRI_RHS)              /* RHS segment is a single segment */
    {
        rc[0] = mchain[posmax].vnum;
        tmp = mchain[posmax].next;
        rc[1] = mchain[tmp].vnum;
        ri = 1;

        vpos = mchain[tmp].next;
        v = mchain[vpos].vnum;

        if ((endv = mchain[mchain[posmax].prev].vnum) == 0)
            endv = global.nseg;
    }
    else                               /* LHS is a single segment */
    {
        tmp = mchain[posmax].next;
        rc[0] = mchain[tmp].vnum;
        tmp = mchain[tmp].next;
        rc[1] = mchain[tmp].vnum;
        ri = 1;

        vpos = mchain[tmp].next;
        v = mchain[vpos].vnum;

        endv = mchain[posmax].vnum;
    }

    while ((v != endv) || (ri > 1))
    {
        if (ri > 0)                   /* reflex chain is non-empty */
        {
            if (CROSS(vert[v].pt, vert[rc[ri - 1]].pt,
                      vert[rc[ri]].pt) > 0)
            {
                /* convex corner: cut it off */
                op[op_idx][0] = rc[ri - 1];
                op[op_idx][1] = rc[ri];
                op[op_idx][2] = v;
                op_idx++;
                ri--;
            }
            else                       /* non-convex */
            {
                /* add v to the chain */
                ri++;
                rc[ri] = v;
                vpos = mchain[vpos].next;
                v = mchain[vpos].vnum;
            }
        }
        else                           /* reflex-chain empty: add v to the */
        {                               /* reflex chain and advance it */

```

```
        rc[++ri] = v;
        vpos = mchain[vpos].next;
        v = mchain[vpos].vnum;
    }
} /* end-while */

/* reached the bottom vertex. Add in the triangle formed */
op[op_idx][0] = rc[ri - 1];
op[op_idx][1] = rc[ri];
op[op_idx][2] = v;
op_idx++;
ri--;

return 0;
}
```

```
#include "basic.h"
#include <sys/time.h>

static int initialise(nseg)
    int nseg;
{
    register int i;

    for (i = 1; i <= nseg; i++)
        seg[i].is_inserted = FALSE;
    generate_random_ordering(nseg);

    return 0;
}

#ifdef STANDALONE

int main(argc, argv)
    int argc;
    char *argv[];
{
    int n, nmonpoly;
    FILE *infile;
    int op[SEGSIZE][3], i;

    if (argc < 2)
    {
        fprintf(stderr, "usage: triangulate <filename>\n");
        exit(1);
    }
    else
        if ((infile = fopen(argv[1], "r")) == NULL)
        {
            perror(argv[1]);
            exit(1);
        }

#ifdef CLOCK
    clock();
#endif

    n = read_segments(infile);
    for (i = 0; i < 1; i++)
    {
        initialise(n);
        construct_trapezoids(n, seg);
        nmonpoly = monotonate_trapezoids(n);
        triangulate_monotone_polygons(nmonpoly, op);
    }

#ifdef CLOCK
    printf("CPU time used: %ld microseconds\n", clock());
#endif

    return 0;
}

#else /* Not standalone. Use this as an interface routine */

/* The points constituting the polygon are specified in anticlockwise
```

```
* order. If there are n points in the polygon, i/p would be the
* points p0, p1...p(n) (where p0 and pn are the same point). The
* output is contained in the array "triangles".
* Every triangle is output in anticlockwise order and the 3
* integers are the indices of the points. Thus, the triangle (i, j, k)
* refers to the triangle formed by the points (pi, pj, pk). Before
* using this routine, please check-out that you do not conflict with
* the global variables defined in basic.h.
*
* n:          number of points in polygon (p0 = pn)
* vertices:   the vertices p0, p1..., p(n) of the polygon
* triangles:  output array containing the triangles
*/
```

```
int triangulate_polygon(n, vertices, triangles)
    int n;
    double vertices[][2];
    int triangles[][3];
{
    register int i;
    int nmonpoly;

    memset((void *)seg, 0, sizeof(seg));
    for (i = 1; i <= n; i++)
    {
        seg[i].is_inserted = FALSE;

        seg[i].v0.x = vertices[i][0]; /* x-cood */
        seg[i].v0.y = vertices[i][1]; /* y-cood */
        if (i == 1)
        {
            seg[n].v1.x = seg[i].v0.x;
            seg[n].v1.y = seg[i].v0.y;
        }
        else
        {
            seg[i - 1].v1.x = seg[i].v0.x;
            seg[i - 1].v1.y = seg[i].v0.y;
        }
    }

    global.nseg = n;
    initialise(n);
    construct_trapezoids(n, seg);
    nmonpoly = monotone_trapezoids(n);
    triangulate_monotone_polygons(nmonpoly, triangles);

    return 0;
}
```

```
/* This function returns TRUE or FALSE depending upon whether the
* vertex is inside the polygon or not. The polygon must already have
* been triangulated before this routine is called.
* This routine will always detect all the points belonging to the
* set (polygon-area - polygon-boundary). The return value for points
* on the boundary is not consistent!!!
*/
```

```
int is_point_inside_polygon(vertex)
    double vertex[2];
```

```
{
    point_t v;
    int trnum, rseg;
    trap_t *t;

    v.x = vertex[0];
    v.y = vertex[1];

    trnum = locate_endpoint(v, v, 1);
    t = &tr[trnum];

    if (t->state == ST_INVALID)
        return FALSE;

    if ((t->lseg <= 0) || (t->rseg <= 0))
        return FALSE;
    rseg = t->rseg;
    return _greater_than_equal_to(&seg[rseg].v1, &seg[rseg].v0);
}

#endif
```

```
/*
 * hot.c - Scan an image for pixels with RGB values that will give
 *         "unsafe" values of chrominance signal or composite signal
 *         amplitude when encoded into an NTSC or PAL colour signal.
 *         (This happens for certain high-intensity high-saturation colours
 *         that are rare in real scenes, but can easily be present
 *         in synthetic images.)
 *
 *         Such pixels can be flagged so the user may then choose other
 *         colours. Or, the offending pixels can be made "safe"
 *         in a manner that preserves hue.
 *
 *         There are two reasonable ways to make a pixel "safe":
 *         We can reduce its intensity (luminance) while leaving
 *         hue and saturation the same. Or, we can reduce saturation
 *         while leaving hue and luminance the same. A #define selects
 *         which strategy to use.
 *
 * Note to the user: You must add your own read_pixel() and write_pixel()
 * routines. You may have to modify pix_decode() and pix_encode().
 * MAXPIX, WID, and HGT are likely to need modification.
 */

/*
 * Originally written as "ikNTSC.c" by Alan Wm Paeth,
 * University of Waterloo, August, 1985
 * Updated by Dave Martindale, ImaX Systems Corp., December 1990
 */

/*
 * Compile-time options.
 *
 * Define either NTSC or PAL as 1 to select the colour system.
 * Define the other one as zero, or leave it undefined.
 *
 * Define FLAG_HOT as 1 if you want "hot" pixels set to black
 * to identify them. Otherwise they will be made safe.
 *
 * Define REDUCE_SAT as 1 if you want hot pixels to be repaired by
 * reducing their saturation. By default, luminance is reduced.
 *
 * CHROMA_LIM is the limit (in IRE units) of the overall
 * chrominance amplitude; it should be 50 or perhaps
 * very slightly higher.
 *
 * COMPOS_LIM is the maximum amplitude (in IRE units) allowed for
 * the composite signal. A value of 100 is the maximum
 * monochrome white, and is always safe. 120 is the absolute
 * limit for NTSC broadcasting, since the transmitter's carrier
 * goes to zero with 120 IRE input signal. Generally, 110
 * is a good compromise - it allows somewhat brighter colours
 * than 100, while staying safely away from the hard limit.
 */

#define NTSC      1
#define PAL       0
#define FLAG_HOT  0
#define REDUCE_SAT 0

#define CHROMA_LIM    50.0          /* chroma amplitude limit */
```



```
#define COMPOS_LIM      110.0          /* max IRE amplitude */

#if NTSC
/*
 * RGB to YIQ encoding matrix.
 */
double code_matrix[3][3] = {
    0.2989,      0.5866,      0.1144,
    0.5959,     -0.2741,     -0.3218,
    0.2113,     -0.5227,      0.3113,
};

#define PEDESTAL        7.5          /* 7.5 IRE black pedestal */
#define GAMMA           2.2
#endif /* NTSC */

#if PAL
/*
 * RGB to YUV encoding matrix.
 */
double code_matrix[3][3] = {
    0.2989,      0.5866,      0.1144,
   -0.1473,     -0.2891,      0.4364,
    0.6149,     -0.5145,     -0.1004,
};

#define PEDESTAL        0.0          /* no pedestal in PAL */
#define GAMMA           2.8
#endif /* PAL */

#define SCALE      8192          /* scale factor: do floats with int math */
#define MAXPIX     255          /* white value */
#define WID        1024         /* FB dimensions */
#define HGT        768

typedef struct {
    unsigned char    r, g, b;
} Pixel;

int      tab[3][3][MAXPIX+1];    /* multiply lookup table */
double   chroma_lim;             /* chroma limit */
double   compos_lim;            /* composite amplitude limit */
long     ichroma_lim2;           /* chroma limit squared (scaled integer) */
int       icompos_lim;           /* composite amplitude limit (scaled integer) */

double   pix_decode(), gc(), inv_gc();
int       pix_encode(), hot();

main()
{
    Pixel    p;
    int      row, col;

    build_tab();

    for (col=0; col<WID; col++) {
        for(row=0; row<HGT; row++) {
            read_pixel(row, col, &p);
            if (hot(&p)) {
                write_pixel(row, col, &p);
            }
        }
    }
}
```

```
    }
}

/*
 * build_tab: Build multiply lookup table.
 *
 * For each possible pixel value, decode value into floating-point
 * intensity. Then do gamma correction required by the video
 * standard. Scale the result by our fixed-point scale factor.
 * Then calculate 9 lookup table entries for this pixel value.
 *
 * We also calculate floating-point and scaled integer versions
 * of our limits here. This prevents evaluating expressions every pixel
 * when the compiler is too stupid to evaluate constant-valued
 * floating-point expressions at compile time.
 *
 * For convenience, the limits are #defined using IRE units.
 * We must convert them here into the units in which YIQ
 * are measured. The conversion from IRE to internal units
 * depends on the pedestal level in use, since as Y goes from
 * 0 to 1, the signal goes from the pedestal level to 100 IRE.
 * Chroma is always scaled to remain consistent with Y.
 */

build_tab()
{
    register double f;
    register int    pv;

    for (pv = 0; pv <= MAXPIX; pv++) {
        f = SCALE * gc(pix_decode(pv));
        tab[0][0][pv] = (int)(f * code_matrix[0][0] + 0.5);
        tab[0][1][pv] = (int)(f * code_matrix[0][1] + 0.5);
        tab[0][2][pv] = (int)(f * code_matrix[0][2] + 0.5);
        tab[1][0][pv] = (int)(f * code_matrix[1][0] + 0.5);
        tab[1][1][pv] = (int)(f * code_matrix[1][1] + 0.5);
        tab[1][2][pv] = (int)(f * code_matrix[1][2] + 0.5);
        tab[2][0][pv] = (int)(f * code_matrix[2][0] + 0.5);
        tab[2][1][pv] = (int)(f * code_matrix[2][1] + 0.5);
        tab[2][2][pv] = (int)(f * code_matrix[2][2] + 0.5);
    }

    chroma_lim = (double)CHROMA_LIM / (100.0 - PEDESTAL);
    compos_lim = ((double)COMPOS_LIM - PEDESTAL) / (100.0 - PEDESTAL);

    ichroma_lim2 = (int)(chroma_lim * SCALE + 0.5);
    ichroma_lim2 *= ichroma_lim2;
    icompos_lim = (int)(compos_lim * SCALE + 0.5);
}

int
hot(p)
Pixel    *p;
{
    register int    r, g, b;
    register int    y, i, q;
    register long   y2, c2;
    double          pr, pg, pb;
#ifdef REDUCE_SAT
```

```
double          py;
#endif
register double fy, fc, t, scale;
#if !FLAG_HOT
static int      prev_r = 0, prev_g = 0, prev_b = 0;
static int      new_r, new_g, new_b;
#endif
extern double   pow(), hypot();

r = p->r;
g = p->g;
b = p->b;

/*
 * Pixel decoding, gamma correction, and matrix multiplication
 * all done by lookup table.
 *
 * "i" and "q" are the two chrominance components;
 * they are I and Q for NTSC.
 * For PAL, "i" is U (scaled B-Y) and "q" is V (scaled R-Y).
 * Since we only care about the length of the chroma vector,
 * not its angle, we don't care which is which.
 */
y = tab[0][0][r] + tab[0][1][g] + tab[0][2][b];
i = tab[1][0][r] + tab[1][1][g] + tab[1][2][b];
q = tab[2][0][r] + tab[2][1][g] + tab[2][2][b];

/*
 * Check to see if the chrominance vector is too long or the
 * composite waveform amplitude is too large.
 *
 * Chrominance is too large if
 *
 *      sqrt(i^2, q^2)  >  chroma_lim.
 *
 * The composite signal amplitude is too large if
 *
 *      y + sqrt(i^2, q^2)  >  compos_lim.
 *
 * We avoid doing the sqrt by checking
 *
 *      i^2 + q^2  >  chroma_lim^2
 * and
 *      y + sqrt(i^2 + q^2)  >  compos_lim
 *      sqrt(i^2 + q^2)  >  compos_lim - y
 *      i^2 + q^2  >  (compos_lim - y)^2
 */

c2 = (long)i * i + (long)q * q;
y2 = (long)compos_lim - y;
y2 *= y2;
if (c2 <= ichroma_lim2 && c2 <= y2)      /* no problems */
    return 0;

/*
 * Pixel is hot, choose desired (compilation time controlled) strategy
 */
#if FLAG_HOT
/*
 * Set the hot pixel to black to identify it.
 */

```

```
    */
    p->r = p->g = p->b = 0;
#else /* FLAG_HOT */
/*
 * Optimization: cache the last-computed hot pixel.
 */
if (r == prev_r && g == prev_g && b == prev_b) {
    p->r = new_r;
    p->g = new_g;
    p->b = new_b;
    return 1;
}
prev_r = r;
prev_g = g;
prev_b = b;

/*
 * Get Y and chroma amplitudes in floating point.
 *
 * If your C library doesn't have hypot(), just use
 * hypot(a,b) = sqrt(a*a, b*b);
 *
 * Then extract linear (un-gamma-corrected) floating-point
 * pixel RGB values.
 */
fy = (double)y / SCALE;
fc = hypot((double)i / SCALE, (double)q / SCALE);

pr = pix_decode(r);
pg = pix_decode(g);
pb = pix_decode(b);

/*
 * Reducing overall pixel intensity by scaling
 * R, G, and B reduces Y, I, and Q by the same factor.
 * This changes luminance but not saturation, since saturation
 * is determined by the chroma/luminance ratio.
 *
 * On the other hand, by linearly interpolating between the
 * original pixel value and a grey pixel with the same
 * luminance (R=G=B=Y), we change saturation without
 * affecting luminance.
 */

#if !REDUCE_SAT
/*
 * Calculate a scale factor that will bring the pixel
 * within both chroma and composite limits, if we scale
 * luminance and chroma simultaneously.
 *
 * The calculated chrominance reduction applies to the
 * gamma-corrected RGB values that are the input to
 * the RGB-to-YIQ operation. Multiplying the
 * original un-gamma-corrected pixel values by
 * the scaling factor raised to the "gamma" power
 * is equivalent, and avoids calling gc() and inv_gc()
 * three times each.
 */
scale = chroma_lim / fc;
t = compos_lim / (fy + fc);
if (t < scale)
```

```
        scale = t;
    scale = pow(scale, GAMMA);

    r = pix_encode(scale * pr);
    g = pix_encode(scale * pg);
    b = pix_encode(scale * pb);
#else /* REDUCE_SAT */
/*
 * Calculate a scale factor that will bring the pixel
 * within both chroma and composite limits, if we scale
 * chroma while leaving luminance unchanged.
 *
 * We have to interpolate gamma-corrected RGB values,
 * so we must convert from linear to gamma-corrected
 * before interpolation and then back to linear afterwards.
 */
    scale = chroma_lim / fc;
    t = (compos_lim - fy) / fc;
    if (t < scale)
        scale = t;

    pr = gc(pr);
    pg = gc(pg);
    pb = gc(pb);
    py = pr * code_matrix[0][0] + pg * code_matrix[0][1]
        + pb * code_matrix[0][2];
    r = pix_encode(inv_gc(py + scale * (pr - py)));
    g = pix_encode(inv_gc(py + scale * (pg - py)));
    b = pix_encode(inv_gc(py + scale * (pb - py)));
#endif /* REDUCE_SAT */

    p->r = new_r = r;
    p->g = new_g = g;
    p->b = new_b = b;
#endif /* FLAG_HOT */
    return 1;
}

/*
 * gc: apply the gamma correction specified for this video standard.
 * inv_gc: inverse function of gc.
 *
 * These are generally just a call to pow(), but be careful!
 * Future standards may use more complex functions.
 * (e.g. SMPTE 240M's "electro-optic transfer characteristic").
 */

double
gc(x)
double x;
{
    extern double pow();

    return pow(x, 1.0 / GAMMA);
}

double
inv_gc(x)
double x;
{
    extern double pow();
```

```
        return pow(x, GAMMA);
    }

/*
 * pix_decode: decode an integer pixel value into a floating-point
 *             intensity in the range [0, 1].
 *
 * pix_encode: encode a floating-point intensity into an integer
 *             pixel value.
 *
 * The code given here assumes simple linear encoding; you must change
 * these routines if you use a different pixel encoding technique.
 */

double
pix_decode(v)
int     v;
{
    return (double)v / MAXPIX;
}

int
pix_encode(v)
double  v;
{
    return (int)(v * MAXPIX + 0.5);
}
```

```
/*
 * C code from the article
 * "Fast Convolution with Packed Lookup Tables"
 * by George Wolberg and Henry Massalin,
 * (wolcc@cunyvm.cuny.edu and qua@microunity.com)
 * in "Graphics Gems IV", Academic Press, 1994
 *
 *      Compile: cc convolve.c -o convolve
 *      Execute: convolve in.bw kernel out.bw
 */

#include <stdio.h>
#include <stdlib.h>

typedef unsigned char    uchar;

typedef struct {          /* image data structure */
    int width;            /* image width  (# cols) */
    int height;           /* image height (# rows) */
    uchar *image;         /* pointer to image data */
} imageS, *imageP;

typedef struct {          /* packed lut structure */
    int lut0[256];        /* stage 0 for 5-pt kernel */
    int lut1[256];        /* stage 1 for 11-pt kernel */
    int lut2[256];        /* stage 2 for 17-pt kernel */
    int bias;              /* accumulated stage biases */
    int stages;            /* # of stages used: 1,2,3 */
} lutS, *lutP;

/* definitions */
#define MASK            0x3FF
#define ROUNDD          1
#define PACK(A,B,C)     (((A)<<20) + ((B)<<10) + (C))
#define INT(A)           ((int) ((A)*262144+32768) >> 16)
#define CLAMP(A,L,H)     ((A)<=(L) ? (L) : (A)<=(H) ? (A) : (H))
#define ABS(A)           ((A) >= 0 ? (A) : -(A))

/* declarations for convolution functions */
void    convolve();
static void    initPackedLuts();
static void    fastconv();

/* declarations for image utility functions */
imageP    allocImage();
imageP    readImage();
int        saveImage();
void        freeImage();

/* ~~~~~~
 * main:
 *
 * Main function to collect input image and kernel values.
 * Pass them to convolve() and save result in output file.
 */
main(argc, argv)
int    argc;
char    **argv;
{
    int    n;
    imageP    I1, I2;
```

```
float    kernel[9];
char     buf[80];
FILE     *fp;

/* make sure the user invokes this program properly */
if(argc != 4) {
    fprintf(stderr, "Usage: convolve in.bw kernel out.bw\n");
    exit(1);
}

/* read input image */
if((I1=readImage(argv[1])) == NULL) {
    fprintf(stderr, "Can't read input file %s\n", argv[1]);
    exit(1);
}

/* read kernel: n lines in file specify a (2n-1)-point kernel
 * Don't exceed 9 kernel values (17-point symmetric kernel is limit)
 */
if((fp=fopen(argv[2], "r")) == NULL) {
    fprintf(stderr, "Can't read kernel file %s\n", argv[2]);
    exit(1);
}
for(n=0; n<9 && fgets(buf, 80, fp); n++) kernel[n] = atof(buf);

/* convolve input I1 with fast convolver */
I2 = allocImage(I1->width, I1->height);
convolve(I1, kernel, n, I2);

/* save output to a file */
if(saveImage(I2, argv[3]) == NULL) {
    fprintf(stderr, "Can't save output file %s\n", argv[3]);
    exit(1);
}
}
```

```
/* ~~~~~
 * convolve:
 *
 * Convolve input image I1 with kernel, a (2n-1)-point symmetric filter
 * kernel containing n entries: h[i] = kernel[ |i| ] for -n < i < n.
 * Output is stored in I2.
 */
void
convolve(I1, kernel, n, I2)
imageP    I1, I2;
float     *kernel;
int       n;
{
    int     x, y, w, h;
    uchar   *src, *dst;
    imageP   II;
    lutS     luts;

    w = I1->width;                /* image width          */
    h = I1->height;               /* image height         */

    II = allocImage(w, h);        /* reserve tmp image    */
    initPackedLuts(kernel, n, &luts); /* init packed luts     */
}
```



```
    for(y=0; y<h; y++) {                                /* process all rows      */
        src = I1->image + y*w;                          /* ptr to input  row    */
        dst = I2->image + y*w;                          /* ptr to output row    */
        fastconv(src, w, 1, &luts, dst); /* w pixels; stride=1 */
    }

    for(x=0; x<w; x++) {                                /* process all columns  */
        src = I1->image + x;                            /* ptr to input  column */
        dst = I2->image + x;                            /* ptr to output column */
        fastconv(src, h, w, &luts, dst); /* h pixels; stride=w  */
    }

    freeImage(I2);                                       /* free temporary image */
}

/* ~~~~~
 * initPackedLuts:
 *
 * Initialize scaled and packed lookup tables in lut.
 * Permit up to 3 cascaded stages for the following kernel sizes:
 *     stage 0:  5-point kernel
 *     stage 1: 11-point kernel
 *     stage 2: 17-point kernel
 * lut->lut0 <= packed entries (i*k2, i*k1, .5*i*k0), for i in [0, 255]
 * lut->lut1 <= packed entries (i*k5, i*k4,   i*k3), for i in [0, 255]
 * lut->lut2 <= packed entries (i*k8, i*k7,   i*k6), for i in [0, 255]
 * where k0,...k8 are taken in sequence from kernel[].
 *
 * Note that in lut0, k0 is halved since it corresponds to the center
 * pixel's kernel value and it appears in both fwd0 and rev0 (see gem).
 */
static void
initPackedLuts(kernel, n, luts)
float  *kernel;
int     n;
lutP    luts;
{
    int     i, k, s, *lut;
    int     b1, b2, b3;
    float   k1, k2, k3;
    float   sum;

    /* enforce flat-field response constraint: sum of kernel values = 1 */
    sum = kernel[0];
    for(i=1; i<n; i++) sum += 2*kernel[i]; /* account for symmetry */
    if(ABS(sum - 1) > .001)
        fprintf(stderr, "Warning: filter sum != 1 (=%f)\n", sum);

    /* init bias added to fields to avoid negative numbers (underflow) */
    luts->bias = 0;

    /* set up lut stages, 3 kernel values at a time */
    for(k=s=0; k<n; s++) {                                /* init lut (stage s) */
        k1 = (k < n) ? kernel[k++] : 0;
        k2 = (k < n) ? kernel[k++] : 0;
        k3 = (k < n) ? kernel[k++] : 0;
        if(k <= 3) k1 *= .5;                             /* kernel[0]: halve k0 */
    }
}
```

```
/* select proper array in lut structure based on stage s */
switch(s) {
case 0: lut = luts->lut0;      break;
case 1: lut = luts->lut1;      break;
case 2: lut = luts->lut2;      break;
}

/* check k1,k2,k3 to avoid overflow in 10-bit fields */
if(ABS(k1) + ABS(k2) + ABS(k3) > 1) {
    fprintf(stderr, " |%f|+|%f|+|%f| > 1\n", k1, k2, k3);
    exit(1);
}

/* compute bias for each field to avoid underflow */
b1 = b2 = b3 = 0;
if(k1 < 0) b1 = -k1 * 1024;
if(k2 < 0) b2 = -k2 * 1024;
if(k3 < 0) b3 = -k3 * 1024;

/* luts->bias will be subtracted in convolve() after adding
 * stages; multiply by 2 because of combined effect of fwd
 * and rev terms
 */
luts->bias += 2*(b1 + b2 + b3);

/* scale and pack kernel values in lut */
for(i=0; i<256; i++) {
    /*
     * INT(A) forms fixed point field:
     * (A*(1<<18)+(1<<15)) >> 16
     */
    lut[i] = PACK(    INT(i*k3) + b3,
                    INT(i*k2) + b2 + ROUND,
                    INT(i*k1) + b1 );
}
luts->stages = s;
}
```

```
/* ~~~~~
 * fastconv:
 *
 * Fast 1D convolver.
 * Convolve len input samples in src with a symmetric kernel packed in luts,
 * a lookup table that is created by initPackedLuts() from kernel values.
 * The output goes into dst.
 */
```

```
static void
fastconv(src, len, offst, luts, dst)
int      len, offst;
uchar    *src, *dst;
lutP     luts;
{
    int      x, padlen, val, bias;
    int      fwd0, fwd1, fwd2;
    int      rev0, rev1, rev2;
    int      *lut0, *lut1, *lut2;
    uchar    *p1, *p2, *ip, *op;
    uchar    buf[1024];
```

```
/* copy and pad src into buf with padlen elements on each end */
padlen = 3*(luts->stages) - 1;
p1 = src; /* pointer to row (or column) of input */
p2 = buf; /* pointer to row of padded buffer */
for(x=0; x<padlen; x++) /* pad left side: replicate first pixel */
    *p2++ = *p1;
for(x=0; x<len; x++) { /* copy input row (or column) */
    *p2++ = *p1;
    p1 += offst;
}
p1 -= offst; /* point to last valid input pixel */
for(x=0; x<padlen; x++) /* pad right side: replicate last pixel */
    *p2++ = *p1;

/* initialize input and output pointers, ip and op, respectively */
ip = buf;
op = dst;

/* bias was added to lut entries to deal with negative kernel values */
bias = luts->bias;

switch(luts->stages) {
case 1: /* 5-pt kernel */
    lut0 = luts->lut0;

    ip += 2; /* ip[0] is center pixel */
    fwd0 = (lut0[ip[-2]] >> 10) + lut0[ip[-1]];
    rev0 = (lut0[ip[0]] << 10) + lut0[ip[1]];

    while(len--) {
        fwd0 = (fwd0 >> 10) + lut0[ip[0]];
        rev0 = (rev0 << 10) + lut0[ip[2]];
        val = ((fwd0 & MASK) + ((rev0 >> 20) & MASK) - bias)
              >> 2;
        *op = CLAMP(val, 0, 255);

        ip++;
        op += offst;
    }
    break;
case 2: /* 11-pt kernel */
    lut0 = luts->lut0;
    lut1 = luts->lut1;

    ip += 5; /* ip[0] is center pixel */
    fwd0 = (lut0[ip[-2]] >> 10) + lut0[ip[-1]];
    rev0 = (lut0[ip[0]] << 10) + lut0[ip[1]];

    fwd1 = (lut1[ip[-5]] >> 10) + lut1[ip[-4]];
    rev1 = (lut1[ip[3]] << 10) + lut1[ip[4]];

    while(len--) {
        fwd0 = (fwd0 >> 10) + lut0[ip[0]];
        rev0 = (rev0 << 10) + lut0[ip[2]];

        fwd1 = (fwd1 >> 10) + lut1[ip[-3]];
        rev1 = (rev1 << 10) + lut1[ip[5]];

        val = ((fwd0 & MASK) + ((rev0 >> 20) & MASK)
              + (fwd1 & MASK) + ((rev1 >> 20) & MASK) - bias)
    }
```

```

        >> 2;
        *op = CLAMP(val, 0, 255);

        ip++;
        op += offst;
    }
    break;
case 3:      /* 17-pt kernel */
    lut0 = luts->lut0;
    lut1 = luts->lut1;
    lut2 = luts->lut2;

    ip += 8;      /* ip[0] is center pixel */
    fwd0 = (lut0[ip[-2]] >> 10) + lut0[ip[-1]];
    rev0 = (lut0[ip[ 0]] << 10) + lut0[ip[ 1]];

    fwd1 = (lut1[ip[-5]] >> 10) + lut1[ip[-4]];
    rev1 = (lut1[ip[ 3]] << 10) + lut1[ip[ 4]];

    fwd2 = (lut2[ip[-8]] >> 10) + lut2[ip[-7]];
    rev2 = (lut2[ip[ 6]] << 10) + lut2[ip[ 7]];

    while(len--) {
        fwd0 = (fwd0 >> 10) + lut0[ip[0]];
        rev0 = (rev0 << 10) + lut0[ip[2]];

        fwd1 = (fwd1 >> 10) + lut1[ip[-3]];
        rev1 = (rev1 << 10) + lut1[ip[ 5]];

        fwd2 = (fwd2 >> 10) + lut2[ip[-6]];
        rev2 = (rev2 << 10) + lut2[ip[ 8]];

        val = ((fwd0 & MASK) + ((rev0 >> 20) & MASK)
              + (fwd1 & MASK) + ((rev1 >> 20) & MASK)
              + (fwd2 & MASK) + ((rev2 >> 20) & MASK) - bias)
              >> 2;
        *op = CLAMP(val, 0, 255);

        ip++;
        op += offst;
    }
    break;
}
}
```

```
/* ~~~~~
 * readImage:
 *
 * Read an image from file.
 * Format: two integers to specify width and height, followed by uchar data.
 * Return image structure pointer.
 */
imageP
readImage(file)
char *file;
{
    int sz[2];
    FILE *fp;
    imageP I = NULL;
```

```
/* open file for reading */
if((fp = fopen(file, "r")) != NULL) { /* open file for read */
    fread(sz, sizeof(int), 2, fp); /* read image dimensions*/
    I = allocImage( sz[0],sz[1]); /* init image structure */
    fread(I->image, sz[0],sz[1],fp); /* read data into I */
    fclose(fp); /* close image file */
}
return(I); /* return image pointer */
}
```

```
/* ~~~~~
* saveImage:
*
* Save image I into file.
* Return NULL for failure, 1 for success.
*/
int
saveImage(I, file)
imageP I;
char *file;
{
    int sz[2], status = NULL;
    FILE *fp;

    if((fp = fopen(file, "w")) != NULL) { /* open file for save */
        sz[0] = I->width;
        sz[1] = I->height;
        fwrite(sz, sizeof(int), 2, fp); /* write dimensions */
        fwrite(I->image,sz[0],sz[1],fp); /* write image data */
        fclose(fp); /* close image file */
        status = 1; /* register success */
    }
    return(status);
}
```







```
/* ~~~~~
* allocImage:
*
* Allocate space for an uchar image of width w and height h.
* Return image structure pointer.
*/
imageP
allocImage(w, h)
int w, h;
{
    imageP I;

    /* allocate memory for image data structure */
    if((I = (imageP) malloc(sizeof(imageS))) != NULL) {
        I->width = w; /* init width */
        I->height = h; /* init height */
        I->image =(uchar*) malloc(w*h); /* init data pointer */
    }
    return(I); /* return image pointer */
}
```

```
/* ~~~~~  
 * freeImage:  
 *  
 * Free image memory.  
 */  
void  
freeImage(I)  
imageP I;  
{  
    free((char *) I->image);  
    free((char *) I);  
}
```








# Index of

## /pubs/tog/GraphicsGems/gemsv/ch7-7/

Name	Last modified	Size	Description
 <a href="#">_Parent Directory</a>			
 <a href="#">GG4D/</a>	29-Jun-00 08:25	1K	
 <a href="#">README</a>	29-Jun-00 08:25	1K	
 <a href="#">libgm/</a>	29-Jun-00 08:25	1K	
 <a href="#">mactbox/</a>	29-Jun-00 08:25	1K	
 <a href="#">vec-h/</a>	29-Jun-00 08:25	1K	

# Index of

## /pubs/tog/GraphicsGems/gemsv/ch7-7/GG4D/
















Name	Last modified	Size	Description
 <a href="#">_Parent Directory</a>			
 <a href="#">_GGems.h</a>	29-Jun-00 08:25	4K	
 <a href="#">_GGems4d.h</a>	29-Jun-00 08:25	1K	
 <a href="#">_Makefile</a>	29-Jun-00 08:25	1K	
 <a href="#">_VecLib.c</a>	29-Jun-00 08:25	10K	
 <a href="#">_VecLib4d.c</a>	29-Jun-00 08:25	18K	
 <a href="#">_example.c</a>	29-Jun-00 08:25	2K	



Dir	Name (dim)	Author(s)	Desc (Gems vol.V, page 403)
-----			
libgm	3+	Scheepers & May	"The compleat C++ graphics library"
mactbox	3+	C. Schlick	"Elegant macros are gentle on the eyes"
vec-h	N+	D. Hatch	"C/C++ macros to go in any dimension"
GG4D	4+	S. Hill	"The 4D tour de force (includes projections)"












# Index of

## /pubs/tog/GraphicsGems/gemsv/ch7-7/libgm/

Name	Last modified	Size	Description
 <a href="#">_ Parent Directory</a>			
 <a href="#">_Imakefile</a>	29-Jun-00 08:25	1K	
 <a href="#">_bool.h</a>	29-Jun-00 08:25	1K	
 <a href="#">_gm.h</a>	29-Jun-00 08:25	1K	
 <a href="#">_gmConst.h</a>	29-Jun-00 08:25	1K	
 <a href="#">_gmMat3.cc</a>	29-Jun-00 08:25	7K	
 <a href="#">_gmMat3.h</a>	29-Jun-00 08:25	2K	
 <a href="#">_gmMat4.cc</a>	29-Jun-00 08:25	14K	
 <a href="#">_gmMat4.h</a>	29-Jun-00 08:25	3K	
 <a href="#">_gmUtils.h</a>	29-Jun-00 08:25	2K	
 <a href="#">_gmVec2.h</a>	29-Jun-00 08:25	4K	
 <a href="#">_gmVec3.h</a>	29-Jun-00 08:25	5K	
 <a href="#">_gmVec4.h</a>	29-Jun-00 08:25	5K	
 <a href="#">_libgm.ps</a>	29-Jun-00 08:25	48K	
 <a href="#">_libgm.tex</a>	29-Jun-00 08:25	14K	





# Index of

## /pubs/tog/GraphicsGems/gemsv/ch7-7/mactbox/

Name	Last modified	Size	Description
 <a href="#">_ Parent Directory</a>			
 <a href="#">_mat2.h</a>	29-Jun-00 08:25	3K	
 <a href="#">_mat3.h</a>	29-Jun-00 08:25	5K	
 <a href="#">_mat4.h</a>	29-Jun-00 08:25	4K	
 <a href="#">_real.h</a>	29-Jun-00 08:25	3K	
 <a href="#">_sint.h</a>	29-Jun-00 08:25	3K	
 <a href="#">_tool.h</a>	29-Jun-00 08:25	11K	
 <a href="#">_uint.h</a>	29-Jun-00 08:25	3K	
 <a href="#">_vec2.h</a>	29-Jun-00 08:25	4K	
 <a href="#">_vec3.h</a>	29-Jun-00 08:25	6K	
 <a href="#">_vec4.h</a>	29-Jun-00 08:25	6K	

# Index of

## /pubs/tog/GraphicsGems/gemsv/ch7-7/vec-h/

Name	Last modified	Size	Description
 <a href="#">_Parent Directory</a>			
 <a href="#">_README</a>	29-Jun-00 08:25	1K	
 <a href="#">_vec.h</a>	29-Jun-00 08:25	30K	
 <a href="#">_vec_h.c</a>	29-Jun-00 08:25	16K	

```
/*
Joining Two Lines with a Circular Arc Fillet
Robert D. Miller
*/
```

```
#include "math.h"
#include "stdio.h"
#include "stdlib.h"

#define pi      3.14159265358974
#define halfpi  1.5707963267949
#define threepi2 4.71238898038469
#define twopi   6.28318530717959
#define degree  57.29577951308232088
#define radian   0.01745329251994329577

typedef struct point { float x; float y; } point;

point    gv1, gv2;
float    xx, yy, sa, sb,
         qx, qy, rx, ry;
int      xl, yl, x2, y2;

float    arccos();
float    arcsin();
float    cross2();
float    dot2();

void      moveto();    /* External draw routine */
void      lineto();    /* External draw routine */
void      drawarc();
float     linetopoint();
void      pointperp();
void      fillet();

float cross2(v1,v2)
    point v1, v2;
{ return (v1.x*v2.y - v2.x*v1.y); }

float dot2(v1,v2)    /* Return angle subtended by two vectors. */
    point v1, v2;
{ float    d, t;

    d= (float) sqrt(((v1.x*v1.x)+(v1.y*v1.y)) * ((v2.x*v2.x)+(v2.y*v2.y)));
    if (d != (float) 0.0)
    {
        t= (v1.x*v2.x+v1.y*v2.y)/d;
        return (arccos(t));
    }
}
```

```
    }
    else
        return ((float) 0.0);
}

/* Draw circular arc in one degree increments. Center is (xc,yc)
   with radius r, beginning at starting angle, startang
   through angle ang. If ang < 0 arc is draw clockwise. */

void drawarc(xc, yc, r, startang, ang)
float xc, yc, r, startang, ang;
{
#define   sindt 0.017452406
#define   cosdt 0.999847695

float    a, x, y, sr;
int      k;

    a= (float) startang*radian;
    x= (float) r*cos(a);
    y= (float) r*sin(a);
    moveto(xc+x, yc+y, 3);
    if (ang >= (float) 0.0)
        sr= (float) sindt;
    else
        sr= (float) -sindt;
    for (k= 1; k <= (int) floor(fabs(ang)); k++)
        {
            x= x*cosdt-y*sr;
            y= x*sr+y*cosdt;
            lineto(xc+x,yc+y);
        }
return;
}

/*      Find a,b,c in Ax + By + C = 0   for line p1,p2.      */

void linecoef(a,b,c,p1,p2)
float    *a, *b, *c;
point    p1, p2;
{
    *c=  (p2.x*p1.y)-(p1.x*p2.y);
    *a=  p2.y-p1.y;
    *b=  p1.x-p2.x;
return;
}

/*      Return signed distance from line Ax + By + C = 0 to point P. */

float linetopoint(a,b,c,p)
float    a, b, c;
point    p;
{
float    d, lp;

    d= sqrt((a*a)+(b*b));
    if (d == 0.0)
        lp = 0.0;
}
```

```
        else
            lp= (a*p.x+b*p.y+c)/d;
return ((float) lp);
}

/*   Given line l = ax + by + c = 0 and point p,
    compute x,y so p(x,y) is perpendicular to l. */

void pointperp(x, y, a, b, c, p)
float      *x, *y, a, b, c;
point      p;

{
float      d, cp;

    *x=  0.0;
    *y=  0.0;
    d=  a*a +b*b;
    cp= a*p.y-b*p.x;
    if (d != 0.0)
        {
            *x= (-a*c-b*cp)/d;
            *y= (a*cp-b*c)/d;
        }
return;
}

/*   Compute a circular arc fillet between lines L1 (p1 to p2) and
    L2 (p3 to p4) with radius R.  The circle center is xc,yc.      */

void fillet(p1, p2, p3, p4, r, xc, yc, pa, aa)
point      *p1, *p2, *p3, *p4;
float      r, *xc, *yc, *pa, *aa;

{
float a1, b1, c1, a2, b2, c2, clp,
      c2p, d1, d2, xa, xb, ya, yb, d, rr;
point mp, pc;

    linecoef(&a1,&b1,&c1,*p1,*p2);
    linecoef(&a2,&b2,&c2,*p3,*p4);

    if ((a1*b2) == (a2*b1)) /* Parallel or coincident lines */
        goto xit;

    mp.x= ((*p3).x + (*p4).x)/2.0;
    mp.y= ((*p3).y + (*p4).y)/2.0;
    d1= linetopoint(a1,b1,c1,mp); /* Find distance p1p2 to p3 */
    if (d1 == 0.0) goto xit;

    mp.x= ((*p1).x + (*p2).x)/2.0;
    mp.y= ((*p1).y + (*p2).y)/2.0;
    d2= linetopoint(a2,b2,c2,mp); /* Find distance p3p4 to p2 */
    if (d2 == 0.0) goto xit;

    rr= r;
    if (d1 <= 0.0) rr= -rr;

    clp= c1-rr*sqrt((a1*a1)+(b1*b1)); /* Line parallel l1 at d */
    rr= r;
```

```
if (d2 <= 0.0) rr= -rr;

c2p= c2-rr*sqrt((a2*a2)+(b2*b2)); /* Line parallel l2 at d */
d= a1*b2-a2*b1;

*xc= (c2p*b1-clp*b2)/d;          /* Intersect constructed lines */
*yc= (clp*a2-c2p*a1)/d;          /* to find center of arc */
pc.x=  *xc;
pc.y=  *yc;

pointperp(&xa,&ya,a1,b1,c1,pc); /* Clip or extend lines as required */
pointperp(&xb,&yb,a2,b2,c2,pc);
(*p2).x= xa; (*p2).y= ya;
(*p3).x= xb; (*p3).y= yb;
gv1.x= xa-*xc; gv1.y= ya-*yc;
gv2.x= xb-*xc; gv2.y= yb-*yc;

*pa= (float) atan2(gv1.y,gv1.x); /* Beginning angle for arc */
*aa= dot2(gv1,gv2);
if (cross2(gv1,gv2) < 0.0) *aa= -*aa; /* Angle subtended by arc */

xit:
return;
}
```



```
/*
 * ANSI C code from the article
 * "Computing the Area of a Spherical Polygon"
 * by Robert D. Miller
 * in "Graphics Gems IV", Academic Press, 1994
 */

/*
 * to test, compile with "cc -DMAIN -o spherical_poly spherical_poly.c -lm"
 * to create subroutine, compile with "cc -c spherical_poly.c"
 */

#include <math.h>
static const double
    HalfPi= 1.5707963267948966192313,
    Degree= 57.295779513082320876798;    /* degrees per radian */

double Hav(double X)
/*  Haversine function: hav(x)= (1-cos(x))/2.  */
{
    return (1.0 -cos(X))/2.0;
}

double SphericalPolyArea(double *Lat, double *Lon, int N)
/*  Returns the area of a spherical polygon in spherical degrees,
    given the latitudes and longitudes in Lat and Lon, respectively.
    The N data points have indices which range from 0 to N-1.  */
{
    int J, K;
    double
        Lam1, Lam2, Beta1, Beta2,
        CosB1, CosB2, HavA,
        T, A, B, C, S, Sum, Excess;

    Sum= 0;
    for (J= 0; J <= N; J++)
    {
        K= J+1;
        if (J == 0)
        {
            Lam1= Lon[J];        Beta1= Lat[J];
            Lam2= Lon[J+1];      Beta2= Lat[J+1];
            CosB1= cos(Beta1);    CosB2= cos(Beta2);
        }
        else
        {
            K= (J+1) % (N+1);
            Lam1= Lam2;          Beta1= Beta2;
            Lam2= Lon[K];        Beta2= Lat[K];
            CosB1= CosB2;        CosB2= cos(Beta2);
        }

        if (Lam1 != Lam2)
        {
            HavA= Hav(Beta2-Beta1) +CosB1*CosB2*Hav(Lam2-Lam1);
            A= 2*asin(sqrt(HavA));
            B= HalfPi -Beta2;
            C= HalfPi -Beta1;
            S= 0.5*(A+B+C);
            T= tan(S/2) * tan((S-A)/2) * tan((S-B)/2) * tan((S-C)/2);
        }
    }
}
```

```
        Excess= fabs(4*atan(sqrt(fabs(T))))*Degree;
        if (Lam2 < Lam1) Excess= -Excess;
```

```
        Sum= Sum + Excess;
```

```
    }
```

```
}
```

```
return fabs(Sum);
```

```
} /* SphericalPolyArea. */
```

```
#ifdef MAIN
```

```
#include <stdlib.h>
```

```
#include <stdio.h>
```

```
double SphericalPolyArea(double *Lat, double *Lon, int N);
```

```
double
```

```
    SqMi= 273218.4, /* Square mi per spherical degree. */
```

```
    SqKm= 707632.4; /* Square km per spherical degree. */
```

```
main() /* Spherical Polygon Area test routine */
```

```
{
```

```
double Lat[100], Lon[100];
```

```
double P, Q, Area;
```

```
int K, LastPt;
```

```
    LastPt= -1;
```

```
    printf("Enter Longitude Latitude. End with: 0 0.\n");
```

```
    while (1)
```

```
    {
```

```
        scanf("%lf %lf",&P,&Q);
```

```
        if ((P == 0.0) && (Q == 0.0))
```

```
            break;
```

```
        Lon[++LastPt]= P; Lat[LastPt]= Q;
```

```
    }
```

```
    for (K= 0; K <= LastPt; K++)
```

```
    {
```

```
        Lat[K]= Lat[K]/Degree; Lon[K]= Lon[K]/Degree;
```

```
    }
```

```
    Area= SphericalPolyArea(Lat, Lon, LastPt);
```

```
    printf("        Area=%12.4lf sq mi., %12.4lf spherical degrees.\n",
```

```
        Area*SqMi, Area);
```



```
    return 0;
```

```
}
```

```
#endif
```

# Index of

## /pubs/tog/GraphicsGems/gemsv/ch3-4/

Name	Last modified	Size	Description
 <a href="#">_Parent Directory</a>			
 <a href="#">_xcoord.c</a>	29-Jun-00 08:22	8K	

```
#include <stdio.h>
#include <stdlib.h>

#define SQR(a) ((a)*(a))

typedef double MATX[10][10];
typedef double VECT[10];
typedef struct {double x; double y;} Point2;

Point2 pt[1023];          /* From coordinates */
double zeta[1023], eta[1023]; /* To    coordinates */

int Gauss(MATX ain, VECT bin, int n, VECT v)
/* Gaussian elimination by converting to upper triangular system.
Row interchanges are done via re-indexing sub[]. See Vandergraft:
Intro. Numerical Methods, 2ed, Academic Press, 1983, Chapter 6. */
{
    MATX a;  VECT b;
    int i, j, k, last, index;
    double big, absv;
    int sub[21];

    for(k=0; k < n; k++) {          /* make local copies */
        for(j=0; j < n; j++) a[k][j] = ain[k][j];
        b[k] = bin[k];
    }

    last= n-1;
    for (k= 0; k <= last; k++) sub[k]= k;
    for (k= 0; k <= last-1; k++) {
        big= 0.0;
        for (i= k; i <= last; i++) {
            absv= abs(a[sub[i]][k]);
            if (absv > big)
                { big= absv; index= i; }
        }
        if (big == 0.0) return 0;
        j= sub[k];
        sub[k]= sub[index];
        sub[index]= j;
        big= 1.0/a[sub[k]][k];
        for (i= k+1; i <= last; i++) {
            a[sub[i]][k]= -a[sub[i]][k]*big;
            for (j= k+1; j <= last; j++)
                a[sub[i]][j] += a[sub[i]][k] * a[sub[k]][j];
            b[sub[i]] += a[sub[i]][k] * b[sub[k]];
        }
    }
    v[last]= b[sub[last]] / a[sub[last]][last];
    for (k= last-1; k >= 0; k--) {
        v[k]= b[sub[k]];
        for (i= k+1; i <= last; i++)
            v[k] = v[k] -a[sub[k]][i] * v[i];
        v[k] = v[k] /a[sub[k]][k];
    }
    return 1;
}

void PrintMatrix(MATX a, VECT v, int size)
{
    int r, c;
    for(r= 0; r < size; r++) {
        for(c= 0; c < size; c++) printf("%14.6lf  ",a[r][c]);
    }
}
```

```
        printf(" %14.6lf\n", v[r]);
    }
    printf("\n");
}

void PrintSolution(VECT v, int vectorsize, char which)
/* Print the solution vector */
{
    int k;
    printf("Solution vector %c\n", which);
    for(k = 0; k < vectorsize; k++)
        if (abs(v[k]) < 1.0E6) printf("%14.6f  ", v[k]);
        else printf("%14.6e  ", v[k]);
    printf("\n");
}

void FirstOrderExact(VECT xv, VECT yv)
{
    int k, ok;  VECT b;  MATX c;
    for(k= 0; k<=2; k++) b[k] = zeta[k];
    for(k= 0; k<=2; k++) {
        c[k][0] = pt[k].x;
        c[k][1] = pt[k].y;
        c[k][2] = 1.0;
    };

    printf("Augmented matrix:\n");
    PrintMatrix(c, b, 3);
    ok =Gauss(c, b, 3, xv);
    PrintSolution(xv, 3, 'X');

    for(k= 0; k<=2; k++) b[k] = eta[k];
    for(k= 0; k<=2; k++) {
        c[k][0] = pt[k].x;
        c[k][1] = pt[k].y;
        c[k][2] = 1.0;
    };

    PrintMatrix(c, b, 3);
    ok =Gauss(c, b, 3, yv);
    PrintSolution(yv, 3, 'Y');
}

void SecondOrderExact(VECT xv, VECT yv)
{
    int k, ok;  VECT b;  MATX c;
    for(k= 0; k<=5; k++) b[k] = zeta[k];
    for(k= 0; k<=5; k++) {
        c[k][0] = pt[k].x*pt[k].x;
        c[k][1] = pt[k].y*pt[k].y;
        c[k][2] = pt[k].x;
        c[k][3] = pt[k].y;
        c[k][4] = pt[k].x*pt[k].y;
        c[k][5] = 1;
    }

    printf("Augmented matrix:\n");
    PrintMatrix(c, b, 6);
    ok =Gauss(c, b, 6, xv);
    printf("x = a5*x^2 + a4*y^2 + a3*x + a2*y + a1*x*y + a0:\n");
    PrintSolution(xv, 6, 'X');

    for(k= 0; k<=5; k++) b[k] = eta[k];
    for(k= 0; k<=5; k++) {
```

```
    c[k][0] = SQR(pt[k].x);
    c[k][1] = SQR(pt[k].y);
    c[k][2] = pt[k].x;
    c[k][3] = pt[k].y;
    c[k][4] = pt[k].x*pt[k].y;
    c[k][5] = 1;
}
```

```
printf("Augmented matrix:\n");
PrintMatrix(c, b, 6);
ok = Gauss(c, b, 6, yv);
printf("y = b5*x^2 + b4*y^2 + b3*x + b2*y + b1*x*y + b0:\n");
PrintSolution(yv, 6, 'Y');
```

```
}
```

```
void FirstOrderLeastSquares(int npoints, VECT xv, VECT yv)
```

```
{    MATX c;    VECT b;
    double sumx= 0, sumxx= 0, sumy= 0, sumyy= 0, sumxy= 0,
           sumd= 0, sumdx= 0, sumdy = 0;
    int k, ok;
    double xt, yt;
    for(k=0; k < npoints; k++) {
        sumx  += pt[k].x;
        sumxx += SQR(pt[k].x);
        sumy  += pt[k].y;
        sumyy += SQR(pt[k].y);
        sumxy += pt[k].x*pt[k].y;
        sumd  += zeta[k];
        sumdx += pt[k].x*zeta[k];
        sumdy += pt[k].y*zeta[k];
    }

    c[0][0] = sumxx;    c[0][1] = sumxy;    c[0][2] = sumx;
    c[1][0] = sumxy;    c[1][1] = sumyy;    c[1][2] = sumy;
    c[2][0] = sumx;     c[2][1] = sumy;     c[2][2] = npoints;
```

```
    b[0] = sumdx;        b[1] = sumdy;        b[2] = sumd;
    ok = Gauss(c, b, 3, xv);
```

```
    sumd = sumdx = sumdy = 0;
```

```
    for(k=0; k < npoints; k++) {
        sumd += eta[k];
        sumdx+= pt[k].x*eta[k];
        sumdy+= pt[k].y*eta[k];
    };
```

```
    b[0] = sumdx;    b[1] = sumdy; b[2] = sumd;
    ok = Gauss(c, b, 3, yv);
    printf("residuals\n");
    for(k=0; k < npoints; k++) {
        xt = zeta[k] -(pt[k].x*xv[0] + pt[k].y*xv[1] + xv[2]);
        yt = eta[k] -(pt[k].x*yv[0] + pt[k].y*yv[1] + yv[2]);
        printf("%4d    %12.6    %12.6\n", xt, yt);
    }
}
```

```
void SecondOrderLeastSquares(MATX c, int npoints, VECT xv, VECT yv)
```

```
{    int j, k, ok;
    VECT b;
    double sumd=0, sumdx=0, sumdx2=0, sumdy=0,
```

```
        sumdy2=0, sumdxy =0;
double px2, py2, xt, yt;

for(j=0; j<= 5; j++)
    for(k=0; k<= 5; k++) c[j][k] = 0;

for(k =0; k < npoints; k++) {
    px2 = SQR(pt[k].x);
    py2 = SQR(pt[k].y);
    c[0][0] += px2 *px2;      /* coefficients for normal equations */
    c[0][1] += px2 *py2;
    c[0][2] += px2 *pt[k].x;
    c[0][3] += px2 *pt[k].y;
    c[0][4] += px2 *pt[k].x *pt[k].y;
    c[0][5] += px2;
    c[1][1] += py2 *py2;
    c[1][2] += pt[k].x *py2;
    c[1][3] += pt[k].y *py2;
    c[1][4] += pt[k].x *py2 *pt[k].y;
    c[1][5] += py2;
    c[2][2] += px2;
    c[2][3] += pt[k].x *pt[k].y;
    c[2][4] += px2 *pt[k].y;
    c[2][5] += pt[k].x;
    c[3][3] += py2;
    c[3][4] += pt[k].x *py2;
    c[3][5] += pt[k].y;
    c[4][4] += px2 *py2;
    c[4][5] += pt[k].x *pt[k].y;

    sumd    += zeta[k];
    sumdx    += pt[k].x *zeta[k];
    sumdx2   += px2 *zeta[k];
    sumdy    += pt[k].y *zeta[k];
    sumdy2   += py2 *zeta[k];
    sumdxy   += pt[k].x *pt[k].y *zeta[k];
}

c[1][0] =c[0][1]; /* Coefficient matrix is symmetric about diagonal */
c[2][0] =c[0][2]; c[2][1] =c[1][2];
c[3][0] =c[0][3]; c[3][1] =c[1][3]; c[3][2] =c[2][3];
c[4][0] =c[0][4]; c[4][1] =c[1][4]; c[4][2] =c[2][4];
                                     c[4][3] =c[3][4];
c[5][0] =c[0][5]; c[5][1] =c[1][5]; c[5][2] =c[2][5];
                                     c[5][3] =c[3][5];
c[5][4] =c[4][5]; c[5][5] =npoints;

b[0] =sumdx2;    b[1] =sumdy2;    b[2] =sumdx; /* new vector */
b[3] =sumdy;     b[4] =sumdxy;    b[5] =sumd;

printf("Augmented matrix:\n");
PrintMatrix(c, b, 6);
ok =Gauss(c, b, 6, xv);
printf("x = a5*x^2 + a4*y^2 + a3*x + a2*y + a1*x*y + a0:\n");
PrintSolution(xv, 6, 'X');

sumd = sumdx = sumdx2 = sumdy = sumdy2 = sumdxy =0;

for(k =0; k < npoints; k++) {
    sumd    += eta[k];
    sumdx    += pt[k].x *eta[k];
```

```
    sumdx2 += px2 *eta[k];
    sumdy  += pt[k].y *eta[k];
    sumdy2 += py2 *zeta[k];
    sumdxy += pt[k].x *pt[k].y *eta[k];
}
```

```
/* Coefficient matrix must remain unchanged. */
```

```
b[0] =sumdx2;    b[1] =sumdy2;    b[2] =sumdx;    /* New vector */
b[3] =sumdy;     b[4] =sumdxy;    b[5] =sumd;
```

```
ok =Gauss(c, b, 6, yv);
printf("y = b5*x^2 + b4*y^2 + b3*x + b2*y + b1*x*y + b0:\n");
PrintSolution(yv, 6, 'Y');
```



```
printf("Residuals:\n");
for(k =0; k < npoints; k++) {
    xt = SQR(pt[k].x)*xv[0] + SQR(pt[k].y)*xv[1] +
        pt[k].x *xv[2] + pt[k].y*xv[3] +
        pt[k].x *pt[k].y*xv[4] +xv[5];
    xt = zeta[k] -xt;
    yt = SQR(pt[k].x)*yv[0] + SQR(pt[k].y)*yv[1] +
        pt[k].x*yv[2] + pt[k].y*yv[3] +
        pt[k].x *pt[k].y*yv[4] +yv[5];
    yt = eta[k] -yt;
    printf("%4d  %12.6f  %12.6f\n", (k+1), xt, yt);
}
```

```
}
```



# Index of

## /pubs/tog/GraphicsGems/gemsv/ch4-8/

Name	Last modified	Size	Description
 <a href="#">_Parent Directory</a>			
 <a href="#">_qbezier.c</a>	29-Jun-00 08:23	2K	

```
/* Quick and Simple Bezier Curve Drawing --- Robert D. Miller
 * This 2-D planar Bezier curve drawing software is 3-D compliant ---
 * redefine Point and change the commented lines as indicated.
 */

#include <stdio.h>
#define MaxCtlPoints 12

typedef struct {float x; float y;} Point;          /* for 2-D curves */
                                                    /* for 3-D space curves */
/* typedef struct {float x; float y; float z;} Point; */
typedef Point PtArray[99];
typedef Point BezArray[MaxCtlPoints];

void BezierForm(int NumCtlPoints, PtArray p, BezArray c)
/* Setup Bezier coefficient array once for each control polygon. */
{
    int k; long n, choose;
    n = NumCtlPoints - 1;
    for(k = 0; k <= n; k++) {
        if (k == 0) choose = 1;
        else if (k == 1) choose = n;
        else choose = choose * (n - k + 1) / k;
        c[k].x = p[k].x * choose;
        c[k].y = p[k].y * choose;
        /* c[k].z = p[k].z * choose; */ /* use for 3-D curves */
    };
}

void BezierCurve(int NumCtlPoints, BezArray c, Point *pt, float t)
/* Return Point pt(t), t <= 0 <= 1 from C, given the number
 of Points in control polygon. BezierForm must be called
 once for any given control polygon. */
{
    int k, n;
    float t1, tt, u;
    BezArray b;

    n = NumCtlPoints - 1; u = t;
    b[0].x = c[0].x;
    b[0].y = c[0].y;
    /* b[0].z = c[0].z; */ /* for 3-D curves */
    for(k = 1; k <= n; k++) {
        b[k].x = c[k].x * u;
        b[k].y = c[k].y * u;
        /* b[k].z = c[k].z * u */ /* for 3-D curves */
        u = u * t;
    };

    (*pt).x = b[n].x; (*pt).y = b[n].y;
    t1 = 1 - t; tt = t1;
    for(k = n - 1; k >= 0; k--) {
        (*pt).x += b[k].x * tt;
        (*pt).y += b[k].y * tt;
        /* (*pt).z += b[k].z * tt; */ /* Again, 3-D */
        tt = tt * t1;
    }
}

float u;
int k;
PtArray pn;
```

```
BezArray bc;
Point pt;
void main ()
{
    pn[0].x = 100;  pn[0].y = 20;
    pn[1].x = 120;  pn[1].y = 40;
    pn[2].x = 140;  pn[2].y = 25;
    pn[3].x = 160;  pn[3].y = 20;
    BezierForm(4, pn, bc);

    for(k =0; k <=10; k++) {
        BezierCurve(4, bc, &pt, (float)k/10.0);
        printf("%3d  %8.4f  %8.4f\n",k, pt.x, pt.y);
        /* draw curve */
        /* if (k == 0) MoveTo(pt.x, pt.y);
           else LineTo(pt.x, pt.y); */
    }
}
```

/\*\*\*\*\*\*

InterPhong shading for Scan-line rendering algorithms

InterPhong shading has been used for rendering the  
synthetic actors Marilyn  
Monroe and Humphrey Bogart in the film "Rendez-vous  
a Montreal" directed by Nadia Magnenat Thalmann and  
Daniel Thalmann, 1987

\*\*\*\*\*/

#include <math.h>

#include "GraphicsGems.h"

#define RESANTI 3839

#define NBMAXSOURCES 10

#define SQRT3\_2 3.464101615

#define NIL 0

#define Trunc(v) ((int)floor(v))

/\* normalize vector, return in pn \*/

#define unity(v,pn) { double len ; \  
len = sqrt((v).x\*(v).x+(v).y\*(v).y+(v).z\*(v).z) ; \  
(pn)->x=(v).x/len; (pn)->y=(v).y/len; \  
(pn)->z=(v).z/len; }

typedef struct {  
double r, g, b;  
} Colors;

typedef struct {  
Colors coul;  
double w, n;  
} RecCoul;

/\* Declaration of types used for the datastructures that  
represent the information of a figure for the treatment by  
scanline ( software rendering )  
\*/

typedef struct blocedge {  
struct blocedge \*edsuiv; /\* next edge in the list \*/  
struct blocpoly \*ptpoly1, \*ptpoly2; /\*  
polygons sharing this edge \*/  
double x, dx; /\* Xmin and Xdelta \*/  
double z, dz; /\* Zmin and Zdelta \*/  
double ymax; /\* maximum Y of edge \*/  
double nx, dnx, ny, dny, nz, dnz;  
double px, dpx, py, dpy, pz, dpz;  
} BlockEdge;

typedef struct blocpoly {  
struct blocpoly \*polsuiv; /\* next polygon in the list \*/  
struct T\_ptedge \*ptlisttrie;  
RecCoul refl; /\* polygon characteristics color, spec.  
coeff., ... \*/  
Colors coulpoly; /\* polygon shading \*/  
Vector3 normalctri;  
double bias, tension;  
} BlockPoly;

```
typedef struct T_ptedge {
    BlockEdge *ptedtrie;
    struct T_ptedge *ptedsuiv;
} PtEdge;

/*
 * Declaration of types concerning the calculated image for the
 * current scanline ( Z-buffer )
 */

typedef struct scanbuf_el {
    Colors c, /* final color of this pixel */
    polycolor; /* initial color of
visible polygon */
} ScanBufType [RESANTI + 1];

typedef struct depthbuf_el {
    double depth; /* depth of opaque
pixel */
} DepthBufType [RESANTI + 1];

typedef struct {
    int xmin, xmax;
} PosBufType;

/*
 * Declaration of data structure types to store light source information
 */

static PosBufType posbuffer;

ScanBufType _scanbuffer; /* Z-buffer */
DepthBufType _depthbuffer;

static
void intphong(nestime, noriginal, bias, tension)
Vector3 *nestime, *noriginal;
double bias, tension;

/*
    Purpose: interphong interpolation
    Arguments
        nestime      : estimated normal
        noriginal     : original normal
        bias, tension : bias and tension
*/

{
    double fact;
    Vector3 vtemp;

    V3Sub (noriginal, nestime, &vtemp);
    fact = fabs(vtemp.x) + fabs(vtemp.y) + fabs(vtemp.z);
    fact = (fact + bias * (SQRT3_2 - fact)) * tension;
    V3Scale (vtemp, fact*V3Length (vtemp));
    V3Add (nestime, &vtemp, nestime);
    V3Normalize (nestime);
}

/*=====*/
```

```
void shadepoly(ptpoly)
BlockPoly      *ptpoly;

/*
  Purpose: shades a polygon on the current scanline
  Arguments
    ptpoly      : polygon to render
    noscane     : current scanline
*/

{
    BlockEdge      *edge1, *edge2;
    PtEdge  *tripedtrie;
    int      xx;
    double   dxx;
    double   zz, dzz;
    double   diffx;
    double   dnnx, dnnny, dnnnz;
    Vector3  normal, unitn;
    double   dppx, dppy, dppz;
    Vector3  point;
    Colors   cc;
    RecCoul  ptrefl;
    register struct scanbuf_el      *scanel ;
    register struct depthbuf_el     *depthel ;

    tripedtrie = ptpoly->ptlisttrie;
    ptrefl = ptpoly->refl;
    while (tripedtrie != (PtEdge *)NIL)
    {
        edge1 = tripedtrie->ptedtrie;
        if (tripedtrie->ptedsuiv != (PtEdge *)NIL)
        {
            tripedtrie = tripedtrie->ptedsuiv;
            edge2 = tripedtrie->ptedtrie;
        }
        else
            abort(" Odd number of edges on scanline");

        dxx = edge2->x - edge1->x;          /* distance between edges
                                           on current scanline */
        if (dxx < 1.0)                      /* crossing edges ? */
            dxx = 1.0;
        dxx = 1.0 / dxx;                    /* increment per pixel */
        xx = Trunc(edge1->x) + 1;            /* first pixel to be
                                           colored */
        diffx = xx - edge1->x;

        dzz = (edge2->z - edge1->z) * dxx;
        zz = edge1->z + dzz * diffx;

        if (xx < posbuffer.xmin)
            posbuffer.xmin = xx;
        if (edge2->x > posbuffer.xmax)
            posbuffer.xmax = Trunc(edge2->x);





        dnnx = (edge2->nx - edge1->nx) * dxx;
        dnnny = (edge2->ny - edge1->ny) * dxx;
        dnnnz = (edge2->nz - edge1->nz) * dxx;
        normal.x = (edge1->nx + dnnx * diffx) + dnnx;
```

```
normal.y = (edge1->ny + dnnny * diffx) + dnnny;
normal.z = (edge1->nz + dnnz * diffx) + dnnz;
dppx = (edge2->px - edge1->px) * dxx;
dppy = (edge2->py - edge1->py) * dxx;
dppz = (edge2->pz - edge1->pz) * dxx;
point.x = (edge1->px + dppx * diffx) + dppx;
point.y = (edge1->py + dppy * diffx) + dppy;
point.z = (edge1->pz + dppz * diffx) + dppz;

while (xx <= edge2->x)
{
    scanel = &_scanbuffer[xx];
    depthel = &_depthbuffer[xx];

    if (zz < depthel->depth)
    {
        unity(normal, &unitn);
        intphong(&unitn, &ptpoly->normalctri,
                ptpoly->bias, ptpoly->tension);
        cc = ptpoly->coulpoly;
        depthel->depth = zz;
        scanel->polycolor = ptrefl.coul;
        scanel->c = cc;
    }
    xx = xx + 1;
    zz = zz + dzz;
    normal.x = normal.x + dnnx;
    normal.y = normal.y + dnnny;
    normal.z = normal.z + dnnz;
    point.x = point.x + dppx;
    point.y = point.y + dppy;
    point.z = point.z + dppz;
}
tripedtrie = tripedtrie->ptedsuiv;
}
```

# Index of /pubs/tog/GraphicsGems/gemsiii/simplex/

Name	Last modified	Size	Description
 <a href="#">_Parent Directory</a>			
 <a href="#">_Makefile</a>	29-Jun-00 08:16	1K	
 <a href="#">_recur.C</a>	29-Jun-00 08:16	1K	
 <a href="#">_symm.C</a>	29-Jun-00 08:16	1K	



```
# code is C++
CFLAGS =

recur.o: recur.C
    CC $(CFLAGS) -c recur.C -o recur.o

symm.o: symm.C
    CC $(CFLAGS) -c symm.C -o symm.o

clean:
    rm -rf recur.o symm.o
```

```
/******
```

Given an s-simplex (with s+1 vertexes) in n dimensions,  
calculate the vertexes of the kth ( $0 \leq k < (1 \leq s)$ )  
subsimplex in the recursive subdivision of the simplex.

Several implementations of the bitCount() function are  
described in "Of Integers, Fields, and Bit Counting",  
Paeth and Schilling, Graphics Gems II.

Entry:

- src\_vtx - list of the vertexes of the original simplex
- n - each n consecutive floats represents one vertex
- s - there are s+1 vertexes in the s-simplex
- k - identifies which subsimplex is to be generated

Exit:

- dst\_vtx - list of the vertexes of the kth subsimplex

```
*****/
```

```
void rec_subsimplex(register float* dst_vtx,  
                  const float* const src_vtx,  
                  int n,  
                  int s,  
                  int k)  
{  
    int id[2];  
    id[1] = n*bitCount(k);  
    id[0] = -id[1];  
  
    for (int j = 0; j <= s; ++j)  
    {  
        for (int i = 0; i < n; ++i)  
            *dst_vtx++ = (src_vtx[i-id[0]] + src_vtx[i+id[1]]) / 2.0;  
        id[ (k&1) ] += n;  
        k >>= 1;  
    }  
}
```

```
/******
```

Given an s-simplex (with s+1 vertexes) in n dimensions,  
calculate the vertexes of the kth ( $0 \leq k < (1 \leq s)$ )  
subsimplex in the symmetric subdivision of the simplex.

Several implementations of the bitCount() function are  
described in "Of Integers, Fields, and Bit Counting",  
Paeth and Schilling, Graphics Gems II.

Entry:

- src\_vtx - list of the vertexes of the original simplex
- n - each n consecutive floats represents one vertex
- s - there are s+1 vertexes in the s-simplex
- k - identifies which subsimplex is to be generated

Exit:

- dst\_vtx - list of the vertexes of the kth subsimplex

```
*****/
```

```
void sym_subsimplex(register float* dst_vtx,  
                  const float* const src_vtx,  
                  int n,  
                  int s,  
                  int k)  
{  
    int id[2];  
    id[1] = n*bitCount(k);  
    id[0] = 0;  
  
    for (int j = 0; j <= s; ++j)  
    {  
        for (int i = 0; i < n; ++i)  
            *dst_vtx++ = (src_vtx[i+id[0]] + src_vtx[i+id[1]]) / 2.0;  
        id[!(k&1)] += n;  
        k >>= 1;  
    }  
}
```

```
/*
Fast Anti-Aliasing Polygon Scan Conversion
by Jack Morrison
from "Graphics Gems", Academic Press, 1990

user provides screenX(), vLerp(), and renderPixel() routines.
*/

/*
* Anti-aliased polygon scan conversion by Jack Morrison
*
* This code renders a polygon, computing subpixel coverage at
* 8 times Y and 16 times X display resolution for anti-aliasing.
* One optimization left out for clarity is the use of incremental
* interpolations. X coordinate interpolation in particular can be
* with integers. See Dan Field's article in ACM Transactions on
* Graphics, January 1985 for a fast incremental interpolator.
*/
#include <math.h>
#include "GraphicsGems.h"

#define SUBYRES 8          /* subpixel Y resolution per scanline */
#define SUBXRES 16         /* subpixel X resolution per pixel */
#define MAX_AREA          (SUBYRES*SUBXRES)
#define MODRES(y)         ((y) & 7)          /*subpixel Y modulo */
#define MAX_X             0x7FFF /* subpixel X beyond right edge */

typedef struct SurfaceStruct { /* object shading surface info */
    int    red, green, blue;    /* color components */
} Surface;

/*
* In real life, SurfaceStruct will contain many more parameters as
* required by the shading and rendering programs, such as diffuse
* and specular factors, texture information, transparency, etc.
*/

typedef struct VertexStruct { /* polygon vertex */
    Vector3 model, world,      /* geometric information */
           normal, image;
    int y;                    /* subpixel display coordinate */
} Vertex;

Vertex *Vleft, *VnextLeft;    /* current left edge */
Vertex *Vright, *VnextRight; /* current right edge */

struct SubPixel { /* subpixel extents for scanline */
    int xLeft, xRight;
} sp[SUBYRES];

int    xLmin, xLmax;          /* subpixel x extremes for scanline */
int    xRmax, xRmin;          /* (for optimization shortcut) */

/* Compute sub-pixel x coordinate for vertex */
extern int screenX(/* Vertex *v */);

/* Interpolate vertex information */
extern void vLerp(/* double alpha, Vertex *Va, *Vb, *Vout */);

/* Render polygon for one pixel, given coverage area */
/* and bitmask */
extern void renderPixel(/* int x, y, Vertex *V,
```

```
int area, unsigned mask[],
Surface *object */);

/*
 * Render shaded polygon
 */
drawPolygon(polygon, numVertex, object)
    Vertex  polygon[];          /*clockwise clipped vertex list */
    int      numVertex;         /*number of vertices in polygon */

    Surface *object;            /* shading parms for this object */
{
    Vertex *endPoly;            /* end of polygon vertex list */
    Vertex VscanLeft, VscanRight; /* interpolated vertices */
/* at scanline */
    double aLeft, aRight;       /* interpolation ratios */
    struct SubPixel *sp_ptr;    /* current subpixel info */
    int xLeft, xNextLeft;       /* subpixel coordinates for */
    int xRight, xNextRight;     /* active polygon edges */
    int i,y;

/* find vertex with minimum y (display coordinate) */
Vleft = polygon;
for (i=1; i<numVertex; i++)
    if (polygon[i].y < Vleft->y)
        Vleft = &polygon[i];
endPoly = &polygon[numVertex-1];

/* initialize scanning edges */
Vright = VnextRight = VnextLeft = Vleft;

/* prepare bottom of initial scanline - no coverage by polygon */
for (i=0; i<SUBYRES; i++)
    sp[i].xLeft = sp[i].xRight = -1;
xLmin = xRmin = MAX_X;
xLmax = xRmax = -1;

/* scan convert for each subpixel from bottom to top */
for (y=Vleft->y; ; y++) {

    while (y == VnextLeft->y) { /* reached next left vertex */
        VnextLeft = (Vleft=VnextLeft) + 1; /* advance */
        if (VnextLeft > endPoly) /* (wraparound) */
            VnextLeft = polygon;
        if (VnextLeft == Vright) /* all y's same? */
            return; /* (null polygon) */
        xLeft = screenX(Vleft);
        xNextLeft = screenX(VnextLeft);
    }

    while (y == VnextRight->y) { /*reached next right vertex */
        VnextRight = (Vright=VnextRight) -1;
        if (VnextRight < polygon) /* (wraparound) */
            VnextRight = endPoly;
        xRight = screenX(Vright);
        xNextRight = screenX(VnextRight);
    }

    if (y>VnextLeft->y || y>VnextRight->y) {
        /* done, mark uncovered part of last scanline */
        for (; MODRES(y); y++)

```

```
        sp[MODRES(y)].xLeft = sp[MODRES(y)].xRight = -1;
        renderScanline(Vleft, Vright, y/SUBYRES, object);
        return;
    }
```

```
/*
 * Interpolate sub-pixel x endpoints at this y,
 * and update extremes for pixel coherence optimization
 */
```

```
    sp_ptr = &sp[MODRES(y)];
    aLeft = (double)(y - Vleft->y) / (VnextLeft->y - Vleft->y);
    sp_ptr->xLeft = LERP(aLeft, xLeft, xNextLeft);
    if (sp_ptr->xLeft < xLmin)
        xLmin = sp_ptr->xLeft;
    if (sp_ptr->xLeft > xLmax)
        xLmax = sp_ptr->xLeft;

    aRight = (double)(y - Vright->y) / (VnextRight->y
                                         - Vright->y);
    sp_ptr->xRight = LERP(aRight, xRight, xNextRight);
    if (sp_ptr->xRight < xRmin)
        xRmin = sp_ptr->xRight;
    if (sp_ptr->xRight > xRmax)
        xRmax = sp_ptr->xRight;

    if (MODRES(y) == SUBYRES-1) {          /* end of scanline */
        /* interpolate edges to this scanline */
        vLerp(aLeft, Vleft, VnextLeft, &VscanLeft);
        vLerp(aRight, Vright, VnextRight, &VscanRight);
        renderScanline(&VscanLeft, &VscanRight, y/SUBYRES, object);
        xLmin = xRmin = MAX_X;              /* reset extremes */
        xLmax = xRmax = -1;
    }
}
```

```
/*
 * Render one scanline of polygon
 */
```

```
renderScanline(Vl, Vr, y, object)
    Vertex *Vl, *Vr;          /* polygon vertices interpolated */
                               /* at scanline */
    int y;                    /* scanline coordinate */
    Surface *object;          /* shading parms for this object */
{
    Vertex Vpixel; /*object info interpolated at one pixel */
    unsigned mask[SUBYRES]; /*pixel coverage bitmask */
    int x;                /* leftmost subpixel of current pixel */

    for (x=SUBXRES*floor((double)(xLmin/SUBXRES)); x<=xRmax; x+=SUBXRES) {
        vLerp((double)(x-xLmin)/(xRmax-xLmin), Vl, Vr, &Vpixel);
        computePixelMask(x, mask);
        renderPixel(x/SUBXRES, y, &Vpixel,
                    /*computePixel*/Coverage(x), mask, object);
    }
}
```

```
/*
 * Compute number of subpixels covered by polygon at current pixel
 */
```

```
*/
/*computePixel*/Coverage(x)
    int x;                /* left subpixel of pixel */
{
    int area;              /* total covered area */
    int partialArea;       /* covered area for current subpixel y */
    int xr = x+SUBXRES-1;  /*right subpixel of pixel */
    int y;

    /* shortcut for common case of fully covered pixel */
    if (x>xLmax && x<xRmin)
        return MAX_AREA;

    for (area=y=0; y<SUBYRES; y++) {
        partialArea = MIN(sp[y].xRight, xr)
            - MAX(sp[y].xLeft, x) + 1;
        if (partialArea > 0)
            area += partialArea;
    }
    return area;
}

/* Compute bitmask indicating which subpixels are covered by
 * polygon at current pixel. (Not all hidden-surface methods
 * need this mask. )
 */
computePixelMask(x, mask)
    int x;                /* left subpixel of pixel */
    unsigned mask[];      /* output bitmask */
{
    static unsigned leftMaskTable[] =
        { 0xFFFF, 0x7FFF, 0x3FFF, 0x1FFF, 0x0FFF, 0x07FF, 0x03FF,
          0x01FF, 0x00FF, 0x007F, 0x003F, 0x001F, 0x000F, 0x0007,
          0x0003, 0x0001 };
    static unsigned rightMaskTable[] =
        { 0x8000, 0xC000, 0xE000, 0xF000, 0xF800, 0xFC00,
          0xFE00, 0xFF00, 0xFF80, 0xFFC0, 0xFFE0, 0xFFFF0,
          0xFFFF8, 0xFFFFC, 0xFFFFE, 0xFFFF };
    unsigned leftMask, rightMask;      /* partial masks */
    int xr = x+SUBXRES-1;              /* right subpixel of pixel */
    int y;

    /* shortcut for common case of fully covered pixel */
    if (x>xLmax && x<xRmin)
        {
            for (y=0; y<SUBYRES; y++)
                mask[y] = 0xFFFF;
        }
    else {
        for (y=0; y<SUBYRES; y++)
            {
                if (sp[y].xLeft < x) /* completely left of pixel*/
                    leftMask = 0xFFFF;
                else if (sp[y].xLeft > xr) /* completely right */
                    leftMask = 0;
                else
                    leftMask = leftMaskTable[sp[y].xLeft -x];

                if (sp[y].xRight > xr) /* completely */
                    /* right of pixel*/
                    rightMask = 0xFFFF;
                else if (sp[y].xRight < x) /*completely left */
                    rightMask = 0;
                else
            }
    }
}
```

```
                rightMask = rightMaskTable[sp[y].xRight -x];
            mask[y] = leftMask & rightMask;
        }
    }
```



```
/*
 * Spherical linear interpolation of unit quaternions with spins
 */
#include <math.h>

#define EPSILON 1.0E-6          /* a tiny number */
#define TRUE 1
#define FALSE 0

typedef struct {                /* quaternion type */
    double w, x, y, z;
} Quaternion;

slerp(alpha, a, b, q, spin)
    double alpha;               /* interpolation parameter (0 to 1) */
    Quaternion *a, *b;          /* start and end unit quaternions */
    Quaternion *q;              /* output interpolated quaternion */
    int spin;                   /* number of extra spin rotations */
{
    double beta;               /* complementary interp parameter */
    double theta;              /* angle between A and B */
    double sin_t, cos_t;       /* sine, cosine of theta */
    double phi;                /* theta plus spins */
    int bflip;                 /* use negation of B? */

    /* cosine theta = dot product of A and B */
    cos_t = a->x*b->x + a->y*b->y + a->z*b->z + a->w*b->w;

    /* if B is on opposite hemisphere from A, use -B instead */
    if (cos_t < 0.0) {
        cos_t = -cos_t;
        bflip = TRUE;
    } else
        bflip = FALSE;

    /* if B is (within precision limits) the same as A,
     * just linear interpolate between A and B.
     * Can't do spins, since we don't know what direction to spin.
     */
    if (1.0 - cos_t < EPSILON) {
        beta = 1.0 - alpha;
    } else {                    /* normal case */
        theta = acos(cos_t);
        phi = theta + spin * M_PI;
        sin_t = sin(theta);
        beta = sin(theta - alpha*phi) / sin_t;
        alpha = sin(alpha*phi) / sin_t;
    }

    if (bflip)
        alpha = -alpha;

    /* interpolate */
    q->x = beta*a->x + alpha*b->x;
    q->y = beta*a->y + alpha*b->y;
    q->z = beta*a->z + alpha*b->z;
    q->w = beta*a->w + alpha*b->w;
}
```

/\* A Digital Dissolve Effect  
by Mike Morton  
from "Graphics Gems", Academic Press, 1990

user must provide copy() function.  
\*/

```
/*
 * Code fragment to advance from one element to the next.
 *
 * int reg;                /* current sequence element
 * reg = 1;                /* start in any non-zero state
 * if (reg & 1)             /* is the bottom bit set?
 *     reg = (reg >>1) ^ MASK; /* yes: toss out 1 bit; XOR in mask
 * else reg = reg >>1;      /* no: toss out 0 bit
 */

int randmasks[32];        /* Gotta fill this in yourself. */

dissolve1 (height, width) /* first version of the dissolve
/* algorithm */
{
    int height, width;    /* number of rows, columns */

    int pixels, lastnum;  /* number of pixels; */
                        /* last pixel's number */
    int regwidth;         /* "width" of sequence generator */
    register long mask;    /* mask to XOR with to*/
                        /* create sequence */
    register unsigned long element; /* one element of random sequence */
    register int row, column; /* row and column numbers for a pixel */

    /* Find smallest register which produces enough pixel numbers */
    pixels = height * width; /* compute number of pixels */
                        /* to dissolve */
    lastnum = pixels-1; /* find last element (they go 0..lastnum) */
    regwidth = bitwidth ((unsigned int)lastnum); /* how wide must the */
                        /* register be? */
    mask = randmasks [regwidth]; /* which mask is for that width? */

    /* Now cycle through all sequence elements. */

    element = 1; /* 1st element (could be any nonzero) */

    do {
        row = element / width; /* how many rows down is this pixel? */
        column = element % width; /* and how many columns across? */
        if (row < height) /* is this seq element in the array? */
            copy (row, column); /* yes: copy the (r,c)'th pixel */

        /* Compute the next sequence element */
        if (element & 1) /* is the low bit set? */
            element = (element >>1)^mask; /* yes: shift value, */
                        /* XOR in mask */
        else element = (element >>1); /* no: just shift the value */
    } while (element != 1); /* loop until we return */
                        /* to original element */
    copy (0, 0); /* kludge: the loop doesn't produce (0,0) */
} /* end of dissolve1() */
```

```
int bitwidth (N)          /* find "bit-width" needed to represent N */
    unsigned int N; /* number to compute the width of */
{
    int width = 0; /* initially, no bits needed to represent N */
    while (N != 0) { /* loop 'til N has been whittled down to 0 */
        N >>= 1; /* shift N right 1 bit (NB: N is unsigned) */
        width++; /* and remember how wide N is */
    } /* end of loop shrinking N down to nothing */
    return (width); /* return bit positions counted */
} /* end of bitwidth() */

dissolve2 (height, width) /* fast version of the dissolve algorithm */
    int height, width; /* number of rows, columns */
{
    int rwidth, cwidth; /* bit width for rows, for columns */
    int regwidth; /* "width" of sequence generator */
    register long mask; /* mask to XOR with to create sequence */
    register int rowshift; /* shift distance to get row */
                                /* from element */
    register int colmask; /* mask to extract column from element */
    register unsigned long element; /* one element of random */
/* sequence */
    register int row, column; /* row and column for one pixel */

    /* Find the mask to produce all rows and columns. */

    rwidth = bitwidth ((unsigned int)height); /* how many bits needed for height? */
    cwidth = bitwidth ((unsigned int)width); /* how many bits needed for width? */
    regwidth = rwidth + cwidth; /* how wide must the register be? */
    mask = randmasks [regwidth]; /* which mask is for that width? */

    /* Find values to extract row and col numbers from each element. */
    rowshift = cwidth; /* find dist to shift to get top bits (row) */
    colmask = (1<<cwidth)-1; /* find mask to extract */
                                /* bottom bits (col) */

    /* Now cycle through all sequence elements. */







    element = 1; /* 1st element (could be any nonzero) */
    do {
        row = element >> rowshift; /* find row number for this pixel */
        column = element & colmask; /* and how many columns across? */
        if ((row < height) /* does element fall in the array? */
            && (column < width)) /* ...must check row AND column */
            copy (row, column); /* in bounds: copy the (r,c)'th pixel */

        /* Compute the next sequence element */
        if (element & 1) /* is the low bit set? */
            element = (element >>1)^mask; /* yes: shift value, */
                                /* XOR in mask */
        else element = (element >>1); /* no: just shift the value */
    } while (element != 1); /* loop until we return to */
                                /* original element */
}
```

```
    copy (0, 0);          /* kludge: element never comes up zero */  
}                          /* end of dissolve2() */
```

# Index of

## /pubs/tog/GraphicsGems/gemsii/Peano/

Name	Last modified	Size	Description
 <a href="#">_Parent Directory</a>			
 <a href="#">_Makefile</a>	29-Jun-00 08:14	1K	
 <a href="#">_main.c</a>	29-Jun-00 08:14	3K	
 <a href="#">_mapply.c</a>	29-Jun-00 08:14	1K	
 <a href="#">_peano.c</a>	29-Jun-00 08:14	3K	
 <a href="#">_types.h</a>	29-Jun-00 08:14	1K	

```
CFLAGS = -O
LIBES = -lm
PROG = peano

all: $(PROG) mapply

# compile main program
$(PROG): peano.o main.o
    cc peano.o main.o $(CFLAGS) -o $(PROG) $(LIBES)

# dependencies on #include files
peano.o main.o : types.h

mapply: mapply.c types.h
    cc mapply.c $(CFLAGS) -o mapply $(LIBES)

clean:
    /bin/rm -f peano.o main.o peano mapply
```

```
/* main.c */

/* copyright Ken Musgrave */
/* June 1986 */

/*
 * creates an 8-bit, 2-D peano curve image file
 * background is 0, curve segments range in value from 1 to 255
 *
 * NOTE: the peano curve can be n-dimensional,
 * this is just an example of the use of the
 * routines in peano.c
 */

#include <stdio.h>
#include <math.h>
#include "types.h"

char          *image_file="im_file";

unsigned char  fb[FB_SIZE + 1][FB_SIZE + 1];
FILE          *outfile, *fopen();
vector        gcoord, glast_coord;

main(argc, argv)
int           argc;
char         *argv[];
{
    int          i;

    /* get command-line arg */
    if (argc != 2) {
        fprintf(stderr, "usage: %s precision\n", argv[0]);
        exit(0);
    }
    dimensions = 2;          /* the dimension is fixed in this example */
    precision = atoi(argv[1]);
    if (precision < 0 || precision > MAX_PRECISION - 1) {
        fprintf(stderr, "%s: can't work with %d bits of precision!\n", argv[0],
precision);
        exit(-1);
    }
    if ((outfile = fopen(image_file, "w")) == NULL) {
        fprintf(stderr, "Error: cannot open file %s\n", image_file);
        exit(-1);
    }
    printf("%s: filling %d dimensions to %d bits of precision.\n",
          argv[0], dimensions, precision);

    /* begin the mysterious & incomprehensible algorithm... */
    for (i=0; i<dimensions; i++) {
        bitmask[i] = 1 << (dimensions - i - 1);
        bytemask |= 1 << i;
        glast_coord[i] = 0;
    }

    /* run the mysterious & incomprehensible algorithm... */
    recurse(gcoord, glast_coord, (int) pow(2.0, (double) precision),
```

```
        dimensions);

        /* inform user that execution of algorithm is complete */
        fprintf(stderr, "Spewing...");

        for (i=FB_SIZE-1; i>=0; i--)
            fwrite(fb[i], 1, FB_SIZE, outfile);

        fprintf(stderr, " done.");

        exit(0);
} /* main() */

/*
 * recursive routine to call "peano()"
 */
recurse(coord, last_coord, iterations, level)
vector      coord, last_coord;
int         iterations, level;
{
    int      i ;
    static int      n=0, scale, offset;

    if (level > 0)
        for (i=0; i<iterations; i++)
            recurse(coord, last_coord, iterations, level - 1);
    else {
        /* get x,y coord of position n on peano curve */
        peano(coord, n++);
        offset = ((int) pow(2.0, (double) precision));
        scale = FB_SIZE / ((int) pow(2.0, (double) precision));
        offset = scale / 2;
        /* draw line between adjacent coords */
        draw_line(scale * coord[1] + offset,
                  FB_SIZE - scale * coord[0] - offset,
                  scale * last_coord[1] + offset,
                  FB_SIZE - scale * last_coord[0] - offset,
                  n);

        /*
         * x = scale*coord[1]+offset; y =
         * FB_SIZE-scale*coord[0]-offset; index = n%256;
         */

        for (i=0; i<precision; i++)
            last_coord[i] = coord[i];
    }
} /* recurse() */

/*
 * draws horizontal and vertical lines into "fb" with color "index"
 */
draw_line(x1, y1, x2, y2, index)
unsigned   x1, y1, x2, y2, index;
{
    int     tmp, i;

```



```
index = index % 256;
if (index == 0)
    index = 1;

if (x1 != x2) {          /* have horizontal line */
    if (x1 > x2) {
        tmp = x1;
        x1 = x2;
        x2 = tmp;
    }
    for (i=x1; i<=x2; i++)
        fb[y1][i] = (unsigned char) index;
} else {                /* vertical line */
    if (y1 > y2) {
        tmp = y1;
        y1 = y2;
        y2 = tmp;
    }
    for (i=y1; i<=y2; i++)
        fb[i][x1] = (unsigned char) index;
}
} /* draw_line() */
```

```
/* mapply.c */
/*
 * maps an 8-bit image into a 24-bit image,
 * using color map in a user-specified file
 *
 * color maps may be generated using ran_ramp
 */

/* Ken Musgrave */
/* June 1986 */

#include <stdio.h>
#include <math.h>
#include "types.h"

char          *image_file = "img";

int           r, g, b;
unsigned char  eight_bit[FB_SIZE], twenty_four_bit[FB_SIZE][3], cmap[256][3];

FILE          *imfile, *cmapfile, *outfile, *fopen();

main(argc, argv)
int           argc;
char          *argv[];
{
    int        i, x, y;

    /* get command-line args */
    if (argc != 3) {
        fprintf(stderr, "usage: %s infile cmap_file\n", argv[0]);
        exit(0);
    }
    if ((imfile = fopen(argv[1], "r")) == NULL) {
        fprintf(stderr, "Error: cannot open file %s\n", argv[1]);
        exit(-1);
    }
    if ((cmapfile = fopen(argv[2], "r")) == NULL) {
        fprintf(stderr, "Error: cannot open file %s\n", argv[2]);
        exit(-1);
    }
    if ((outfile = fopen(image_file, "w")) == NULL) {
        fprintf(stderr, "Error: cannot open file %s\n", image_file);
        exit(-1);
    }
    printf("%s: mapping \"%s\" to \"%s\" using map in \"%s\"\n",
           argv[0], argv[1], image_file, argv[2]);

    fscanf(cmapfile, "%d %d\n", &r, &g);
    for (i = 0; i < 256; i++) {
        fscanf(cmapfile, "%d %d %d\n", &r, &g, &b);
        cmap[i][0] = (unsigned char) r;
        cmap[i][1] = (unsigned char) g;
        cmap[i][2] = (unsigned char) b;
    }
    cmap[0][0] = cmap[1][0];
    cmap[0][1] = cmap[1][1];
    cmap[0][2] = cmap[1][2];
}
```

```
cmap[255][0] = cmap[254][0];
cmap[255][1] = cmap[254][1];
cmap[255][2] = cmap[254][2];

for (y = 0; y < FB_SIZE; y++) {
    fread(eight_bit, 1, FB_SIZE, imfile);
    for (x = 0; x < FB_SIZE; x++) {
        if (eight_bit[x]) {
            twenty_four_bit[x][0] = cmap[eight_bit[x]][0];
            twenty_four_bit[x][1] = cmap[eight_bit[x]][1];
            twenty_four_bit[x][2] = cmap[eight_bit[x]][2];
        } else {
            twenty_four_bit[x][0] = 0;
            twenty_four_bit[x][1] = 0;
            twenty_four_bit[x][2] = 0;
        }
    }
    fwrite(twenty_four_bit, 3, FB_SIZE, outfile);
}

} /* main() */
```

```
/* peano.c */

/* copyright Ken Musgrave */
/* June 1986 */

/*
 * space-filling peano curve algorithm.
 * fills n-space with a 1-D peano curve.
 *
 * the algorithm is utterly incomprehensible,
 * so expect a paucity of sensible comments in this code.
 * it consists largely of bit-wise logical operations,
 * and is therefore quite impenetrable.
 * but it works.
 */

#include "types.h"

/*
 * determine the n-space coordinate of "point" on the peano curve
 */
peano(coord, point)
vector      coord;
int         point;
{
    int      i;

    zero(sigma);          /* initialize necessary arrays */
    zero(tilde_sigma);
    zero(tilde_tau);

    build_rho(point);
    for (i=0; i<precision; i++)
        J[i] = principal_pos(rho[i]);
    build_sigma();
    build_tau();
    build_tilde_sigma();
    build_tilde_tau();
    build_omega();
    build_alpha();

    v_convert(alpha, coord);
} /* peano() */

/*
 * build "rho" array
 */
build_rho(point)
int         point;
{
    int      i, mask=bytemask;

    for (i=0; i<precision; i++) {
        rho[precision - i - 1] = (point & mask) >> (i * dimensions);
        mask <=< dimensions;
    }
}
```

```
} /* build_rho() */

/*
 * find principal position of "a_byte"
 */
principal_pos(a_byte)
byte          a_byte;
{
    int          nth_bit, i=1;

    nth_bit = a_byte & 0x01;
    for (i=1; i<dimensions; i++) {
        if (((a_byte & bitmask[dimensions - i - 1]) >> i) != nth_bit)
            return (dimensions - i);
    }

    return (dimensions);    /* all bits are the same */
} /* principal_pos() */

/*
 * build "sigma" array
 */
build_sigma()
{
    int          i, bit;

    for (i=0; i<precision; i++) {
        sigma[i] |= rho[i] & bitmask[0];
        for (bit=1; bit<dimensions; bit++) {
            sigma[i] |= (rho[i] & bitmask[bit])
                ^ ((rho[i] & bitmask[bit - 1]) >> 1);
        }
    }
}

} /* build_sigma() */

/*
 * build "tau" array
 */
build_tau()
{
    int          parity, bit, i, j;
    byte          temp_byte;

    for (i=0; i<precision; i++) {
        parity = 0;

        /* complement nth bit */
        if (sigma[i] & bitmask[dimensions - 1])
            tau[i] = sigma[i] - 1; /* nth bit was 1 */
        else
            tau[i] = sigma[i] + 1; /* nth bit was 0 */

        for (j=0; j<dimensions; j++) /* find parity */
            if (tau[i] & bitmask[j])
                parity++;
    }
}
```

```
        if (ODD(parity)) {          /* complement principal bit */
            bit = J[i] - 1; /* get index of principal bit */
                               /* get bit in question */
            temp_byte = tau[i] & bitmask[bit];
            tau[i] |= bitmask[bit]; /* set the bit to 1 */
            tau[i] &= ~temp_byte;   /* assign complement */
        }
    }

} /* build_tau() */

/*
 * build "tilde_sigma" array
 */
build_tilde_sigma()
{
    int          i, shift=0;

    tilde_sigma[0] = sigma[0];
    for (i=1; i<precision; i++) {
        shift += J[i - 1] - 1;
        shift %= dimensions;
        tilde_sigma[i] = RT_CSHFT(sigma[i], shift, dimensions, bytemask);
    }
}

} /* build_tilde_sigma() */

/*
 * build "tilde_tau" array
 */
build_tilde_tau()
{
    int          i, shift=0;

    tilde_tau[0] = tau[0];
    for (i=1; i<precision; i++) {
        shift += J[i - 1] - 1;
        shift %= dimensions;
        tilde_tau[i] = RT_CSHFT(tau[i], shift, dimensions, bytemask);
    }
}

} /* build_tilde_tau() */

/*
 * build "omega" array
 */
build_omega()
{
    int          i;

    omega[0] = 0;
    for (i=1; i<precision; i++)
        omega[i] = omega[i - 1] ^ tilde_tau[i - 1];
} /* build_omega() */

/*
```

```
* build "alpha" array
*/
build_alpha()
{
    int          i;

    for (i=0; i<precision; i++)
        alpha[i] = omega[i] ^ tilde_sigma[i];
} /* build_alpha() */

/*
 * initialize "array" to zeros
 */
zero(array)

    r_array      array;

{
    int          i;

    for (i=0; i<precision; i++)
        array[i] = 0;
} /* zero() */

/*
 * convert "alpha" array into n_space coordinate vector
 */
v_convert(alph, coord)
r_array    alph;
vector     coord;
{
    int          i, j, bit, a_bitmask=1;

    for (i=0; i<dimensions; i++) {
        coord[i] = 0;
        bit = precision;
        for (j=0; j<precision; j++) /* extract each bit of coord
                                     * i */
            coord[i] |= ((alph[j] & a_bitmask) << --bit) >> i;
        a_bitmask <<= 1;
    }
} /* v_convert() */
```

```
/* types.h */
/* types and #define's for peano curve algorithm */

/* copyright Ken Musgrave */
/* June 1986 */

#define FB_SIZE          1024          /* frame buffer size */

#define MAX_DIMENSIONS   5             /* dimensionality of space */
#define MAX_PRECISION    11           /* number of bits/dimension -1 */

#define ODD(x)           ( ( (x) & 0x1 ) ? 1 : 0 )

/* right circular shift */
#define RT_CSHFT( byte, shift, dimensions, bytemask ) \
    (((byte) >> (shift)) | ((byte) << dimensions - (shift))) & bytemask

typedef char    byte;                  /* size must be >= MAX_PRECISION */

typedef int     vector[MAX_DIMENSIONS]; /* n-space vector */

typedef byte    r_array[MAX_PRECISION]; /* vector of type "r" in algo */

/* global variable section */

int    J[MAX_PRECISION];              /* storage for principal positions */

/* global arrays */
r_array rho, sigma, tau, tilde_sigma, tilde_tau, omega, alpha;

byte    bitmask[MAX_DIMENSIONS];      /* to be filled with bit masks */

int     dimensions;                   /* number of dimensions being filled */
int     precision;                    /* number of bits of precision used */
byte    bytemask;                     /* masks "dimensions" bits */
```



```
/* copyright Ken Musgrave */
/* March 1985 */

/* ran_ramp.c */

/*****
 *
 * Usage: ran_ramp [-g] [-z]
 *
 * Performs random continuous changes to the color map of a frame buffer.
 *
 * The idea is to use three DDA's with endpoint input from a random
 * number generator. The three DDA's generate random sawtooth waves
 * of values for red, green, and blue. These waves of values are pushed
 * through the lookup table from entry 254 down to entry 0.
 *
 * NOTE: Entry 255 remains black, as this was designed for animating
 * the Mandelbrot set, whose center should remain black.
 *
 *****/

#include <stdio.h>
#include <ctype.h>
#include <signal.h>

#define TRUE 1
#define FALSE 0

#define MAXENTRY 256
#define MAXINDEX 255
#define DELAY 8 /* kludge to modulate speed of animation */
                /* you may want/need to change this value */

/*
 * pseudo-random number generator; period 65536; requires seed between 0 and
 * 65535; returns random numbers between 0 and 65536.
 */
#define MULTIPLIER 25173
#define INCREMENT 13849
#define MODULUS 65535
#define RANDOM(x) (MULTIPLIER * x + INCREMENT) & MODULUS

int quit = FALSE; /* signal variables */
int stop = FALSE; /* used in trapping <ctrl-C> and <ctrl-Z> */

int seed, maxsteps; /* user input variables */

/* arrays for color lookup table values */
unsigned char r[MAXENTRY], g[MAXENTRY], b[MAXENTRY];

main(argc, argv)
int argc;
char *argv[];
{
    register int i;
    int reply, delay;

    int inter(); /* signal functions, see below */
    int suspend();
```

```
signal(SIGINT, inter);          /* traps <ctrl-C> */
signal(SIGTSTP, suspend);       /* traps <ctrl-Z> */

printf("Please enter seed: ");
scanf("%d", &seed);

                                /* "(10-100)" is just a suggestion... */
printf("\nPlease enter maximum length of color ramp:  (10-100)");
scanf("%d", &maxsteps);
printf("\n");

fb_init();                      /* generic frame buffer init routine */

                                /* if specified, set initial map option */
if (argc == 2) {
    /* "-g" option for gray scale */
    if (argv[1][0] == '-' && argv[1][1] == 'g') {
        for (i=0; i<MAXINDEX; i++)
            r[i] = g[i] = b[i] = i;
        r[MAXINDEX] = g[MAXINDEX] = b[MAXINDEX] = 0;
    }
    /* "-z" option for zebra scale */
    else if (argv[1][0] == '-' && argv[1][1] == 'z') {
        for (i=0; i<MAXINDEX; i+=4) {
            r[i]=r[i+1]=g[i]=g[i+1]=b[i]=b[i+1]=i * 4 / 5;
            r[i+2]=r[i+3]=g[i+2]=g[i+3]=b[i+2]=b[i+3]=
                i * 4/5 + 51;
        }
        r[MAXINDEX] = g[MAXINDEX] = b[MAXINDEX] = 0;
    }
} else {
    /* initialize the color map to black */
    for (i=0; i<MAXINDEX; i++)
        r[i] = g[i] = b[i] = 0;
}

                                /* generic routine to write frame buffer color map */
fb_setmap(r, g, b);

                                /* loop until <ctrl-C> is trapped */
while (!quit) {
    /* trapped <ctrl-Z> */
    if (stop) {
        fb_done();          /* release frame buffer */
        printf("\nSave lookup table? (y/n) ");
        while (isspace(reply = getchar()));
        if (reply == 'y')
            save_lut();
                                /* returns when job running again */
        kill(getpid(), SIGSTOP);
        stop = FALSE;
        fb_init();          /* get back in action */
    }

    /*
     * main animation loop:
     *
     * move each entry in color map arrays down one place
     */
    for (i=0; i<(MAXINDEX-1); i++) {
```

```
        r[i] = r[i+1];
        g[i] = g[i+1];
        b[i] = b[i+1];
    }
    /*
     * get new high color map entries
     */
    r[MAXINDEX-1] = dda_red();
    g[MAXINDEX-1] = dda_green();
    b[MAXINDEX-1] = dda_blue();
    /*
     * send new color maps to the frame buffer
     * NOTE: we may do this several times (the value of
     * "DELAY") to slow down the animation
     */
    for (delay=0; delay<DELAY; delay++)
        fb_setmap(0, MAXINDEX, r, g, b);

}

fb_done();      /* release the frame buffer */

} /* main() */

/*
 * function for trapping <ctrl-C>
 * NOTE: the only reason to use this is so that
 * we can call fb_done() before exiting - this
 * may or may not be necessary
 */
inter()
{
    quit = TRUE;
}

/*
 * function for trapping <ctrl-Z>
 * NOTE: this is used so that the user may stop the
 * animation at any time and dump the color map to
 * a file
 */
suspend()
{
    stop = TRUE;
}

/*
 * produces linear ramps in intensity for the red portion of the color
 * lookup table
 */

dda_red()
{
    register int    temp;
    static float    ry1, ry2=0., rinc, r_xsteps=0.;
    static int      r_xcount=0;

    /* if at end of ramp... */

```

```
    if (r_xcount >= (int) r_xsteps) {
        /*
         * make the end of last ramp the beginning of next ramp
         */
        ry1 = ry2;
        /* define end of next ramp */
        seed = RANDOM(seed);
        /* assign a new (scaled) end point */
        ry2 = MAXINDEX * (seed / 65535.0);
        seed = RANDOM(seed);
        /* get a new ramp length */
        r_xsteps = (maxsteps * (seed / 65535.0));
        /* find the intensity increment per step */
        if (r_xsteps != 0)
            rinc = (ry2 - ry1) / r_xsteps;
        else
            rinc = 0;
        r_xcount = 0;
    }
    temp = (int) ry1;
    ry1 += rinc;
    r_xcount++;
    return temp;
} /* dda_red() */

/*
 * produces linear ramps in intensity for the green portion of the color
 * lookup table
 */

dda_green()
{
    register int    temp;
    static float    gy1, gy2=0., ginc, g_xsteps=0.;
    static int      g_xcount=0;

    if (g_xcount >= (int) g_xsteps) {
        /*
         * make the end of last ramp the beginning of next ramp
         */
        gy1 = gy2;
        /* define end of next ramp */
        seed = RANDOM(seed);
        gy2 = MAXINDEX * (seed / 65535.0);
        seed = RANDOM(seed);
        g_xsteps = (maxsteps * (seed / 65535.0));
        /* find the intensity increment per step */
        if (g_xsteps != 0)
            ginc = (gy2 - gy1) / g_xsteps;
        else
            ginc = 0;
        g_xcount = 0;
    }
    temp = (int) gy1;
    gy1 += ginc;
    g_xcount++;
    return temp;
} /* dda_green() */

/*
```

```
* produces linear ramps in intensity for the blue portion of the color
* lookup table
*/
```

```
dda_blue()
{

    register int    temp;
    static float    by1, by2=0., binc, b_xsteps=0.;
    static int      b_xcount=0;

    if (b_xcount >= (int) b_xsteps) {
        /*
         * make the end of last ramp the beginning of next ramp
         */
        by1 = by2;
        /* define end of next ramp */
        seed = RANDOM(seed);
        by2 = MAXINDEX * (seed / 65535.0);
        seed = RANDOM(seed);
        b_xsteps = (maxsteps * (seed / 65535.0));
        /* find the intensity increment per step */
        if (b_xsteps != 0)
            binc = (by2 - by1) / b_xsteps;
        else
            binc = 0;
        b_xcount = 0;
    }
    temp = (int) by1;
    by1 += binc;
    b_xcount++;
    return temp;
} /* dda_blue() */
```

```
/*
 * save the lookup table to a file
 */
```

```
save_lut()
{
    FILE          *fp, *fopen();
    char          filename[40];
    int           i;

    getchar();          /* read leading newline char */
    printf("Enter filename for lookup table: ");
    gets(filename);
    fp = fopen(filename, "w");

    for (i=0; i<MAXINDEX; i++)
        fprintf(fp, "%3d %3d %3d\n", r[i], g[i], b[i]);

    fclose(fp);
} /* save_lut() */
```



```
/*
 * Panoramic virtual screen implementation code fragment.
 *
 * Copyright (C) 1991, F. Kenton Musgrave
 * All rights reserved.
 *
 * This code is an extension of Rayshade 4.0
 * Copyright (C) 1989, 1991, Craig E. Kolb, Rod G. Bogart
 * All rights reserved.
 */

/* Various declarations and vector routines are a part of Rayshade; most are
 * easy enough to do yourself.
 */

RSViewing()
{
    Float magnitude;
    RSMatrix trans;
    Vector eyeOffset;
    int x, y;

    VecSub(Camera.lookp, Camera.pos, &Camera.dir);
    Screen.firstray = Camera.dir;

    Camera.lookdist = VecNormalize(&Camera.dir);
    if (VecNormCross(&Camera.dir, &Camera.up, &Screen.scrni) == 0.)
        RLError(RL_PANIC,
                "The view and up directions are identical?\n");
    (void)VecNormCross(&Screen.scrni, &Camera.dir, &Screen.scrnj);

    /* construct screen "x" (horizontal) direction vector */
    if (!Options.panorama) {
        /* standard virtual screen setup */
        magnitude = 2.*Camera.lookdist * tan(deg2rad(0.5*Camera.hfov)) /
                    Screen.xres;
        VecScale(magnitude, Screen.scrni, &Screen.scrnx);
    } else {
        /* For panorama option, we need an array of screen "x" vectors
         * which we will build later in the code. At this point, we
         * just construct the required rotation matrix (rotations being
         * about the "up" vector) and point the "scrnx" vector to the
         * edge of the screen.
         */
        Camera.lookdist = 1.;
        magnitude = -deg2rad(Camera.hfov);
        Screen.hincr = magnitude / Screen.xres;
        Screen.scrnx = Camera.dir;
        RotationMatrix( Camera.up.x, Camera.up.y, Camera.up.z,
                        -0.5*magnitude, &trans );
        VecTransform( &Screen.scrnx, &trans );
    }

    /* construct screen "y" (vertical) direction vector */
    magnitude = 2.*Camera.lookdist * tan(deg2rad(0.5*Camera.vfov)) /
                Screen.yres;
    VecScale(magnitude, Screen.scrnj, &Screen.scrny);

    if (!Options.panorama) {
        /* Construct ray direction for standard virtual screen */
        Screen.firstray.x -= 0.5*(Screen.xres*Screen.scrnx.x +
```

```
Screen.yres*Screen.scrny.x);
Screen.firstray.y -= 0.5*(Screen.xres*Screen.scrnx.y +
                          Screen.yres*Screen.scrny.y);
Screen.firstray.z -= 0.5*(Screen.xres*Screen.scrnx.z +
                          Screen.yres*Screen.scrny.z);
} else {
/* Panorama option: requires that we allocate & fill horizontal
 * and vertical direction arrays.
 */
Screen.horizdir = (Vector *)Malloc((Screen.xres+1) *
                                   sizeof(Vector));

/* Set eye separation for stereo rendering */
if (Options.stereo) {
    if (Options.eyesep == UNSET)
        RLError(RL_PANIC,
                "No eye separation specified.\n");
    Screen.eyepoints = (Vector *)Malloc((Screen.xres+1) *
                                         sizeof(Vector));

    if (Options.stereo == LEFT)
        magnitude = .5 * Options.eyesep;
    else
        magnitude = -.5 * Options.eyesep;
    eyeOffset.x = magnitude * Screen.scrni.x;
    eyeOffset.y = magnitude * Screen.scrni.y;
    eyeOffset.z = magnitude * Screen.scrni.z;
}

/* Fill the array of horizontal directions and, if stereo
 * rendering, eyepoints.
 * The horizontal ("x") direction array contains rotations of
 * "scrnx". Each entry requires construction of an appropriate
 * rotation matrix; rotation again being around the "up" vector.
 */
for ( x=0; x<=Screen.xres; x++ ) {
    Screen.horizdir[x] = Screen.scrnx;
    RotationMatrix( Camera.up.x, Camera.up.y, Camera.up.z,
                   x*Screen.hincr, &trans );
    VecTransform( &Screen.horizdir[x], &trans );
    /* Offset the eyepoints for stereo panorama */
    if (Options.stereo) {
        Screen.eyepoints[x] = eyeOffset;
        VecTransform( &Screen.eyepoints[x], &trans );
        VecAdd( Screen.eyepoints[x], Camera.pos,
               &Screen.eyepoints[x] );
    }
}

/* The vertical ("y") array varies as the tangent of "scrny". */
Screen.vertdir = (Vector *)Malloc((Screen.yres+1) *
                                   sizeof(Vector));
for ( y=0; y<=Screen.yres; y++ ) {
    Screen.vertdir[y] = Screen.scrny;
    magnitude = 0.5*Camera.vfov -
               Camera.vfov * ((Float)y/Screen.yres);
    magnitude = tan(deg2rad(magnitude));
    VecScale(-magnitude, Screen.scrnj, &Screen.vertdir[y]);
}
}

} /* RSViewing() */
```



```
SampleScreen(x, y, ray, color)
Float x, y;          /* Screen position to sample */
Ray *ray;            /* ray, with origin and medium properly set */
Pixel *color;        /* resulting color */
{
    Float dist;
    HitList hitlist;
    Color ctmp, fullintens;
    extern void ShadeRay();
    int ix, iy;

    /*
     * Calculate ray direction.
     */
    Stats.EyeRays++;
    if (Options.panorama) {
        /* Construct ray direction from vectors in tables,
         * using linear interpolation for jittering.
         */
        ix = (int)x;
        iy = (int)y;
        if (Options.stereo)
            ray->origin = Screen.eyepoints[ix];
        ray->dir.x = Screen.horizdir[ix].x +
            (Screen.horizdir[ix+1].x - Screen.horizdir[ix].x)
            * (x-ix) +
            Screen.vertdir[iy].x +
            (Screen.horizdir[iy+1].x - Screen.horizdir[iy].x)
            * (y-iy);
        ray->dir.y = Screen.horizdir[ix].y +
            (Screen.horizdir[ix+1].y - Screen.horizdir[ix].y)
            * (x-ix) +
            Screen.vertdir[iy].y +
            (Screen.horizdir[iy+1].y - Screen.horizdir[iy].y)
            * (y-iy);
        ray->dir.z = Screen.horizdir[ix].z +
            (Screen.horizdir[ix+1].z - Screen.horizdir[ix].z)
            * (x-ix) +
            Screen.vertdir[iy].z +
            (Screen.horizdir[iy+1].z - Screen.horizdir[iy].z)
            * (y-iy);
    } else {
        ray->dir.x = Screen.firstray.x + x*Screen.scrnx.x +
            y*Screen.scrny.x;
        ray->dir.y = Screen.firstray.y + x*Screen.scrnx.y +
            y*Screen.scrny.y;
        ray->dir.z = Screen.firstray.z + x*Screen.scrnx.z +
            y*Screen.scrny.z;
    }

    (void)VecNormalize(&ray->dir);






















    /*
     * Do the actual ray trace.
     */
    fullintens.r = fullintens.g = fullintens.b = 1.;
    dist = FAR_AWAY;
    hitlist.nodes = 0;
    (void)TraceRay(ray, &hitlist, EPSILON, &dist);
}
```



```
    ShadeRay(&hitlist, ray, dist, &Screen.background, &ctmp, &fullintens);  
    color->r = ctot.o;  
    color->g = ctmp.g;  
    color->b = ctmp.b;
```

```
} /* SampleScreen() */
```

# Index of

## /pubs/tog/GraphicsGems/gemsv/ch5-5/

Name	Last modified	Size	Description
 <a href="#">Parent Directory</a>			
 <a href="#">camera.cxx</a>	29-Jun-00 08:23	1K	
 <a href="#">csg.cxx</a>	29-Jun-00 08:23	1K	
 <a href="#">finish.cxx</a>	29-Jun-00 08:23	1K	
 <a href="#">finite.cxx</a>	29-Jun-00 08:23	1K	
 <a href="#">global.h</a>	29-Jun-00 08:23	33K	
 <a href="#">infinite.cxx</a>	29-Jun-00 08:23	1K	
 <a href="#">lightsrc.cxx</a>	29-Jun-00 08:23	1K	
 <a href="#">main.cxx</a>	29-Jun-00 08:23	7K	
 <a href="#">makefile</a>	29-Jun-00 08:23	1K	
 <a href="#">misc.cxx</a>	29-Jun-00 08:23	1K	
 <a href="#">normal.cxx</a>	29-Jun-00 08:23	1K	
 <a href="#">patch.cxx</a>	29-Jun-00 08:23	1K	
 <a href="#">pigment.cxx</a>	29-Jun-00 08:23	1K	
 <a href="#">pov.1</a>	29-Jun-00 08:23	10K	
 <a href="#">pov.y</a>	29-Jun-00 08:24	22K	
 <a href="#">random.pov</a>	29-Jun-00 08:24	9K	
 <a href="#">readme.txt</a>	29-Jun-00 08:24	2K	
 <a href="#">show.c</a>	29-Jun-00 08:24	3K	
 <a href="#">show.m</a>	29-Jun-00 08:24	1K	
 <a href="#">test.pov</a>	29-Jun-00 08:24	1K	

 <a href="#">__texture.cxx</a>	29-Jun-00 08:24	1K
 <a href="#">__voronoi.cxx</a>	29-Jun-00 08:24	8K
 <a href="#">__voronoi.h</a>	29-Jun-00 08:24	15K

```
#include <iostream.h>

#include "global.h"

ostream& operator<<(ostream& o, camera& c) {
    o<<"camera {\n";
    o<<"location "<<c.l<<"\n";
    o<<"sky "<<c.s<<"\n";
    o<<"direction "<<c.d<<"\n";
    o<<"up "<<c.u<<"\n";
    o<<"right "<<c.r<<"\n";
    o<<"look at "<<c.a<<"\n";
    o<<c.T;
    o<<"}\n";
    return o;
}

camera actcamera;
```

```
#include <iostream.h>
```

```
#include "global.h"
```

```
ostream& operator<<(ostream& o, csg& c) {c.out(o); return o;}  
ostream& operator<<(ostream& o, csguni& c) {c.out(o); return o;}  
ostream& operator<<(ostream& o, csgmer& c) {c.out(o); return o;}  
ostream& operator<<(ostream& o, csgint& c) {c.out(o); return o;}  
ostream& operator<<(ostream& o, csgdif& c) {c.out(o); return o;}
```

```
#include <iostream.h>
#include <math.h>
#include "global.h"

ostream& operator<<(ostream& o, finish& f) {
    o<<"finish {\n";
    o<<"diffuse "<<f.kd<<" "<<"brilliance "<<f.kb<<"\n";
    o<<"crand "<<f.kc<<" "<<"ambient "<<f.ka<<"\n";
    o<<"reflection "<<f.kr<<"\n";
    o<<"phong "<<f.ks<<" "<<"phong_size "<<f.kp<<"\n";
    o<<"specular "<<f.kh<<" "<<"roughness "<<f.kg<<"\n";
    if(f.km) o<<"metallic\n";
    o<<"refraction "<<f.kt<<" "<<"ior "<<f.ki<<"\n";
    o<<"}\n";
    return o;
}

extern vector norm(vector& v);

intensity finish::surfoptics(color& c, vector& n, ray& r, intersect& i){
    intensity I(this->ka, this->ka, this->ka);
    for(lightsource *ls=lightsources.first(); ls; ls=lightsources.next()) {
        list<light*>* L=ls->illum(i);
        for(light *l=L->first(); l; l=L->next()) {
            double f=n%l->v;
            vector h=norm(l->v+(-r.d));
            double j=n%h;
            double g=pow(j,this->kp);
            if(f>0.)
                I+=c*l->i*(this->kd*f+ this->ks*g);
            delete l;
        }
        delete L;
    }
    if(this->kr) {
        vector d=norm(r.d-n*((n%r.d)*2));
        ray rr(i.p,d,r.l+1,2*r.c+1);
        I+=trace(rr)*this->kr ;
    }
    if(this->kt) {
        vector l=-r.d;
        double ci=l%n;
        double ct=1+(this->ki*this->ki)*(ci*ci-1);
        if(ct>=0.) {
            vector d=norm(r.d*this->ki+n*(this->ki*ci-sqrt(ct)));
            ray rt(i.p,d,r.l+1,2*r.c+2);
            I+=trace(rt)*this->kt;
        }
    }
    return I;
}
```

```
#include <iostream.h>
#include <math.h>

#include "global.h"

ostream& operator<<(ostream& o, sphere& s) {s.out(o); return o;}

list<intersect*> *sphere::shape(ray& r) {
    list<intersect*> *l=new list<intersect*>;
    double b=r.d%(r.o-this->c);
    double C=(r.o-this->c)%(r.o-this->c)-this->r*this->r;
    double d=b*b-C;
    if(d<0.) return l;
    d=sqrt(d); double t1=(-b-d); double t2=(-b+d);
    if(t1>EPS) {
        intersect* i=new intersect; i->t=t1; i->p=r.o+r.d*t1;
        i->n=norm(i->p-this->c); i->o=this;
        *l+=i;
    }
    if(t2>EPS) {
        intersect* i=new intersect; i->t=t2; i->p=r.o+r.d*t2;
        i->n=norm(i->p-this->c); i->o=this;
        *l+=i;
    }
    return l;
}
```



```
#include <math.h>
#include <stdlib.h>

//      NUMERIC CONSTANTS AND RELATED THINGS

const double Pi=3.1415926535;
inline double RAD(double r) {return Pi*r/180.;}      // DEG-TO-RAD CONVERSION
const double EPS=1e-4;                               // VERY SMALL NUMBER

//      LIST TEMPLATE

template <class T> struct listelem {
    T o;                                              // OBJECT
    listelem<T> *n;                                  // NEXT ELEMENT
    listelem(T o) {this->o=o; n=(listelem<T>*)0;}
};

template <class T> class list {
    listelem<T> *h, *t, *a;                          // HEAD, TAIL, ACTUAL
public:
    list() {h=t=(listelem<T>*)0;}
    ~list() {for(;h;a=h) {a=h->n; delete h;}}
    void operator+=(T o) {                            // INSERT INTO LIST
        listelem<T> *e=new listelem<T>(o);
        if(t) {t->n=e; t=e;} else {h=e; t=e;}
    }
    T first() {a=h; return a?a->o:(T)0;}
    T next() {a=a->n; return a?a->o:(T)0;}
    int operator[](T o) {                             // o ON LIST?
        for(listelem<T>*e=h;e;e=e->n) if(e->o==o) return 1;
        return 0;
    }
};

//      DECLARATION TEMPLATE (SIMILAR TO LIST BUT MAY BE ENHANCED)

template <class T> struct declaration {
    char *i;                                          // IDENTIFIER
    T *p;                                            // OBJECT
    declaration<T> *n;                              // NEXT
    declaration(char* i, T* p) {this->i=i; this->p=p;}
    ~declaration() {delete i; delete p;}
};

template <class T> class declarations {
    declaration<T> *h;                              // HEAD
public:
    declarations() {h=(declaration<T>*)0;}
    ~declarations() {
        for(declaration<T>* a; h; h=a) {a=h->n; delete h;}
    }
    void operator+=(declaration<T>* d) {d->n=h; h=d;}      // INSERT
    T* operator[](char* i) {                          // GET
        for(declaration<T>* d=h; d; d=d->n)
            if(strcmp(d->i,i)==0)
                return d->p;
        cout<<"identifier "<<i<<" not found\n";
        exit(1);
        return (T*)0;
    }
};
```

```
//      VECTORS AND MATRICES

class xform;

class vector {
    friend ostream& operator<<(ostream& o, vector& v);
    friend vector operator-(vector& v);
    friend class xform;
    double x, y, z;
public:
    vector() {x=0.; y=0.; z=0.;}
    vector(double x, double y, double z) {
        this->x=x; this->y=y; this->z=z;
    }
    vector operator+(vector& v) {return vector(x+v.x,y+v.y,z+v.z);}
    vector operator-(vector& v) {return vector(x-v.x,y-v.y,z-v.z);}
    vector operator*(vector& v) {return vector(x*v.x,y*v.y,z*v.z);}
    vector operator/(vector& v) {return vector(x/v.x,y/v.y,z/v.z);}
    vector operator+(double d) {return vector(x+d,y+d,z+d);}
    vector operator-(double d) {return vector(x-d,y-d,z-d);}
    vector operator*(double d) {return vector(x*d,y*d,z*d);}
    vector operator/(double d) {return vector(x/d,y/d,z/d);}
    double operator%(vector& v) {return x*v.x+y*v.y+z*v.z;}
    operator double() {return y;}
    double operator[](int i) {return i==0?this->x:i==1?this->y:this->z;}
    double get_x() { return this->x;}
    double get_y() { return this->y;}
    double get_z() { return this->z;}
};

extern vector norm(vector& v);

struct matrix {
    double m11, m12, m13;           // 1ST ROW
    double m21, m22, m23;           // 2ND ROW
    double m31, m32, m33;           // 3RD ROW
    matrix(double m11=1., double m12=0., double m13=0.,
            double m21=0., double m22=1., double m23=0.,
            double m31=0., double m32=0., double m33=1.) {
        this->m11=m11; this->m12=m12; this->m13=m13;
        this->m21=m21; this->m22=m22; this->m23=m23;
        this->m31=m31; this->m32=m32; this->m33=m33;
    }
    matrix operator*(matrix& m) {
        return matrix(
            m11*m.m11+m12*m.m21+m13*m.m31,
            m11*m.m12+m12*m.m22+m13*m.m32,
            m11*m.m13+m12*m.m23+m13*m.m33,

            m21*m.m11+m22*m.m21+m23*m.m31,
            m21*m.m12+m22*m.m22+m23*m.m32,
            m21*m.m13+m22*m.m23+m23*m.m33,

            m31*m.m11+m32*m.m21+m33*m.m31,
            m31*m.m12+m32*m.m22+m33*m.m32,
            m31*m.m13+m32*m.m23+m33*m.m33
        );
    }
    vector operator*(vector& v) {
        return vector(

```

```
        vector(m11,m12,m13)%v,
        vector(m21,m22,m23)%v,
        vector(m31,m32,m33)%v
    );
}
vector operator/(vector& v) {                // INVERSE
    return vector(
        v%vector(m11,m21,m31),
        v%vector(m12,m22,m32),
        v%vector(m13,m23,m33)
    );
}
};

//      RAYS AND INTERSECTIONS

struct ray {
    vector o;                                // ORIGIN
    vector d;                                // DIRECTION
    int l;                                    // LEVEL
    int c;                                    // CODE
    ray(vector& o, vector& d, int l, int c)
        {this->o=o; this->d=d; this->l=l; this->c=c;}
};

class object;

struct intersect {
    double t;                                // RAY PARAMETER
    vector p;                                // SURFACE POINT
    vector n;                                // SURFACE NORMAL
    object *o;                                // OBJECT SURFACE
    intersect() {o=(object*)0;}
    intersect(double t, vector& p, vector& n, object *o) {
        this->t=t; this->p=p; this->n=n; this->o=o;
    }
};

//      HALF-SPACES

struct halfspace {
    vector n;                                // NORMAL
    double d;                                // DISTANCE FROM ORIGIN
    halfspace()                               // DEFAULT CONSTRUCTOR
        {this->n=vector(0.,0.,1.); this->d=0.;}
    halfspace(vector& n, double d)             // USUAL DEFINITION
        {this->n=norm(n); this->d=d;}
    halfspace(vector& u, vector& v)            // BISECTOR PLANE
        {n=norm(v-u); d=((u+v)/2.)%n;}
    int operator&(vector& p) {return n%p<=d;} // IF CONTAINS POINT p
};

//      GEOMETRIC TRANSFORMATIONS (RIGID MOTIONS)

class xform {
    friend ostream& operator<<(ostream& o, xform& T);
    matrix A;                                // ORTHONORMAL ORIENTATION
    vector s;                                // DIAGONAL SCALE
    vector t;                                // TRANSLATION VECTOR
public:
    xform() {s=vector(1.,1.,1.); t=vector(0.,0.,0.);}
```

```

void operator+=(vector& t) {this->t=this->t+t;} // TRANSLATE
void operator*=(vector& s) { // SCALE
    this->s=this->s*s; this->t=s*this->t;
}
void operator<=(vector& r) { // ROTATE
    double sa=sin(RAD(r.x)), ca=cos(RAD(r.x));
    double sb=sin(RAD(r.y)), cb=cos(RAD(r.y));
    double sc=sin(RAD(r.z)), cc=cos(RAD(r.z));
    matrix Ap=matrix(cc,-sc,0., sc,cc,0., 0.,0.,1.)*
        (matrix(cb,0.,sb, 0.,1.,0., -sb,0.,cb)*
        matrix(1.,0.,0., 0.,ca,-sa, 0.,sa,ca));
    matrix B(
        s.x*Ap.m11, s.y*Ap.m12, s.z*Ap.m13,
        s.x*Ap.m21, s.y*Ap.m22, s.z*Ap.m23,
        s.x*Ap.m31, s.y*Ap.m32, s.z*Ap.m33
    );
    s=vector(
        sqrt(B.m11*B.m11+B.m12*B.m12+B.m13*B.m13),
        sqrt(B.m21*B.m21+B.m22*B.m22+B.m23*B.m23),
        sqrt(B.m31*B.m31+B.m32*B.m32+B.m33*B.m33)
    );
    A=matrix(
        B.m11/s.x, B.m12/s.x, B.m13/s.x,
        B.m21/s.y, B.m22/s.y, B.m23/s.y,
        B.m31/s.z, B.m32/s.z, B.m33/s.z
    );
    t=Ap*t;
}
vector operator*(vector& v) {return s*(A*v)+t;} // TRANSFORM
vector operator/(vector& v) {return (A/(v-t))/s;} // INVERSE
vector operator<<(vector& v) {return s*(A*v);} // ROTATE
vector operator>>(vector& v) {return (A/v)/s;} // INVERSE
ray operator*(ray& r)
    {return ray((*this)*r.o,(*this)<<r.d,r.l,r.c);}
ray operator/(ray& r)
    {return ray((*this)/r.o,(*this)>>r.d,r.l,r.c);}
intersect operator*(intersect& i)
    {return intersect(i.t,(*this)*i.p,norm((*this)<<i.n),i.o);}
list<intersect*> *operator*(list<intersect*> *l) {
    for(intersect* i=l->first(); i; i=l->next())
        *i=(*this)*(*i);
    return l;
}
halfspace operator*(halfspace& h) {
    vector p=(*this)*(h.n*h.d); vector n=(*this)<<h.n;
    return halfspace(n,p%n);
}
halfspace operator/(halfspace& h) {
    vector p=(*this)/(h.n*h.d); vector n=(*this)>>h.n;
    return halfspace(n,p%n);
}
};

// CAMERA MODEL

class camera {
    friend ostream& operator<<(ostream& o, camera& c);
    vector l; // LOCATION
    vector s; // SKY
    vector d; // DIRECTION
    vector u; // UP

```

```
vector r; // RIGHT
vector a; // LOOK_AT
xform T; // TRANSFORMATION

public:
    camera() {
        l=vector(0.,0.,0.); s=vector(0.,1.,0.); d=vector(0.,0.,1.);
        u=vector(0.,1.,0.); r=vector(1.33,0.,0.); a=vector(0.,0.,1.);
    }
    void setl(vector& v) {l=v;}
    void sets(vector& v) {s=v;}
    void setd(vector& v) {d=v;}
    void setu(vector& v) {u=v;}
    void setr(vector& v) {r=v;}
    void seta(vector& v) {a=v;}
    void traT(vector& v) {T+=v;}
    void rotT(vector& v) {T<=v;}
    void scaT(vector& v) {T*=v;}
    ray getray(double x, double y) {
        return this->T*ray(l,norm((r*x)+(u*y)+d),0,0);
    }
};

// TEXTURES - I. PIGMENT

class icolor {
    unsigned char r, g, b; // RED, GREEN, BLUE
public:
    icolor(unsigned char r=0, unsigned char g=0, unsigned char b=0) {
        this->r=r; this->g=g; this->b=b;
    }
};

class color;

class intensity {
    friend ostream& operator<<(ostream& o, intensity& i);
    friend class color;
    double r, g, b; // RED, GREEN, BLUE
public:
    intensity(double r=0., double g=0., double b=0.) {
        this->r=r; this->g=g; this->b=b;
    }
    intensity operator*(double d) {return intensity(r*d,g*d,b*d);}
    intensity operator+(intensity& i)
        {return intensity(r+i.r,g+i.g,b+i.b);}
    void operator+=(intensity& i) {r+=i.r; g+=i.g; b+=i.b;}
    operator icolor() {
        unsigned char ir=r<1.?(unsigned char)(r*255.):255;
        unsigned char ig=g<1.?(unsigned char)(g*255.):255;
        unsigned char ib=b<1.?(unsigned char)(b*255.):255;
        return icolor(ir, ig, ib);
    }
};

class color {
    friend ostream& operator<<(ostream& o, color& c);
    double r, g, b; // RED, GREEN, BLUE
    double f; // FILTER
public:
    color(double r=0., double g=0., double b=0., double f=0.) {
        this->r=r; this->g=g; this->b=b; this->f=f;
    }
};
```

```
    }
    void setr(double d) {r=d;}
    void setg(double d) {g=d;}
    void setb(double d) {b=d;}
    void setf(double d) {f=d;}
    double filter() {return f;}
    color operator*(color& c){return color(r*c.r,g*c.g,b*c.b,f*c.f);}
    operator intensity() {return intensity(r,g,b);}
    intensity operator*(intensity& i)
        {return intensity(r*i.r,g*i.g,b*i.b);}
};

class finish;

class fog {
};

class pigment {
    friend ostream& operator<<(ostream& o, pigment& p);
protected:
    color q; // QUICK COLOR
    vector t; // TURBULENCE
    int o; // OCTAVES
    double m; // OMEGA
    double l; // LAMBDA
    double f, p; // FREQUENCY, PHASE
    xform T; // TRANSFORMATION
public:
    virtual void out(ostream& o) {}
    pigment() {t=vector(0.,0.,0.); o=6; m=.5; l=2.; f=1.; p=0.;}
    virtual ~pigment() {}
    void operator|=(pigment& p) {
        t=p.t; o=p.o; m=p.m; l=p.l; f=p.f; this->p=p.p; T=p.T;
    }
    void setq(color& q) {this->q=q;}
    void sett(vector& t) {this->t=t;}
    void seto(int o) {this->o=o;}
    void setm(double m) {this->m=m;}
    void setl(double l) {this->l=l;}
    void setf(double f) {this->f=f;}
    void setp(double p) {this->p=p;}
    void traT(vector& v) {T+=v;}
    void rotT(vector& v) {T<=v;}
    void scaT(vector& v) {T*=v;}
    virtual pigment* copy() {pigment *p=new pigment; *p=*this; return p;}
    virtual color paint(vector& p) {return q;}
    color surfcolor(vector& p) {return paint(T/p);}
};

class solid : public pigment {
    color c;
public:
    solid() {c=color(1.,1.,1.); setq(c);}
    solid(color& c) {this->c=c; setq(c);}
    void out(ostream& o) {o<<"color "<<c<<"\n";}
    pigment* copy() {solid *p=new solid; *p=*this; return p;}
    color paint(vector& p) {return c;}
};

class checker : public pigment {
    color c1, c2;
```

```
public:
    checker() {c1=color(0.,0.,1.,0.); c2=color(0.,1.,0.,0.);}
    checker(color& c1) {this->c1=c1; c2=color(0.,1.,0.,0.);}
    checker(color& c1, color& c2) {this->c1=c1; this->c2=c2;}
    void out(ostream& o) {o<<"checker color "<<c1<<" color "<<c2<<"\n";}
    pigment* copy() {checker *p=new checker; *p=*this; return p;}
    color paint(vector& p)
        {return (((int)p[0]+(int)p[1]+(int)p[2])%2)?c2:c1;}
};

class hexagon : public pigment {
    color c1, c2, c3;
public:
    hexagon() {
        c1=color(0.,0.,1.,0.); c2=color(0.,1.,0.,0.);
        c3=color(1.,0.,0.,0.);
    }
    hexagon(color& c1)
        {this->c1=c1; c2=color(0.,1.,0.,0.); c3=color(1.,0.,0.,0.);}
    hexagon(color& c1, color& c2)
        {this->c1=c1; this->c2=c2; c3=color(1.,0.,0.,0.);}
    hexagon(color& c1, color& c2, color& c3)
        {this->c1=c1; this->c2=c2; this->c3=c3;}
    pigment* copy() {hexagon *p=new hexagon; *p=*this; return p;}
    color paint(vector& p) {return c1;}
};

struct colormapitem {
    double b, e; // BEGIN, END VALUES
    color cb, ce; // BEGIN, END COLORS
    colormapitem() {b=0.; e=1.; cb=color(0.,0.,0.); ce=color(1.,1.,1.);}
    colormapitem(double b, double e, color& cb, color& ce)
        {this->b=b; this->e=e; this->cb=cb; this->ce=ce;}
};

class colormap {
    friend ostream& operator<<(ostream& o, colormap& m);
    friend class colormapped;
    colormapitem *I;
    int nI;
public:
    colormap() {I=new colormapitem; nI=1;}
    virtual ~colormap() {delete I;}
    colormap& operator=(colormap& m) {
        delete I; I=new colormapitem[m.nI];
        for(register int i=0; i<m.nI; i++) I[i]=m.I[i]; nI=m.nI;
        return *this;
    }
    void operator+=(colormapitem& J) {
        colormapitem *K=new colormapitem[nI+1];
        for(register int i=0; i<nI; i++) K[i]=I[i];
        K[nI]=J; delete I; I=K; nI=nI+1;
    }
    colormap* copy() {colormap*p=new colormap; *p=*this; return p;}
};

class colormapped : public pigment {
    friend ostream& operator<<(ostream& o, colormapped& m);
public:
    colormap cm;
    colormapped() {}
};
```

```
    virtual ~colormapped() {}
    pigment* copy() {colormapped*p=new colormapped; *p=*this; return p;}
};

class gradient : public colormapped {
};

class marble : public colormapped {
};

class wood : public colormapped {
};

class onion : public colormapped {
};

class leopard : public colormapped {
};

class granite : public colormapped {
};

class bozo : public colormapped {
};

class spotted : public colormapped {
};

class agate : public colormapped {
};

class mandel : public colormapped {
};

class radial : public colormapped {
};

//      TEXTURES - II. NORMAL

class normal {
    friend ostream& operator<<(ostream& o, normal& n);
    vector t;                                // TURBULENCE
    double f, p;                             // FREQUENCY, PHASE
    xform T;                                 // TRANSFORMATION
public:
    virtual void out(ostream& o) {}
    normal() {t=vector(0.,0.,0.); f=1.; p=0.;}
    virtual ~normal() {}
    void operator|=(normal& n) {
        t=n.t; f=n.f; p=n.p; T=n.T;
    }
    void sett(vector& t) {this->t=t;}
    void setf(double f) {this->f=f;}
    void setp(double p) {this->p=p;}
    void traT(vector& v) {T+=v;}
    void rotT(vector& v) {T<=v;}
    void scaT(vector& v) {T*=v;}
    virtual normal* copy() {normal *n=new normal; *n=*this; return n;}
    virtual vector perturbate(vector& n,vector& p) {return n;}
    vector surfnormal(vector& n,vector& p) {
        return norm(T<<perturbate(norm(T>>n),T*p));
    }
};
```



```
    }
};

class bumps : public normal {
    double p;                                // PERTURBATION
public:
    void out(ostream& o) {o<<"bumps "<<p<<"\n";}
    bumps() {p=.4;}
    bumps(double d) {p=d;}
    normal* copy() {bumps *n=new bumps; *n=*this; return n;}
    vector perturbate(vector& n,vector& p) {
        double theta,phi,nx,ny,nz;
        theta=acos(n.get_z())+RAD((double)rand()*this->p);
        phi=atan(n.get_y()/n.get_x())+RAD((double)rand()*this->p);
        /*double k=RAD(p*p);
        theta=acos(n.get_z())+RAD(sin(k)*this->p);
        phi=atan(n.get_y()/n.get_x())+RAD(cos(k)*this->p);*/
        nz=cos(theta);
        nx=sqrt((1-nz*nz)/(1+tan(phi)*tan(phi)));
        ny=nx*tan(phi);
        n=vector(nx,ny,nz);
        return n;
    }
};

class dents : public normal {
    double d;                                // DENTING
public:
    void out(ostream& o) {o<<"dents "<<d<<"\n";}
    dents() {d=.4;}
    dents(double d) {this->d=d;}
    normal* copy() {dents *n=new dents; *n=*this; return n;}
    vector perturbate(vector& n,vector& p) {return n;}
};

class ripples : public normal {
};

class waves : public normal {
};

class wrinkles : public normal {
};

class bumpmap : public normal {
};

//      TEXTURES - III. FINISH

class finish {
    friend ostream& operator<<(ostream& o, finish& f);
    double kd, kb, kc;                        // DIFFUSE, BRILLIANCE, CRAND
    double ka;                                // AMBIENT
    double kr;                                // REFLECTION
    double ks, kp, kh, kg;                    // PHONG, SPECULAR HIGHLIGHTS
    int km;                                    // METALLIC
    double kt, ki;                            // REFRACTION, INDEX OF REFR.
public:
    finish() {
        kd=.6; kb=0.; kc=0.; ka=.1; kr=0.; ks=0.; kp=40.;
        kh=0.; kg=.05; km=0; kt=0.; ki=1.;
    }
};
```

```
    }
    void setkd(double d) {kd=d;}
    void setkb(double d) {kb=d;}
    void setkc(double d) {kc=d;}
    void setka(double d) {ka=d;}
    void setkr(double d) {kr=d;}
    void setks(double d) {ks=d;}
    void setkp(double d) {kp=d;}
    void setkh(double d) {kh=d;}
    void setkg(double d) {kg=d;}
    void setkm(int i) {km=i;}
    void setkt(double d) {kt=d;}
    void setki(double d) {ki=d;}
    finish* copy() {finish *f=new finish; *f=*this; return f;}
    intensity surfoptics(color& c, vector& n, ray& r, intersect& i);
};

//      TEXTURES - IV. TOP LEVEL

class texture {
    friend ostream& operator<<(ostream& o, texture& t);
    pigment *p;
    normal *n;
    finish *f;
protected:
    list<texture*> *l;
    xform T;
public:
    texture() {p=new solid;n=new normal;f=new finish;l=new list<texture*>;}
    virtual ~texture() {
        delete p; delete n; delete f;
        for(texture* t=l->first(); t; t=l->next()) delete t;
        delete l;
    }
    texture& operator=(texture& t) {
        delete p; p=t.p->copy();
        delete n; n=t.n->copy();
        delete f; f=t.f->copy();
        for(texture* x=l->first(); x; x=l->next()) delete x;
        delete l; l=new list<texture*>;
        for(texture* p=t.l->first(); p; p=t.l->next()) *l+=p->copy();
        return *this;
    }
    void operator+=(texture* t) {*l+=t;}
    virtual void out(ostream& o) {
        o<<"texture {\n";
        o<<*p; o<<*n; o<<*f;
        o<<"T<<"\n";
        for(texture* x=l->first(); x; x=l->next()) o<<*x;
    }
    void setp(pigment& p) {delete this->p; this->p=p.copy();}
    void setn(normal& n) {delete this->n; this->n=n.copy();}
    void setf(finish& f) {delete this->f; this->f=f.copy();}
    void traT(vector& v) {T+=v;}
    void rotT(vector& v) {T<=v;}
    void scaT(vector& v) {T*=v;}
    pigment* copyp() {return p->copy();}
    normal* copyn() {return n->copy();}
    finish* copyf() {return f->copy();}
    virtual texture* copy() {texture* t=new texture; *t=*this; return t;}
    virtual intensity shade(ray& r, intersect& i) {
```

```
        i.p=T/i.p;
        color c=p->surfcOLOR(i.p);
        vector v=n->surfnORMAL(i.n, i.p);
        return f->surfoptics(c, v, r, i);
    }
};

class tiles : public texture {
public:
    void out(ostream& o) {
        o<<"texture {\n";
        o<<"tiles\n"<<*(texture*)this<<"tile2\n"<<(1->first());
        o<<"T<<"\n";
    }
    texture* copy() {tiles* t=new tiles; *t=*this; return t;}
    intensity shade(ray& r, intersect& i)
        {return texture::shade(r,i);}
};

class materialmap : public texture {
};

//      OBJECTS IN GENERAL

class object {
protected:
    friend ostream& operator<<(ostream& o, object& p);
    texture t; // TEXTURE
    xform T; // TRANSFORMATION
    virtual list<intersect*> *shape(ray& r) // RAY INTERSECTION
        {return new list<intersect*>;}
public:
    void outopt(ostream& o) {o<<t<<"T;"}
    virtual void out(ostream& o) {o<<"undefined {\n";outopt(o);o<<"}\n";}
    virtual ~object() {}
    void sett(texture& t) {this->t=t;}
    void setp(pigment& p) {t.setp(p);}
    void setn(normal& n) {t.setn(n);}
    void setf(finish& f) {t.setf(f);}
    void traT(vector& v) {T+=v;}
    void rotT(vector& v) {T<=v;}
    void scaT(vector& v) {T*=v;}
    virtual object* copy() {object* o=new object; *o=*this; return o;}
    list<intersect*> *test(ray& r) {return T*shape(T/r);}
    intensity shade(ray& r, intersect& i) {return t.shade(r,i);}
    virtual list<vector*> *particles() {
        vector *v=new vector; *v=(this->T)*vector(0.,0.,0.);
        list<vector*> *l=new list<vector*>; *l+=v;
        return l;
    }
    virtual int operator&(halfspace& h) {return 1;} // INTERSECTS h
};

//      INTERSECTION ACCELERATION TECHNIQUES

class method {
    list<object*> *lo; // SCENE OBJECTS
    virtual list<object*> *firstlist(ray& r)
        {return lo;} // BRUTE-FORCE METHOD
    virtual list<object*> *nextlist()
        {return (list<object*>*)0;} // BRUTE-FORCE METHOD
```

```
public:
    method() {lo=new list<object*>;}
    virtual void preprocess(list<object*> *l)
        {delete lo; lo=l;} // NO PREPROCESSING
    intersect operator()(ray& r) { // GENERAL SCHEME
        intersect imin, *i;
        for(list<object*>*lo=firstlist(r); lo; lo=nextlist()) {
            for(object *o=lo->first(); o; o=lo->next()) {
                list<intersect*> *li=o->test(r);
                if(i=li->first()) {
                    if(!imin.o || i->t<imin.t )
                        imin=*i;
                }
                for(i=li->first(); i; i=li->next())
                    delete i;
                delete li;
            }
        }
        return imin;
    }
};

extern method* query; // IN FILE main

//      OBJECTS SPECIFIED - I. SOLID FINITE PRIMITIVES

class sphere : public object {
    vector c; // CENTER
    double r; // RADIUS
    list<intersect*> *shape(ray& r);
public:
    void out(ostream& o) {
        o<<"sphere {\n"<<c<<"\n"<<r<<"\n"; outopt(o); o<<"}\n";
    }
    friend ostream& operator<<(ostream& o, sphere& s);
    sphere() {c=vector(0.,0.,0.); r=1.;}
    sphere(vector& c, double r) {this->c=c;this->r=r;}
    object* copy() {sphere* o=new sphere; *o=*this; return o;}
    list<vector*> *particles() {
        vector *v=new vector; *v=T*c;
        list<vector*> *l=new list<vector*>; *l+=v;
        return l;
    }
    int operator&(halfspace& h)
        {halfspace H=T/h; return (c-H.n*(r+H.d))*H.n<=EPS;}
};

class box : public object {
};

class cone : public object {
};

class cylinder : public object {
};

class torus : public object {
};

class blob : public object {
};
```

```
//      OBJECTS SPECIFIED -   II. FINITE PATCH PRIMITIVES

//      OBJECTS SPECIFIED -   III. INFINITE SOLID PRIMITIVES

class plane : public object, public halfspace {
};

class quadric : public object {
};

//      OBJECTS SPECIFIED -   IV. CONSTRUCTIVE SOLID GEOMETRY (CSG)

class csg : public object {
    friend ostream& operator<<(ostream& o, csg& c);
protected:
    list<object*> *l;
public:
    csg(list<object*>*ol=(list<object*>*)0) {l=ol?ol:new list<object*>;}
    ~csg() {for(object* o=l->first();o;o=l->next()) delete o; delete l;}
    void outobj(ostream& o) {
        for(object *p=l->first();p;p=l->next()) o<<*p;
    }
    csg& operator=(csg& c) {
        object* o;
        for(o=l->first(); o; o=l->next()) delete o;
        delete l; l=new list<object*>;
        for(o=c.l->first(); o; o=c.l->next()) *l+=o->copy();
        return *this;
    }
    virtual object* copy() {csg* o=new csg; *o=*this; return o;}
    list<vector*> *particles() {
        list<vector*> *L=new list<vector*>;
        for(object*o=l->first();o;o=l->next()) {
            list<vector*> *M=o->particles();
            for(vector*v=M->first();v;v=M->next())
                {*v=T*(*v); *L+=v;}
            delete M;
        }
        return L;
    }
    int operator&(halfspace& h) {
        halfspace H=T/h;
        for(object*o=l->first();o;o=l->next()) if(*o&H) return 1;
        return 0;
    }
};

class csguni : public csg {
    friend ostream& operator<<(ostream& o, csguni& c);
public:
    csguni(list<object*>*ol=(list<object*>*)0) : csg(ol) {}
    ~csguni() {}
    void out(ostream& o) {
        o<<"union {\n"; outobj(o); outopt(o); o<<"}\n";
    }
    object* copy() {csguni* o=new csguni; *o=*this; return o;}
};
```

```
class csgmer : public csg {
    friend ostream& operator<<(ostream& o, csgmer& c);
public:
    csgmer(list<object*>*ol=(list<object*>*)0) : csg(ol) {}
    ~csgmer() {}
    void out(ostream& o) {
        o<<"merge {\n"; outobj(o); outopt(o); o<<"}\n";
    }
    object* copy() {csgmer* o=new csgmer; *o=*this; return o;}
};

class csgint : public csg {
    friend ostream& operator<<(ostream& o, csgint& c);
public:
    csgint(list<object*>*ol=(list<object*>*)0) : csg(ol) {}
    ~csgint() {}
    void out(ostream& o) {
        o<<"intersection {\n";outobj(o);outopt(o);o<<"}\n";
    }
    object* copy() {csgint* o=new csgint; *o=*this; return o;}
};

class csgdif : public csg {
    friend ostream& operator<<(ostream& o, csgdif& c);
public:
    csgdif(list<object*>*ol=(list<object*>*)0) : csg(ol) {}
    ~csgdif() {}
    void out(ostream& o) {
        o<<"difference {\n";outobj(o);outopt(o);o<<"}\n";
    }
    object* copy() {csgdif* o=new csgdif; *o=*this; return o;}
};

//      OBJECTS SPECIFIED -   V. LIGHT SOURCES

struct light {
    intensity i;
    vector v;
    light(intensity& i, vector& v) {this->i=i; this->v=v;}
};

class lightsource : public object {
    friend ostream& operator<<(ostream& o, lightsource& l);
protected:
    vector p;                                // POSITION
    color c;                                // COLOR
public:
    virtual void out(ostream& o) {
        o<<"light_source {\n"<<p<<"\n"<<c<<"\n"; o<<"T; o<<"}\n";
    }
    lightsource() {p=vector(0.,0.,0.); c=color(1.,1.,1.);}
    void setp(vector& p) {this->p=p;}
    void setc(color& c) {this->c=c;}
    object* copy() {lightsource* o=new lightsource; *o=*this; return o;}
    list<intersect*> *shape(ray& r) {return new list<intersect*>;}
    virtual list<light*>* illum(intersect& i) {
        list<light*> *L=new list<light*>;
        if((*query)(ray(this->p,norm(i.p-this->p),0,-1)).o!=i.o)
            return L;
        light *l=new light((intensity)c,norm(this->p-i.p));
    }
};
```

```
        *L+=1;
        return L;
    }
    list<vector*> *particles() {
        vector *v=new vector; *v=T*p;
        list<vector*> *l=new list<vector*>; *l+=v;
        return l;
    }
};

class spotlight : public lightsource {
    friend ostream& operator<<(ostream& o, spotlight& l);
    vector a; // POINT AT
    double r; // RADIUS
    double f; // FALLOFF
    double t; // TIGHTNESS
public:
    virtual void out(ostream& o) {
        o<<"light_source {\n"<<p<<"\n"<<c<<"\n";
        o<<"spotlight\npoint_at "<<a<<"\nradius "<<r<<"\n";
        o<<"falloff "<<f<<"\ntightness "<<t<<"\n";
        o<<T;
        o<<"}\n";
    }
    spotlight() {a=vector(0.,0.,0.); r=0.; f=0.; t=10;}
    spotlight(vector a, double r, double f, double t)
        {this->a=a; this->r=r; this->f=f; this->t=t;}
    void seta(vector& a) {this->a=a;}
    void setr(double r) {this->r=r;}
    void setf(double f) {this->f=f;}
    void sett(double t) {this->t=t;}
    object* copy() {spotlight* o=new spotlight; *o=*this; return o;}
};

//      ACCELERATION TECHNIQUES - I. VIA VORONOI-DIAGRAM

struct cell; // DECLARED IN voronoi.cxx

class voronoi : public method {
    cell *C; // CELL CLOSEST TO CENTROID
    cell **s; // ARRAY OF STARTING CELLS
    long traverse; // TRAVERSE CODE (disperse)
    void disperse(object *o, cell*C);
    cell* a; ray* r; double t; // USED BY step()
    void step(); // USED BY first-,nextlist()
    list<object*> *firstlist(ray& r);
    list<object*> *nextlist();
public:
    void preprocess(list<object*> *lo);
};

//      GLOBAL OBJECTS

extern int lmax; // MAXIMAL LEVEL OF RECURSION
extern int verbose; // PRINTOUTS
extern list<object*> objects; // SCENE OBJECTS
extern list<lightsource*> lightsources; // LIGHTSOURCES
extern camera actcamera; // CAMERA
extern intensity trace(ray& r); // RECURSIVE RAY TRACING
extern fog actfog; // FOG
```

```
#include <iostream.h>
#include <math.h>

#include "global.h"

static list<int> li;
```



```
#include <iostream.h>

#include "global.h"

ostream& operator<<(ostream& o, lightsource& l) {l.out(o); return o;}
ostream& operator<<(ostream& o, spotlight& l) {l.out(o); return o;}

list<lightsource*> lightsources;
```

```
#include <stdio.h>
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>

#include "global.h"

//      GLOBAL CONSTANTS

int verbose=0;                                // WHAT TO PRINT OUT

//      INPUT FILE CONTAINING RANDOM OBJECTS

char* randomfilename="random.pov";

int nrandom=0;                                // NUMBER OF RANDOM OBJECTS
double rho=.0001;                             // DEFAULT DENSITY OF RANDOM SPHERES

double random(double a, double b) {           // a<=random(a,b)<=b
//      double r=(double)rand()/(double)RAND_MAX;    // 0.<=r<=1.
    double r=drand48();                        // 0.<=r<=1.
    return a+r*(b-a);
}

void randomobjects(char* filename) {           // GENERATE RANDOM SPHERES
    ofstream o(filename, ios::out);
    o<<"#declare WHITE=color rgb<1,1,1>\n";
    o<<"#declare BLACK=color rgb<0,0,0>\n";
    o<<"#declare S=sphere{<0,0,0>,1 pigment{color WHITE}finish{diffuse 1}}\n";
    o<<"\n";
    o<<"camera{location<0,0,0> direction<0,0,1> right<1,0,0> up<0,1,0>}\n";
    o<<"light_source{<0,0,0> color WHITE}\n";
    o<<"light_source{<0,0,0> color WHITE}\n";
    o<<"background {color BLACK}\n";
    o<<"\n";
    double h=pow(nrandom/rho, 1./3.)/2.;        // HALF WIDTH OF SCENE CUBE
    for(register int i=0; i<nrandom; i++) {
        o<<"object{S "<<"translate<";
        o<<random(-h,h)<<" "<<random(-h,h)<<" "<<random(0.,2*h);
        o<<">}\n";
    }
}

//      EXTERNALS

extern FILE* inputfile;                       // DEFINED IN THE lex FILE
extern char* inputfilename;                   // DEFINED IN THE lex FILE
extern int yyparse(void);                     // GENERATED BY yacc FILE
extern int intensity background;              // DEFINED IN yacc FILE

//      DEBUGGING

#ifdef COUNTMEM
extern "C" {void* malloc(size_t); void free(void*);}
long nbytes=0L;
void *operator new(size_t size) {
    nbytes+=size;
    void *p=malloc(size+8); cout<<nbytes<<"\n";
    *(long*)p=size;
    return p+8;
}
#endif
```

```
void operator delete(void *p) {
    p-=8;
    nbytes-=*(long*)p;
    free(p);
}
#endif

//      PICTURE BUFFER

const int MAXRES=1024;
icolor buf[MAXRES];           // ONE ROW OF R,G,B BYTES
int xres=200, yres=200;       // ACTUAL RESOLUTION

//      OUTPUT FILE

char* outputfilename="picture"; // DEFAULT OUTPUT NAME

void outinit(int xres, int yres) { // INITIALIZE OUTPUT FILE
    FILE *f;
    if(!(f=fopen(outputfilename,"wb")))
        return;
    fwrite((void*)&xres, sizeof(int), 1, f);
    fwrite((void*)&yres, sizeof(int), 1, f);
    fclose(f);
}

void output(icolor b[]) { // OUTPUT ONE ROW
    FILE *f;
    if(!(f=fopen(outputfilename,"ab")))
        return;
    fwrite((void*)b, sizeof(icolor), xres, f);
    fclose(f);
}

//      INTERSECTION ACCELERATION METHODS

method* query;                // ACCELERATION METHOD

//      RECURSIVE RAY TRACING

int lmax=4;                    // MAXIMAL LEVEL OF RECURSION

intensity trace(ray& r) {
    if(r.l>lmax)
        return background;
    intersect i=(*query)(r);
    if(!i.o)
        return background;
    return i.o->shade(r,i);
}

//      MAIN PROGRAM

void usage() {
    cout<<"usage: oopov [SWITCHes] {FILENAME | -r #}\n";
    cout<<"where\n";
    cout<<"      SWITCH can be\n";
    cout<<"      -a v           :accelerate via Voronoi-diagram\n";
    cout<<"      -a b           :brute-force intersection (default)\n";
    cout<<"      -o FILENAME    :name of output file\n";
}
```

```
cout<<"          -r #                :creates # number of random objects\n";
cout<<"                        (also writes them into random.pov)\n";
cout<<"          -r -1              :reads objects from random.pov\n";
cout<<"          -v                  :verbose printout\n";
cout<<"          -x #                :horizontal resolution of image\n";
cout<<"          -y #                :vertical resolution of image\n";
cout<<"      # is an integer\n";
exit(0);
```

```
}
```

```
main(int argc, char** argv) {
    query=new method;
    while(argv[1] && argv[1][0]!='-') {
        switch(argv[1][1]) {
            case 'a':
                switch(argv[2][0]) {
                    case 'B': case 'b':
                        delete query;
                        query=new method;
                        break;
                    case 'V': case 'v':
                        delete query;
                        query=new voronoi;
                        break;
                }
                argc--; argv++;
                break;
            case 'o':
                outputfilename=argv[2];
                argc--; argv++;
                break;
            case 'r':
                nrandom=atoi(argv[2]);
                argc--; argv++;
                break;
            case 'v':
                verbose=1;
                break;
            case 'x':
                xres=atoi(argv[2]);
                argc--; argv++;
                break;
            case 'y':
                yres=atoi(argv[2]);
                argc--; argv++;
                break;
        }
        argc--;
        argv++;
    }
    if(argc>1)
        inputfilename=argv[1];
    else if(nrandom>0) {
        inputfilename=randomfilename;
        randomobjects(inputfilename);
    } else if(nrandom<0)
        inputfilename=randomfilename;
    else
        usage();
    inputfile=fopen(inputfilename, "r");
    if(yyparse()!=0)
```

```
// DEFAULT BRUTE FORCE
// PROCESS ARGUMENTS
// ACCELERATION METHOD
// BRUTE-FORCE
// VIA VORONOI-DIAGRAM
// OUTPUT FILE NAME
// USE RANDOM OBJECTS
// VERBOSE OUTPUT
// X-RESOLUTION
// Y-RESOLUTION
// NAME OF INPUT FILE
// COMES AS ARGUMENT
// USE RANDOM OBJECTS
// GENERATE OBJECTS
// USE RANDOM OBJECTS
// NOT GIVEN -> QUIT
// OPEN INPUT FILE
// READ INPUT FILE
```

```
        exit(1);
fprintf(stderr, "\n");
query->preprocess(&objects);           // PREPROCESS OBJECTS
outinit(xres, yres);                   // INITIALIZE OUTPUT
for(register int i=0; i<yres; i++) {    // TRACE
    for(register int j=0; j<xres; j++) {
        double x=(2*j-xres+0.5)/xres;
        double y=(yres-2*i-0.5)/yres;
        intensity I=trace(actcamera.getray(x,y));
        buf[j]=(icolor)I;
    }
    output(buf);
    fprintf(stderr, "rows remained: %4d\r", yres-i-1);
}
fprintf(stderr, "\n");
exit(0);
```

}

```
CC = g++ -g

.cxx.o:
    $(CC) -c $<

PROG = oopov
HEADER = global.h
OBJS = camera.o csg.o finish.o finite.o infinite.o lightsrc.o main.o \
      misc.o normal.o parser.o pigment.o texture.o voronoi.o
LEX = pov
YACC = pov

$(PROG) : $(OBJS)
    $(CC) -o $(PROG) -L/usr/local/lib $(OBJS) -ll -ly -lm -lg++

voronoi.o : voronoi.cxx voronoi.h $(HEADER)

$(OBJS) : $$(@:.o=.cxx) $(HEADER)

parser.cxx : y.tab.c lex.yy.c
    cat y.tab.c lex.yy.c > parser.cxx
    touch parser.cxx

y.tab.c: $(YACC).y $(HEADER)
    yacc $(YACC).y

lex.yy.c: $(LEX).l
    lex $(LEX).l
```

```
#include <iostream.h>

#include "global.h"

list<object*> objects;

ostream& operator<<(ostream& o, vector& v) {
    o<<"<<v.x<<" "<<v.y<<" "<<v.z<<">;
    return o;
}

vector operator-(vector& v) {return vector(-v.x,-v.y,-v.z);}

ostream& operator<<(ostream& o, xform& T) {
    o<<"xform {\n";
    o<<"\t"<<"orientation {\n";
    o<<"\t"<<"\t"<<T.A.m11<<"\t"<<T.A.m12<<"\t"<<T.A.m13<<"\n";
    o<<"\t"<<"\t"<<T.A.m21<<"\t"<<T.A.m22<<"\t"<<T.A.m23<<"\n";
    o<<"\t"<<"\t"<<T.A.m31<<"\t"<<T.A.m32<<"\t"<<T.A.m33<<"\n";
    o<<"\t"<<"}\n";
    o<<"\t"<<"scale "<<T.s<<"\n";
    o<<"\t"<<"translate "<<T.t<<"\n";
    o<<"}\n";
    return o;
}

ostream& operator<<(ostream& o, color& c) {
    o<<"red "<<c.r<<" green "<<c.g<<" blue "<<c.b<<" filter "<<c.f<<">;
    return o;
}

ostream& operator<<(ostream& o, intensity& i) {
    o<<"red "<<i.r<<" green "<<i.g<<" blue "<<i.b<<">;
    return o;
}

ostream& operator<<(ostream& o, object& p) {p.out(o); return o;}

vector norm(vector& v) {return v*(1./sqrt(v%v));}
```

```
#include <iostream.h>

#include "global.h"

ostream& operator<<(ostream& o, normal& n) {
    o<<"normal {\n";
    n.out(o);
    o<<"turbulence "<<n.t<<"\n";
    o<<"frequency "<<n.f<<"\n"<<"phase "<<n.p<<"\n"<<n.T;
    o<<"}\n";
    return o;
}
```



```
#include <iostream.h>
```

```
#include "global.h"
```

```
#include <iostream.h>
#include <math.h>

#include "global.h"

ostream& operator<<(ostream& o, pigment& p) {
    o<<"pigment {\n";
    p.out(o);
    o<<"turbulence "<<p.t<<"\n"<<"octaves "<<p.o<<"\n";
    o<<"omega "<<p.m<<"\n"<<"lambda "<<p.l<<"\n";
    o<<"frequency "<<p.f<<"\n"<<"phase "<<p.p<<"\n";
    o<<"quick_color "<<p.q<<"\n"<<p.T;
    o<<"}\n";
    return o;
}

ostream& operator<<(ostream& o, colormap& m) {
    o<<"color_map{\n";
    for(register int i=0; i<m.nI; i++) {
        o<<"    ["<<m.I[i].b<<" , "<<m.I[i].e;
        o<<"    "<<m.I[i].cb<<"    "<<m.I[i].ce<<" ]\n";
    }
    o<<"}\n";
    return o;
}

ostream& operator<<(ostream& o, colormapped& m) {
    o<<"undefined colormapped {\n"<<m.cm<<"}\n"; return o;
}
```

```
%e      3000
%p      7000
%n      1000
%a      3000
```

```
%s      COMMENT
```

```
W      [ \t]+
D      [0-9]
E      [DEde][-+]?{D}+
```

```
%{
```

```
#include <stdio.h>
```

```
int comment;
```

```
template <class T> struct stackelem {
    T v;
    stackelem<T> *n;
};
```

```
template <class T> class stack {
    stackelem<T> *h;
public:
    stack() {h=(stackelem<T>*)0;}
    ~stack() {stackelem<T> *n; for(;h;n) {n=h->n; delete h;}}
    void operator<<(T& v) {
        stackelem<T> *e=new stackelem<T>; e->v=v; e->n=h; h=e;
    }
    void operator>>(T& v) {stackelem<T> *e=h; h=e->n; v=e->v; delete e;}
    int empty() {return h==(stackelem<T>*)0;}
};
```

```
struct fileident {
    char* n;
    FILE* f;
    int l;
    fileident() {}
    fileident(char* n, FILE* f, int l) {this->n=n; this->f=f; this->l=l;}
};
```

```
stack<fileident> files;
FILE* inputfile=stdin;
char* inputfilename="";
int line=1;
```

```
INPUT() {
    int c=fgetc(inputfile);
    fileident f;
    switch(c) {
        case EOF:
            fprintf(stderr, "\n");
            if(files.empty())
                return 0;
            fclose(inputfile); delete inputfilename;
            files>>f; inputfile=f.f; inputfilename=f.n; line=f.l;
            return '\n';
        case '\n':
            line++;
    }
```

```
        fprintf(stderr,"reading line %d of file %s\r",line,inputfilename);
    }
    return c;
}

UNPUT(int c) {return ungetc(c, inputfile);}

#undef input
#define input() INPUT()
#undef unput
#define unput(c) UNPUT(c)

void include(char* s) {
    register int i; char *n; fileident f(inputfilename,inputfile,line);
    files<<f;
    for(i=0; s[i]!='\0'; i++); n=&s[i+1];
    for(i=0; n[i]!='\0'; i++); n[i]='\0';
    inputfilename=new char[strlen(n)+1]; strcpy(inputfilename,n);
    inputfile=fopen(n, "r");
    line=1;
}

%}

%%

"/*"                {comment=1; BEGIN COMMENT;}
<COMMENT>"/*"       {comment++;}
<COMMENT>"*/"       {comment--; if(!comment) {BEGIN INITIAL;}}
<COMMENT>."         ;

<INITIAL>"*/"       printf("unmatched */ (ignored)\n");
<INITIAL>"//".*$    ;

<INITIAL>#include{W}\("[^"\n]*\"    include((char*)yytext);

<INITIAL>adaptive   return ADAPTIVE;
<INITIAL>agate      return AGATE;
<INITIAL>agate_turb return AGATE_TURB;
<INITIAL>all        return ALL;
<INITIAL>alpha      return ALPHA;
<INITIAL>ambient    return AMBIENT;
<INITIAL>area_light return AREA_LIGHT;
<INITIAL>background return BACKGROUND;
<INITIAL>bicubic_patch return BICUBIC_PATCH;
<INITIAL>blob       return BLOB;
<INITIAL>blue       return BLUE;
<INITIAL>bounded_by return BOUNDED_BY;
<INITIAL>box        return BOX;
<INITIAL>bozo       return BOZO;
<INITIAL>brick      return BRICK;
<INITIAL>brilliance return BRILLIANCE;
<INITIAL>bumps      return BUMPS;
<INITIAL>bumpy1     return BUMPY1;
<INITIAL>bumpy2     return BUMPY2;
<INITIAL>bumpy3     return BUMPY3;
<INITIAL>bump_map   return BUMP_MAP;
<INITIAL>bump_size  return BUMP_SIZE;
<INITIAL>camera{W}identifier return CAMERA_ID;
<INITIAL>camera     return CAMERA;
<INITIAL>checker    return CHECKER;
```

<INITIAL>clipped_by	return CLIPPED_BY;
<INITIAL>clock	return CLOCK;
<INITIAL>colour{W}identifier	return COLOR_ID;
<INITIAL>colour{W}map{W}identifier	return COLOR_MAP_ID;
<INITIAL>color_map	return COLOR_MAP;
<INITIAL>colour_map	return COLOR_MAP;
<INITIAL>color	return COLOR;
<INITIAL>colour	return COLOR;
<INITIAL>component	return COMPONENT;
<INITIAL>composite	return COMPOSITE;
<INITIAL>cone	return CONE;
<INITIAL>crand	return CRAND;
<INITIAL>cubic	return CUBIC;
<INITIAL>cylinder	return CYLINDER;
<INITIAL>declare	return DECLARE;
<INITIAL>default	return DEFAULT;
<INITIAL>dents	return DENTS;
<INITIAL>difference	return DIFFERENCE;
<INITIAL>diffuse	return DIFFUSE;
<INITIAL>direction	return DIRECTION;
<INITIAL>disc	return DISC;
<INITIAL>distance	return DISTANCE;
<INITIAL>dump	return DUMP;
<INITIAL>End{W}of{W}File	return END_OF_FILE;
<INITIAL>falloff	return FALLOFF;
<INITIAL>filter	return FILTER;
<INITIAL>finish{W}identifier	return FINISH_ID;
<INITIAL>finish	return FINISH;
<INITIAL>flatness	return FLATNESS;
<INITIAL>float{W}identifier	return FLOAT_ID;
<INITIAL>float{W}constant	return FLOAT;
<INITIAL>fog	return FOG;
<INITIAL>frequency	return FREQUENCY;
<INITIAL>gif	return GIF;
<INITIAL>gradient	return GRADIENT;
<INITIAL>granite	return GRANITE;
<INITIAL>green	return GREEN;
<INITIAL>height_field	return HEIGHT_FIELD;
<INITIAL>hexagon	return HEXAGON;
<INITIAL>undeclared{W}identifier	return IDENTIFIER;
<INITIAL>iff	return IFF;
<INITIAL>image_map	return IMAGE_MAP;
<INITIAL>interpolate	return INTERPOLATE;
<INITIAL>intersection	return INTERSECTION;
<INITIAL>inverse	return INVERSE;
<INITIAL>ior	return IOR;
<INITIAL>jitter	return JITTER;
<INITIAL>lambda	return LAMBDA;
<INITIAL>leopard	return LEOPARD;
<INITIAL>light_source	return LIGHT_SOURCE;
<INITIAL>location	return LOCATION;
<INITIAL>looks_like	return LOOKS_LIKE;
<INITIAL>look_at	return LOOK_AT;
<INITIAL>mandel	return MANDEL;
<INITIAL>map_type	return MAP_TYPE;
<INITIAL>marble	return MARBLE;
<INITIAL>material_map	return MATERIAL_MAP;
<INITIAL>max_intersections	return MAX_INTERSECTIONS;
<INITIAL>max_trace_level	return MAX_TRACE_LEVEL;
<INITIAL>merge	return MERGE;
<INITIAL>metallic	return METALLIC;

<INITIAL>mortar	return MORTAR;
<INITIAL>normal	return NORMAL;
<INITIAL>no_shadow	return NO_SHADOW;
<INITIAL>object{W}identifier	return OBJECT_ID;
<INITIAL>object	return OBJECT;
<INITIAL>octaves	return OCTAVES;
<INITIAL>omega	return OMEGA;
<INITIAL>once	return ONCE;
<INITIAL>onion	return ONION;
<INITIAL>open	return OPEN;
<INITIAL>painted1	return PAINTED1;
<INITIAL>painted2	return PAINTED2;
<INITIAL>painted3	return PAINTED3;
<INITIAL>phase	return PHASE;
<INITIAL>phong_size	return PHONG_SIZE;
<INITIAL>phong	return PHONG;
<INITIAL>pigment{W}identifier	return PIGMENT_ID;
<INITIAL>pigment	return PIGMENT;
<INITIAL>plane	return PLANE;
<INITIAL>point_at	return POINT_AT;
<INITIAL>poly	return POLY;
<INITIAL>pot	return POT;
<INITIAL>quadric	return QUADRIC;
<INITIAL>quartic	return QUARTIC;
<INITIAL>quick_color	return QUICK_COLOR;
<INITIAL>quick_colour	return QUICK_COLOR;
<INITIAL>radial	return RADIAL;
<INITIAL>radius	return RADIUS;
<INITIAL>raw	return RAW;
<INITIAL>red	return RED;
<INITIAL>reflection	return REFLECTION;
<INITIAL>refraction	return REFRACTION;
<INITIAL>rgbf	return RGBF;
<INITIAL>rgb	return RGB;
<INITIAL>right	return RIGHT;
<INITIAL>ripples	return RIPPLES;
<INITIAL>rotate	return ROTATE;
<INITIAL>roughness	return ROUGHNESS;
<INITIAL>scale	return SCALE;
<INITIAL>sky	return SKY;
<INITIAL>smooth	return SMOOTH;
<INITIAL>smooth_triangle	return SMOOTH_TRIANGLE;
<INITIAL>specular	return SPECULAR;
<INITIAL>sphere	return SPHERE;
<INITIAL>spotlight	return SPOTLIGHT;
<INITIAL>spotted	return SPOTTED;
<INITIAL>string	return STRING;
<INITIAL>sturm	return STURM;
<INITIAL>texture{W}identifier	return TEXTURE_ID;
<INITIAL>texture	return TEXTURE;
<INITIAL>tga	return TGA;
<INITIAL>threshold	return THRESHOLD;
<INITIAL>tightness	return TIGHTNESS;
<INITIAL>tile2	return TILE2;
<INITIAL>tiles	return TILES;
<INITIAL>tnormal{W}identifier	return TNORMAL_ID;
<INITIAL>torus	return TORUS;
<INITIAL>track	return TRACK;
<INITIAL>transform{W}identifier	return TRANSFORM_ID;
<INITIAL>transform	return TRANSFORM;
<INITIAL>translate	return TRANSLATE;

```
<INITIAL>triangle                return TRIANGLE;
<INITIAL>turbulence               return TURBULENCE;
<INITIAL>type                    return TYPE;
<INITIAL>union                   return UNION;
<INITIAL>up                      return UP;
<INITIAL>use_color               return USE_COLOR;
<INITIAL>use_colour              return USE_COLOR;
<INITIAL>use_index              return USE_INDEX;
<INITIAL>u_steps                return U_STEPS;
<INITIAL>vector{W}identifier     return VECTOR_ID;
<INITIAL>version                return VERSION;
<INITIAL>v_steps                return V_STEPS;
<INITIAL>water_level            return WATER_LEVEL;
<INITIAL>waves                  return WAVES;
<INITIAL>wood                   return WOOD;
<INITIAL>wrinkles               return WRINKLES;
<INITIAL>x                      return X;
<INITIAL>y                      return Y;
<INITIAL>z                      return Z;

<INITIAL>{D}+                    |
<INITIAL>{D}+"."{D}*({E})?       |
<INITIAL>{D}*"."{D}+({E})?       |
<INITIAL>{D}+{E}                 {yyval.dval=atof((char*)yytext);return LITERAL;}
<INITIAL>[a-zA-Z][a-zA-Z0-9]*     {yyval.sval=(char*)yytext; return IDENTIFIER;}

<INITIAL>{W}                      ;
<INITIAL>\n                       ;

<INITIAL>.{                       {return yytext[0];}
```

```
%{  
  
#include <iostream.h>  
#include <string.h>  
#include <stdlib.h>  
  
#include "global.h"  
  
//      DECLARATIONS  
  
texture *deftexture;  
declarations<object> odecls;  
declarations<texture> tdecls;  
declarations<pigment> pdecls;  
declarations<normal> ndecls;  
declarations<finish> fdecls;  
declarations<color> cdecls;  
declarations<colormap> cmdecls;  
declarations<vector> vdecls;  
declarations<camera> camdecls;  
  
int max_intersections=64;  
double version=2.2;  
double clock=0.;  
intensity background=intensity(0.,0.,0.);  
  
%}  
  
%start  items  
  
%union {  
    int ival;  
    double dval;  
    char* sval;  
    vector* vval;  
    object* oval;  
    color* cval;  
    list<object*>* olval;  
    texture* tval;  
    pigment* pval;  
    normal* nval;  
    finish* fval;  
    camera* camval;  
    colormap* cmval;  
    lightsource *lval;  
}  
  
/*  
*      KEYWORD TOKENS  
*/  
%token <ival> ADAPTIVE  
%token <ival> AGATE  
%token <ival> AGATE_TURB  
%token <ival> ALL  
%token <ival> ALPHA  
%token <ival> AMBIENT  
%token <ival> AREA_LIGHT  
%token <ival> BACKGROUND  
%token <ival> BICUBIC_PATCH  
%token <ival> BLOB  
%token <ival> BLUE
```



%token <ival> BOUNDED\_BY  
%token <ival> BOX  
%token <ival> BOZO  
%token <ival> BRICK  
%token <ival> BRILLIANCE  
%token <ival> BUMPS  
%token <ival> BUMPY1  
%token <ival> BUMPY2  
%token <ival> BUMPY3  
%token <ival> BUMP\_MAP  
%token <ival> BUMP\_SIZE  
%token <ival> CAMERA\_ID  
%token <ival> CAMERA  
%token <ival> CHECKER  
%token <ival> CLIPPED\_BY  
%token <ival> CLOCK  
%token <ival> COLOR\_ID  
%token <ival> COLOR\_MAP\_ID  
%token <ival> COLOR\_MAP  
%token <ival> COLOR  
%token <ival> COMPONENT  
%token <ival> COMPOSITE  
%token <ival> CONE  
%token <ival> CRAND  
%token <ival> CUBIC  
%token <ival> CYLINDER  
%token <ival> DECLARE  
%token <ival> DEFAULT  
%token <ival> DENTS  
%token <ival> DIFFERENCE  
%token <ival> DIFFUSE  
%token <ival> DIRECTION  
%token <ival> DISC  
%token <ival> DISTANCE  
%token <ival> DUMP  
%token <ival> END\_OF\_FILE  
%token <ival> FALLOFF  
%token <ival> FILTER  
%token <ival> FINISH\_ID  
%token <ival> FINISH  
%token <ival> FLATNESS  
%token <ival> FLOAT\_ID  
%token <ival> FLOAT  
%token <ival> FOG  
%token <ival> FREQUENCY  
%token <ival> GIF  
%token <ival> GRADIENT  
%token <ival> GRANITE  
%token <ival> GREEN  
%token <ival> HEIGHT\_FIELD  
%token <ival> HEXAGON  
%token <ival> IFF  
%token <ival> IMAGE\_MAP  
%token <ival> INCLUDE  
%token <ival> INTERPOLATE  
%token <ival> INTERSECTION  
%token <ival> INVERSE  
%token <ival> IOR  
%token <ival> JITTER  
%token <ival> LAMBDA  
%token <ival> LEOPARD

%token <ival> LIGHT\_SOURCE  
%token <ival> LOCATION  
%token <ival> LOOKS\_LIKE  
%token <ival> LOOK\_AT  
%token <ival> MANDEL  
%token <ival> MAP\_TYPE  
%token <ival> MARBLE  
%token <ival> MATERIAL\_MAP  
%token <ival> MAX\_INTERSECTIONS  
%token <ival> MAX\_TRACE\_LEVEL  
%token <ival> MERGE  
%token <ival> METALLIC  
%token <ival> MORTAR  
%token <ival> NO\_SHADOW  
%token <ival> OBJECT\_ID  
%token <ival> OBJECT  
%token <ival> OCTAVES  
%token <ival> OMEGA  
%token <ival> ONCE  
%token <ival> ONION  
%token <ival> OPEN  
%token <ival> PAINTED1  
%token <ival> PAINTED2  
%token <ival> PAINTED3  
%token <ival> PHASE  
%token <ival> PHONG\_SIZE  
%token <ival> PHONG  
%token <ival> PIGMENT\_ID  
%token <ival> PIGMENT  
%token <ival> PLANE  
%token <ival> POINT\_AT  
%token <ival> POLY  
%token <ival> POT  
%token <ival> QUADRIC  
%token <ival> QUARTIC  
%token <ival> QUICK\_COLOR  
%token <ival> RADIAL  
%token <ival> RADIUS  
%token <ival> RAW  
%token <ival> RED  
%token <ival> REFLECTION  
%token <ival> REFRACTION  
%token <ival> RGBF  
%token <ival> RGB  
%token <ival> RIGHT  
%token <ival> RIPPLES  
%token <ival> ROTATE  
%token <ival> ROUGHNESS  
%token <ival> SCALE  
%token <ival> SKY  
%token <ival> SMOOTH  
%token <ival> SMOOTH\_TRIANGLE  
%token <ival> SPECULAR  
%token <ival> SPHERE  
%token <ival> SPOTLIGHT  
%token <ival> SPOTTED  
%token <ival> STRING  
%token <ival> STURM  
%token <ival> TEXTURE\_ID  
%token <ival> TEXTURE  
%token <ival> TGA

```
%token <ival> THRESHOLD
%token <ival> TIGHTNESS
%token <ival> TILE2
%token <ival> TILES
%token <ival> TNORMAL_ID
%token <ival> NORMAL
%token <ival> TORUS
%token <ival> TRACK
%token <ival> TRANSFORM_ID
%token <ival> TRANSFORM
%token <ival> TRANSLATE
%token <ival> TRIANGLE
%token <ival> TURBULENCE
%token <ival> TYPE
%token <ival> UNION
%token <ival> UP
%token <ival> USE_COLOR
%token <ival> USE_INDEX
%token <ival> U_STEPS
%token <ival> VECTOR_ID
%token <ival> VERSION
%token <ival> V_STEPS
%token <ival> WATER_LEVEL
%token <ival> WAVES
%token <ival> WOOD
%token <ival> WRINKLES
%token <ival> X
%token <ival> Y
%token <ival> Z

/*
 *      OTHER TOKENS
 */
%token <dval> LITERAL
%token <sval> IDENTIFIER

/*
 *      NONTERMINALS
 */
%type <vval> expr
%type <oval> object objopts
%type <olval> objects moreobjects
%type <cval> color colorvalues
%type <tval> texture textureitems
%type <pval> pigment pigmentitems pigmentpatt
%type <nval> normal normalitems normalpatt
%type <fval> finish finishitems
%type <camval> camera cameraitems
%type <cmval> colormap colormapitems
%type <sval> identifier
%type <lval> lightsource lightsrcitems lightsrcype spotlight
%type <vval> tightness

%left '+' '-'
%left '*' '/'
%left UMINUS

%%
/*
 *      SCENE FILE ITEMS AT TOP LEVEL
 */
items      :      /* empty */
```

```

        {deftexture=new texture;}
    |
    items item
;

item      :      camera
            {actcamera=*$1; delete $1;}
    |
    object
            {{if(verbose) cout<<*$1;} objects+=*$1;}
    |
    lightsource
            {lightsources+=*$1;}
    |
    '#' DECLARE identifier '=' object
            {odecls+=new declaration<object>($3,$5);}
    |
    '#' DECLARE identifier '=' texture
            {tdecls+=new declaration<texture>($3,$5);}
    |
    '#' DECLARE identifier '=' pigment
            {pdecls+=new declaration<pigment>($3,$5);}
    |
    '#' DECLARE identifier '=' normal
            {nddecls+=new declaration<normal>($3,$5);}
    |
    '#' DECLARE identifier '=' finish
            {fdecls+=new declaration<finish>($3,$5);}
    |
    '#' DECLARE identifier '=' color
            {cdecls+=new declaration<color>($3,$5);}
    |
    '#' DECLARE identifier '=' colormap
            {cmdecls+=new declaration<colormap>($3,$5);}
    |
    '#' DECLARE identifier '=' expr
            {vdecls+=new declaration<vector>($3,$5);}
    |
    '#' DECLARE identifier '=' camera
            {camdecls+=new declaration<camera>($3,$5);}
    |
    '#' MAX_TRACE_LEVEL expr
            {lmax=(int)((double)*$3; delete $3;}
    |
    '#' MAX_INTERSECTIONS expr
            {max_intersections=(int)((double)*$3; delete $3;}
    |
    '#' VERSION expr
            {version=(double)*$3; delete $3;}
    |
    '#' DEFAULT '{' texture '}'
            {delete deftexture; deftexture=$4;}
    |
    '#' DEFAULT '{' pigment '}'
            {deftexture->setp(*$4); delete $4;}
    |
    '#' DEFAULT '{' normal '}'
            {deftexture->setn(*$4); delete $4;}
    |
    '#' DEFAULT '{' finish '}'
            {deftexture->setf(*$4); delete $4;}
    |
    BACKGROUND '{' color '}'
            {background=(intensity)(*$3); delete $3;}
;

identifier      :      IDENTIFIER
                    {$$=new char[strlen($1)+1]; strcpy($$, $1);}
;

/*
 *      FLOAT AND VECTOR EXPRESSIONS UNIFIED IN ORDER TO GET LR(1) GRAMMAR
 */
expr      :      LITERAL
                    {$$=new vector($1,$1,$1);}
    |
    identifier
                    {$$=new vector; *$$=*vdecls[$1]; delete $1;}
    |
    X
                    {$$=new vector(1.,0.,0.);}
    |
    Y
                    {$$=new vector(0.,1.,0.);}

```

```

|      Z
|      { $$=new vector(0.,0.,1.); }
|      '<' expr ',' expr ',' expr '>'
|      { $$=new vector((double)*$2,(double)*$4,(double)*$6);
|        delete $2; delete $4; delete $6; }
|      expr '+' expr
|      { $$=new vector; *$$=*$1+*$3; delete $1; delete $3; }
|      expr '-' expr
|      { $$=new vector; *$$=*$1-*$3; delete $1; delete $3; }
|      expr '*' expr
|      { $$=new vector; *$$=*$1*(*$3); delete $1; delete $3; }
|      expr '/' expr
|      { $$=new vector; *$$=*$1/(*$3); delete $1; delete $3; }
|      '-' expr %prec UMINUS
|      { $$=new vector; *$$=-*$2; delete $2; }
|      '(' expr ')'
|      { $$=$2; }
|
;

/*
*
*/
camera      :      CAMERA '{' cameraitems '}'
              { $$=$3; }
;

cameraitems :      /* empty */
              { $$=new camera; }
|      cameraitems identifier
              { $$=$1; *$$=actcamera /* *camdecls[$2] */ ; delete $2; }
|      cameraitems LOCATION expr
              { $$=$1; $$->setl(*$3); delete $3; }
|      cameraitems SKY expr
              { $$=$1; $$->sets(*$3); delete $3; }
|      cameraitems DIRECTION expr
              { $$=$1; $$->setd(*$3); delete $3; }
|      cameraitems UP expr
              { $$=$1; $$->setu(*$3); delete $3; }
|      cameraitems RIGHT expr
              { $$=$1; $$->setr(*$3); delete $3; }
|      cameraitems LOOK_AT expr
              { $$=$1; $$->seta(*$3); delete $3; }
|      cameraitems TRANSLATE expr
              { $$=$1; $$->traT(*$3); delete $3; }
|      cameraitems ROTATE expr
              { $$=$1; $$->rotT(*$3); delete $3; }
|      cameraitems SCALE expr
              { $$=$1; $$->scaT(*$3); delete $3; }
;

/*
*
*/
lightsource :      LIGHT_SOURCE '{' expr color lightsrcitems '}'
                  { $$=$5; $$->setp(*$3); $$->setc(*$4);
                    delete $3; delete $4; }
;

lightsrcitems :      /* empty */
                  { $$=new lightsource; }
|      lightsrcitems lightsrcctype
                  { $$=$2; *$$=*$1; delete $1; }
|      lightsrcitems TRANSLATE expr

```

```

                                {$$=$1; $$->traT(*$3); delete $3;}
|                                lightsrcitems ROTATE expr
                                {$$=$1; $$->rotT(*$3); delete $3;}
|                                lightsrcitems SCALE expr
                                {$$=$1; $$->scaT(*$3); delete $3;}
;

lightsrctype      :      spotlight
                    {$$=$1;}
;

spotlight        :      SPOTLIGHT
                    POINT_AT expr RADIUS expr FALLOFF expr tightness
                        {$$=new spotlight(*$3,(double)*$5,(double)*$7,
                                           (double)*$8);
                        delete $3; delete $5; delete $7; delete $8;}
;

tightness        :      /* empty */
                    {$$=new vector(10.,10.,10.);}
|      TIGHTNESS expr
                    {$$=$2;}
;

/*
*      OBJECT GEOMETRY
*/
object           :      OBJECT {' identifier objopts '}
                    {$$=odecls[$3]->copy(); *$$=*$4;
                    delete $3; delete $4;}
|      SPHERE {' expr ',' expr objopts '}
                    {$$=new sphere(*$3,(double)*$5); *$$=*$6;
                    delete $3; delete $5; delete $6;}
|      UNION {' objects objopts '}
                    {$$=new csguni($3); *$$=*$4; delete $4;}
|      MERGE {' objects objopts '}
                    {$$=new csgmer($3); *$$=*$4; delete $4;}
|      INTERSECTION {' objects objopts '}
                    {$$=new csgint($3); *$$=*$4; delete $4;}
|      DIFFERENCE {' objects objopts '}
                    {$$=new csgdif($3); *$$=*$4; delete $4;}
;

objects          :      object object moreobjects
                    {$$=$3; *$$+=*$2; *$$+=*$1;}
;

moreobjects      :      /* empty */
                    {$$=new list<object*>;}
|      moreobjects object
                    {$$=$1; *$$+=*$2;}
;

/*
*      OBJECT OPTIONS
*/
objopts          :      /* empty */
                    {$$=new object; $$->sett(*deftexture);}
|      objopts texture
                    {$$=$1; $$->sett(*$2); delete $2;}
|      objopts pigment
                    {$$=$1; $$->setp(*$2); delete $2;}
|      objopts normal
```

```

        {$$=$1; $$->setn(*$2); delete $2;}
|      objopts finish
        {$$=$1; $$->setf(*$2); delete $2;}
|      objopts TRANSLATE expr
        {$$=$1; $$->traT(*$3); delete $3;}
|      objopts ROTATE expr
        {$$=$1; $$->rotT(*$3); delete $3;}
|      objopts SCALE expr
        {$$=$1; $$->scaT(*$3); delete $3;}
;

/*
*      TEXTURES
*/
texture      :      TEXTURE '{' textureitems '}'
                {$$=$3;}
|      texture TEXTURE '{' textureitems '}'
                {$$=$1; *$$+= $4;}
;

textureitems :      /* empty */
                {$$=new texture; *$$=*deftexture;}
|      textureitems identifier
                {delete $1; $$=tdecls[$2]->copy(); delete $2;}
|      textureitems PIGMENT '{' pigmentitems '}'
                {$$=$1; $$->setp(*$4); delete $4;}
|      textureitems NORMAL '{' normalitems '}'
                {$$=$1; $$->setn(*$4); delete $4;}
|      textureitems FINISH '{' finishitems '}'
                {$$=$1; $$->setf(*$4); delete $4;}
|      textureitems TRANSLATE expr
                {$$=$1; $$->traT(*$3); delete $3;}
|      textureitems ROTATE expr
                {$$=$1; $$->rotT(*$3); delete $3;}
|      textureitems SCALE expr
                {$$=$1; $$->scaT(*$3); delete $3;}
;

/*
*      PIGMENT
*/
pigment      :      PIGMENT '{' pigmentitems '}'
                {$$=$3;}
;

pigmentitems :      /* empty */
                {$$=deftexture->copyp();}
|      pigmentitems identifier
                {delete $1; $$=pdecls[$2]->copy(); delete $2;}
|      pigmentitems pigmentpatt
                {$$=$2; *$$|=*$1; delete $1;}
|      pigmentitems TURBULENCE expr
                {$$=$1; $$->sett(*$3); delete $3;}
|      pigmentitems OCTAVES expr
                {$$=$1; $$->seto((int)((double)*$3)); delete $3;}
|      pigmentitems OMEGA expr
                {$$=$1; $$->setm((double)*$3); delete $3;}
|      pigmentitems LAMBDA expr
                {$$=$1; $$->setl((double)*$3); delete $3;}
|      pigmentitems FREQUENCY expr
                {$$=$1; $$->setf((double)*$3); delete $3;}
|      pigmentitems PHASE expr
                {$$=$1; $$->setp((double)*$3); delete $3;}
;
```

```
|      pigmentitems TRANSLATE expr
|          {$$=$1; $$->traT(*$3); delete $3;}
|      pigmentitems ROTATE expr
|          {$$=$1; $$->rotT(*$3); delete $3;}
|      pigmentitems SCALE expr
|          {$$=$1; $$->scaT(*$3); delete $3;}
|
pigmentpatt :      color
|          {$$=new solid(*$1); delete $1;}
|      CHECKER
|          {$$=new checker;}
|      CHECKER color
|          {$$=new checker(*$2); delete $2;}
|      CHECKER color color
|          {$$=new checker(*$2,$$3);delete $2;delete $3;}
|
color :      COLOR identifier
|          {$$=new color; *$$=*cdecls[$2]; delete $2;}
|      COLOR RGB '<' expr ',' expr ',' expr '>'
|          {$$=new color((double)*$4,(double)*$6,
|              (double)*$8);
|              delete $4; delete $6; delete $8;}
|      COLOR RGBF '<' expr ',' expr ',' expr ',' expr '>'
|          {$$=new color((double)*$4,(double)*$6,
|              (double)*$8,(double)*$10);
|              delete $4; delete $6; delete $8; delete $10;}
|      COLOR colorvalues
|          {$$=$2;}
|
colorvalues :      RED expr
|          {$$=new color;$$->setr((double)*$2);delete $2;}
|      GREEN expr
|          {$$=new color;$$->setg((double)*$2);delete $2;}
|      BLUE expr
|          {$$=new color;$$->setb((double)*$2);delete $2;}
|      FILTER expr
|          {$$=new color;$$->setf((double)*$2);delete $2;}
|      colorvalues RED expr
|          {$$=$1; $$->setr((double)*$3); delete $3;}
|      colorvalues GREEN expr
|          {$$=$1; $$->setg((double)*$3); delete $3;}
|      colorvalues BLUE expr
|          {$$=$1; $$->setb((double)*$3); delete $3;}
|      colorvalues FILTER expr
|          {$$=$1; $$->setf((double)*$3); delete $3;}
|
colormap :      COLOR_MAP '{' identifier colormapitems '}'
|          {$$=cmdecls[$3]->copy(); delete $3; delete $4;}
|      COLOR_MAP '{' colormapitems '}'
|          {$$=$3;}
|
colormapitems :      /* empty */
|          {$$=new colormap;}
|
/*
*      NORMAL
```



```
*/
normal      :      NORMAL '{' normalitems '}'
                { $$=$3; }
                ;

normalitems :      /* empty */
                { $$=deftexture->copyn(); }
                | normalitems identifier
                { delete $1; $$=ndecls[$2]->copy(); delete $2; }
                | normalitems normalpatt
                { $$=$2; *$$|=*$1; delete $1; }
                | normalitems TURBULENCE expr
                { $$=$1; $$->setf((double)*$3); delete $3; }
                | normalitems FREQUENCY expr
                { $$=$1; $$->setf((double)*$3); delete $3; }
                | normalitems PHASE expr
                { $$=$1; $$->setp((double)*$3); delete $3; }
                | normalitems TRANSLATE expr
                { $$=$1; $$->traT(*$3); delete $3; }
                | normalitems ROTATE expr
                { $$=$1; $$->rotT(*$3); delete $3; }
                | normalitems SCALE expr
                { $$=$1; $$->scaT(*$3); delete $3; }
                ;

normalpatt  :      BUMPS expr
                { $$=new bumps((double)*$2); delete $2; }
                | DENTS expr
                { $$=new dents((double)*$2); delete $2; }
                ;

/*
*      FINISH
*/
finish      :      FINISH '{' finishitems '}'
                { $$=$3; }
                ;

finishitems :      /* empty */
                { $$=deftexture->copyf(); }
                | finishitems identifier
                { delete $1; $$=fdecls[$2]->copy(); delete $2; }
                | finishitems DIFFUSE expr
                { $$=$1; $$->setkd((double)*$3); delete $3; }
                | finishitems BRILLIANCE expr
                { $$=$1; $$->setkb((double)*$3); delete $3; }
                | finishitems CRAND expr
                { $$=$1; $$->setkc((double)*$3); delete $3; }
                | finishitems AMBIENT expr
                { $$=$1; $$->setka((double)*$3); delete $3; }
                | finishitems REFLECTION expr
                { $$=$1; $$->setkr((double)*$3); delete $3; }
                | finishitems PHONG expr
                { $$=$1; $$->setks((double)*$3); delete $3; }
                | finishitems PHONG_SIZE expr
                { $$=$1; $$->setkp((double)*$3); delete $3; }
                | finishitems SPECULAR expr
                { $$=$1; $$->setkh((double)*$3); delete $3; }
                | finishitems ROUGHNESS expr
                { $$=$1; $$->setkg((double)*$3); delete $3; }
                | finishitems METALLIC
                { $$=$1; $$->setkm(1); }
```

```
|      finishitems REFRACTION expr  
|      { $$=$1; $$->setkt((double)*$3); delete $3; }  
|      finishitems IOR expr  
|      { $$=$1; $$->setki((double)*$3); delete $3; }  
;
```

%%

```
void yyerror(char* s) {cout<<s<<"\n";}
```

```
#declare WHITE=color rgb<1,1,1>
#declare BLACK=color rgb<0,0,0>
#declare S=sphere{<0,0,0>,1 pigment{color WHITE}finish{diffuse 1}}

camera{location<0,0,0> direction<0,0,1> right<1,0,0> up<0,1,0>}
light_source{<0,0,0> color WHITE}
light_source{<0,0,0> color WHITE}
background {color BLACK}

object{S translate<-13.0446,42.8985,44.5176>}
object{S translate<-6.73007,-22.8433,111.683>}
object{S translate<-61.0327,10.5947,20.0792>}
object{S translate<-14.6509,24.065,7.41576>}
object{S translate<50.3785,-42.3906,20.0418>}
object{S translate<4.16589,13.1213,73.4155>}
object{S translate<-28.9818,-13.7989,36.9662>}
object{S translate<30.5376,-25.3842,9.5172>}
object{S translate<-11.9714,45.0268,118.681>}
object{S translate<20.5154,43.6532,0.347119>}
object{S translate<-4.73992,4.10684,99.2662>}
object{S translate<-29.531,60.823,38.6525>}
object{S translate<12.707,13.6973,26.7656>}
object{S translate<48.6197,-24.6117,19.1331>}
object{S translate<-20.4533,-14.177,81.0898>}
object{S translate<31.9457,13.0548,66.981>}
object{S translate<-5.12028,19.2123,41.2222>}
object{S translate<56.2392,-16.6259,118.923>}
object{S translate<-62.0602,2.09146,34.367>}
object{S translate<-59.9346,11.5855,25.8238>}
object{S translate<47.5864,-55.5161,32.8641>}
object{S translate<-24.842,49.3253,62.769>}
object{S translate<26.4616,-26.9102,108.974>}
object{S translate<22.1167,-5.22994,120.907>}
object{S translate<34.6069,-15.5536,28.8067>}
object{S translate<-18.3276,-25.1583,84.3852>}
object{S translate<27.5881,8.30975,103.876>}
object{S translate<-13.782,40.162,106.338>}
object{S translate<-40.2585,55.8644,53.5324>}
object{S translate<2.60372,-54.7255,115.095>}
object{S translate<48.2026,32.9298,50.2611>}
object{S translate<23.7189,32.953,51.0279>}
object{S translate<-47.2154,-1.93601,28.0285>}
object{S translate<47.0103,3.63133,0.178148>}
object{S translate<45.4219,-60.6403,102.67>}
object{S translate<-32.3946,-23.3625,121.675>}
object{S translate<54.8771,38.986,62.0019>}
object{S translate<-35.2607,9.61997,36.4154>}
object{S translate<-22.5441,-30.0714,21.9211>}
object{S translate<-62.7671,-57.3536,30.3861>}
object{S translate<-10.6525,25.4031,27.9508>}
object{S translate<0.492672,-54.5508,49.5229>}
object{S translate<-2.5858,-35.5119,27.6567>}
object{S translate<52.4383,-18.8709,24.2779>}
object{S translate<-36.3821,16.8428,6.74882>}
object{S translate<35.7076,-59.1314,55.9527>}
object{S translate<-40.7694,54.4513,114.609>}
object{S translate<-3.42134,46.8307,87.636>}
object{S translate<54.2005,-5.65824,50.2163>}
object{S translate<49.5532,24.3854,105.674>}
object{S translate<30.2002,19.0588,85.4421>}
object{S translate<9.72814,-28.5853,117.852>}
```

```
object{S translate<20.403,-57.0225,46.9999>}
object{S translate<14.8916,-44.2701,47.5375>}
object{S translate<18.2174,-59.7596,106.007>}
object{S translate<-53.2923,30.57,32.2522>}
object{S translate<50.6292,-15.4154,40.2582>}
object{S translate<-36.3569,18.7139,31.6637>}
object{S translate<-34.1887,-31.3818,118.838>}
object{S translate<-45.771,-28.9705,69.1328>}
object{S translate<-22.1719,45.9972,37.3823>}
object{S translate<22.6861,41.9739,110.408>}
object{S translate<18.8566,-53.7841,113.21>}
object{S translate<-31.0468,14.0253,106.071>}
object{S translate<41.8318,-16.053,95.3821>}
object{S translate<-49.2358,44.1995,70.4711>}
object{S translate<45.0523,-19.7685,87.1575>}
object{S translate<-19.5039,49.5886,120.79>}
object{S translate<-47.6584,60.6708,6.93269>}
object{S translate<14.4609,-58.1924,47.4326>}
object{S translate<3.22601,-27.4759,70.6234>}
object{S translate<13.5118,39.8689,56.2599>}
object{S translate<-59.5802,-3.54617,35.8949>}
object{S translate<-26.1152,-38.3419,2.21487>}
object{S translate<41.5269,9.21827,13.2015>}
object{S translate<29.3532,-47.9907,28.2082>}
object{S translate<56.3503,30.1186,103.484>}
object{S translate<41.09,-31.3746,32.3164>}
object{S translate<-20.3864,-14.0832,66.438>}
object{S translate<-29.4306,-12.4454,109.726>}
object{S translate<-57.214,-25.8821,49.672>}
object{S translate<7.55983,-23.835,103.638>}
object{S translate<-3.08914,-51.5127,33.0605>}
object{S translate<52.5443,60.2791,41.8411>}
object{S translate<50.6582,-32.678,47.0158>}
object{S translate<31.786,-5.31342,113.472>}
object{S translate<-9.71468,8.30584,122.271>}
object{S translate<-18.4111,-8.62964,22.5641>}
object{S translate<-35.8675,-20.5144,57.2467>}
object{S translate<-57.361,23.1677,7.8303>}
object{S translate<6.47288,-23.161,33.7933>}
object{S translate<0.0965093,-55.1169,121.561>}
object{S translate<28.9247,38.5933,19.1476>}
object{S translate<25.8306,28.9728,106.157>}
object{S translate<14.8581,57.3828,60.3991>}
object{S translate<3.41256,-59.2936,76.5543>}
object{S translate<-57.0027,11.6882,50.7303>}
object{S translate<50.9505,-21.4021,39.1631>}
object{S translate<44.7202,29.501,12.4262>}
object{S translate<-20.101,-12.38,124.123>}
object{S translate<16.3279,41.8246,55.0736>}
object{S translate<36.6923,-55.5043,17.0259>}
object{S translate<5.52147,54.2335,111.49>}
object{S translate<-13.8724,-10.7651,107.958>}
object{S translate<-26.8667,-49.4196,72.6566>}
object{S translate<34.9757,-45.0736,101.887>}
object{S translate<31.2579,-11.3362,59.8608>}
object{S translate<-19.5969,-10.4827,2.97943>}
object{S translate<3.25198,27.4526,29.737>}
object{S translate<24.5515,35.2716,96.1005>}
object{S translate<-47.6801,-16.3137,116.952>}
object{S translate<33.5147,-26.5113,123.933>}
object{S translate<-9.05796,-58.2319,75.3844>}
```

```
object{S translate<-61.0311,32.3452,16.4645>}
object{S translate<-15.8035,-35.5933,72.7979>}
object{S translate<-37.0841,36.1624,94.2886>}
object{S translate<-5.62726,-27.0545,107.121>}
object{S translate<-6.29866,42.2089,54.2076>}
object{S translate<-21.2932,56.8063,31.356>}
object{S translate<-18.0428,-10.3476,62.6248>}
object{S translate<48.5439,7.95834,86.3676>}
object{S translate<58.934,-40.019,71.539>}
object{S translate<-59.2585,-37.2985,19.8771>}
object{S translate<5.99652,-37.2948,26.6204>}
object{S translate<-54.5175,-38.6673,113.585>}
object{S translate<35.536,-24.8721,53.1696>}
object{S translate<61.6763,37.4348,48.8184>}
object{S translate<-40.073,-34.6817,92.9044>}
object{S translate<4.22163,20.7554,25.4291>}
object{S translate<-3.92795,8.37813,12.2797>}
object{S translate<-28.0933,-60.8149,71.0449>}
object{S translate<48.7513,-54.1527,7.81415>}
object{S translate<-31.0814,-24.3083,112.97>}
object{S translate<-46.1039,56.4079,54.1079>}
object{S translate<-53.1527,35.7398,55.899>}
object{S translate<-9.93043,25.4361,23.9159>}
object{S translate<-47.0933,-31.7425,82.8937>}
object{S translate<-45.6715,-22.3289,103.63>}
object{S translate<-61.2522,53.3838,47.4176>}
object{S translate<10.34,-30.0651,52.1241>}
object{S translate<23.6483,31.9668,21.5859>}
object{S translate<-46.7173,-42.9645,36.8852>}
object{S translate<45.0539,-27.8597,86.7359>}
object{S translate<42.6974,39.6846,6.12502>}
object{S translate<-53.8059,-8.69613,94.4721>}
object{S translate<-43.851,-11.0536,12.9162>}
object{S translate<32.8994,17.8639,92.5928>}
object{S translate<5.97376,-50.4117,92.2671>}
object{S translate<3.63232,2.45364,36.0967>}
object{S translate<56.3588,-9.3062,62.2773>}
object{S translate<25.5103,-30.4621,102.226>}
object{S translate<47.932,-62.854,13.4937>}
object{S translate<-28.5685,60.9784,45.4987>}
object{S translate<39.9142,44.7602,36.4638>}
object{S translate<62.9313,-61.8925,8.03647>}
object{S translate<58.9257,23.077,37.9937>}
object{S translate<28.4015,53.3683,81.7706>}
object{S translate<52.3834,4.55105,123.054>}
object{S translate<-33.58,-10.2609,73.3121>}
object{S translate<35.3153,-37.7762,113.525>}
object{S translate<-46.7258,54.0085,96.3668>}
object{S translate<-2.55246,7.1031,1.52302>}
object{S translate<-14.6483,-15.4132,91.7918>}
object{S translate<5.5193,22.3706,74.3058>}
object{S translate<-27.37,-49.4108,37.6381>}
object{S translate<3.62486,51.0284,55.0975>}
object{S translate<56.8441,-33.7225,65.5156>}
object{S translate<14.6827,-39.2747,123.378>}
object{S translate<0.402843,-42.8299,67.9769>}
object{S translate<-38.7348,-31.6159,41.601>}
object{S translate<-43.9862,15.4445,26.6754>}
object{S translate<55.9182,15.3726,26.9215>}
object{S translate<-3.31679,43.1804,55.5857>}
object{S translate<-48.0002,52.9693,84.9521>}
```

```
object{S translate<-42.2425,35.2808,89.3943>}
object{S translate<-39.7101,34.8662,27.1124>}
object{S translate<-30.9303,20.5192,121.03>}
object{S translate<55.6989,38.8063,100.459>}
object{S translate<61.289,25.1845,83.7244>}
object{S translate<-48.1458,23.9002,19.5785>}
object{S translate<-21.7101,-6.14602,51.854>}
object{S translate<0.940811,43.6278,103.931>}
object{S translate<8.66713,24.1877,17.6945>}
object{S translate<43.7604,25.3561,26.1496>}
object{S translate<-54.5605,-7.65762,106.46>}
object{S translate<-25.0707,-20.2771,67.7116>}
object{S translate<-58.8475,47.7121,22.8606>}
object{S translate<58.5568,28.6657,56.0682>}
object{S translate<37.7065,-0.580805,41.7127>}
object{S translate<-43.6446,29.8876,56.5593>}
object{S translate<-18.8624,14.909,118.622>}
object{S translate<-45.6238,32.5909,5.64485>}
object{S translate<10.9157,19.846,1.24777>}
object{S translate<58.8135,-20.6291,106.412>}
object{S translate<14.0836,-61.4398,107.383>}
object{S translate<1.41138,-33.4558,84.4708>}
object{S translate<38.067,-55.402,95.8094>}
object{S translate<14.9387,53.424,45.5809>}
object{S translate<-37.3561,-21.9732,70.0706>}
object{S translate<47.1737,56.4608,3.82599>}
```

IMPORTANT REMARKS  
about

OOPOV

written by  
Gabor Marton  
Department of Process Control, Technical University of Budapest  
Budapest, Muegyetem rkp. 9/R., H-1111 Hungary  
marton@seeger.fsz.bme.hu

1. The executable 'oopov' can be generated by 'make'. In case when GNU C++ compiler ('g++') is not present at the site, the macro CC in the 'makefile' should be changed properly. The command line option '-L/usr/local/lib' in the linking command should probably also be changed.

2. Starting 'oopov' without any command line option prints out the following:

```
usage: oopov [SWITCHes] {FILENAME | -r #}
```

where

SWITCH can be

-a v	:accelerate via Voronoi-diagram
-a b	:brute-force intersection (default)
-o FILENAME	:name of output file
-r #	:creates # number of random objects (also writes them into random.pov)
-r -1	:reads objects from random.pov
-v	:verbose printout
-x #	:horizontal resolution of image
-y #	:vertical resolution of image

# is an integer

A good trial is typing in first 'oopov -r 200 -a v', which generates 200 random spheres and renders them using the Voronoi-diagram acceleration technique, then typing in 'oopov -r -1', which renders the same spheres by the brute-force method, and comparing the running times.

3. The program 'oopov' is a framework for an object oriented implementation of the widely known ray tracer POV-Ray. Now it can render spheres only (although the parser recognizes CSG objects and also builds the corresponding object structure, the intersection calculation is not implemented for CSG). It also recognizes a few textures (pigment{solid},pigment{checker}).

4. The structure of the image file is as simple as possible (similar to that of Heckbert's minimal ray tracer):

```
+-----+-----+
| (int) XSIZE | (int) YSIZE |
+-----+-----+-----+-----+
| (unsigned char) R1 | (unsigned char) G1 | (unsigned char) B1 |
+-----+-----+-----+-----+
| (unsigned char) R2 | (unsigned char) G2 | (unsigned char) B2 |
+-----+-----+-----+-----+
.
.
.
+-----+-----+-----+-----+
| (unsigned char) Rn | (unsigned char) Gn | (unsigned char) Bn |
+-----+-----+-----+-----+
```

where  $n = \text{XSIZE} * \text{YSIZE}$

5. The image viewer 'show' can be generated by 'make -f show.m', although it can be used only if Starbase is installed on the (HP) workstation. Probably the paths should also be modified in 'show.m'.



```
#include <malloc.h>
#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <time.h>
#include <starbase.c.h>

typedef unsigned char icolor[3];

char command[100]="/usr/bin/X11/xwcreate -depth 24 -geometry ";
char screen[100]="/dev/screen/";
int watchmode=0;

main(ac, av)
int ac;
char *av[];
{
    FILE *f;
    int width, height, w, h;
    int fildes;
    icolor *row;
    unsigned char *r, *g, *b;
    register int i, j;
    register long t;

    while(av[1]&&*av[1]!='-') {
        switch(av[1][1]) {
            case 'w':
                watchmode=1;
                break;
        }
        ac--;
        av++;
    }
    if(ac!=2) {
        printf("usage: show file\n");
        exit(1);
    }
    if(!(f=fopen(av[1],"rb"))) {
        printf("failed to open file %s\n", av[1]);
        exit(1);
    }
    fread(&width, sizeof(int), 1, f);
    fread(&height, sizeof(int), 1, f);
    row=(icolor*)malloc(width*sizeof(icolor));
    r=(unsigned char*)malloc(width*sizeof(unsigned char));
    g=(unsigned char*)malloc(width*sizeof(unsigned char));
    b=(unsigned char*)malloc(width*sizeof(unsigned char));
    if(!row||!r||!g||!b) {
        printf("not enough memory\n");
        exit(1);
    }
    if(!system(NULL)) {
        printf("command processor not available\n");
        exit(1);
    }
    sprintf(&command[strlen(command)], "%dx%d+10+10 %s", width, height, av[1]);
    system(command);
    if(errno) {
        printf("failed to create window\n");
        exit(1);
    }
}
```

```
}
strcat(screen, av[1]);
if((fildes=open(screen,OUTDEV,0,INIT))<0) {
    printf("failed to open screen device\n");
    exit(1);
}
shade_mode(fildes, CMAP_FULL|INIT, 0);
for(;;) {
    for(i=0; i<height; i++) {
        if(fread(row, sizeof(icolor), width, f)!=width)
            break;
        for(j=0; j<width; j++) {
            r[j]=row[j][0];
            g[j]=row[j][1];
            b[j]=row[j][2];
        }
        bank_switch(fildes, 2, 0);
        dcblock_write(fildes, 0, i, width, 1, r, 1);
        bank_switch(fildes, 1, 0);
        dcblock_write(fildes, 0, i, width, 1, g, 1);
        bank_switch(fildes, 0, 0);
        dcblock_write(fildes, 0, i, width, 1, b, 1);
    }
    fclose(f);
    if(!watchmode)
        break;
    for(j=0; j<width; j++)
        r[j]=0;
    for(;; i<height; i++) {
        bank_switch(fildes, 2, 0);
        dcblock_write(fildes, 0, i, width, 1, r, 1);
        bank_switch(fildes, 1, 0);
        dcblock_write(fildes, 0, i, width, 1, r, 1);
        bank_switch(fildes, 0, 0);
        dcblock_write(fildes, 0, i, width, 1, r, 1);
    }
    for(t=clock(); (clock()-t)/CLOCKS_PER_SEC<2L);
    if(!(f=fopen(av[1],"rb"))) {
        printf("failed to open file %s\n", av[1]);
        exit(1);
    }
    fread(&w, sizeof(int), 1, f);
    fread(&h, sizeof(int), 1, f);
    if(!(w==width&&h==height)) {
        printf("picture size changed\n");
        exit(1);
    }
}
exit(0);
}
```

```
RM = rm -f
CFLAGS= -g

CO=gcc

DEFINES= -DSYSV
CFLAGS= -I/usr/include/X11R4/ -L/usr/lib/X11R4/

OBJECTS = \
    show.o

all: show

show: $(OBJECTS)
    cc -g $(OBJECTS) $(CFLAGS) -o show -lX11

show.o: show.c
    cc -g -c $(CFLAGS) show.c
```

```
#declare Black=color rgb<0,0,0>
#declare Red=color rgb<1,0,0>
#declare White=color rgb<1,1,1>
#declare Yellow=color rgb<1,1,0>
#declare Blue=color rgb<0,0,1>

camera{location<0,0,-4> direction<0,0,1> right<1,0,0> up<0,1,0>}
light_source{<0,0,-4> color White }

background {color Black}

sphere {5*z,3 pigment {color Yellow} finish {phong 0.9 phong_size 60 metallic}}
sphere {<0,0,3>,2 rotate <0,45,0> pigment {checker color Red color Yellow} finish{diffuse
.9}}
sphere {<-2,-1,2> ,2 pigment {color Blue} normal {bumps 0.7 scale 0.8} finish{ phong 0.9
phong_size 80}}

sphere {<0,0,10>,.5 pigment {color White} finish{diffuse .7}}
sphere {<5,0,10>,.5 pigment {color White} finish{diffuse .7}}
sphere {<0,5,10>,.5 pigment {color White} finish{diffuse .7}}
sphere {<6,6,10>,.5 pigment {color White} finish{diffuse .7}}
sphere {<7,4,10>,.5 pigment {color White} finish{diffuse .7}}
sphere {<4,8,10>,.5 pigment {color White} finish{diffuse .7}}
sphere {<3,3,10>,.5 pigment {color White} finish{diffuse .7}}
sphere {<2,5,10>,.5 pigment {color White} finish{diffuse .7}}
sphere {<5,1,10>,.5 pigment {color White} finish{diffuse .7}}
sphere {<2,2,10>,.5 pigment {color White} finish{diffuse .7}}
sphere {<3,-3,10>,.5 pigment {color White} finish{diffuse .7}}
```

```
#include <iostream.h>
```

```
#include "global.h"
```

```
ostream& operator<<(ostream& o, texture& t) {t.out(o); return o;}
```

```
#include <iostream.h>
#include <math.h>

#include "global.h"
#include "voronoi.h"                                     // D-DIM. VORONOI TEMPLATE

struct cell : public VECTOR<3> {                        // 3-DIMENSIONAL VORONOI-CELL
    vector p;                                           // POSITION OF PARTICLE
    list<cell*> ln;                                     // CONTIGUOUS CELLS
    list<object*> lh;                                   // HOST OBJECT(S)
    list<object*> lo;                                   // LIST OF INTERSECTING OBJECTS
    long t;                                             // LOCAL TRAVERSE CODE (WORK)
    cell *b;                                            // BACK (voronoi::disperse)
    cell() {t=0L;}
    cell(vector& v, object *o=(object*)0) {
        double x[3]; x[0]=v[0]; x[1]=v[1]; x[2]=v[2];
        *((VECTOR<3>*)this)=VECTOR<3>(x); p=v; {if(o)lh+=o;} t=0L;
    }
    int operator&(vector& p) {                          // CELL CONTAINS POINT p
        for(cell*n=ln.first();n;n=ln.next())
            if(!(halfspace(this->p,n->p)&p)) return 0;
        return 1;
    }
    int operator&(object& o) {                          // CELL INTERSECTS OBJECT o
        for(cell*n=ln.first();n;n=ln.next())
            if(!(o&halfspace(this->p,n->p))) return 0;
        return 1;
    }
};

// GLOBAL METHODS (NOT REALLY AN OBJECT ORIENTED APPROACH!)

static void initcells(VERTEX<3>*v) {                  // VORONOI-CELLS FROM VORONOI<3>
    for(register int i=0; i<3; i++) {
        cell *ci=(cell*)(v->p[i]);
        for(register int j=i+1; j<4; j++) {
            cell *cj=(cell*)(v->p[j]);
            if(!ci->ln[cj]) ci->ln+=cj;
            if(!cj->ln[ci]) cj->ln+=ci;
        }
    }
}

static int lexcmp(const void *v1, const void *v2) {
    cell *c1=(cell**)v1; cell *c2=(cell**)v2;
    const double EPS=1e-6;
    for(register int i=0; i<3; i++) {
        double d=(*c1)[i]-(*c2)[i];
        if(d<=-EPS) return -1;
        if(d>EPS) return 1;
    }
    return 0;
}

// METHODS OF class voronoi

void voronoi::disperse(object *o, cell *C) {
    traverse++;
    if(traverse==--1L) {                                // OVERFLOW
        traverse=-3L; disperse((object*)0,C);          // REINITIALIZE
        traverse=0L;
    }
}
```

```
    }
    cell *c=C; c->b=(cell*)0; // PARTICULAR CASE
    cell *n=c->ln.first(); // ACTUAL NEIGHBOR
    register int back=0; // DON'T STEP BACK YET
    for(;;) { // ITERATIVE TRAVERSE
        c->t=traverse; // MARK AS TRAVERSED
        register int go=0; // DON'T GO YET
        do { // ACTION ON ACTUAL c
            if(back) { // STEP BACKWARDS
                cell *b=c->b; // FROM WHERE WE CAME
                if(o) c->lo+=o; // PERFORM ACTION
                c=b; back=0; continue; // TAKE BACK CELL
            }
            if(n==c->b) continue; // DON'T STEP BACK YET
            if(n->t==traverse) continue; // ALREADY TRAVERSED
            if(o && !(*n&(*o))) continue; // PARTIAL TRAVERSE
            go=1; break; // GO ON
        } while(n=c->ln.next()); // UNTIL NOT ALL DONE
        if(go) { // STEP FORWARDS
            n->b=c; // ESTABLISH BACK LINK
            c=n; n=c->ln.first(); continue; // LET'S GO AHEAD
        }
        if(c==C) break; // RETURNED
        back=1; // GO BACK IF NO BETTER
    }
    if(o) C->lo+=o; C->t=traverse; // C IS THE LAST ONE
}
```

```
void voronoi::preprocess(list<object*> *lo) { // PREPROCESSING
    list<cell*> *lc=new list<cell*>;
    register int n=0;
    for(object* o=lo->first(); o; o=lo->next()) { // OBJECTS
        list<vector*> *lp=o->particles();
        for(vector* p=lp->first(); p; p=lp->next())
            {(*lc)+=new cell(*p,o); n++; delete p;}
        delete lp;
    }
    for(o=lightsources.first(); o; o=lightsources.next()) { // LIGHTSRC.S
        list<vector*> *lp=o->particles();
        for(vector* p=lp->first(); p; p=lp->next())
            {(*lc)+=new cell(*p); n++; delete p;}
        delete lp;
    }
    (*lc)+=new cell(actcamera.getray(0.,0.).o); n++; // EYE
    cell **C=new cell*[n];
    n=0;
    for(cell* c=lc->first(); c; c=lc->next()) C[n++]=c;
    delete lc;
    qsort((void*)C, (size_t)n, sizeof(cell*), lexcmp); // LEXICOGRAPHIC
    list<VECTOR<3>*> *lv=new list<VECTOR<3>*>;
    (*lv)+=C[0]; cell* p=C[0];
    vector cm=C[0]->p; // CENTER OF MASS
    for(register int i=1; i<n; i++) {
        cm=cm+C[i]->p;
        if(lexcmp((const void*)&C[i], // MULTIPLE PARTICLE
            (const void*)&p)==0) {
            for(o=C[i]->lh.first(); o; o=C[i]->lh.next())
                p->lh+=o;
            delete C[i];
        } else {(*lv)+=C[i]; p=C[i];} // SINGLE (YET)
    }
}
```

```

    cm=cm/(double)n; // CENTER OF MASS
    delete C;
    VECTOR<3>*b[4]; for(i=0;i<4;i++) b[i]=new cell; // BOUNDARY CELLS
    VORONOI<3> V(lv,b); // VERTEX STRUCTURE
    for(i=0;i<4;i++) { // BOUNDARY CELLS
        cell* c=(cell*)b[i];
        c->p=vector((*c)[0],(*c)[1],(*c)[2]); // UPDATE
    }
    V(initcells); // 3D VORONOI-CELLS
    traverse=0L; // INITIALIZE disperse
    double ddmin; register int first=1; // INITIALIZE SEARCH
    for(c=(cell*)lv->first();c;c=(cell*)lv->next()) {
        for(o=c->lh.first();o;o=c->lh.next()) // BUILD OBJECT LISTS
            disperse(o,c); // PARTIAL TRAVERSE
        vector u=c->p-cm; double dd=u%u; // DISTANCE^2 FROM cm
        if(first || dd<ddmin) // FIND CLOSEST TO cm
            {ddmin=dd;this->C=c;}
        first=0;
    }
    n=1<<(lmax+1);
    s=new cell*[n]; for(i=0;i<n;i++) s[i]=this->C; // START FOR firstlist
    delete lv; // ???
}

```

```

void voronoi::step() { // ONE STEP ALONG RAY r
    double tmin=0.; cell* nmin=(cell*)0; // INITIALIZE SEARCH
    for(cell*n=a->ln.first(); n; n=a->ln.next()) { // NEIGHBORING CELLS
        vector u=n->p-a->p; // NORMAL OF BISECTOR
        double d=r->d%u; // DENOMINATOR
        if(fabs(d)<EPS) continue; // PARALLEL FACE
        vector v=(n->p+a->p)*.5-r->o;
        double t=(v%u)/d; // t AT INTERSECTION
        if(t<=this->t) continue; // WOULD BE A BACK STEP
        if(t<EPS) // r->o ON BISECTOR PL.
            if(u%r->d>0.) {tmin=EPS; nmin=n; continue;}
            else continue;
        if(!tmin || t<tmin) {tmin=t; nmin=n;} // CLOSER INTERSECTION
    }
    this->t=tmin; a=nmin;
}

```

```

list<object*>* voronoi::firstlist(ray& r) {
    register int i=r.c>=0?r.c:(1<<(lmax+1))-1; // INDEX INTO s
    a=s[i]; // INITIALIZE step()
    if(!((*a)&r.o)) { // COHERENCE FAILED
        ray R(a->p,norm(r.o-a->p),0,0); // PATH OF WALK
        this->r=&R; t=0.; // INITIALIZE step()
        for(step(); a; step()) // WALK
            if((*a)&r.o) {s[i]=a; break;} // SUCCESS
    }
    this->r=&r; t=0.; // INITIALIZE step()
    return a?&a->lo:(list<object*>*)0;
}

```

```

list<object*>* voronoi::nextlist() {
    step(); // ONE STEP
    return a?&a->lo:(list<object*>*)0; // SUCCESS OR FAIL
}

```



```
//      PERMUTATION TEMPLATE (USED IN GAUSSIAN ELIMINATION)

template <int D> class PERMUTATION {
    int n[D];                                     // ELEMENTS
public:
    PERMUTATION() {for(register int i=0; i<D; i++) n[i]=i;}
    int operator[](int i) {return n[i];}
    void operator()(int i, int j) {               // SWAP
        if(i==j) return;
        register int t=n[i]; n[i]=n[j]; n[j]=t;
    }
};

//      D-DIMENSIONAL VECTOR TEMPLATE

template <int D> class VECTOR {
    friend ostream& operator<<(ostream& o, VECTOR<D>& v);
    double x[D];                                 // COORDINATES
public:
    VECTOR() {for(register int i=0; i<D; i++) x[i]=0.;}
    VECTOR(double x[D]) {for(register int i=0; i<D; i++) this->x[i]=x[i];}
    VECTOR(double x[D], PERMUTATION<D>& p) {
        for(register int i=0; i<D; i++) this->x[i]=x[p[i]];
    }
    double operator[](int i) {return x[i];}
    void operator--=(VECTOR<D>& v) {
        for(register int i=0; i<D; i++) x[i]-=v.x[i];
    }
    VECTOR<D> operator*(double d) {
        VECTOR<D> w; for(register int i=0; i<D; i++) w.x[i]=x[i]*d;
        return w;
    }
    VECTOR<D> operator/(double d) {
        VECTOR<D> w; for(register int i=0; i<D; i++) w.x[i]=x[i]/d;
        return w;
    }
    double operator*(VECTOR<D>& v) {
        double d=0.; for(register int i=0; i<D; i++) d+=x[i]*v.x[i];
        return d;
    }
    VECTOR<D> operator+(VECTOR<D>& v) {
        VECTOR<D> w; for(register int i=0; i<D; i++) w.x[i]=x[i]+v.x[i];
        return w;
    }
    VECTOR<D> operator-(VECTOR<D>& v) {
        VECTOR<D> w; for(register int i=0; i<D; i++) w.x[i]=x[i]-v.x[i];
        return w;
    }
};

//      D-DIMENSIONAL SQUARE MATRIX TEMPLATE

template <int D> class MATRIX {
    friend ostream& operator<<(ostream& o, MATRIX<D>& A);
    VECTOR<D> a[D];                               // ROWS
public:
    MATRIX(VECTOR<D> a[D]) {for(register int i=0; i<D; i++) this->a[i]=a[i];}
    MATRIX(double a[D][D]) {
        for(register int i=0; i<D; i++) this->a[i]=VECTOR<D>(a[i]);
    }
    VECTOR<D> operator*(VECTOR<D>& x) {
```

```

        double y[D];
        for(register int i=0; i<D; i++) y[i]=a[i]*x;
        return VECTOR<D>(y);
    }
// int operator()(VECTOR<D>& x, VECTOR<D>& b) {      // SOLVE (*this)x=b
//     GAUSSIAN ELIMINATION METHOD
//     const double EPS=1e-10;
//     VECTOR<D> B[D]; double c[D]; PERMUTATION<D> p;
//     register int i, j, k;
//     for(i=0; i<D; i++) {B[i]=a[i]; c[i]=b[i];}      // COPY
//     for(i=0; i<D; i++) {                            // THROUGH ROWS
//         double a, amax=0., e, emain;
//         for(j=i; j<D; j++)                            // MAIN ELEMENT
//             if((a=fabs(e=B[p[j]][i]))>amax)
//                 {emain=e; amax=a; k=j;}
//         if(amax<EPS) return 0;                        // SINGULAR
//         p(i,k);                                        // SWAP
//         for(j=i+1; j<D; j++) {                        // NULL BELOW
//             double s=B[p[j]][i]/emain;
//             B[p[j]]-=B[p[i]]*s;
//             c[p[j]]-=c[p[i]]*s;
//         }
//     }
//     for(i=D-1; i>=0; i--) {                            // BUILD SOLUTION
//         for(j=D-1; j>i; j--) c[p[i]]-=B[p[i]][j]*c[p[j]];
//         c[p[i]]/=B[p[i]][i];
//     }
//     x=VECTOR<D>(c,p); return 1;
// }
};

// VORONOI-VERTEX TEMPLATE

template <int D> struct VERTEX {
    VECTOR<D>* p[D+1];      // FORMING POINTS
    VERTEX<D>* v[D+1];      // CONTIGUOUS VERTICES
    VECTOR<D> c;            // POSITION
    double rr;              // RADIUS SQUARE
    int b;                  // BACK INDEX (WORK)
    int i;                  // ACTUAL INDEX (WORK)
    long t;                 // TRAVERSE CODE (WORK)
private:
    void initialize(VECTOR<D>* f[D+1]) {
        register int i;
        for(i=0; i<D+1; i++) {p[i]=f[i]; v[i]=(VERTEX<D>*)0;}
        this->b=-1; this->i=0; this->t=0L;
        VECTOR<D> A[D]; double b[D];
        for(i=0; i<D; i++) {
            A[i]=(*f[i+1])-(f[i]);
            b[i]=(((f[i+1])+(f[i]))*0.5)*A[i];
        }
        if(!MATRIX<D>(A)(c,VECTOR<D>(b))) {      // EQUATION A*c=b
            rr=-1.;                               // DEGENERATE
            return;
        }
        rr=(*p[0]-c)*(*p[0]-c);
    }
public:
    VERTEX(VECTOR<D>* f[D+1]) {initialize(f);}      // FORMING POINTS
    VERTEX(VECTOR<D> *q, VERTEX<D> *v, int i) {    // POINT q AND RING i
        VECTOR<D> *f[D+1]; f[0]=q;
    }
};

```

```
        for(register int j=1; j<D+1; j++) f[j]=v->p[(j+i)%(D+1)];
        initialize(f);
    }
};

//      VORONOI-DIAGRAM TEMPLATE

template <int D> class VORONOI {
    friend ostream& operator<<(ostream& o, VORONOI<D>& v);
    VECTOR<D> C; // CENTROID
    VECTOR<D> *b[D+1], B[D+1]; // BOUNDING SIMPLEX
    VERTEX<D> *c; // CLOSEST TO CENTROID
    double ll(VECTOR<D>& v) {return v*v;} // LENGTH SQUARE
    double dd(VECTOR<D>& v, VECTOR<D>& w) { // DISTANCE SQUARE
        VECTOR<D> d=v-w; return d*d;
    }
    void normals(int d, double n[D+1][D]) { // NORMAL VECTORS FOR bound()
        register int i, j;
        if(d==2) {
            n[0][0]=1.0; n[0][1]=1.0;
            n[1][0]=-1.0; n[1][1]=1.0;
            n[2][0]=0.0; n[2][1]=-1.0;
            return;
        }
        normals(d-1, n);
        for(i=0; i<d; i++) n[i][d-1]=1.0;
        for(i=0; i<d-1; i++) n[d][i]=0.0;
        n[d][d-1]=-1.0;
    }
    void bound(list<VECTOR<D>*>* l); // BUILD BOUNDING SIMPLEX
    VECTOR<D>* q; // ACTUAL POINT
    list<VERTEX<D>*> *ld; // VERTICES TO DELETE
    void find(); // FIND A VERTEX TO DELETE
    void search(); // FIND ALL VERTICES TO DELETE
    list<VERTEX<D>*> *ln; // NEW VERTICES
    void create(); // CREATE NEW VERTICES
    int samering(VERTEX<D>*v, int iv, VERTEX<D>*w, int iw) {
        for(register int i=(iv+1)%(D+1); i!=iv; i=(i+1)%(D+1)) {
            VECTOR<D> *p=v->p[i];
            for(register int j=(iw+1)%(D+1); j!=iw; j=(j+1)%(D+1))
                if(w->p[j]==p) {j=-1; break;}
            if(j>=0) return 0;
        }
        return 1;
    }
    void link(); // LINK NEW VERTICES TO EACH O.
    void build(list<VECTOR<D>*>* l) { // DISJOINT PARTICLES
        traverse=0L;
        bound(l);
        for(VECTOR<D>* p=l->first(); p; p=l->next())
            {q=p; find(); search(); create(); link();}
    }
    long traverse; // TRAVERSE CODE
    static void free(VERTEX<D>*v){delete v;} // FOR DESTRUCTOR
    static void donothing(VERTEX<D>*v){} // FOR REINITIALIZE traverse
public:
    VORONOI(list<VECTOR<D>*>* l) {
        for(register int i=0; i<D+1; i++) b[i]=&B[i];
        build(l);
    }
    VORONOI(list<VECTOR<D>*>* l, VECTOR<D> *b[D+1]) {
```

```
        for(register int i=0; i<D+1; i++) this->b[i]=b[i];
        build(1);
    }
    void operator()(void (*f)(VERTEX<D>* v));           // TRAVERSE VERTICES
    ~VORONOI() {(*this)(free);}                          // TRAVERSE AND DELETE
};

template <int D> void VORONOI<D>::bound(list<VECTOR<D>*>* l) {
    register int i, j;

    //      NORMAL VECTORS FOR FACES OF BOUNDING SIMPLEX

    double a[D+1][D]; normals(D, a);
    VECTOR<D> n[D+1];
    for(i=0; i<D+1; i++) n[i]=VECTOR<D>(a[i])/sqrt(11(VECTOR<D>(a[i])));

    //      MAXIMAL DISTANCES IN DIRECTION OF NORMALS

    double d, dmax[D+1], dmin[D+1]; register int first=1;
    for(VECTOR<D>* p=l->first(); p; p=l->next()) {
        for(i=0; i<D+1; i++) {
            d=n[i]*(*p);
            if(first || d>dmax[i]) dmax[i]=d;
            if(first || d<dmin[i]) dmin[i]=d;
        }
        first=0;
    }

    //      VERTICES OF BOUNDING SIMPLEX (INTERSECT FACES CYCLICALLY)

    for(i=0; i<D+1; i++) dmax[i]+=(dmax[i]-dmin[i])*0.5;           // INACCURACY
    VECTOR<D> A[D]; double t[D];
    for(i=0; i<D+1; i++) {
        for(j=0; j<D; j++) {
            A[j]=n[(i+j)%(D+1)];
            t[j]=dmax[(i+j)%(D+1)];
        }
        (void)MATRIX<D>(A)(*b[i],VECTOR<D>(t)); // EQUATION A*b[i]=t
    }

    //      CENTRAL VERTEX AND CENTROID

    c=new VERTEX<D>(b);
    for(i=0; i<D+1; i++) C=C+(*b[i]);
    C=C*(1./(double)(D+1));
}

template <int D> void VORONOI<D>::find() {
    register int i, j;
    double P[D+1][D+1]; for(j=0; j<D+1; j++) P[D][j]=1.;
    double q[D+1]; for(j=0; j<D; j++) q[j]=(*this->q)[j]; q[D]=1.;
    VERTEX<D> *v=c;
    VECTOR<D+1> a;                                           // BARICENTRIC COORDINATES OF q
    for(;;) {
        for(i=0; i<D; i++) for(j=0; j<D+1; j++) P[i][j]=(*v->p[j])[i];
        (void)MATRIX<D+1>(P)(a,VECTOR<D+1>(q));           // SOLVE P*a=q
        double aminus=0.;
        for(j=0; j<D+1; j++) if(a[j]<aminus) {aminus=a[j]; i=j;}
        if(aminus<0.) {v=v->v[i]; continue;}
        break;                                           // q INSIDE
    }
}
```

```

    ld=new list<VERTEX<D>*>; *ld+=v;
}

template <int D> void VORONOI<D>::search() {
    VERTEX<D> *vstart=ld->first(), *v=vstart; v->b=-1;
    register int back=0;
    for(;;) {
        register int go=0, i; VERTEX<D> *n;
        do {
            if(back) {
                // STEP BACKWARDS
                *ld+=v;
                i=v->b; v->b=-1; v=v->v[i]; back=0; continue;
            }
            if(v->i==v->b) continue;
            if(v->v[v->i]==(VERTEX<D>*)0) continue;
            n=v->v[v->i]; // NEIGHBOR
            if(n->b>=0) continue; // ALREADY TRAVERSED
            if((*ld)[n]) continue; // ALREADY ON LIST
            if(dd(*q,n->c)<n->rr) go=1; // GO IF q IN SPHERE
            if(go) break;
        } while((v->i=(v->i+1)%(D+1))!=0);
        if(go) {
            // STEP FORWARDS
            for(i=0; i<D+1; i++) // COMPUTE BACK INDEX
                if(v==n->v[i]) {n->b=i; break;}
            v=n; continue;
        }
        if(v==vstart) break;
        back=1;
    }
}

template <int D> void VORONOI<D>::create() {
    VERTEX<D> *v;
    ln=new list<VERTEX<D>*>;
    for(v=ld->first(); v; v=ld->next()) {
        // TAKE VERTICES
        for(register int i=0; i<D+1; i++) {
            // TAKE NEIGHBORS
            VERTEX<D> *n=v->v[i];
            if((*ld)[n]) continue; // ALSO DELETED
            VERTEX<D> *m=new VERTEX<D>(q,v,i); // POINT q + RING i
            if(m->rr<0.)
                {delete m;*ld+=n;continue;} // DEGENERACY
            *ln+=m; // STORE
            register int j;
            for(j=0; j<D+1; j++)
                if(m->p[j]==q) m->v[j]=n; // OUTER LINK
            if(n==(VERTEX<D>*)0) continue; // NO REAL NEIGHBOR
            for(j=0; j<D+1; j++)
                if(n->v[j]==v) n->v[j]=m; // BACK LINK
        }
    }
    if((*ld)[c]) {
        // NEW c NEEDED
        double d, ddmin; register int first=1;
        for(v=ln->first(); v; v=ln->next()) {
            d=dd(v->c,C);
            if(first || d<ddmin) {c=v; ddmin=d;}
            first=0;
        }
    }
    for(v=ld->first(); v; v=ld->next()) {delete v;} // DELETE VERTICES
    delete ld;
}

```

```

template <int D> void VORONOI<D>::link() {
    register int i, j, n, iv, iw;
    VERTEX<D> *v, *w;
    n=0; for(v=ln->first(); v; v=ln->next()) n++;
    VERTEX<D> *N[n];
    n=0; for(v=ln->first(); v; v=ln->next()) N[n++]=v;
    for(i=0; i<n-1; i++) {
        v=N[i];
        for(j=i+1; j<n; j++) {
            w=N[j];
            for(iv=0; iv<D+1; iv++) {
                for(iw=0; iw<D+1; iw++) {
                    if(samering(v,iv,w,iw))
                        {v->v[iv]=w; w->v[iw]=v;}
                }
            }
        }
    }
    delete ln;
}

template <int D> void VORONOI<D>::operator()(void (*f)(VERTEX<D>* v)) {
    traverse++;
    if(traverse==--1L) {
        traverse=-3L; (*this)(donothing); // OVERFLOW
        traverse=0L; // REINITIALIZE
    }
    VERTEX<D> *v=c; v->b=D+1; // PARTICULAR CASE
    register int back=0;
    for(;;) { // ITERATIVE TRAVERSE
        v->t=traverse; // MARK AS TRAVERSED
        register int go=0; // DON'T GO YET
        VERTEX<D> *n; // ACTUAL NEIGHBOR
        do { // ACTION ON ACTUAL v
            if(back) { // STEP BACKWARDS
                VERTEX<D>*n=v->v[v->b]; // FROM WHERE WE CAME
                v->b=-1; // FOR NEXT USAGE
                f(v); // PERFORM ACTION
                v=n; back=0; continue; // TAKE BACK VERTEX
            }
            if(v->i==v->b) continue; // DON'T STEP BACK YET
            if(v->v[v->i]==(VERTEX<D>*)0)
                continue; // NO REAL NEIGHBOR
            n=v->v[v->i]; // WHERE WE SHOULD GO
            if(n->t==traverse) continue; // ALREADY TRAVERSED
            go=1; break; // GO ON
        } while((v->i=(v->i+1)%(D+1))!=0); // UNTIL NOT ALL DONE
        if(go) { // STEP FORWARDS
            for(register int i=0; i<D+1; i++) // FIND BACK LINK
                if(v==n->v[i])
                    {n->b=i; break;} // BOOK
            v=n; continue; // LET'S GO
        }
        if(v==c) break; // RETURNED
        back=1; // GO BACK IF NO BETTER
    }
    f(c); c->b=-1; c->t=traverse; // THE LAST ONE
}

```

```
/*
Fast Bitmap Stretching
Tomas Mšller
*/

/* user provided routines */

short ReadPixel(long x,long y);/* returns color of (x,y) in source bitmap*/
void SetColor(short Col); /* set the current writing color to Col */
void WritePixel(long x,long y); /* write a pixel at (x,y) in destination bitmap with the
current writing color */

#define sign(x) ((x)>0 ? 1:-1)

void RectStretch(long xs1,long ys1,long xs2,long ys2,long xd1,long yd1,long xd2,long yd2)
;
void Stretch(long x1,long x2,long y1,long y2,long yr,long yw) ;
void CircleStretch(long SBMINX,long SBMAXX,long xc,long yc,long r) ;
void Stretch2Lines(long x1,long x2,long y1,long y2,long yr1,long yw1,long yr2,long yw2) ;

/*****
RectStretch enlarges or diminishes a source rectangle of
a bitmap to a destination rectangle. The source
rectangle is selected by the two points (xs1,ys1) and
(xs2,ys2), and the destination rectangle by (xd1,yd1) and
(xd2,yd2). Since readability of source-code is wanted,
some optimizations have been left out for the reader:
It's possible to read one line at a time, by first
stretching in x-direction and then stretching that bitmap
in y-direction.
Entry:
    xs1,ys1 - first point of source rectangle
    xs2,ys2 - second point of source rectangle
    xd1,yd1 - first point of destination rectangle
    xd2,yd2 - second point of destination rectangle
*****/
void RectStretch(long xs1,long ys1,long xs2,long ys2,long xd1,long yd1,long xd2,long yd2)
{
    long dx,dy,e,d,dx2;
    short sx,sy;
    dx=abs((int)(yd2-yd1));
    dy=abs((int)(ys2-ys1));
    sx=sign(yd2-yd1);
    sy=sign(ys2-ys1);
    e=(dy<<1)-dx;
    dx2=dx<<1;
    dy<=1;
    for(d=0;d<=dx;d++)
    {
```

```
        Stretch(xd1,xd2,xs1,xs2,ys1,yd1);
        while(e>=0)
        {
            ys1+=sy;
            e-=dx2;
        }
        yd1+=sx;
        e+=dy;
    }
}
```

```
}
```

```
/******
```

Stretches a horizontal source line onto a horizontal destination line. Used by RectStretch.

Entry:

x1,x2 - x-coordinates of the destination line  
y1,y2 - x-coordinates of the source line  
yr - y-coordinate of source line  
yw - y-coordinate of destination line

```
*****/
```

```
void Stretch(long x1,long x2,long y1,long y2,long yr,long yw)
```

```
{
```

```
    long dx,dy,e,d,dx2;
    short sx,sy,color;
    dx=abs((int)(x2-x1));
    dy=abs((int)(y2-y1));
    sx=sign(x2-x1);
    sy=sign(y2-y1);
    e=(dy<<1)-dx;
    dx2=dx<<1;
    dy<=<=1;
    for(d=0;d<=dx;d++)
    {
        color=ReadPixel(y1,yr);
        SetColor(color);
        WritePixel(x1,yw);
        while(e>=0)
        {
            y1+=sy;
            e-=dx2;
        }
        x1+=sx;
        e+=dy;
    }
}
```

```
}
```

```
/******
```

CircleStretch stretches a source rectangle, selected by the two points (SBMINX,0) and (SBMAXX,2\*r-1), onto a Bresenham circle at (xc,yc) with radius=r. Instead of writing pixels on the circle, horizontal lines of the source rectangle are being stretched onto all horizontal lines of the circle.

Entry:

SBMINX - min x of source rectangle  
SBMAXX - max x of source rectangle  
xc,yc - center of the circle  
r - radius of circle

```
*****/
```

```
void CircleStretch(long SBMINX,long SBMAXX,long xc,long yc,long r)
```

```
{
```



```
long p=3-(r<<1),x=0,y=r;
while(x<y)
{
    /* stretch lines in first octant */
    Stretch2Lines(xc-y,xc+y,SBMINX,SBMAXX,r-x,yc-x,r+x,yc+x);
    if(p<0) p=p+(x<<2)+6;
    else
    {
        /* stretch lines in second octant */
        Stretch2Lines(xc-x,xc+x,SBMINX,SBMAXX,r-y,yc-y,r+y,yc+y);
        p=p+((x-y)<<2)+10;
        y--;
    }
    x++;
}
if(x==y) Stretch2Lines(xc-x,xc+x,SBMINX,SBMAXX,r-y,yc-y,r+y,yc+y);
}
```

\*\*\*\*\*

Stretch2Lines stretches two source lines with same length and different y-coordinates onto two destination lines with same length and different y-coordinates. Used by CircleStretch.

Entry:

```
x1,x2 - x-coordinates of the destination line
y1,y2 - x-coordinates of the source line
yr1    - y-coordinate of source line # 1
yw1    - y-coordinate of destination line # 1
yr2    - y-coordinate of source line # 2
yw2    - y-coordinate of destination line # 2
```

\*\*\*\*\*/

```
void Stretch2Lines(long x1,long x2,long y1,long y2,long yr1,long yw1,long yr2,long yw2)
{
```





```
    long dx,dy,e,d,dx2;
    short sx,sy,color;
    dx=abs((int)(x2-x1));
    dy=abs((int)(y2-y1));
    sx=sign(x2-x1);
    sy=sign(y2-y1);
    e=(dy<<1)-dx;
    dx2=dx<<1;
    dy<=<1;
    for(d=0;d<=dx;d++)
    {
        color=ReadPixel(y1,yr1);
        SetColor(color);
        WritePixel(x1,yw1);
        color=ReadPixel(y1,yr2);
        SetColor(color);
        WritePixel(x1,yy.

        while(e>=0)
        {
            y1+=sy;
            e-=dx2;
        }
        x1+=sx;
        e+=dy;
    }
}
```

}

# Index of

## /pubs/tog/GraphicsGems/gemsv/ch5-3/

Name	Last modified	Size	Description
 <a href="#">_Parent Directory</a>			
 <a href="#">_rayscan.cc</a>	29-Jun-00 08:23	10K	
 <a href="#">_vector-bookver.h</a>	29-Jun-00 08:23	4K	
 <a href="#">_vector.h</a>	29-Jun-00 08:23	4K	

```

/*****
 * Scanline-rejection routines *
 * for spheres and polygons.   *
 * By Tomas Moller             *
 *****/
#include "vector.h"

#define EPSILON 1e-6           /* a small number */
#define INFINITY 1e8           /* a large number */
enum StatusFlag {OnScanline,OffScanline,Interval,NeverAgain};

struct FirstHitStatus
{
    StatusFlag Flag;           /* se enum above */
    short IntervalMin,IntervalMax; /* the interval */
};

/*****
 * SphereComputeFirstHitStatus - computes the first hit status
 * for a sphere and the Interval (if any) for the scanline-plane
 * given by the first two parameters.
 *
 * Entry:
 *   Ns - normal of the scanline-plane
 *   ds - "d-value" for the scanline-plane
 *   LeftMost - the leftmost point on the scanline. Referred to
 *             as L in the text.
 *   ScanLineDir - the direction of the scanline in space.
 *             Observe that it is constructed by subtracting two
 *             adjacent points on the scanline from each other.
 *   width - number of pixels per scanline
 *   Ui,Vi - the indices to the uv-plane. 0==x,1==y,2==z
 *   Eyepos - the position of the eye (or camera)
 *   Sphcen - the centre of the sphere
 *   Sr - the radius of the sphere
 *
 * The function returns a struct FirstHitStatus with all
 * necessary information.
 *****/
struct FirstHitStatus SphereComputeFirstHitStatus(Vector &Ns,float ds,
    Vector LeftMost,Vector ScanLineDir,int width,int Ui,int Vi,
    Vector Eyepos,Vector Sphcen,float Sr)
{
    struct FirstHitStatus FHstatus;
    float signed_dist=Ns*Sphcen+ds;
    if(signed_dist>Sr) FHstatus.Flag=NeverAgain; /* sphere is above plane */
    else if(signed_dist<-Sr) FHstatus.Flag=OffScanline; /* below plane */
    else
    {
        Vector D,H,Cc,Nd; /* Cc=Circle Origo */
        float sintheta,costheta,centerdist;
        float cr2,d2,t1,t2,t3,V1v,V1u,V2v,V2u; /* squared circle radius */
        Cc=Sphcen-Ns*signed_dist;
        cr2=Sr*Sr-signed_dist*signed_dist; /* the * is dot-product */
        D=Cc-Eyepos;
        d2=D*D; /* D dot D=squared length of D */
        if(d2<=cr2)
        { /* we are inside the sphere */
            FHstatus.Flag=OnScanline;
            return FHstatus;
        }
    }
}

```

```

/* Macro used by PolyComputeFirstHitStatus. It projects
 * the point (x,y,z) onto the scanline and saves the
 * result in [realIntervalMin,realIntervalMax].
 */

```

\\  
\\  
\\  
\\  
\\  
\\  
\\  
\\  
\\  
\\  
\\  
\\

```
}
denom=ScanLineDir[Ui]*(Eyepos[Vi]-v)-ScanLineDir[Vi]*(Eyepos[Ui]-u);
if(denom==0.0)
{
    if(ScanLineDir[Ui]!=0.0) alpha=(u-Eyepos[Ui])/ScanLineDir[Ui];
    else alpha=(v-Eyepos[Vi])/ScanLineDir[Vi];
    if(alpha>0.0) realIntervalMax=width-1;
    else realIntervalMin=0;
}
else
{
    alpha=(Eyepos[Ui]-LeftMost[Ui])*(Eyepos[Vi]-v);
    alpha-=(Eyepos[Ui]-u)*(Eyepos[Vi]-LeftMost[Vi]);
    alpha/=denom;
    if(alpha>realIntervalMax) realIntervalMax=alpha;
    if(alpha<realIntervalMin) realIntervalMin=alpha;
}

/*****
* PolyComputeFirstHitStatus - computes the first hit status for a
* polygon and the Interval (if any) for the scanline-plane
* given by the first two parameters.
*
* Entry:
*   Ns - normal of the scanline-plane
*   ds - "d-value" for the scanline-plane
*   LeftMost - the leftmost point on the scanline. Referred to
*   as L in the text.
*   ScanLineDir - the direction of the scanline in space.
*   Observe that it is constructed by subtracting two
*   adjacent points on the scanline from each other.
*   width - number of pixels per scanline
*   Ui,Vi - the indices to the uv-plane. 0==x,1==y,2==z
*   Eyepos - the position of the eye (or camera)
*   x,y,z - the points of the polygon
*   NrVert - Number of vertices
*
* The function returns a struct FirstHitStatus with all
* necessary information.
*****/
struct FirstHitStatus PolyComputeFirstHitStatus(Vector Ns,float ds,
    Vector LeftMost,Vector ScanLineDir,int width,int Ui,int Vi,
    Vector Eyepos,float *x,float *y,float *z,short NrVert)
{
    struct FirstHitStatus FHstatus;
    Vector isect,dir;
    float dist,prevdist=0,denom,alpha,u,v;
    float realIntervalMax=-INFINITY,realIntervalMin=INFINITY;
    prevdist=Ns.X()*x[NrVert-1]+Ns.Y()*y[NrVert-1]+Ns.Z()*z[NrVert-1]+ds;
    /* start with last point */
    for(int i=0;i<NrVert;i++)
    {
        dist=Ns.X()*x[i]+Ns.Y()*y[i]+Ns.Z()*z[i]+ds;
        if(dist==0.0)
        {
            /* point i is on the plane, project it on the scanline */
            PROJECTPOINT(x[i],y[i],z[i]);
        }
        else if((prevdist<0.0 && dist>0.0) || (prevdist>0.0 && dist<0.0))
        /* intersection */
        {
            isect.Set(x[i],y[i],z[i]);

```

```
        if(i==0)
        {
            dir.SetX(x[NrVert-1]-x[0]);
            dir.SetY(y[NrVert-1]-y[0]);
            dir.SetZ(z[NrVert-1]-z[0]);
        }
        else dir.Set(x[i-1]-x[i],y[i-1]-y[i],z[i-1]-z[i]);
        alpha=(-ds-Ns*isect)/(Ns*dir);
        isect+=dir*alpha;          /* intersection point calculated */
        PROJECTPOINT(isect.X(),isect.Y(),isect.Z());
    }
    prevdist=dist;
}
if(realIntervalMax==--INFINITY)          // no intersection
{
    if(dist>0) FHstatus.Flag=NeverAgain;
    else FHstatus.Flag=OffScanline;
}
else if(realIntervalMax<0 || realIntervalMin>=width)
    FHstatus.Flag=OffScanline;
else
{
    FHstatus.IntervalMax=(int)floor(realIntervalMax);
    FHstatus.IntervalMin=(int)ceil(realIntervalMin);
    if(FHstatus.IntervalMax>=width) FHstatus.IntervalMax=width-1;
    if(FHstatus.IntervalMin<0) FHstatus.IntervalMin=0;
    if(FHstatus.IntervalMin==0 && FHstatus.IntervalMax==width-1)
        FHstatus.Flag=OnScanline;
    else FHstatus.Flag=Interval;
}
if(FHstatus.IntervalMin>FHstatus.IntervalMax) FHstatus.Flag=OffScanline;
return FHstatus;
}
```

```

/*****
* vector.h - a vector class written in c++
* functions for +, -, dotproduct, crossproduct, scaling,
* length & normalizing, many of these are operators
* By Tomas Moller
*****/
#ifndef VECTOR_H
#define VECTOR_H

#include <stream.h>
#include <string.h>
#include <math.h>

#define Xi 0          // indices into vector
#define Yi 1
#define Zi 2

class Vector
{
protected:
    float fx,fy,fz;
public:
    Vector() {fx=0.0;fy=0.0;fz=0.0;} // constructor with no argument
    Vector(float x,float y,float z); // constructor with coords
    Vector(Vector& a);               // constructor with vector
    void Set(float x,float y,float z); // assign new values to vector
    void SetX(float x);               // set x
    void SetY(float y);               // set y
    void SetZ(float z);               // set z;
    void SetIndex(int index,float value);
    // set x,y or z to value depending on index
    float X(void);                   // return fx
    float Y(void);                   // return fy
    float Z(void);                   // return fz
    void Add(float x,float y,float z); // addition to this vector
    void Sub(float x,float y,float z); // subtraction
    void Scale(float a);              // scaling of vector
    float Length(void);               // length of vector
    void Normalize(void);             // normalize vector

    void operator=(Vector& a);        // operator: assignment
    Vector operator*(float t);        // operator: scaling
    Vector operator+(Vector& a);      // operator: addition
    Vector operator-(Vector& a);      // operator: subtraction
    Vector operator+(void);           // unary +
    Vector operator-(void);           // unary -
    void operator+=(Vector& a);       // operator: +=
    void operator-=(Vector& a);       // operator: -=
    void operator*=(float t);         // operator: *= (scaling)
    float operator*(Vector& a);       // operator: dot product
    Vector operator%(Vector& a);      // operator: cross product
    float operator[](short index);
    // if short=0 then X, short=1 then Y, else Z, see constants above
};

/* here follows the inline functions and operators */

inline Vector::Vector(float x,float y,float z)
{ fx=x; fy=y; fz=z; }

inline Vector::Vector(Vector& a)
```

```
{ fx=a.fx; fy=a.fy; fz=a.fz; }

inline void Vector::Set(float x,float y,float z)
{ fx=x; fy=y; fz=z; }

inline void Vector::SetX(float x)
{ fx=x;}

inline void Vector::SetY(float y)
{ fy=y; }

inline void Vector::SetZ(float z)
{ fz=z; }

inline void Vector::SetIndex(int index,float value)
{
    switch(index)
    {
        case Xi: fx=value;
        case Yi: fy=value;
        case Zi: fz=value;
    }
}

inline float Vector::X(void)
{ return fx; }

inline float Vector::Y(void)
{ return fy; }

inline float Vector::Z(void)
{ return fz; }

inline void Vector::Add(float x,float y,float z)
{ fx+=x; fy+=y; fz+=z; }

inline void Vector::Sub(float x,float y,float z)
{ fx-=x; fy-=y; fz-=z; }

inline void Vector::Scale(float a)
{ fx*=a; fy*=a; fz*=a; }

inline float Vector::Length(void)
{ return sqrt((*this)*(*this)); // square root of Dot(this,this)
}

inline void Vector::Normalize(void)
{
    if(Length()==0.0) cout<<"Error:normalize\n";
    else Scale(1.0/Length());
}

/***** Operators *****/
inline void Vector::operator=(Vector& a) // assignment
{ fx=a.fx; fy=a.fy; fz=a.fz; }

inline Vector Vector::operator+(void) // unary +
{ return *this; }

inline Vector Vector::operator*(float t) // scaling
{ Vector temp; temp.Set(fx*t,fy*t,fz*t); return temp; }
```



```
inline Vector Vector::operator+(Vector& a)
{ Vector sum; sum.Set(fx+a.fx,fy+a.fy,fz+a.fz); return sum; }

inline Vector Vector::operator-(Vector& a)
{ Vector sum; sum.Set(fx-a.fx,fy-a.fy,fz-a.fz); return sum; }

inline Vector Vector::operator-(void)          // unary -
{ Vector neg; neg.Set(-fx,-fy,-fz); return neg; }

inline void Vector::operator+=(Vector& a)
{ Set(fx+a.fx,fy+a.fy,fz+a.fz); }

inline void Vector::operator-=(Vector& a)
{ Set(fx-a.fx,fy-a.fy,fz-a.fz); }

inline void Vector::operator*=(float t)        // scaling
{ Set(fx*t,fy*t,fz*t); }

inline float Vector::operator*(Vector& a)      // dot product
{ return fx*a.fx+fy*a.fy+fz*a.fz; }

inline Vector Vector::operator%(Vector& a)     // cross product
{
    Vector cross;
    cross.Set(fy*a.fz-fz*a.fy,fz*a.fx-fx*a.fz,fx*a.fy-fy*a.fx);
    return cross;
}

inline float Vector::operator[](short index)
{
    switch(index)
    {
        case Xi: return fx;
        case Yi: return fy;
        case Zi: return fz;
    }
    return 0.0;          // if invalid index
}
/***** End of Operators *****/
#endif
```

```

/*****
* vector.h - a vector class written in c++
* functions for +, -, dotproduct, crossproduct, scaling,
* length & normalizing, many of these are operators
* By Tomas Moller
*****/
#ifndef VECTOR_H
#define VECTOR_H

#include <stream.h>
#include <string.h>
#include <math.h>

#define Xi 0          // indices into vector
#define Yi 1
#define Zi 2

class Vector
{
    protected:
        float fx,fy,fz;
    public:
        Vector() {fx=0.0;fy=0.0;fz=0.0;}           // constructor with no
argument
        Vector(float x,float y,float z);           // constructor
with coords
        Vector(Vector& a);                           //
constructor with vector
        void Set(float x,float y,float z);           // assign new
values to vector
        void SetX(float x);                           // set x
        void SetY(float y);                           // set y
        void SetZ(float z);                           // set z;
        void SetIndex(int index,float value);        // set x,y or z to value
depending on index
        float X(void);                               // return
fx
        float Y(void);                               // return
fy
        float Z(void);                               // return
fz
        void Add(float x,float y,float z);           // addition to
this vector
        void Sub(float x,float y,float z);           // subtraction
        void Scale(float a);                           //
scaling of vector
        float Length(void);                           // length
of vector
        void Normalize(void);                           // normalize
vector

        void operator=(Vector& a);                   // operator:
assignment
        Vector operator*(float t);                   // operator:
scaling
        Vector operator+(Vector& a);                 // operator: addition
        Vector operator-(Vector& a);                 // operator: subtraction
        Vector operator+(void);                       // unary +
        Vector operator-(void);                       // unary -
        void operator+=(Vector& a);                   // operator: +=
        void operator-=(Vector& a);                   // operator: -=

```

```
void operator*=(float t); // operator: *=
(scaling) float operator*(Vector& a); // operator: dot
product Vector operator%(Vector& a); // operator: cross
product float operator[](short index); // if short=0 then X,
short=1 then Y, else Z, see constants above
void print(void); // print
coords

};

/* here follows the inline functions and operators */

inline Vector::Vector(float x,float y,float z)
{ fx=x; fy=y; fz=z; }

inline Vector::Vector(Vector& a)
{ fx=a.fx; fy=a.fy; fz=a.fz; }

inline void Vector::Set(float x,float y,float z)
{ fx=x; fy=y; fz=z; }

inline void Vector::SetX(float x)
{ fx=x; }

inline void Vector::SetY(float y)
{ fy=y; }

inline void Vector::SetZ(float z)
{ fz=z; }

inline void Vector::SetIndex(int index,float value)
{
    switch(index)
    {
        case Xi: fx=value;
        case Yi: fy=value;
        case Zi: fz=value;
    }
}

inline float Vector::X(void)
{ return fx; }

inline float Vector::Y(void)
{ return fy; }

inline float Vector::Z(void)
{ return fz; }

inline void Vector::Add(float x,float y,float z)
{ fx+=x; fy+=y; fz+=z; }

inline void Vector::Sub(float x,float y,float z)
{ fx-=x; fy-=y; fz-=z; }

inline void Vector::Scale(float a)
{ fx*=a; fy*=a; fz*=a; }
```

```
inline float Vector::Length(void)
{ return sqrt((*this)*(*this)); // square root of Dot(this,this)
}

inline void Vector::Normalize(void)
{
    if(Length()==0.0) cout<<"Error:normalize\n";
    else Scale(1.0/Length());
}

/***** Operators *****/
inline void Vector::operator=(Vector& a) // assignment
{ fx=a.fx; fy=a.fy; fz=a.fz; }

inline Vector Vector::operator+(void) // unary +
{ return *this; }

inline Vector Vector::operator*(float t) // scaling
{ Vector temp; temp.Set(fx*t,fy*t,fz*t); return temp; }

inline Vector Vector::operator+(Vector& a)
{ Vector sum; sum.Set(fx+a.fx,fy+a.fy,fz+a.fz); return sum; }

inline Vector Vector::operator-(Vector& a)
{ Vector sum; sum.Set(fx-a.fx,fy-a.fy,fz-a.fz); return sum; }

inline Vector Vector::operator-(void) // unary -
{ Vector neg; neg.Set(-fx,-fy,-fz); return neg; }

inline void Vector::operator+=(Vector& a)
{ Set(fx+a.fx,fy+a.fy,fz+a.fz); }

inline void Vector::operator-=(Vector& a)
{ Set(fx-a.fx,fy-a.fy,fz-a.fz); }

inline void Vector::operator*=(float t) // scaling
{ Set(fx*t,fy*t,fz*t); }

inline float Vector::operator*(Vector& a) // dot product
{ return fx*a.fx+fy*a.fy+fz*a.fz; }

inline Vector Vector::operator%(Vector& a) // cross product
{
    Vector cross;
    cross.Set(fy*a.fz-fz*a.fy,fz*a.fx-fx*a.fz,fx*a.fy-fy*a.fx);
    return cross;
}

inline float Vector::operator[](short index)
{
    switch(index)
    {
        case Xi: return fx;
        case Yi: return fy;
        case Zi: return fz;
    }
    return 0.0; // if invalid index
}

/***** End of Operators *****/

inline void Vector::print(void)
```

```
{  
    cout<<form("x:%.6f y:%.6f z:%.6f\n",fx,fy,fz);  
}  
#endif
```

```
/*
 * C code from the article
 * "Fast Linear Approximations of Euclidean Distance in Higher Dimensions"
 * by Yoshikazu Ohashi, yoshi@cognex.com
 * in "Graphics Gems IV", Academic Press, 1994
 */

/*
 *      2-D Euclidean distance approximation
 *      c1 = 123/128, c2 = 51 /128 and max(e) = 4.0 %
 */

int veclen2 (ix,iy)
    int      ix,iy;
{
    int      t;

    ix= (ix<0 ? -ix : ix);      /* absolute values */
    iy= (iy<0 ? -iy : iy);

    if(ix<iy)                   /* swap ix and iy if (ix < iy) */
    {                           /* See Wyvill (G1, 436) */
        ix=ix^iy;
        iy=ix^iy;
        ix=ix^iy;
    }

    t = iy + (iy>>1);

    return (ix - (ix>>5) - (ix>>7)  + (t>>2) + (t>>6));
}

/*
 *      3-D Euclidean distance approximation
 *      c1 = 15/16 , c2 = c3 = 3/8 and max(e) = 7.7 %
 */

int veclen3 (ix,iy,iz)
    int ix,iy,iz;
{
    int      t;

    ix= (ix<0 ? -ix : ix);      /* absolute values */
    iy= (iy<0 ? -iy : iy);
    iz= (iz<0 ? -iz : iz);

    if(ix<iy)                   /* needs only two comparisons */
    {
        ix=ix^iy;
        iy=ix^iy;
        ix=ix^iy;
    }

    if(ix<iz)
    {
        ix=ix^iz;
        iz=ix^iz;
        ix=ix^iz;
    }
}
```

```
}  
/* now ix is the largest */  
  
t = iy + iz;  
  
return (ix - (ix>>4) + (t>>2) + (t>>3));  
}
```

```
/*
A Fast 2D Point-On-Line Test
by Alan Paeth
from "Graphics Gems", Academic Press, 1990
*/

#include "GraphicsGems.h"

int PntOnLine(px,py,qx,qy,tx,ty)
    int px, py, qx, qy, tx, ty;
{
/*
* given a line through P:(px,py) Q:(qx,qy) and T:(tx,ty)
* return 0 if T is not on the line through      <--P--Q-->
*      1 if T is on the open ray ending at P: <--P
*      2 if T is on the closed interior along:  P--Q
*      3 if T is on the open ray beginning at Q:  Q-->
*
* Example: consider the line P = (3,2), Q = (17,7). A plot
* of the test points T(x,y) (with 0 mapped onto '.') yields:
*
*      8 | . . . . . . . . . . . . . . . . . . . 3 3
*      Y 7 | . . . . . . . . . . . . . . . . . . . 2 2 Q 3 3      Q = 2
*      6 | . . . . . . . . . . . . . . . . . . . 2 2 2 2 2 . . .
*      a 5 | . . . . . . . . . . . . . . . . . . . 2 2 2 2 2 . . .
*      x 4 | . . . . . . 2 2 2 2 2 2 . . . . . . . . . . . . .
*      i 3 | . . . . 2 2 2 2 2 . . . . . . . . . . . . . . .
*      s 2 | 1 1 P 2 2 . . . . . . . . . . . . . . . . . . .      P = 2
*      1 | 1 1 . . . . . . . . . . . . . . . . . . .
*
*      +-----+
*      1 2 3 4 5 X-axis 10          15          19
*
* Point-Line distance is normalized with the Infinity Norm
* avoiding square-root code and tightening the test vs the
* Manhattan Norm. All math is done on the field of integers.
* The latter replaces the initial ">= MAX(...)" test with
* "> (ABS(qx-px) + ABS(qy-py))" loosening both inequality
* and norm, yielding a broader target line for selection.
* The tightest test is employed here for best discrimination
* in merging collinear (to grid coordinates) vertex chains
* into a larger, spanning vectors within the Lemming editor.
*/

    /* addenda: this first set of tests has been added to detect
    * the case where the line is of zero length. Remove this if
    * such a case is impossible.
    */
    if ((px == qx) && (py == qy))
        if ((tx == px) && (ty == py)) return 2;
        else return 0;

    if ( ABS((qy-py)*(tx-px)-(ty-py)*(qx-px)) >=
        (MAX(ABS(qx-px), ABS(qy-py))) ) return(0);
    if (((qx<px)&&(px<tx)) || ((qy<py)&&(py<ty))) return(1);
    if (((tx<px)&&(px<qx)) || ((ty<py)&&(py<qy))) return(1);
    if (((px<qx)&&(qx<tx)) || ((py<qy)&&(qy<ty))) return(3);
    if (((tx<  }
&(qx<px)) || ((ty<qy)&&(qy<py))) return(3);
    return(2);
}
```



```
/*
Median Finding on a 3-by-3 Grid
by Alan Paeth
from "Graphics Gems", Academic Press, 1990
*/

#define s2(a,b) {register int t; if ((t=b-a)<0) {a+=t; b-=t;}}
#define mn3(a,b,c) s2(a,b); s2(a,c);
#define mx3(a,b,c) s2(b,c); s2(a,c);
#define mnm3(a,b,c) mx3(a,b,c); s2(a,b);
#define mnm4(a,b,c,d) s2(a,b); s2(c,d); s2(a,c); s2(b,d);
#define mnm5(a,b,c,d,e) s2(a,b); s2(c,d); mn3(a,c,e); mx3(b,d,e);
#define mnm6(a,b,c,d,e,f) s2(a,d); s2(b,e); s2(c,f);\
                           mn3(a,b,c); mx3(d,e,f);

med3x3(b1, b2, b3)
    int *b1, *b2, *b3;
/*
 * Find median on a 3x3 input box of integers.
 * b1, b2, b3 are pointers to the left-hand edge of three
 * parallel scan-lines to form a 3x3 spatial median.
 * Rewriting b2 and b3 as b1 yields code which forms median
 * on input presented as a linear array of nine elements.
 */
    {
        register int r1, r2, r3, r4, r5, r6;
        r1 = *b1++; r2 = *b1++; r3 = *b1++;
        r4 = *b2++; r5 = *b2++; r6 = *b2++;
        mnm6(r1, r2, r3, r4, r5, r6);
        r1 = *b3++;
        mnm5(r1, r2, r3, r4, r5);
        r1 = *b3++;
        mnm4(r1, r2, r3, r4);
        r1 = *b3++;
        mnm3(r1, r2, r3);
        return(r2);
    }

/* t2(i,j) transposes elements in A[] such that A[i] <= A[j] */

#define t2(i, j) s2(A[i-1], A[j-1])

int median25(A)
    int A[25];
    {
/*
 * median25(A) partitions the array A[0..24] such that element
 * A[12] is the median and subarrays A[0..11] and A[13..24]
 * are partitions containing elements of smaller and larger
 * value (rank), respectively.
 *
 * The exchange table lists element indices on the range 1..25,
 * this accounts for the "-1" offsets in the macro t2 and in
 * the final return value used to adjust subscripts to C-code
 * conventions (array indices begin at zero).
 */
        t2( 1, 2); t2( 4, 5); t2( 3, 5); t2( 3, 4); t2( 7, 8);
        t2( 6, 8); t2( 6, 7); t2(10,11); t2( 9,11); t2( 9,10);
        t2(13,14); t2(12,14); t2(12,13); t2(16,17); t2(15,17);
        t2(15,16); t2(19,20); t2(18,20); t2(18,19); t2(22,23);
    }
```

```
t2(21,23); t2(21,22); t2(24,25); t2( 3, 6); t2( 4, 7);
t2( 1, 7); t2( 1, 4); t2( 5, 8); t2( 2, 8); t2( 2, 5);
t2(12,15); t2( 9,15); t2( 9,12); t2(13,16); t2(10,16);
t2(10,13); t2(14,17); t2(11,17); t2(11,14); t2(21,24);
t2(18,24); t2(18,21); t2(22,25); t2(19,25); t2(19,22);
t2(20,23); t2( 9,18); t2(10,19); t2( 1,19); t2( 1,10);
t2(11,20); t2( 2,20); t2( 2,11); t2(12,21); t2( 3,21);
t2( 3,12); t2(13,22); t2( 4,22); t2( 4,13); t2(14,23);
t2( 5,23); t2( 5,14); t2(15,24); t2( 6,24); t2( 6,15);
t2(16,25); t2( 7,25); t2( 7,16); t2( 8,17); t2( 8,20);
t2(14,22); t2(16,24); t2( 8,14); t2( 8,16); t2( 2,10);
t2( 4,12); t2( 6,18); t2(12,18); t2(10,18); t2( 5,11);
t2( 7,13); t2( 8,15); t2( 5, 7); t2( 5, 8); t2(13,15);
t2(11,15); t2( 7, 8); t2(11,13); t2( 7,11); t2( 7,18);
t2(13,18); t2( 8,18); t2( 8,11); t2(13,19); t2( 8,13);
t2(11,19); t2(13,21); t2(11,21); t2(11,13);
return(A[13-1]);
}
```

```
/*
Mapping RGB Triples onto Four Bits
by Alan Paeth
from "Graphics Gems", Academic Press, 1990
*/

remap8(R, G, B, R2, G2, B2)
    float R, G, B, *R2, *G2, *B2;
    {
/*
 * remap8 maps floating (R,G,B) triples onto quantized
 * (R2,B2,B2) triples and returns the code (vertex)
 * value/color table entry for the quantization. The
 * points (eight) are the vertices of the cube.
 */
        int code;
        *R2 = *G2 = *B2 = 0.0;
        code = 0;
        if (R >= 0.5) { *R2 = 1.0; code |= 1; }
        if (G >= 0.5) { *G2 = 1.0; code |= 2; }
        if (B >= 0.5) { *B2 = 1.0; code |= 4; }
        return(code);
    }

/*
 * remap14 maps floating (R,G,B) triples onto quantized
 * (R2,B2,B2) triples and returns the code (vertex)
 * value/color table entry for the quantization. The
 * points (fourteen) are the vertices of the cuboctahedron.
 */

float rval[] = { 0.,.5 ,.5 , 1.,.0 , 0., 0.,.5,
                 .5 , 1., 1., 1., 0.,.5 ,.5 , 1.};
float gval[] = { 0.,.5 , 0., 0.,.5 , 1., 0.,.5,
                 .5 , 1., 0.,.5 , 1., 1.,.5 , 1.};
float bval[] = { 0., 0.,.5 , 0.,.5 , 0., 1.,.5,
                 .5 , 0., 1.,.5 , 1.,.5 , 1., 1.};

int remap14(R, G, B, R2, G2, B2)
    float R, G, B, *R2, *G2, *B2;
    {
        int code = 0;
        if ( R + G + B > 1.5) code |= 8;
        if (-R + G + B > 0.5) code |= 4;
        if ( R - G + B > 0.5) code |= 2;
        if ( R + G - B > 0.5) code |= 1;
        *R2 = rval[code];
        *G2 = gval[code];
        *B2 = bval[code];
        return(code);
    }
```

```
/*
Proper Treatment of Pixels as Integers
by Alan Paeth
from "Graphics Gems", Academic Press, 1990
*/

#define Min      code[2]
#define Med      code[1]
#define Max      code[0]
#define NCODE    3

#include <string.h>      /* or <strings.h> */

/*
 * A call to getplanes of the form:
 * getplanes(&red, &green, &blue, 256, "grb");
 *
 * fills the first three integer pointers with (near) identical
 * values which maximize red*green*blue <= 256. The final parameter
 * string defines tie-break order, here green>=red>=blue (the usual
 * default). The present code procedure calls "err(string, arg)"
 * given bad parameters; it is a simple task to rewrite the code as
 * a function which returns a success/failure code(s), as needed.
 *
 * In the example given above the code fills in the values
 * red = 6, green = 7, blue = 6.
 */





getplanes(r, g, b, n, bias)
    int *r, *g, *b;
    char *bias;
{
    int i, code[NCODE];
    if(strlen(bias) != NCODE )
        err("bias string \"%s\" wrong length",bias);
    Min = Med = Max = 0;
    *r = *g = *b = 0;
    while(Min*Min*Min <= n) Min++;
    Min--;
    while(Med*Med*Min <= n) Med++;
    Med--;
    Max = n/(Min*Med);
    for( i = 0; i < NCODE; i++ )
    {
        switch(bias[i])
        {
            case 'r': case 'R': *r = code[i]; break;
            case 'g': case 'G': *g = code[i]; break;
            case 'b': case 'B': *b = code[i]; break;
            default: err("bad bias character: \"%c\"",bias[i]); break;
        }
    }
    if (!(*r && *g && *b)) err("bias string \"%s\" deficient", bias);
}
```

```
/*
A Fast Approximation to the Hypotenuse
by Alan Paeth
from "Graphics Gems", Academic Press, 1990
*/

int idist(x1, y1, x2, y2)
    int x1, y1, x2, y2;
{
/*
 * gives approximate distance from (x1,y1) to (x2,y2)
 * with only overestimations, and then never by more
 * than (9/8) + one bit uncertainty.
 */
    if ((x2 -= x1) < 0) x2 = -x2;
    if ((y2 -= y1) < 0) y2 = -y2;
    return (x2 + y2 - (((x2>y2) ? y2 : x2) >> 1) );
}

int PntOnCirc(xp, yp, xc, yc, r)
    int xp, yp, xc, yc, r;
{
/* returns true IFF a test point (xp, yp) is to within a
 * pixel of the circle of center (xc, yc) and radius r.
 * "d" is an approximate length to circle's center, with
 *  $1.0*r < d < 1.12*r < (9/8)*r$  used for coarse testing.
 * The 9/8 ratio suggests the code:  $(x)<<3$  and  $((x)<<3)-(x)$ .
 * Variables xp, yp, r and d should be of 32-bit precision.
 *
 * Note: (9/8) forms a very tight, proper inner bound but
 * must be slackened by one pixel for the outside test (#2)
 * to account for the -1/2 pixel absolute error introduced
 * when "idist" halves an odd integer; else rough clipping
 * will trim occasional points on the circle's perimeter.
 */
    int d = idist(xp, yp, xc, yc);
    if ( r > d) return(0); /* far-in */
    if (9*r < 8*(d-1)) return(0); /* far-out */
/* full test:  $r < \text{hypot}(xp-xc, yp-yc) < r+1$  */
    xp -= xc;
    yp -= yc;
    d = xp*xp + yp*yp;
    if (d < r*r) return(0); /* near-in */
    r += 1;
    if (d > r*r) return(0); /* near-out */
    return(1); /* WITHIN */
}
```

# Index of /pubs/tog/GraphicsGems/gemsii/BitCounting/

Name	Last modified	Size	Description
 <a href="#">_Parent Directory</a>			
 <a href="#">_Makefile</a>	29-Jun-00 08:14	1K	
 <a href="#">_bit32.c</a>	29-Jun-00 08:14	1K	
 <a href="#">_test.c</a>	29-Jun-00 08:14	3K	

```
CFLAGS = -g

tester : test.c bit32.o
        $(CC) $(CFLAGS) test.c -o tester bit32.o

bit32.o : bit32.c
        $(CC) -c $(CFLAGS) bit32.c -o bit32.o

clean:
        /bin/rm -f bit32.o tester
```

```
/*
 * bit32on - count the 1 bits in a 32-bit long integer. Three variants are
 *           provided (the second accepts further code revision); one will
 *           be ``best'' based on the nature of the data and the hardware.
 *           Use of "register" directives may also yield further speed-up.
 *           The user is urged to perform comparative analysis for a given
 *           architecture and to study the assembler output, or to hand code.
 *           This code has been thoroughly tested by the second author.
 *
 * Alan W. Paeth
 * David Schilling
 */

int bit32on1(a)
    unsigned long a;
{
    int c;
    c = 0;
    while( a != 0 )          /* until no bits remain, */
    {
        c++;                /* "tally" ho, then */
        a = a &~ -a;        /* clear lowest bit */
    }
    return(c);
}

int bit32on2(a)
    unsigned long a;          /* a: 32  1-bit tallies */
{
    a = (a&0x55555555) + ((a>>1) &(0x55555555)); /* a: 16  2-bit tallies */
    a = (a&0x33333333) + ((a>>2) &(0x33333333)); /* a:  8  4-bit tallies */
    a = (a&0x07070707) + ((a>>4) &(0x07070707)); /* a:  4  8-bit tallies */
    /* a %= 255; return(a); may replace what follows */
    a = (a&0x000F000F) + ((a>>8) &(0x000F000F)); /* a:  2 16-bit tallies */
    a = (a&0x0000001F) + ((a>>16)&(0x0000001F)); /* a:  1 32-bit tally */
    return(a);
}

int bit32on3(a)
    unsigned long a;
{
    unsigned long mask, sum;
    if (a == 0)              /* a common case */
        return(0);
    else if (a == 0xffffffff) /* ditto, but the early return is essential: */
        return(32);          /* it leaves mod 31 (not 33) final states */
    mask = 0x42108421L;
    sum = a & mask;           /* 5x: accumulate through a 1-in-5 sieve */
    sum += (a>>1) & mask;
    sum += (a>>2) & mask;
    sum += (a>>4) & mask;
    sum += (a>>8) & mask;
    sum %= (mask = 31);       /* casting out mod 31 (save that constant) */
    return(sum ? sum : mask); /* return bits (zero indicated 31 bits on) */
}
```



```
/*
 *
 * Here is some code used to test the bit counting algorithms
 * described in "Of Integers, Fields and Bit Counting" by
 *
 *                               Alan W. Paeth
 *                               NeuralWare Incorporated
 *                               Pittsburgh, Pennsylvania
 *
 *                               David Schilling
 *                               Software Consultant
 *                               Bellevue, Washington
 *
 * It assumes that rand() returns an integer in [0..32767].
 * If long's were returned, the code for generating random samples
 * becomes greatly simplified.
 *
 * srand() is the random seed function. It is used to produce the
 * SAME random sample each time the test is run so that if bugs are
 * found, they can be reproduced.
 * rand() and srand() are defined in stdlib.h.
 *
 * Feel free to modify this code for the purposes of testing the bit
 * counting algorithms on your machine, and also to determine which
 * version is the fastest on your setup. It is highly recommended
 * that the code which is generated by a compiler for the bit-counting
 * routines be manually examined.
 */
```

```
#include <stdio.h>
#include <stdlib.h>
```

```
extern int bit32on1( unsigned long a );
extern int bit32on2( unsigned long a );
extern int bit32on3( unsigned long a );
```

```
int CorrectCount( unsigned long a )
{
    int c;

    c = 0;
    while( a != 0 )
    {
        c++;
        a = a & ~-a;
    }
    return( c );
}
```

```
void Error( unsigned long i, int count, char *fn )
{
    printf( "\nError: %s( %08lx ) = %d. Should be %d",
           fn, i, count, CorrectCount( i ) );
}
```

```
test( int (*count)(unsigned long), char *fn )
{
    unsigned long i;
    unsigned int j, k;
```

```
srand( 100 );

printf( "Starting... [%s] \n", fn );

for( j=0; j <= 65000; j++ )                /* first do some random testing. */
{
    i = ((unsigned long)rand() << 21) ^      /* a random long */
        ((unsigned long)rand() << 17) ^
        (unsigned long)rand();

    k = count(i);

    if( k != CorrectCount( i ) )
        Error( i, k, fn );
}

for( j=0; j <= 65000; j++ )                /* test low # of bits */
{
    i = ( ((unsigned long)rand() << 21) & ((unsigned long)rand() << 21) ) ^
        ( ((unsigned long)rand() << 17) & ((unsigned long)rand() << 17) ) ^
        ( (unsigned long)rand()          & (unsigned long)rand()          );

    k = count(i);

    if( k != CorrectCount( i ) )
        Error( i, k, fn );
}

for( j=0; j <= 65000; j++ )                /* test high # of bits */
{
    i = ( ((unsigned long)rand() << 21) | ((unsigned long)rand() << 21) ) ^
        ( ((unsigned long)rand() << 17) | ((unsigned long)rand() << 17) ) ^
        ( (unsigned long)rand()          | (unsigned long)rand()          );

    k = count(i);

    if( k != CorrectCount( i ) )
        Error( i, k, fn );
}

i = 1L;                                     /* Now try all permutations */
for( j =0; j < 33; j++ )                  /* with only 1 bit. */
{                                           /* termination includes all 0s */
    k = count(i);
    if( i != 0 )
    {
        if( k != 1 )
            Error( i, k, fn );
    }
    else
    {
        if( k != 0 )
            Error( i, k, fn );
    }
    i <<= 1;
}

i = 1L;                                     /* Finally, all permutations */
for( j =0; j < 33; j++ )                  /* with only one 0 bit. */
```

```
/* termination includes all 1s */
```

```
{
k = count( ~i );
if( i != 0 )
{
    if( k != 31 )
        Error( ~i, k, fn );
}
else
{
    if( k != 32 )
        Error( ~i, k, fn );
}
i <<= 1;
}
```

```
printf( "... Done.\n" );
}
```

```
void main( void )
{
test( bit32on1, "bit32on1" );
test( bit32on2, "bit32on2" );
test( bit32on3, "bit32on3" );
}
```

```
/*
 * Graphics Gems III: Fast Generation of Cyclic Sequences
 *
 * programmed by Alan Paeth (awpaeth@alumni.caltech.edu)
 *
 * These functions are implemented as macros which fall into two classes:
 *
 *   sequences -- generate a cyclic set of N arbitrary values
 *   triggers  -- generate a cyclic set of N boolean test conditions.
 *
 * Macros come as a set. The first initializes (declarative), the second steps.
 * These are paired for VARIABLE / BOOLEAN triggering use.
 * The first are called "sequN" such as "getch()" and
 * will not introduce side-effects. A third macro named "testM" provides
 * boolean triggering expressions without altering the current step. Its value
 * is "true" after step "i" if the ith character in string M is non-zero.
 *
 * Note: these macros have been tuned for performance and internal consistency.
 * Unlike the book exposition, a cycle's current value is undefined until
 * *after* the first call to "cyclN" or "stepN". This style facilitates fast
 * testing as in "do { } while(condition)" loops at the expense of additional
 * first-use overhead, total amortized loop time remaining unchanged. The code
 * takes advantage of both register variables and the C "comma" operator.
 * Loops may be cycled symbolically, thereby reordering the initialization or
 * step macros to take advantage of specific compiler or problem optimizations.
 */

/*
 * Sequences
 *
 * The macro "sequN(parm1,...,parmN) initializes a sequence N values;
 * the macro "cyclN()" advances to (and returns in "t1") the next cyclic value.
 */

/* cycle: [a, b] (eqn 2.2) */

#define sequ2(a,b) register int t1, t2; t1 = (int)(b); t2 = t1^(int)(a)
#define cycl2() (t1 ^= t2)

/* cycle: [a, b, c] (eqn 3.3) */

#define sequ3(a,b,c) register int t1, t2, t3; \
    t1 = (int)(c); t2 = t1^(int)(b); t3 = t2^(int)(a)
#define cycl3() (t2 ^= (t1 ^ t3), t1 ^= t2)

/* cycle: [a, b, c, d, e, f] (eqn 6.2)
 *
 * register t1 may be removed if value is not kept; t4 provides only an offset
 * a second table may be easily added to the LHS of the innermost subexpression
 * values may be cast as ints or the macro rewritten to support floats, etc.
 */

#define sequ6(a,b,c,d,e,f) register int t1,t2,t3,t4[7],*t5;t2=t3=1;t5= &t4[3];\
    t4[1]=(a); t4[0]=(b); t4[2]=(c); t4[5]=(d); t4[6]=(e); t4[4]=(f)
#define cycl6() (t1=t5[(t2 += t3, t3 += (~t2))])

/* cycle: [1, 2, 3] (eqn 3.3b, end) */

#define sequ123() register int t1, t2; t1 = 3; t2 = 1
#define cycl123() (t2 ^= t1, t1 ^= t2)
```

```
/*
 * Triggers
 *
 * The macro "trigN()" initializes a boolean cycle of N states.
 * the macro "stepN()" advances the state
 * the macro "testM" is a true expression at step i if M's ith char is '1'.
 */

/* trigger mod 2 (eqn 2.1) */

#define trig2() register int t1 = 0
#define step2() (t1 = !t1)
#define test01() (!t1)
#define test10() (t1)

/* trigger mod 3 (eqn 3.2) */




#define trig3() register int t1, t2; t1 = t2 = 1
#define step3() (t1 ^= t2, t2 ^= t1)
#define test100() (!t1)
#define test010() (!t2)
#define test001() (t1 == t2)
#define test110() (t1 != t2)

/* trigger mod 6 (eqn 6.3) */

#define trig6() register int t1, t2; t1 = -1; t2 = 1
#define step6() (t1 += t2, t2 += ~t1)
#define test100000() ((t1 == 0) && (t2 == 0))
#define test100100() (!t2)
#define test101000() (t1 == t2)
#define test110000() (!t1)
#define test110100() ((t1 == 0) || (t2 == 0))
#define test010111() (t1 != t2)
#define test011111() ((t1 != 0) || (t2 != 0))
```

# Index of

## /pubs/tog/GraphicsGems/gemsv/ch2-7/

Name	Last modified	Size	Description
 <a href="#">_Parent Directory</a>			
 <a href="#">_len4.c</a>	29-Jun-00 08:22	1K	
 <a href="#">_testlen4.c</a>	29-Jun-00 08:22	1K	

```
/*
 * len4.c - find length of the 4-vector v having components v = [a b c d]
 * programmed by: Alan Paeth, 17-Oct-1994
 *
 * This Euclidean distance estimator
 *
 * 1) can only overestimate, making it useful for containment heuristics,
 * 2) is exact for vectors [1 0 0 0] (len4 = 1) and [1 1 1 1] (len4 = 2)
 *    under all permutations, sign changes and (integer) scaling (*),
 * 3) is nearly exact for vectors [1 1 0 0] and [1 1 1 0] (len4=irrational)
 *    subject to the same conditions as in (2) above,
 * 4) offers best 3D fit using the nested form len3(a,b,c) = len4(a,b,c,0)
 * 5) may be easily reprogrammed for floating point operation by rewriting
 *    all occurrences of the string "int" with "double" or "float" and by
 *    removing the "a++" rounding fudge immediately before the return.
 * (*) - exact estimates hold only in the presence of the mod: one line up.
 *
 * Note: worst-case error occurs for vectors of the form [60c 25c 19c 16c]
 *       and is worsened (because of increased a++ rounding imprecision)
 *       with vector [2 1 1 1]. Otherwise (c large), the maximum error in
 *       overestimation is always bounded by 16% [1.1597+ = sqrt(538)/20].
 */
```

```
#define absv(x) if (x < 0) x = -x
#define inorder(x,y) {int t; if ((t = a - b) < 0) {a -= t; b += t; } }
```

```
int len4(a, b, c, d)
{
    absv(a); absv(b);          /* get the absolute values */
    absv(c); absv(d);          /* (component magnitudes) */
    inorder(a, b); inorder(c, d); /* everyone has a chance to play */
    inorder(a, c); inorder(b, d); /* (a,d) are big (winner, loser) */
    inorder(b, c);             /* playoff for 2nd and 3rd slots */
    a += (25*b + 19*c + 16*d)/60; /* compute 4D approximate length */
/* a += (5*b + 4*c + 3*d)/12;    .. only .1% worse; easy to eval */
    a++;                        /* Roundoff -> underestimation */
    return(a);                 /* omit the above one bit jitter */
}
```

```
/*
 * testlen4.c - len4 exerciser
 *
 * % cc -g testlen4.c len4.c -lm -o testlen4
 * % ./testlen4 2 1 1 1
 * % ./testlen4 60000000 25000000 19000000 16000000
 */

#include <stdio.h>
#include <stdlib.h>      /* for atoi() */
#include <math.h>

main(argc, argv)
    char *argv[];
{
    long a, b, c, d, l;
    double l2;
    a = atoi(argv[1]);
    b = atoi(argv[2]);
    c = atoi(argv[3]);
    d = atoi(argv[4]);
    l = len4(a,b,c,d);
    l2 = hypot(hypot((double)(a),(double)(b)),
               hypot((double)(c),(double)(d)));
    printf("(%d,%d,%d,%d) = %d (%g)\n", a, b, c, d, l, l/l2);
}
```



```
/*
/* When Spawning a Refraction Ray:
/* Mask = 0x01 << Spawning_ray_level;
/* path = path | Mask; /* Turn on correct bit. */
/* trace( / refraction ray / );
/* path = path & ~Mask;
/*
/* When Spawning Reflection Ray:
/* Mask = 0x01 << Spawning_ray_level;
/* path = path & ~Mask; /* Turn off correct bit. */
/* trace( / reflection ray / );
/**/

typedef struct _stree {
    TRIANGLE_REC *last_object;
    TRIANGLE_REC **last_voxel;
    struct _stree *refraction_ray;
    struct _stree *reflection_ray;
} SHADOW_TREE;

float check_shadowing(ray, light, path, Spawning_ray_level)
RAY_REC *ray; /* ray from shading point to light source */
LIGHT_REC *light; /* the light source we are interested in */
int path; /* bit table describing current position in vision ray tree */
int Spawning_ray_level; /* level of the ray spawning this shadow ray */
{
    unsigned int Mask;
    SHADOW_TREE *cache;
    int i, hit ;
    float shadow_percent;
    /* user needs to define cache, object, voxel structures */

    cache = light->cache_tree;
    Mask = 0x01;
    /* If the spawning ray's level is 0 (primary ray), then we */
    /* use the head of the cache_tree. */
    for (i = 0; i < Spawning_ray_level; ++i) {
        if (Mask & path) cache = cache->refraction_ray;
        else cache = cache->reflection_ray;
        Mask = Mask << 1; /* Shift mask left 1 bit */
    }

    if (cache->last_object != NULL) {
        /* intersect_object() marks object as having been */
        /* intersected by this ray. */
        hit = intersect_object( ray, cache->last_object, &object);

        if (hit) {
            return(1.0); /* full shadowing */
        }
        cache->last_object = NULL; /* object was not hit */

        if (cache->last_voxel != NULL) { /* implied !hit */

            /* intersect_object_in_voxel_for_shadows() returns hit = TRUE */
            /* on first affirmed intersection with an opaque object. */
            /* It ignores transparent objects altogether. */
            hit = intersect_objects_in_voxel_for_shadows( ray,
                                                         cache->last_voxel, &object);

            if (hit) {
                cache->last_object = object;
            }
        }
    }
}
```

```
        return(1.0);
    }
    cache->last_voxel = NULL; /* voxel did not supply a hit */
}














/* traverse_voxels_for_shadows() DOES intersect transparent objects and */
/* sorts the intersections for proper attenuation of the light          */
/* intensity. If multiple objects are hit, then one of the              */
/* intersections must be transparent, and the object returned is the     */
/* transparent one. Tracing of the shadow ray halts once the light       */
/* source has been reached. */
hit = traverse_voxels_for_shadows(ray, &object, &voxel, &shadow_percent);

if (!hit) {
    cache->last_object = NULL;
    cache->last_voxel = NULL;
    return(0.0); /* No shadowing was found. */
}
if (object->transparency_value > 0.0) {
    /* the object is transparent */
    cache->last_object = NULL;
    cache->last_voxel = NULL;
}
else {
    /* The object was NOT transparent, cache the info. */
    cache->last_object = object;
    cache->last_voxel = voxel;
}
return ( shadow_percent );
}

/*
- Andrew Pearce, Alias, someplace in Toronto - pearce@alias.com
*/
```

# Index of

## /pubs/tog/GraphicsGems/gemsiv/nurb\_polyg/

Name	Last modified	Size	Description
 <a href="#">_</a> <a href="#">Parent Directory</a>			
 <a href="#">FakeWindow.c</a>	29-Jun-00 08:20	1K	
 <a href="#">GGVecLib.c</a>	29-Jun-00 08:20	3K	
 <a href="#">GraphicsGems.h</a>	29-Jun-00 08:20	5K	
 <a href="#">Main.c</a>	29-Jun-00 08:20	3K	
 <a href="#">NurbEval.c</a>	29-Jun-00 08:20	6K	
 <a href="#">NurbRefine.c</a>	29-Jun-00 08:20	3K	
 <a href="#">NurbSubdiv.c</a>	29-Jun-00 08:20	17K	
 <a href="#">NurbUtils.c</a>	29-Jun-00 08:20	1K	
 <a href="#">README</a>	29-Jun-00 08:20	1K	
 <a href="#">drawing.h</a>	29-Jun-00 08:20	1K	
 <a href="#">makefile</a>	29-Jun-00 08:20	1K	
 <a href="#">nurbs.h</a>	29-Jun-00 08:20	2K	

```
/* This is a fake version, used to test on other machines. */
```

```
extern void MakeWindow(void);
```

```
void MakeWindow()  
{}
```

```
extern void MoveTo( short, short );
```

```
extern void LineTo( short, short );
```

```
void LineTo( short x, short y )  
{  
    printf("L %3d, %3d\n", (int) x, (int) y );  
}
```

```
void MoveTo( short x, short y )  
{  
    printf("M %3d, %3d\n", (int) x, (int) y );  
}
```

```
/*
2d and 3d Vector C Library
by Andrew Glassner
from "Graphics Gems", Academic Press, 1990
*/
#include <stdlib.h>

#include <math.h>
#include "GraphicsGems.h"

/*****
/*    3d Library    */
*****/

/* returns squared length of input vector */
double V3SquaredLength(Vector3 *a)
{
    return((a->x * a->x)+(a->y * a->y)+(a->z * a->z));
}

/* returns length of input vector */
double V3Length(Vector3 *a)
{
    return(sqrt(V3SquaredLength(a)));
}

/* negates the input vector and returns it */
Vector3 *V3Negate(Vector3 *v)
{
    v->x = -v->x;  v->y = -v->y;  v->z = -v->z;
    return(v);
}

/* normalizes the input vector and returns it */
Vector3 *V3Normalize(Vector3 *v)
{
    double len = V3Length(v);
    if (len != 0.0) { v->x /= len;  v->y /= len;  v->z /= len; }
    return(v);
}

/* scales the input vector to the new length and returns it */
Vector3 *V3Scale(Vector3 *v, double newlen)
{
    double len = V3Length(v);
    if (len != 0.0) {
        v->x *= newlen/len;  v->y *= newlen/len;  v->z *= newlen/len;
    }
    return(v);
}

/* return vector sum c = a+b */
Vector3 *V3Add(Vector3 *a, Vector3 *b, Vector3 *c)
{
    c->x = a->x+b->x;  c->y = a->y+b->y;  c->z = a->z+b->z;
    return(c);
}

/* return vector difference c = a-b */
Vector3 *V3Sub(Vector3 *a, Vector3 *b, Vector3 *c)
```

```
{
    c->x = a->x-b->x;  c->y = a->y-b->y;  c->z = a->z-b->z;
    return(c);
}

/* return the dot product of vectors a and b */
double V3Dot(Vector3 *a, Vector3 *b)
{
    return((a->x*b->x)+(a->y*b->y)+(a->z*b->z));
}

/* linearly interpolate between vectors by an amount alpha */
/* and return the resulting vector. */
/* When alpha=0, result=lo.  When alpha=1, result=hi. */
Vector3 *V3Lerp(Vector3 *lo, Vector3 *hi, double alpha, Vector3 *result)
{
    result->x = LERP(alpha, lo->x, hi->x);
    result->y = LERP(alpha, lo->y, hi->y);
    result->z = LERP(alpha, lo->z, hi->z);
    return(result);
}

/* make a linear combination of two vectors and return the result. */
/* result = (a * ascl) + (b * bscl) */
Vector3 *V3Combine (Vector3 *a, Vector3 *b, Vector3 *result, double ascl, double bscl)
{
    result->x = (ascl * a->x) + (bscl * b->x);
    result->y = (ascl * a->y) + (bscl * b->y);
    result->z = (ascl * a->z) + (bscl * b->z);
    return(result);
}

/* multiply two vectors together component-wise and return the result */
Vector3 *V3Mul (Vector3 *a, Vector3 *b, Vector3 *result)
{
    result->x = a->x * b->x;
    result->y = a->y * b->y;
    result->z = a->z * b->z;
    return(result);
}

/* return the distance between two points */
double V3DistanceBetween2Points(Point3 * a, Point3 * b)
{
    double dx = a->x - b->x;
    double dy = a->y - b->y;
    double dz = a->z - b->z;
    return(sqrt((dx*dx)+(dy*dy)+(dz*dz)));
}

/* return the cross product c = a cross b */
Vector3 *V3Cross(Vector3 *a, Vector3 *b, Vector3 *c)
{
    c->x = (a->y*b->z) - (a->z*b->y);
    c->y = (a->z*b->x) - (a->x*b->z);
    c->z = (a->x*b->y) - (a->y*b->x);
    return(c);
}

/* create, initialize, and return a new vector */
```

```
Vector3 *V3New(double x, double y, double z)
{
    Vector3 *v = NEWTYPE(Vector3);
    v->x = x;   v->y = y;   v->z = z;
    return(v);
}

/* create, initialize, and return a duplicate vector */
Vector3 *V3Duplicate(Vector3 *a)
{
    Vector3 *v = NEWTYPE(Vector3);
    v->x = a->x;   v->y = a->y;   v->z = a->z;
    return(v);
}
```

```
/*
 * GraphicsGems.h
 * Version 1.0 - Andrew Glassner
 * from "Graphics Gems", Academic Press, 1990
 */

#ifndef GG_H

#define GG_H 1

/*****
 * 2d geometry types */
*****/

typedef struct Point2Struct {    /* 2d point */
    double x, y;
} Point2;
typedef Point2 Vector2;

typedef struct IntPoint2Struct {    /* 2d integer point */
    int x, y;
} IntPoint2;

typedef struct Matrix3Struct {    /* 3-by-3 matrix */
    double element[3][3];
} Matrix3;

typedef struct Box2dStruct {        /* 2d box */
    Point2 min, max;
} Box2;

/*****
 * 3d geometry types */
*****/

typedef struct Point3Struct {    /* 3d point */
    double x, y, z;
} Point3;
typedef Point3 Vector3;

typedef struct IntPoint3Struct {    /* 3d integer point */
    int x, y, z;
} IntPoint3;

typedef struct Matrix4Struct {    /* 4-by-4 matrix */
    double element[4][4];
} Matrix4;

typedef struct Box3dStruct {        /* 3d box */
    Point3 min, max;
} Box3;

/*****
 * one-argument macros */
*****/

/* absolute value of a */
```



```
#define ABS(a)      ((a)<0) ? -(a) : (a))

/* round a to nearest integer towards 0 */
#define FLOOR(a)     ((a)>0 ? (int)(a) : -(int)(-a))

/* round a to nearest integer away from 0 */
#define CEILING(a) \
((a)==(int)(a) ? (a) : (a)>0 ? 1+(int)(a) : -(1+(int)(-a)))

/* round a to nearest int */
#define ROUND(a)     ((a)>0 ? (int)(a+0.5) : -(int)(0.5-a))

/* take sign of a, either -1, 0, or 1 */
#define ZSGN(a)      ((a)<0) ? -1 : (a)>0 ? 1 : 0)

/* take binary sign of a, either -1, or 1 if >= 0 */
#define SGN(a)       ((a)<0) ? -1 : 0)

/* shout if something that should be true isn't */
#define ASSERT(x) \
if (!(x)) fprintf(stderr," Assert failed: x\n");

/* square a */
#define SQR(a)       ((a)*(a))

/*****/
/* two-argument macros */
/*****/

/* find minimum of a and b */
#define MIN(a,b)     ((a)<(b))?(a):(b))

/* find maximum of a and b */
#define MAX(a,b)     ((a)>(b))?(a):(b))

/* swap a and b (see Gem by Wyvill) */
#define SWAP(a,b)    { a^=b; b^=a; a^=b; }

/* linear interpolation from l (when a=0) to h (when a=1)*/
/* (equal to (a*h)+(1-a)*l) */
#define LERP(a,l,h)  ((l)+(((h)-(l))*(a)))

/* clamp the input to the specified range */
#define CLAMP(v,l,h)  ((v)<(l) ? (l) : (v) > (h) ? (h) : v)

/*****/
/* memory allocation macros */
/*****/

/* create a new instance of a structure (see Gem by Hultquist) */
#define NEWSTRUCT(x)  (struct x *) (malloc((unsigned)sizeof(struct x)))

/* create a new instance of a type */
#define NEWTYPE(x)    (x *) (malloc((unsigned)sizeof(x)))

/*****/
/* useful constants */
/*****/
```

```
#define PI      3.141592    /* the venerable pi */
#define PITIMES2  6.283185    /* 2 * pi */
#define PIOVER2   1.570796    /* pi / 2 */
#define E        2.718282    /* the venerable e */
#define SQRT2     1.414214    /* sqrt(2) */
#define SQRT3     1.732051    /* sqrt(3) */
#define GOLDEN    1.618034    /* the golden ratio */
#define DTOR      0.017453    /* convert degrees to radians */
#define RTOD      57.29578    /* convert radians to degrees */

/*****/
/* booleans */
/*****/

#define TRUE      1
#define FALSE     0
#define ON        1
#define OFF       0
typedef int boolean;          /* boolean data type */
typedef boolean flag;         /* flag data type */

/*****/
/* 3d Library */
/*****/

/* returns squared length of input vector */
extern double V3SquaredLength(Vector3 *);
/* returns length of input vector */
extern double V3Length(Vector3 *);
/* negates the input vector and returns it */
extern Vector3 *V3Negate(Vector3 *);
/* normalizes the input vector and returns it */
extern Vector3 *V3Normalize(Vector3 *);
/* scales the input vector to the new length and returns it */
extern Vector3 *V3Scale(Vector3 *, double );
/* return vector sum c = a+b */
extern Vector3 *V3Add(Vector3 *, Vector3 *, Vector3 *);
/* return vector difference c = a-b */
extern Vector3 *V3Sub(Vector3 *, Vector3 *, Vector3 *);
/* return the dot product of vectors a and b */
extern double V3Dot(Vector3 *, Vector3 *);
/* linearly interpolate between vectors by an amount alpha */
/* and return the resulting vector. */
/* When alpha=0, result=lo. When alpha=1, result=hi. */
extern Vector3 *V3Lerp(Vector3 *, Vector3 *, double , Vector3 *);
/* make a linear combination of two vectors and return the result. */
/* result = (a * ascl); + (b * bscl); */
extern Vector3 *V3Combine (Vector3 *, Vector3 *, Vector3 *, double , double);
/* multiply two vectors together component-wise and return the result */
extern Vector3 *V3Mul (Vector3 *, Vector3 *, Vector3 *);
/* return the distance between two points */
extern double V3DistanceBetween2Points(Point3 *, Point3 * );
/* return the cross product c = a cross b */
extern Vector3 *V3Cross(Vector3 *a, Vector3 *, Vector3 *);
/* create, initialize, and return a new vector */
extern Vector3 *V3New(double, double, double);
/* create, initialize, and return a duplicate vector */
extern Vector3 *V3Duplicate(Vector3 *);
```

#endif

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "nurbs.h"
#include "drawing.h"

extern void MakeWindow( void );      /* External system routine to create a window */

void (*DrawTriangle)();             /* Pointer to triangle drawing function */

/* This generates the NURB surface for a torus, centered about the origin */

static NurbSurface *
generateTorus(double majorRadius, double minorRadius)
{
    /* These define the shape of a unit torus centered about the origin. */
    double xvalues[] = { 0.0, -1.0, -1.0, -1.0, 0.0, 1.0, 1.0, 1.0, 0.0 };
    double yvalues[] = { 1.0, 1.0, 0.0, -1.0, -1.0, -1.0, 0.0, 1.0, 1.0 };
    double zvalues[] = { 0.0, 1.0, 1.0, 1.0, 0.0, -1.0, -1.0, -1.0, 0.0 };
    double offsets[] = { -1.0, -1.0, 0.0, 1.0, 1.0, 1.0, 0.0, -1.0, -1.0 };

    /* Piecewise Bezier knot vector for a quadratic curve with four segments */
    long knots[] = { 0, 0, 0, 1, 1, 2, 2, 3, 3, 4, 4, 4 };

    long i, j;

    double r2over2 = sqrt( 2.0 ) / 2.0;
    double weight;

    NurbSurface * torus = (NurbSurface *) malloc( sizeof(NurbSurface) );
    CHECK( torus );

    /* Set up the dimension and orders of the surface */

    torus->numU = 9;      /* A circle is formed from nine points */
    torus->numV = 9;
    torus->orderU = 3;     /* Quadratic in both directions */
    torus->orderV = 3;

    /* After the dimension and orders are set, AllocNurb creates the dynamic
     * storage for the control points and the knot vectors */

    AllocNurb( torus, NULL, NULL );

    for (i = 0; i < 9; i++)
    {
        for (j = 0; j < 9; j++)
        {
            weight = ((j & 1) ? r2over2 : 1.0) * ((i & 1) ? r2over2 : 1.0);
            /* Notice how the weights are pre-multiplied with the x, y and z values */
            torus->points[i][j].x = xvalues[j]
                                * (majorRadius + offsets[i] * minorRadius) * weight;
            torus->points[i][j].y = yvalues[j]
                                * (majorRadius + offsets[i] * minorRadius) * weight;
            torus->points[i][j].z = (zvalues[i] * minorRadius) * weight;
            torus->points[i][j].w = weight;
        }
    }

    /* The knot vectors define piecewise Bezier segments (the same in both U and V). */
}
```

```
    for (i = 0; i < torus->numU + torus->orderU; i++)
        torus->kvU[i] = torus->kvV[i] = (double) knots[i];

    return torus;
}

/* These drawing routines assume a window of around 400x400 pixels */

void
ScreenProject( Point4 * worldPt, Point3 * screenPt )
{
    screenPt->x = worldPt->x / worldPt->w * 100 + 200;
    screenPt->y = worldPt->y / worldPt->w * 100 + 200;
    screenPt->z = worldPt->z / worldPt->w * 100 + 200;
}

static void
LineTriangle( SurfSample * v0, SurfSample * v1, SurfSample * v2 )
{
    MoveTo( (short) (v0->point.x * 100 + 200), (short) (v0->point.y * 100 + 200) );
    LineTo( (short) (v1->point.x * 100 + 200), (short) (v1->point.y * 100 + 200) );
    LineTo( (short) (v2->point.x * 100 + 200), (short) (v2->point.y * 100 + 200) );
    LineTo( (short) (v0->point.x * 100 + 200), (short) (v0->point.y * 100 + 200) );
}

main()
{
    Nurbsurface * torus;

    MakeWindow();          /* Create a window on the screen */

    /* Set up the subdivision tolerance (facets span about two pixels) */
    SubdivTolerance = 2.0;

    DrawTriangle = LineTriangle;

    torus = generateTorus( 1.3, 0.3 );

    DrawSubdivision( torus );
    /* DrawEvaluation( torus );    */    /* Alternate drawing method */
}
```

```
/*
 * NurbEval.c - Code for evaluating NURB surfaces.
 *
 * John Peterson
 */

#include <stdlib.h>
#include <stdio.h>
#include <math.h>

#include "nurbs.h"
#include "drawing.h"

/*
 * Return the current knot the parameter u is less than or equal to.
 * Find this "breakpoint" allows the evaluation routines to concentrate on
 * only those control points actually effecting the curve around u.]
 *
 *      m   is the number of points on the curve (or surface direction)
 *      k   is the order of the curve (or surface direction)
 *      kv  is the knot vector ([0..m+k-1]) to find the break point in.
 */

long
FindBreakPoint( double u, double * kv, long m, long k )
{
    long i;

    if (u == kv[m+1])    /* Special case for closed interval */
        return m;

    i = m + k;
    while ((u < kv[i]) && (i > 0))
        i--;
    return( i );
}

/*
 * Compute  $B_{i,k}(u)$ , for  $i = 0..k$ .
 * u          is the parameter of the spline to find the basis functions for
 * brkPoint   is the start of the knot interval ("segment")
 * kv         is the knot vector
 * k          is the order of the curve
 * bvals      is the array of returned basis values.
 *
 * (From Bartels, Beatty & Barsky, p.387)
 */

static void
BasisFunctions( double u, long brkPoint, double * kv, long k, double * bvals )
{
    register long r, s, i;
    register double omega;

    bvals[0] = 1.0;
    for (r = 2; r <= k; r++)
    {
        i = brkPoint - r + 1;
        bvals[r - 1] = 0.0;
        for (s = r-2; s >= 0; s--)
```

```
{
    i++;
    if (i < 0)
        omega = 0;
    else
        omega = (u - kv[i]) / (kv[i + r - 1] - kv[i]);
    bvals[s + 1] = bvals[s + 1] + (1 - omega) * bvals[s];
    bvals[s] = omega * bvals[s];
}
}

/*
 * Compute derivatives of the basis functions Bi,k(u)'
 */
static void
BasisDerivatives( double u, long brkPoint, double * kv, long k, double * dvals )
{
    register long s, i;
    register double omega, knotScale;

    BasisFunctions( u, brkPoint, kv, k - 1, dvals );

    dvals[k-1] = 0.0;          /* BasisFunctions misses this */

    knotScale = kv[brkPoint + 1L] - kv[brkPoint];

    i = brkPoint - k + 1L;
    for (s = k - 2L; s >= 0L; s--)
    {
        i++;
        omega = knotScale * ((double)(k-1L)) / (kv[i+k-1L] - kv[i]);
        dvals[s + 1L] += -omega * dvals[s];
        dvals[s] *= omega;
    }
}

/*
 * Calculate a point p on NurbSurface n at a specific u, v using the tensor product.
 * If utan and vtan are not nil, compute the u and v tangents as well.
 *
 * Note the valid parameter range for u and v is
 * (kvU[orderU] <= u < kvU[numU], (kvV[orderV] <= v < kvV[numV])
 */
void
CalcPoint(double u, double v, NurbSurface * n,
          Point3 * p, Point3 * utan, Point3 * vtan)
{
    register long i, j, ri, rj;
    register Point4 * cp;
    register double tmp;
    register double wsqrdiv;
    long ubrkPoint, ufirst;
    double bu[MAXORDER], buprime[MAXORDER];
    long vbrkPoint, vfirst;
    double bv[MAXORDER], bvprime[MAXORDER];
    Point4 r, rutan, rvtan;

    r.x = 0.0;
    r.y = 0.0;
```

```
r.z = 0.0;
r.w = 0.0;

rutan = r;
rvtan = r;

/* Evaluate non-uniform basis functions (and derivatives) */

ubrkJoint = FindBreakPoint( u, n->kvU, n->numU-1, n->orderU );
ufirst = ubrkJoint - n->orderU + 1;
BasisFunctions( u, ubrkJoint, n->kvU, n->orderU, bu );
if (utan)
    BasisDerivatives( u, ubrkJoint, n->kvU, n->orderU, buprime );

vbrkJoint = FindBreakPoint( v, n->kvV, n->numV-1, n->orderV );
vfirst = vbrkJoint - n->orderV + 1;
BasisFunctions( v, vbrkJoint, n->kvV, n->orderV, bv );
if (vtan)
    BasisDerivatives( v, vbrkJoint, n->kvV, n->orderV, bvprime );

/* Weight control points against the basis functions */

for (i = 0; i < n->orderV; i++)
    for (j = 0; j < n->orderU; j++)
    {
        ri = n->orderV - 1L - i;
        rj = n->orderU - 1L - j;

        tmp = bu[rj] * bv[ri];
        cp = &( n->points[i+vfirst][j+ufirst] );
        r.x += cp->x * tmp;
        r.y += cp->y * tmp;
        r.z += cp->z * tmp;
        r.w += cp->w * tmp;

        if (utan)
        {
            tmp = buprime[rj] * bv[ri];
            rutan.x += cp->x * tmp;
            rutan.y += cp->y * tmp;
            rutan.z += cp->z * tmp;
            rutan.w += cp->w * tmp;
        }
        if (vtan)
        {
            tmp = bu[rj] * bvprime[ri];
            rvtan.x += cp->x * tmp;
            rvtan.y += cp->y * tmp;
            rvtan.z += cp->z * tmp;
            rvtan.w += cp->w * tmp;
        }
    }

/* Project tangents, using the quotient rule for differentiation */

wsqrdiv = 1.0 / (r.w * r.w);
if (utan)
{
    utan->x = (r.w * rutan.x - rutan.w * r.x) * wsqrdiv;
    utan->y = (r.w * rutan.y - rutan.w * r.y) * wsqrdiv;
    utan->z = (r.w * rutan.z - rutan.w * r.z) * wsqrdiv;
```



```
}
if (vtan)
{
    vtan->x = (r.w * rvtan.x - rvtan.w * r.x) * wsqrdiv;
    vtan->y = (r.w * rvtan.y - rvtan.w * r.y) * wsqrdiv;
    vtan->z = (r.w * rvtan.z - rvtan.w * r.z) * wsqrdiv;
}

p->x = r.x / r.w;
p->y = r.y / r.w;
p->z = r.z / r.w;
}

/*
 * Draw a mesh of points by evaluating the surface at evenly spaced
 * points.
 */
void
DrawEvaluation( NurbSurface * n )
{
    Point3 p, utan, vtan;
    register long i, j;
    register double u, v, d;
    SurfSample ** pts ;

    long Granularity = 10; /* Controls the number of steps in u and v */

    /* Allocate storage for the grid of points generated */

    CHECK( pts = (SurfSample**) malloc( (Granularity+1L) * sizeof( SurfSample* ) ));
    CHECK( pts[0] = (SurfSample*) malloc( (Granularity+1L)*(Granularity+1L)
        * sizeof( SurfSample ) ));
    for (i = 1; i <= Granularity; i++)
        pts[i] = &(pts[0][(Granularity+1L) * i]);

    /* Compute points on curve */

    for (i = 0; i <= Granularity; i++)
    {
        v = ((double) i / (double) Granularity)
            * (n->kvV[n->numV] - n->kvV[n->orderV-1])
            + n->kvV[n->orderV-1];

        for (j = 0; j <= Granularity; j++)
        {
            u = ((double) j / (double) Granularity)
                * (n->kvU[n->numU] - n->kvU[n->orderU-1])
                + n->kvU[n->orderU-1];

            CalcPoint( u, v, n, &(pts[i][j].point), &utan, &vtan );
            (void) V3Cross( &utan, &vtan, &p );
            d = V3Length( &p );
            if (d != 0.0)
            {
                p.x /= d;
                p.y /= d;
                p.z /= d;
            }
            else
            {
                p.x = 0;
            }
        }
    }
}
```

```
        p.y = 0;
        p.z = 0;
    }
    pts[i][j].normLen = d;
    pts[i][j].normal = p;
    pts[i][j].u = u;
    pts[i][j].v = v;
}

/* Draw the grid */

for (i = 0; i < Granularity; i++)
    for (j = 0; j < Granularity; j++)
    {
        (*DrawTriangle)( &pts[i][j], &pts[i+1][j+1], &pts[i+1][j] );
        (*DrawTriangle)( &pts[i][j], &pts[i][j+1], &pts[i+1][j+1] );
    }

free( pts[0] );
free( pts );
}
```

```
/*
 * NurbRefine.c - Given a refined knot vector, add control points to a surface.
 *
 * John Peterson
 */

#include <stdlib.h>
#include <stdio.h>
#include <math.h>

#include "nurbs.h"

/*
 * Given the original knot vector ukv, and a new knotvector vkv, compute
 * the "alpha matrix" used to generate the corresponding new control points.
 * This routines allocates the alpha matrix if it isn't allocated already.
 *
 * This is from Bartels, Beatty & Barsky, p. 407
 */
static void
CalcAlpha( double * ukv, double * wkv, long m, long n, long k, double *** alpha )
{
    register long i, j;
    long brkPoint, r, rml, last, s;
    double omega;
    double aval[MAXORDER];

    if (! *alpha)          /* Must allocate alpha */
    {
        CHECK( *alpha = (double **) malloc( (long) ((k+1) * sizeof( double * ))) );
        for (i = 0; i <= k; i++)
            CHECK( (*alpha)[i] = (double *) malloc( (long) ((m + n + 1)
                                                         * sizeof( double ))) );
    }

    for (j = 0; j <= m + n; j++)
    {
        brkPoint = FindBreakPoint( wkv[j], ukv, m, k );
        aval[0] = 1.0;
        for (r = 2; r <= k; r++)
        {
            rml = r - 1;
            last = MIN( rml, brkPoint );
            i = brkPoint - last;
            if (last < rml)
                aval[last] = aval[last] * (wkv[j + r - 1] - ukv[i])
                               / (ukv[i + r - 1] - ukv[i]);
            else
                aval[last] = 0.0;

            for (s = last - 1; s >= 0; s-- )
            {
                i++;
                omega = (wkv[j + r - 1] - ukv[i]) / (ukv[i + r - 1] - ukv[i]);
                aval[s + 1] = aval[s+1] + (1 - omega) * aval[s];
                aval[s] = omega * aval[s];
            }
        }
        last = MIN( k - 1, brkPoint );
        for (i = 0; i <= k; i++)
            (*alpha)[i][j] = 0.0;
    }
}
```

```
    for (s = 0; s <= last; s++)
        (*alpha)[last - s][j] = aval[s];
}

/*
 * Apply the alpha matrix computed above to the rows (or columns)
 * of the surface.  If dirflag is true do the U's (row), else do V's (col).
 */
void
RefineSurface( NurbSurface * src, NurbSurface * dest, Boolean dirflag )
{
    register long i, j, out;
    register Point4 * dp, * sp;
    long il, brkPoint, maxj, maxout;
    register double tmp;
    double ** alpha = NULL;

    /* Compute the alpha matrix and indexing variables for the requested direction */

    if (dirflag)
    {
        CalcAlpha( src->kvU, dest->kvU, src->numU - 1, dest->numU - src->numU,
                   src->orderU, &alpha );
        maxj = dest->numU;
        maxout = src->numV;
    }
    else
    {
        CalcAlpha( src->kvV, dest->kvV, src->numV - 1, dest->numV - src->numV,
                   src->orderV, &alpha );
        maxj = dest->numV;
        maxout = dest->numU;
    }

    /* Apply the alpha matrix to the original control points, generating new ones */

    for (out = 0; out < maxout; out++)
        for (j = 0; j < maxj; j++)
        {
            if (dirflag)
            {
                dp = &(dest->points[out][j]);
                brkPoint = FindBreakPoint( dest->kvU[j], src->kvU,
                                           src->numU-1, src->orderU );
                il = MAX( brkPoint - src->orderU + 1, 0 );
                sp = &(src->points[out][il]);
            } else {
                dp = &(dest->points[j][out]);
                brkPoint = FindBreakPoint( dest->kvV[j], src->kvV,
                                           src->numV-1, src->orderV );
                il = MAX( brkPoint - src->orderV + 1, 0 );
                sp = &(src->points[il][out]);
            }
            dp->x = 0.0;
            dp->y = 0.0;
            dp->z = 0.0;
            dp->w = 0.0;
            for (i = il; i <= brkPoint; i++)
            {
                tmp = alpha[i - il][j];
```

```
        sp = (dirflag ? &(amp;src->points[out][i]) : &(amp;src->points[i][out]) );
        dp->x += tmp * sp->x;
        dp->y += tmp * sp->y;
        dp->z += tmp * sp->z;
        dp->w += tmp * sp->w;
    }
}

/* Free up the alpha matrix */
for (i = 0; i <= (dirflag ? src->orderU : src->orderV); i++)
    free( alpha[i] );
free( alpha );
}
```

```
/*
 * NurbSubdiv.c - Perform adaptive subdivision on a NURB surface.
 *
 * John Peterson
 *
 */

#include <stdlib.h>
#include <stdio.h>
#include <math.h>

#include "nurbs.h"
#include "drawing.h"

#define EPSILON 0.0000001 /* Used to determine when things are too small. */
#define DIVPT( p, dn ) { ((p).x) /= (dn); ((p).y) /= (dn); ((p).z) /= (dn); }

double SubdivTolerance; /* Size, in pixels of facets produced */

#define maxV(surf) ((surf)->numV-1L)
#define maxU(surf) ((surf)->numU-1L)

/*
 * Split a knot vector at the center, by adding multiplicity k knots near
 * the middle of the parameter range. Tries to start with an existing knot,
 * but will add a new knot value if there's nothing in "the middle" (e.g.,
 * a Bezier curve).
 */
static long
SplitKV( double * srckv,
         double ** destkv,
         long * splitPt, /* Where the knot interval is split */
         long m, long k )
{
    long i, last;
    long middex, extra, same; /* "midx" ==> "middle index" */
    double midVal;

    extra = 0L;
    last = (m + k);

    middex = last / 2;
    midVal = srckv[midx];

    /* Search forward and backward to see if multiple knot is already there */

    i = middex+1L;
    same = 1L;
    while ((i < last) && (srckv[i] == midVal)) {
        i++;
        same++;
    }

    i = middex-1L;
    while ((i > 0L) && (srckv[i] == midVal)) {
        i--;
        middex--; /* middex is start of multiple knot */
        same++;
    }

    if (i <= 0L) /* No knot in middle, must create it */

```

```
{
    midVal = (srckv[0L] + srckv[last]) / 2.0;
    middex = last / 2L;
    while (srckv[middex + 1L] < midVal)
        middex++;
    same = 0L;
}

extra = k - same;
CHECK( *destkv = (double *) malloc( (long) (sizeof( double )
                                         * (m+k+extra+1L) ) ) );

if (same < k)          /* Must add knots */
{
    for (i = 0L; i <= middex; i++)
        (*destkv)[i] = srckv[i];

    for (i = middex+1L; i <= middex+extra; i++)
        (*destkv)[i] = midVal;

    for (i = middex + k - same + 1L; i <= m + k + extra; i++)
        (*destkv)[i] = srckv[i - extra];
}
else
{
    for (i = 0L; i <= m + k; i++)
        (*destkv)[i] = srckv[i];
}

*splitPt = (extra < k) ? middex - 1L : middex;
return( extra );
}

/*
 * Given a line defined by firstPt and lastPt, project midPt onto
 * that line.  Used for fixing "cracks".
 */
static void
ProjectToLine( Point3 * firstPt, Point3 * lastPt, Point3 * midPt )
{
    Point3 base, v0, vm;
    double fraction, denom;

    base = *firstPt;

    (void) V3Sub( lastPt, &base, &v0 );
    (void) V3Sub( midPt, &base, &vm );

    denom = V3SquaredLength( &v0 );
    fraction = (denom == 0.0) ? 0.0 : (V3Dot( &v0, &vm ) / denom);

    midPt->x = base.x + fraction * v0.x;
    midPt->y = base.y + fraction * v0.y;
    midPt->z = base.z + fraction * v0.z;
}

/*
 * If a normal has collapsed to zero (normLen == 0.0) then try
 * and fix it by looking at its neighbors.  If all the neighbors
 * are sick, then re-compute them from the plane they form.
 * If that fails too, then we give up...
 */
```

```
*/
static void
FixNormals( SurfSample * s0, SurfSample * s1, SurfSample * s2 )
{
    Boolean goodnorm;
    long i, j, ok;
    double dist;
    SurfSample * V[3];
    Point3 norm;

    V[0] = s0; V[1] = s1; V[2] = s2;

    /* Find a reasonable normal */
    for (ok = 0, goodnorm = FALSE;
        (ok < 3L) && !(goodnorm = (V[ok]->normLen > 0.0)); ok++);

    if (! goodnorm)      /* All provided normals are zilch, try and invent one */
    {
        norm.x = 0.0; norm.y = 0.0; norm.z = 0.0;

        for (i = 0; i < 3L; i++)
        {
            j = (i + 1L) % 3L;
            norm.x += (V[i]->point.y - V[j]->point.y) * (V[i]->point.z + V[j]->point.z);
            norm.y += (V[i]->point.z - V[j]->point.z) * (V[i]->point.x + V[j]->point.x);
            norm.z += (V[i]->point.x - V[j]->point.x) * (V[i]->point.y + V[j]->point.y);
        }
        dist = V3Length( &norm );
        if (dist == 0.0)
            return;      /* This sucker's hopeless... */

        DIVPT( norm, dist );

        for (i = 0; i < 3; i++)
        {
            V[i]->normal = norm;
            V[i]->normLen = dist;
        }
    }
    else
        /* Replace a sick normal with a healthy one nearby */
    {
        for (i = 0; i < 3; i++)
            if ((i != ok) && (V[i]->normLen == 0.0))
                V[i]->normal = V[ok]->normal;
    }
    return;
}

/*
 * Normalize the normal in a sample.  If it's degenerate,
 * flag it as such by setting the normLen to 0.0
 */
static void
AdjustNormal( SurfSample * samp )
{
    /* If it's not degenerate, do the normalization now */
    samp->normLen = V3Length( &(samp->normal) );

    if (samp->normLen < EPSILON)
        samp->normLen = 0.0;
    else

```



```
        DIVPT( (samp->normal), samp->normLen );
    }

/*
 * Compute the normal of a corner point of a mesh.  The
 * base is the value of the point at the corner, indU and indV
 * are the mesh indices of that point (either 0 or numU|numV).
 */
static void
GetNormal( NurbSurface * n, long indV, long indU )
{
    Point3 tmpL, tmpR; /* "Left" and "Right" of the base point */
    SurfSample * crnr;

    if ( (indU && indV) || ((! indU) && (!indV)) )
    {
        if (indU)
            crnr = &(n->cnn);
        else
            crnr = &(n->c00);
        DIVW( &(n->points[indV][(indU ? (indU-1L) : 1L)]), &tmpL );
        DIVW( &(n->points[(indV ? (indV-1L) : 1L)][indU]), &tmpR );
    }
    else
    {
        if (indU)
            crnr = &(n->c0n);
        else
            crnr = &(n->cn0);
        DIVW( &(n->points[indV][(indU ? (indU-1L) : 1L)]), &tmpR );
        DIVW( &(n->points[(indV ? (indV-1L) : 1L)][indU]), &tmpL );
    }

    (void) V3Sub( &tmpL, &(crnr->point), &tmpL );
    (void) V3Sub( &tmpR, &(crnr->point), &tmpR );
    (void) V3Cross( &tmpL, &tmpR, &(crnr->normal) );
    AdjustNormal( crnr );
}

/*
 * Build the new corners in the two new surfaces, computing both
 * point on the surface along with the normal.  Prevent cracks that may occur.
 */
static void
MakeNewCorners( NurbSurface * parent,
                NurbSurface * kid0,
                NurbSurface * kid1,
                Boolean dirflag )
{
    DIVW( &(kid0->points[maxV(kid0)][maxU(kid0)]), &(kid0->cnn.point) );
    GetNormal( kid0, maxV(kid0), maxU(kid0) );

    if (dirflag)
    {
        kid0->strUn = FALSE; /* Must re-test new edge straightness */

        DIVW( &(kid0->points[0L][maxU(kid0)]), &(kid0->c0n.point) );
        GetNormal( kid0, 0L, maxU(kid0) );
    }
    /*
     * Normals must be re-calculated for kid1 in case the surface
     * was split at a c1 (or c0!) discontinuity
     */
}
```

```
    */
    kid1->c00.point = kid0->c0n.point;
    GetNormal( kid1, 0L, 0L );
    kid1->cn0.point = kid0->cnn.point;
    GetNormal( kid1, maxV(kid1), 0L );

    /*
    * Prevent cracks from forming by forcing the points on the seam to
    * lie along any straight edges.  (Must do this BEFORE finding normals)
    */
    if (parent->strV0)
        ProjectToLine( &(amp;parent->c00.point),
                        &(amp;parent->c0n.point),
                        &(amp;kid0->c0n.point) );
    if (parent->strVn)
        ProjectToLine( &(amp;parent->cn0.point),
                        &(amp;parent->cnn.point),
                        &(amp;kid0->cnn.point) );

    kid1->c00.point = kid0->c0n.point;
    kid1->cn0.point = kid0->cnn.point;
    kid1->strU0 = FALSE;
}
else
{
    kid0->strVn = FALSE;

    DIVW( &(amp;kid0->points[maxV(kid0)][0]), &(amp;kid0->cn0.point) );
    GetNormal( kid0, maxV(kid0), 0L );
    kid1->c00.point = kid0->cn0.point;
    GetNormal( kid1, 0L, 0L );
    kid1->c0n.point = kid0->cnn.point;
    GetNormal( kid1, 0L, maxU(kid1) );

    if (parent->strU0)
        ProjectToLine( &(amp;parent->c00.point),
                        &(amp;parent->cn0.point),
                        &(amp;kid0->cn0.point) );
    if (parent->strUn)
        ProjectToLine( &(amp;parent->c0n.point),
                        &(amp;parent->cnn.point),
                        &(amp;kid0->cnn.point) );

    kid1->c00.point = kid0->cn0.point;
    kid1->c0n.point = kid0->cnn.point;
    kid1->strV0 = FALSE;
}
}

/*
* Split a surface into two halves.  First inserts multiplicity k knots
* in the center of the parametric range.  After refinement, the two
* resulting surfaces are copied into separate data structures.  If the
* parent surface had straight edges, the points of the children are
* projected onto those edges.
*/
static void
SplitSurface( NurbSurface * parent,
              NurbSurface * kid0, NurbSurface * kid1,
              Boolean dirflag )    /* If true subdivided in U, else in V */
{
```

```
NurbSurface tmp;
double * newkv;
long i, j, splitPt;

/*
 * Add a multiplicity k knot to the knot vector in the direction
 * specified by dirflag, and refine the surface. This creates two
 * adjacent surfaces with c0 discontinuity at the seam.
 */

tmp = *parent;          /* Copy order, # of points, etc. */
if (dirflag)
{
    tmp.numU = parent->numU + SplitKV( parent->kvU,
                                        &newkv,
                                        &splitPt,
                                        maxU(parent),
                                        parent->orderU );

    AllocNurb( &tmp, newkv, NULL );
    for (i = 0L; i < tmp.numV + tmp.orderV; i++)
        tmp.kvV[i] = parent->kvV[i];
}
else
{
    tmp.numV = parent->numV + SplitKV( parent->kvV,
                                        &newkv,
                                        &splitPt,
                                        maxV(parent),
                                        parent->orderV );

    AllocNurb( &tmp, NULL, newkv );
    for (i = 0L; i < tmp.numU + tmp.orderU; i++)
        tmp.kvU[i] = parent->kvU[i];
}
RefineSurface( parent, &tmp, dirflag );

/*
 * Build the two child surfaces, and copy the data from the refined
 * version of the parent (tmp) into the two children
 */

/* First half */

*kid0 = *parent;        /* copy various edge flags and orders */

kid0->numU = dirflag ? splitPt+1L : parent->numU;
kid0->numV = dirflag ? parent->numV : splitPt+1L;
kid0->kvU = kid0->kvV = NULL;
kid0->points = NULL;
AllocNurb( kid0, NULL, NULL );

for (i = 0L; i < kid0->numV; i++) /* Copy the point and kv data */
    for (j = 0L; j < kid0->numU; j++)
        kid0->points[i][j] = tmp.points[i][j];
for (i = 0L; i < kid0->orderU + kid0->numU; i++)
    kid0->kvU[i] = tmp.kvU[i];
for (i = 0L; i < kid0->orderV + kid0->numV; i++)
    kid0->kvV[i] = tmp.kvV[i];

/* Second half */

splitPt++;
```

```
*kid1 = *parent;

kid1->numU = dirflag ? tmp.numU - splitPt : parent->numU;
kid1->numV = dirflag ? parent->numV : tmp.numV - splitPt;
kid1->kvU = kid1->kvV = NULL;
kid1->points = NULL;
AllocNurb( kid1, NULL, NULL );

for (i = 0L; i < kid1->numV; i++) /* Copy the point and kv data */
    for (j = 0L; j < kid1->numU; j++)
        kid1->points[i][j]
            = tmp.points[dirflag ? i: (i + splitPt) ][dirflag ? (j + splitPt) : j];
for (i = 0L; i < kid1->orderU + kid1->numU; i++)
    kid1->kvU[i] = tmp.kvU[dirflag ? (i + splitPt) : i];
for (i = 0L; i < kid1->orderV + kid1->numV; i++)
    kid1->kvV[i] = tmp.kvV[dirflag ? i : (i + splitPt)];

/* Construct new corners on the boundry between the two kids */
MakeNewCorners( parent, kid0, kid1, dirflag );

FreeNurb( &tmp ); /* Get rid of refined parent */
}

/*
 * Test if a particular row or column of control points in a mesh
 * is "straight" with respect to a particular tolerance. Returns true
 * if it is.
 */

#define GETPT( i ) (( dirflag ? &(n->points[crvInd][i]) : &(n->points[i][crvInd]) ))

static Boolean
IsCurveStraight( NurbSurface * n,
                 double tolerance,
                 long crvInd,
                 Boolean dirflag ) /* If true, test in U direction, else test in V */
{
    Point3 p, vec, prod;
    Point3 cp, e0;
    long i, last;
    double linelen, dist;

    /* Special case: lines are automatically straight. */
    if ((dirflag ? n->numU : n->numV) == 2L)
        return( TRUE );

    last = (dirflag ? n->numU : n->numV) - 1L;
    ScreenProject( GETPT( 0L ), &e0 );

    /* Form an initial line to test the other points against (skipping degen lines) */

    linelen = 0.0;
    for (i = last; (i > 0L) && (linelen < EPSILON); i--)
    {
        ScreenProject( GETPT( i ), &cp );
        (void) V3Sub( &cp, &e0, &vec );

        linelen = sqrt( V3SquaredLength( &vec ) );
    }

    DIVPT( vec, linelen );
}
```

```
if (linelen > EPSILON)      /* If no non-degenerate lines found, it's all degen */
for (i = 1L; i <= last; i++)
{
    /* The cross product of the vector defining the
    * initial line with the vector of the current point
    * gives the distance to the line. */
    ScreenProject( GETPT( i ), &cp );
    (void) V3Sub( &cp,&e0,&p );

    (void) V3Cross( &p, &vec, &prod );
    dist = V3Length( &prod );

    if (dist > tolerance)
        return( FALSE );
}

return( TRUE );
}

/*
 * Check to see if a surface is flat. Tests are only performed on edges and
 * directions that aren't already straight. If an edge is flagged as straight
 * (from the parent surface) it is assumed it will stay that way.
 */
static Boolean
TestFlat( NurbSurface * n, double tolerance )
{
    long i;
    Boolean straight;
    Point3 cp00, cp0n, cpn0, cpnn, planeEqn;
    double dist,d ;

    /* Check edge straightness */

    if (! n->strU0)
        n->strU0 = IsCurveStraight( n, tolerance, 0L, FALSE );
    if (! n->strUn)
        n->strUn = IsCurveStraight( n, tolerance, maxU(n), FALSE );
    if (! n->strV0)
        n->strV0 = IsCurveStraight( n, tolerance, 0L, TRUE );
    if (! n->strVn)
        n->strVn = IsCurveStraight( n, tolerance, maxV(n), TRUE );

    /* Test to make sure control points are straight in U and V */

    straight = TRUE;
    if ( (! n->flatU) && (n->strV0) && (n->strVn) )
        for (i = 1L;
            (i < maxV(n)) && (straight = IsCurveStraight( n, tolerance, i, TRUE ));
            i++);

    if (straight && n->strV0 && n->strVn)
        n->flatU = TRUE;

    straight = TRUE;
    if ( (! n->flatV) && (n->strU0) && (n->strUn) )
        for (i = 1L;
            (i < maxU(n)) && (straight = IsCurveStraight( n, tolerance, i, FALSE ));
            i++);
```

```
if (straight && n->strU0 && n->strUn)
    n->flatV = TRUE;

if ( (! n->flatV) || (! n->flatU) )
    return( FALSE );

/* The surface can pass the above tests but still be twisted. */

ScreenProject( &(n->points[0L][0L]),          &cp00 );
ScreenProject( &(n->points[0L][maxU(n)]),      &cp0n );
ScreenProject( &(n->points[maxV(n)][0L]),      &cpn0 );
ScreenProject( &(n->points[maxV(n)][maxU(n)]), &cpnn );

(void) V3Sub( &cp0n, &cp00, &cp0n ); /* Make edges into vectors */

(void) V3Sub( &cpn0, &cp00, &cpn0 );

/*
 * Compute the plane equation from two adjacent sides, and
 * measure the distance from the far point to the plane. If it's
 * larger than tolerance, the surface is twisted.
 */

(void) V3Cross( &cpn0, &cp0n, &planeEqn );

(void) V3Normalize( &planeEqn ); /* Normalize to keep adds in sync w/ mults */

d = V3Dot( &planeEqn, &cp00 );
dist = fabs( V3Dot( &planeEqn, &cpnn ) - d );

if ( dist > tolerance ) /* Surface is twisted */
    return( FALSE );
else
    return( TRUE );
}

/*
 * Turn a sufficiently flat surface into triangles.
 */
static void
EmitTriangles( NurbSurface * n )
{
    Point3 vecnn, vec0n; /* Diagonal vectors */
    double len2nn, len20n; /* Diagonal lengths squared */
    double u0, un, v0, vn; /* Texture coords;

    /*
     * Measure the distance along the two diagonals to decide the best
     * way to cut the rectangle into triangles.
     */

    (void) V3Sub( &n->c00.point, &n->cnn.point, &vecnn );
    (void) V3Sub( &n->c0n.point, &n->cn0.point, &vec0n );

    len2nn = V3SquaredLength( &vecnn ); /* Use these to reject triangles */
    len20n = V3SquaredLength( &vec0n ); /* that are too small to render */

    if (MAX(len2nn, len20n) < EPSILON)
        return; /* Triangles are too small to render */

    /*
```

```
* Assign the texture coordinates
*/
u0 = n->kvU[n->orderU-1L];
un = n->kvU[n->numU];
v0 = n->kvV[n->orderV-1L];
vn = n->kvV[n->numV];
n->c00.u = u0; n->c00.v = v0;
n->c0n.u = un; n->c0n.v = v0;
n->cn0.u = u0; n->cn0.v = vn;
n->cnn.u = un; n->cnn.v = vn;

/*
 * If any normals are sick, fix them now.
 */
if ((n->c00.normLen == 0.0) || (n->cnn.normLen == 0.0) || (n->cn0.normLen == 0.0))
    FixNormals( &(n->c00), &(n->cnn), &(n->cn0) );
if (n->c0n.normLen == 0.0)
    FixNormals( &(n->c00), &(n->c0n), &(n->cnn) );

if ( len2nn < len20n )
{
    (*DrawTriangle)( &n->c00, &n->cnn, &n->cn0 );
    (*DrawTriangle)( &n->c00, &n->c0n, &n->cnn );
}
else
{
    (*DrawTriangle)( &n->c0n, &n->cnn, &n->cn0 );
    (*DrawTriangle)( &n->c0n, &n->cn0, &n->c00 );
}
}

/*
 * The recursive subdivision algorithm. Test if the surface is flat.
 * If so, split it into triangles. Otherwise, split it into two halves,
 * and invoke the procedure on each half.
 */
static void
DoSubdivision( NurbSurface * n, double tolerance, Boolean dirflag, long level )
{
    NurbSurface left, right;    /* ...or top or bottom. Whatever spins your wheels. */

    if (TestFlat( n, tolerance ))
    {
        EmitTriangles( n );
    }
    else
    {
        if ( (!! n->flatV) && (!! n->flatU) ) || ((n->flatV) && (n->flatU)) )
            dirflag = ! dirflag;    /* If twisted or curved in both directions, */
        else                        /* then alternate subdivision direction */
        {
            if (n->flatU)            /* Only split in directions that aren't flat */
                dirflag = FALSE;
            else
                dirflag = TRUE;
        }
        SplitSurface( n, &left, &right, dirflag );
        DoSubdivision( &left, tolerance, dirflag, level + 1L );
        DoSubdivision( &right, tolerance, dirflag, level + 1L );
        FreeNurb( &left );
        FreeNurb( &right );        /* Deallocate surfaces made by SplitSurface */
    }
}
```

```
    }  
}  
  
/*  
 * Main entry point for subdivision */  
void  
DrawSubdivision( NurbSurface * surf )  
{  
    surf->flatV = FALSE;  
    surf->flatU = FALSE;  
    surf->strU0 = FALSE;  
    surf->strUn = FALSE;  
    surf->strV0 = FALSE;  
    surf->strVn = FALSE;  
  
    /*  
     * Initialize the projected corners of the surface  
     * and the normals.  
     */  
    DIVW( &(amp;surf->points[0L][0L]),                &surf->c00.point );  
    DIVW( &(amp;surf->points[0L][surf->numU-1L]),        &surf->c0n.point );  
    DIVW( &(amp;surf->points[surf->numV-1L][0L]),        &surf->cn0.point );  
    DIVW( &(amp;surf->points[surf->numV-1L][surf->numU-1L]), &surf->cnn.point );  
  
    GetNormal( surf, 0L, 0L );  
    GetNormal( surf, 0L, maxU(surf) );  
    GetNormal( surf, maxV(surf), 0L );  
    GetNormal( surf, maxV(surf), maxU(surf) );  
  
    DoSubdivision( surf, SubdivTolerance, TRUE, 0L );  
    /* Note surf is deallocated by the subdivision process */  
}
```



```
/*
 * NurbUtils.c - Code for Allocating, freeing, & copying NURB surfaces.
 *
 * John Peterson
 */

#include <stdlib.h>
#include <stdio.h>
#include <math.h>

#include "nurbs.h"

/*
 * Allocate memory for a NURB (assumes numU, numV, orderU
 * and orderV have been set).  If ukv or vkv are not NIL, they
 * are assumed to be pointers to valid knot vectors.
 */

void
AllocNurb( NurbSurface * n, double * ukv, double * vkv )
{
    long i;

    if (! ukv)
        CHECK( n->kvU = (double *) malloc( (n->numU + n->orderU) * sizeof( double ) ) )
    else
        n->kvU = ukv;
    if (! vkv)
        CHECK( n->kvV = (double *) malloc( (n->numV + n->orderV) * sizeof( double ) ) )
    else
        n->kvV = vkv;

    CHECK( n->points = (Point4 **) malloc( (long) n->numV
                                           * (long) sizeof( Point4 * ) ) );
    for (i = 0; i < n->numV; i++)
        CHECK( n->points[i] = (Point4 *) malloc( (long) n->numU
                                                  * (long) sizeof( Point4 ) ) );
}

/*
 * Release storage for a patch
 */

void
FreeNurb( NurbSurface * n )
{
    long i;

    if (n->kvU) free( n->kvU );
    n->kvU = NULL;
    if (n->kvV) free( n->kvV );
    n->kvV = NULL;
    for (i = 0; i < n->numV; i++)
        free( n->points[i] );
    free( n->points );
}

/*
 * Clone a nurb (deep copy)
 */
```

```
void
CloneNurb( NurbSurface * src, NurbSurface * dst )
{
    long i, j;
    double * srcp, *dstp;

    *dst = *src;          /* Copy fields that don't change */
    dst->kvU = NULL;
    dst->kvV = NULL;      /* So they get allocated */
    dst->points = NULL;

    AllocNurb( dst, NULL, NULL );

    /* Copy kv's */
    srcp = src->kvU;
    dstp = dst->kvU;
    for (i = 0; i < src->numU + src->orderU; i++)
        *dstp++ = *srcp++;

    srcp = src->kvV;
    dstp = dst->kvV;
    for (i = 0; i < src->numV + src->orderV; i++)
        *dstp++ = *srcp++;

    /* Copy control points */
    for (i = 0; i < src->numV; i++)
        for (j = 0; j < src->numU; j++)
            dst->points[i][j] = src->points[i][j];
}
```

ANSI C code from the article

"Tessellation of NURB Surfaces"

by John W. Peterson, [jp@blowfish.taligent.com](mailto:jp@blowfish.taligent.com)

in "Graphics Gems IV", Academic Press, 1994

```
/*  
 * These routines need to be provided by your rendering package  
 */  
  
extern void FastTriangle( SurfSample *, SurfSample *, SurfSample * );  
extern void SlowTriangle( SurfSample *, SurfSample *, SurfSample * );  
  
extern void (*DrawTriangle)( SurfSample *, SurfSample *, SurfSample * );  
  
extern void ScreenProject( Point4 *, Point3 * );
```

```
OBJS = NurbRefine.o NurbSubdiv.o NurbUtils.o NurbEval.o GGVecLib.o Main.o \  
      FakeWindow.o
```

```
nurb_polyg: $(OBJS)  
            $(CC) -o $@ $(OBJS) -lm
```

```
.c.o:  
      $(CC) -c $<
```

```
NurbRefine.c: nurbs.h GraphicsGems.h  
NurbSubdiv.c: nurbs.h drawing.h GraphicsGems.h  
NurbUtils.c:  nurbs.h GraphicsGems.h  
NurbEval.c:   nurbs.h GraphicsGems.h  
GGVecLib.c:   GraphicsGems.h  
Main.c:       nurbs.h drawing.h GraphicsGems.h
```

```
/*
 * Nurbs.h Nurb surface processing code.
 *
 * John Peterson
 */

#include "GraphicsGems.h"

#ifndef THINK_C
typedef unsigned char Boolean;
#endif

/* Rational (homogeneous) point */

typedef struct Point4Struct {
    double x, y, z, w;
} Point4;
typedef Point4 Vector4;

/*
 * Sampled point on a surface. This contains the point, normal and
 * surface coordinates (u,v). This structure is passed to the rendering
 * code for shading, etc.
 */
typedef struct SurfSample {
    Point3 point, normal; /* Point on surface, normal at that point */
    double normLen; /* Used for normalizing normals */
    double u, v; /* Parameters, e.g., used for texture mapping. */
    /* Note the parameter's range is determined by the surface's knot vector,
     * i.e., u goes from kvU[orderU-1] to kvU[numU], and likewise for v */
} SurfSample;

#define MAXORDER 20 /* Maximum order allowed (for local array sizes) */

typedef struct NurbSurface {
    /* Number of Points in the U and V directions, respectively */
    long numU, numV;
    /* Order of the surface in U and V (must be >= 2, < MAXORDER) */
    long orderU, orderV;
    /* Knot vectors, indexed as [0..numU+orderU-1] and [0..numV+orderV-1] */
    double * kvU, * kvV;
    /* Control points, indexed as points[0..numV-1][0..numU-1] */
    /* Note the w values are *premultiplied* with the x, y and z values */
    Point4 ** points;

    /* These fields are added to support subdivision */
    Boolean strV0, strVn, /* Edge straightness flags for subdivision */
           strU0, strUn;
    Boolean flatV, flatU; /* Surface flatness flags for subdivision */
    SurfSample c00, c0n,
               cn0, cnn; /* Corner data structures for subdivision */
} NurbSurface;

extern double SubdivTolerance; /* Screen space tolerance for subdivision */

#define CHECK( n ) \
    { if (!(n)) { fprintf( stderr, "Ran out of memory\n" ); exit(-1); } }

#define DIVW( rpt, pt ) \
    { (pt)->x = (rpt)->x / (rpt)->w; \
      (pt)->y = (rpt)->y / (rpt)->w; \
```

```
(pt)->z = (rpt)->z / (rpt)->w; }
```

```
/* Function prototypes */
```

```
extern void DrawSubdivision( NurbSurface * );
```

```
extern void DrawEvaluation( NurbSurface * );
```

```
extern long FindBreakPoint( double u, double * kv, long m, long k );
```

```
extern void AllocNurb( NurbSurface *, double *, double * );
```

```
extern void CloneNurb( NurbSurface *, NurbSurface * );
```

```
extern void FreeNurb( NurbSurface * );
```

```
extern void RefineSurface( NurbSurface *, NurbSurface *, Boolean );
```

```
extern void CalcPoint( double, double, NurbSurface *, Point3 *, Point3 *, Point3 * );
```

```
/* lines_intersect:  AUTHOR: Mukesh Prasad
*
*   This function computes whether two line segments,
*   respectively joining the input points (x1,y1) -- (x2,y2)
*   and the input points (x3,y3) -- (x4,y4) intersect.
*   If the lines intersect, the output variables x, y are
*   set to coordinates of the point of intersection.
*
*   All values are in integers.  The returned value is rounded
*   to the nearest integer point.
*
*   If non-integral grid points are relevant, the function
*   can easily be transformed by substituting floating point
*   calculations instead of integer calculations.
*
*   Entry
*       x1, y1,  x2, y2   Coordinates of endpoints of one segment.
*       x3, y3,  x4, y4   Coordinates of endpoints of other segment.
*
*   Exit
*       x, y             Coordinates of intersection point.
*
*   The value returned by the function is one of:
*
*       DONT_INTERSECT    0
*       DO_INTERSECT      1
*       COLLINEAR         2
*
*   Error conditions:
*
*       Depending upon the possible ranges, and particularly on 16-bit
*       computers, care should be taken to protect from overflow.
*
*       In the following code, 'long' values have been used for this
*       purpose, instead of 'int'.
*/

#define DONT_INTERSECT    0
#define DO_INTERSECT      1
#define COLLINEAR        2

/*****
*
*   NOTE:  The following macro to determine if two numbers
*   have the same sign, is for 2's complement number
*   representation.  It will need to be modified for other
*   number systems.
*****/

#define SAME_SIGNS( a, b ) \
    (((long) ((unsigned long) a ^ (unsigned long) b)) >= 0 )

int lines_intersect( x1, y1, /* First line segment */
                    x2, y2,
                    x3, y3, /* Second line segment */
                    x4, y4,
                    x,
```



```

        y          /* Output value:
                   * point of intersection */
    )
long
x1, y1, x2, y2, x3, y3, x4, y4,
*x, *y;
{
    long a1, a2, b1, b2, c1, c2; /* Coefficients of line eqns. */
    long r1, r2, r3, r4;          /* 'Sign' values */
    long denom, offset, num;       /* Intermediate values */

    /* Compute a1, b1, c1, where line joining points 1 and 2
     * is "a1 x + b1 y + c1 = 0".
     */

    a1 = y2 - y1;
    b1 = x1 - x2;
    c1 = x2 * y1 - x1 * y2;

    /* Compute r3 and r4.
     */

    r3 = a1 * x3 + b1 * y3 + c1;
    r4 = a1 * x4 + b1 * y4 + c1;

    /* Check signs of r3 and r4.  If both point 3 and point 4 lie on
     * same side of line 1, the line segments do not intersect.
     */

    if ( r3 != 0 &&
        r4 != 0 &&
        SAME_SIGNS( r3, r4 ))
        return ( DONT_INTERSECT );

    /* Compute a2, b2, c2 */

    a2 = y4 - y3;
    b2 = x3 - x4;
    c2 = x4 * y3 - x3 * y4;

    /* Compute r1 and r2 */

    r1 = a2 * x1 + b2 * y1 + c2;
    r2 = a2 * x2 + b2 * y2 + c2;

    /* Check signs of r1 and r2.  If both point 1 and point 2 lie
     * on same side of second line segment, the line segments do
     * not intersect.
     */

    if ( r1 != 0 &&
        r2 != 0 &&
        SAME_SIGNS( r1, r2 ))
        return ( DONT_INTERSECT );

    /* Line segments intersect: compute intersection point.
     */

    denom = a1 * b2 - a2 * b1;
    if ( denom == 0 )

```

```
    return ( COLLINEAR );
offset = denom < 0 ? - denom / 2 : denom / 2;

/* The denom/2 is to get rounding instead of truncating.  It
 * is added or subtracted to the numerator, depending upon the
 * sign of the numerator.
 */

num = b1 * c2 - b2 * c1;
*x = ( num < 0 ? num - offset : num + offset ) / denom;

num = a2 * c1 - a1 * c2;
*y = ( num < 0 ? num - offset : num + offset ) / denom;

return ( DO_INTERSECT );
} /* lines_intersect */
```

```
/* A main program to test the function.
 */
```

```
main()
{
    long x1, x2, x3, x4, y1, y2, y3, y4;
    long x, y;

    for (;;) {
        printf( "X1, Y1: " );
        scanf( "%ld %ld", &x1, &y1 );
        printf( "X2, Y2: " );
        scanf( "%ld %ld", &x2, &y2 );
        printf( "X3, Y3: " );
        scanf( "%ld %ld", &x3, &y3 );
        printf( "X4, Y4: " );
        scanf( "%ld %ld", &x4, &y4 );

        switch ( lines_intersect( x1, y1, x2, y2, x3, y3, x4, y4, &x, &y ) ) {
            case DONT_INTERSECT:
                printf( "Lines don't intersect\n" );
                break;
            case COLLINEAR:
                printf( "Lines are collinear\n" );
                break;
            case DO_INTERSECT:
                printf( "Lines intersect at %ld,%ld\n", x, y );
                break;
        }
    }
} /* main */
```

```
/*
 * C code from the article
 * "Fast Collision Detection of Moving Convex Polyhedra"
 * by Rich Rabbitz, rrabbitz%pgn138fs@serling.motown.ge.com
 * in "Graphics Gems IV", Academic Press, 1994
 */

#include <math.h>

typedef long          Boolean;

#define FUZZ          (0.00001)
#define TRUE          (1)
#define FALSE         (0)
#define MAX_VERTS     (1026)

#define ABS(x)        ( (x) > 0 ? (x) : -(x) )
#define EQZ(x)        ( ABS((x)) < FUZZ ? TRUE : FALSE )

#define DOT3(u,v)      ( u[0]*v[0] + u[1]*v[1] + u[2]*v[2] )
#define VECADD3(r,u,v) { r[0]=u[0]+v[0]; r[1]=u[1]+v[1]; r[2]=u[2]+v[2]; }
#define VECADDS3(r,a,u,v){r[0]=a*u[0]+v[0]; r[1]=a*u[1]+v[1]; r[2]=a*u[2]+v[2];}
#define VECSMULT3(a,u) { u[0]= a * u[0]; u[1]= a * u[1]; u[2]= a * u[2]; }
#define VECSUB3(r,u,v) { r[0]=u[0]-v[0]; r[1]=u[1]-v[1]; r[2]=u[2]-v[2]; }
#define CPVECTOR3(u,v) { u[0]=v[0];      u[1]=v[1];      u[2]=v[2]; }
#define VECNEGATE3(u)  { u[0]=(-u[0]);    u[1]=(-u[1]);    u[2]=(-u[2]); }

#define GET(u,i,j,s)  (*(u+i*s+j))
#define GET3(u,i,j,k,s) (*(u+i*(s*2)+(j*2)+k))

/*****
 *
 * The structure polyhedron is used to store the geometry of the primitives
 * used in this collision detection example.  Since the collision detection
 * algorithm only needs to operate on the vertex set of a polyhedron, and
 * no rendering is done in this example, the faces and edges of a
 * polyhedron are not stored.  Adding faces and edges to the structure for
 * rendering purposes should be straight forward and will have no effect on
 * the collision detection computations.
 *
 *****/

typedef struct polyhedron {
    double  verts[MAX_VERTS][3]; /* 3-D vertices of polyhedron. */
    int     m;                   /* number of 3-D vertices. */
    double  trn[3];              /* translational position in world coords. */
    double  itrn[3];             /* inverse of translational position. */
} *Polyhedron;

/*****
 *
 * The structure couple is used to store the information required to
 * repeatedly test a pair of polyhedra for collision.  This information
 * includes : a reference to each polyhedron, a flag indicating if there
 * is a cached prospective proper separating plane, two points for
 * constructing the proper separating plane, and possibly a cached set
 * of points from each polyhedron for speeding up the distance algorithm.
 *
 *****/

typedef struct couple {
```

```
Polyhedron  polyhdrn1;      /* First polyhedron of collision test. */
Polyhedron  polyhdrn2;      /* Second polyhedron of collision test. */
Boolean     plane_exists;    /* prospective separating plane flag */
double      pln_pnt1[3];     /* 1st point used to form separating plane. */
double      pln_pnt2[3];     /* 2nd point used to form separating plane. */
int         vert_indx[4][2]; /* cached points for distance algorithm. */
int         n;               /* number of cached points, if any. */
} *Couple;
```

```
/** Arrays for vertex sets of three primitives **/
```

```
double box[24];
double cyl[108];
double sphere[1026];
```

```
/** RJR 08/20/93 ****
```

```
*
*   Function to create vertex set of a polyhedron used to represent a
*   box primitive.
*
*   On Entry:
*       box_verts - an empty array of type double of size 24 or more.
*
*   On Exit:
*       box_verts - vertices of a polyhedron representing a box with
*                   dimensions length = 5.0, width = 5.0, and height = 5.0.
*
*   Function Return : none.
*
```

```
****
```

```
void mak_box(box_verts)
double      box_verts[];
{
    int      i;
    static double verts[24] =
        {-5.0,  5.0, 5.0, -5.0,  5.0, -5.0,  5.0,  5.0, -5.0,
         5.0,  5.0, 5.0, -5.0, -5.0, 5.0, -5.0, -5.0, -5.0,
         5.0, -5.0, -5.0,  5.0, -5.0, 5.0};

    for (i = 0; i < 24; i++)
        box_verts[i] = verts[i];
}
```

```
/** RJR 08/20/93 ****
```

```
*
*   Function to create vertex set of a polyhedron used to approximate
*   a cylinder primitive.
*
*   On Entry:
*       cyl_verts - an empty array of type double of size 108 or more.
*
*   On Exit:
*       cyl_verts - vertices of a polyhedron approximating a cylinder with
*                   a base radius = 5.0, and height = 5.0.
*
*   Function Return : none.
*
```

```
****
```

```
void mak_cyl(cyl_verts)
double      cyl_verts[];
{
    int      i;
    double    *pD_1, *pD_2, rads, stp, radius;

    pD_1 = cyl_verts;      pD_2 = cyl_verts + 54;
    stp = 0.34906585;      rads = 0.0;          radius = 5.0;

    for (i = 0; i < 18; i++) {
        pD_1[0] = pD_2[0] = radius * cos(rads);    /* X for top and bot. */
        pD_1[1] = 5.0;                             /* Y for top */
        pD_2[1] = 0.0;                             /* Y for bot. */
        pD_1[2] = pD_2[2] = -(radius * sin(rads)); /* Z for top and bot. */
        rads += stp;    pD_1 += 3;    pD_2 += 3;
    }
}

/**/ RJR 08/20/93 /**/
*
*   Function to create vertex set of a polyhedron used to approximate
*   a sphere primitive.
*
*   On Entry:
*       sph_verts - an empty array of type double of size 1026 or more.
*
*   On Exit:
*       sph_verts - vertices of a polyhedron approximating a sphere with
*                   a radius = 5.25.
*   Function Return : none.
*
***/
void mak_sph(sph_verts)
double      sph_verts[];
{
    int      i, j;
    double    rads_1, rads_2, stp_1, stp_2, *pD_1, *pD_2, radius;

    rads_1 = 1.570796327;    stp_1 = 0.174532935;
    rads_2 = 0.0;           stp_2 = 0.34906585;
    pD_1 = sph_verts;      radius = 5.25;

    for (i = 0; i < 19; i++) {
        pD_1[0] = radius * cos(rads_1);    pD_1[1] = radius * sin(rads_1);
        pD_1[2] = 0.0;
        if (EQZ(pD_1[0]))
            pD_1[0] = 0.01;
        rads_2 = 0.0;    stp_2 = 0.34906585;

        for (j = 0; j < 18; j++) {
            pD_2 = pD_1 + j * 3;
            pD_2[0] = pD_1[0] * cos(rads_2) - pD_1[2] * sin(rads_2); /* X */
            pD_2[1] = pD_1[1]; /* Y */
            pD_2[2] = -(pD_1[0] * sin(rads_2) + pD_1[2] * cos(rads_2)); /* Z */
            rads_2 += stp_2;
        }
        pD_1 += 54;    rads_1 -= stp_1;
    }
}
```

```
/** RJR 05/26/93 *****/
*
* Function to evaluate the support and contact functions at A for a given
* polytope. See equations (6) & (7).
*
* On Entry:
*   P   - table of 3-element points containing polytope vertices.
*   r   - number of points in table.
*   A   - vector at which support and contact functions will be evaluated.
*   Cp  - empty 3-element array.
*   P_i - pointer to an int.
*
* On Exit:
*   Cp  - contact point of P w.r.t. A.
*   P_i - index into P of contact point.
*
* Function Return :
*   the result of the evaluation of eq. (6) for P and A.
*
*****/
```

```
double Hp(P, r, A, Cp, P_i)
double      P[][3], A[], Cp[];
int         r, *P_i;
{
    int         i;
    double      max_val, val;

    max_val = DOT3(P[0], A);      *P_i = 0;

    for (i = 1; i < r; i++) {
        val = DOT3(P[i], A);
        if (val > max_val) {
            *P_i = i;
            max_val = val;
        }
    }
    CPVECTOR3(Cp, P[*P_i]);

    return max_val;
}
```

```
/** RJR 05/26/93 *****/
*
* Function to evaluate the support and contact functions at A for the
* set difference of two polytopes. See equations (8) & (9).
*
* On Entry:
*   P1   - table of 3-element points containing first polytope's vertices.
*   m1   - number of points in P1.
*   P2   - table of 3-element points containing second polytope's vertices.
*   m2   - number of points in P2.
*   A    - vector at which to evaluate support and contact functions.
*   Cs   - an empty array of size 3.
*   P1_i - a pointer to an int.
*   P2_i - a pointer to an int.
*
* On Exit:
```

```
*      Cs      - solution to equation 9.
*      P1_i    - index into P1 for solution to equation 9.
*      P2_i    - index into P2 for solution to equation 9.
*
*      Function Return :
*      the result of the evaluation of eq. (8) for P1 and P2 at A.
*
*****/
```

```
double Hs(P1, m1, P2, m2, A, Cs, P1_i, P2_i)
double      P1[][3], P2[][3], A[], Cs[];
int         m1, m2, *P1_i, *P2_i;
{
    double    Cp_1[3], Cp_2[3], neg_A[3], Hp_1, Hp_2;

    Hp_1 = Hp(P1, m1, A, Cp_1, P1_i);

    CPVECTOR3(neg_A, A);
    VECNEGATE3(neg_A);
    Hp_2 = Hp(P2, m2, neg_A, Cp_2, P2_i);

    VECSUB3(Cs, Cp_1, Cp_2);

    return (Hp_1 + Hp_2);
}
```

```
/*** RJR 05/26/93 *****/
*
*      Alternate function to compute the point in a polytope closest to the
*      origin in 3-space. The polytope size m is restricted to 1 < m <= 4.
*      This function is called only when comp_sub_dist fails.
*
*      On Entry:
*      stop_index - number of sets to test.
*      D_P        - array of determinants for each set.
*      Di_P       - cofactors for each set.
*      Is        - indices for each set.
*      c2        - row size offset.
*
*      On Exit:
*
*      Function Return :
*      the index of the set that is numerically closest to eq. (14).
*
*****/
```

```
int sub_dist_back(P, stop_index, D_P, Di_P, Is, c2)
double      P[][3], Di_P[][4], *D_P;
int         stop_index, *Is, c2;
{
    Boolean    first, pass;
    int        i, k, s, is, best_s;
    float      sum, v_aff, best_v_aff;

    first = TRUE;  best_s = -1;
    for (s = 0; s < stop_index; s++) {
        pass = TRUE;
        if (D_P[s] > 0.0) {
            for (i = 1; i <= GET(Is,s,0,c2); i++) {
                is = GET(Is,s,i,c2);
                if (Di_P[s][is] <= 0.0)
```

```

        pass = FALSE;
    }
}
else
    pass = FALSE;

if (pass) {

    /*** Compute equation (33) in Gilbert ***/

    k = GET(Is,s,1,c2);
    sum = 0;
    for (i = 1; i <= GET(Is, s, 0, c2); i++) {
        is = GET(Is,s,i,c2);
        sum += Di_P[s][is] * DOT3(P[is],P[k]);
    }
    v_aff = sqrt(sum / D_P[s]);
    if (first) {
        best_s = s;
        best_v_aff = v_aff;
        first = FALSE;
    }
    else {
        if (v_aff < best_v_aff) {
            best_s = s;
            best_v_aff = v_aff;
        }
    }
}
}
if (best_s == -1) {
    printf("backup failed\n");
    exit(0);
}
return best_s;
}

/*** RJR 05/26/93 *****/
*
*   Function to compute the point in a polytope closest to the origin in
*   3-space. The polytope size m is restricted to 1 < m <= 4.
*
*   On Entry:
*       P   - table of 3-element points containing polytope's vertices.
*       m   - number of points in P.
*       jo  - table of indices for storing Dj_P cofactors in Di_P.
*       Is  - indices into P for all sets of subsets of P.
*       IsC - indices into P for complement sets of Is.
*       near_pnt - an empty array of size 3.
*       near_indx - an empty array of size 4.
*       lambda - an empty array of size 4.
*
*   On Exit:
*       near_pnt - the point in P closest to the origin.
*       near_indx - indices for a subset of P which is affinely independent.
*                   See eq. (14).
*       lambda - the lambda as in eq. (14).
*
*   Function Return :
*       the number of entries in near_indx and lambda.
*

```



\*\*\*\*\*/

```
int comp_sub_dist(P, m, jo, Is, IsC, near_pnt, near_indx, lambda)
double      P[][3], near_pnt[], lambda[];
int         m, *jo, *Is, *IsC, near_indx[];
{
    Boolean      pass, fail;
    int          i, j, k, isp, is, s, row, col, stop_index, c1, c2;
    double       D_P[15], x[3], Dj_P, Di_P[15][4];
    static int    combinations[5] = {0,0,3,7,15};

    stop_index = combinations[m];    /** how many subsets in P **/
    c1 = m;  c2 = m + 1;            /** row offsets for IsC and Is **/

    /** Initialize Di_P for singletons **/

    Di_P[0][0] = Di_P[1][1] = Di_P[2][2] = Di_P[3][3] = 1.0;
    s = 0;  pass = FALSE;

    while ((s < stop_index) && (!pass)) {    /** loop through each subset */
        D_P[s] = 0.0;  fail = FALSE;
        for (i = 1; i <= GET(Is,s,0,c2); i++) {    /** loop through all Is **/
            is = GET(Is,s,i,c2);
            if (Di_P[s][is] > 0.0)                /** Condition 2 Theorem 2 **/
                D_P[s] += Di_P[s][is];            /** sum from eq. (16)      **/
            else
                fail = TRUE;
        }

        for (j = 1; j <= GET(IsC,s,0,c1); j++) {    /** loop through all IsC **/
            Dj_P = 0;  k = GET(Is,s,1,c2);
            isp = GET(IsC,s,j,c1);

            for (i = 1; i <= GET(Is,s,0,c2); i++) {
                is = GET(Is,s,i,c2);
                VECSUB3(x, P[k], P[isp]);            /** Wk - Wj  eq. (18) **/
                Dj_P += Di_P[s][is] * DOT3(P[is], x); /** sum from eq. (18) **/
            }
            row = GET3(jo,s,isp,0,c1);
            col = GET3(jo,s,isp,1,c1);
            Di_P[row][col] = Dj_P;                    /** add new cofactors **/

            if (Dj_P > 0.00001)                        /** Condition 3 Theorem 2 **/
                fail = TRUE;
        }
        if ((!fail) && (D_P[s] > 0.0))    /** Conditions 2 && 3 && 1 Theorem 2 **/
            pass = TRUE;
        else
            s++;
    }
    if (!pass) {
        printf("*** using backup procedure in sub_dist\n");
        s = sub_dist_back(P, stop_index, D_P, Di_P, Is, c2);
    }

    near_pnt[0] = near_pnt[1] = near_pnt[2] = 0.0;
    j = 0;
    for (i = 1; i <= GET(Is,s,0,c2); i++) {    /** loop through all Is **/
        is = GET(Is,s,i,c2);
        near_indx[j] = is;
        lambda[j] = Di_P[s][is] / D_P[s];        /** eq. (17)  **/
    }
```

```

        VECADDS3(near_pnt, lambda[j], P[is], near_pnt);  /** eq. (17)  */
        j++;
    }

    return (i-1);
}

/**** RJR 05/26/93 *****/
*
*   Function to compute the point in a polytope closest to the origin in
*   3-space.  The polytope size m is restricted to 1 < m <= 4.
*
*   On Entry:
*       P - table of 3-element points containing polytope's vertices.
*       m - number of points in P.
*       near_pnt - an empty array of size 3.
*       near_indx - an empty array of size 4.
*       lambda - an empty array of size 4.
*
*   On Exit:
*       near_pnt - the point in P closest to the origin.
*       near_indx - indices for a subset of P which is affinely independent.
*                   See eq. (14).
*       lambda - the lambda as in eq. (14).
*
*   Function Return :
*       the number of entries in near_indx and lambda.
*
*****/

int sub_dist(P, m, near_pnt, near_indx, lambda)
double      P[][3], near_pnt[], lambda[];
int         near_indx[], m;
{
    int      size;

/*
*   Tables to index the Di_P cofactor table in comp_sub_dist.  The s,i
*   entry indicates where to store the cofactors computed with Is_C.
*/

    static int      jo_2[2][2][2] = { { {0,0}, {2,1}},
                                         { {2,0}, {0,0}} };

    static int      jo_3[6][3][2] = { { {0,0}, {3,1}, {4,2}},
                                         { {3,0}, {0,0}, {5,2}},
                                         { {4,0}, {5,1}, {0,0}},
                                         { {0,0}, {0,0}, {6,2}},
                                         { {0,0}, {6,1}, {0,0}},
                                         { {6,0}, {0,0}, {0,0}} };

    static int      jo_4[14][4][2] = { { {0,0}, {4,1}, {5,2}, {6,3}},
                                         { {4,0}, {0,0}, {7,2}, {8,3}},
                                         { {5,0}, {7,1}, {0,0}, {9,3}},
                                         { {6,0}, {8,1}, {9,2}, {0,0}},
                                         { {0,0}, {0,0}, {10,2}, {11,3}},
                                         { {0,0}, {10,1}, {0,0}, {12,3}},
                                         { {0,0}, {11,1}, {12,2}, {0,0}},
                                         { {10,0}, {0,0}, {0,0}, {13,3}},

```

```

        {{11,0},{0,0},{13,2},{0,0}},
        {{12,0},{13,1},{0,0},{0,0}},
        {{0,0},{0,0},{0,0},{14,3}},
        {{0,0},{0,0},{14,2},{0,0}},
        {{0,0},{14,1},{0,0},{0,0}},
        {{14,0},{0,0},{0,0},{0,0}}};

/*
 * These tables represent each Is. The first column of each row indicates
 * the size of the set.
 */
static int    Is_2[3][3] = { {1,0,0},{1,1,0},{2,0,1}};

static int    Is_3[7][4] = { {1,0,0,0},{1,1,0,0},{1,2,0,0},{2,0,1,0},
                             {2,0,2,0},{2,1,2,0},{3,0,1,2}};

static int    Is_4[15][5] = { {1,0,0,0,0},{1,1,0,0,0},{1,2,0,0,0},
                              {1,3,0,0,0},{2,0,1,0,0},{2,0,2,0,0},
                              {2,0,3,0,0},{2,1,2,0,0},{2,1,3,0,0},
                              {2,2,3,0,0},{3,0,1,2,0},{3,0,1,3,0},
                              {3,0,2,3,0},{3,1,2,3,0},{4,0,1,2,3}};

/*
 * These tables represent each Is complement. The first column of each row
 * indicates the size of the set.
 */
static int    IsC_2[3][2] = { {1,1},{1,0},{0,0}};

static int    IsC_3[7][3] = { {2,1,2},{2,0,2},{2,0,1},{1,2,0},{1,1,0},
                              {1,0,0},{0,0,0}};

static int    IsC_4[15][4] = { {3,1,2,3},{3,0,2,3},{3,0,1,3},{3,0,1,2},
                              {2,2,3,0},{2,1,3,0},{2,1,2,0},{2,0,3,0},
                              {2,0,2,0},{2,0,1,0},{1,3,0,0},{1,2,0,0},
                              {1,1,0,0},{1,0,0,0},{0,0,0,0}};

/** Call comp_sub_dist with appropriate tables according to size of P */
switch (m) {
    case 2:
        size = comp_sub_dist(P, m, jo_2[0][0], Is_2[0], IsC_2[0], near_pnt,
                             near_indx, lambda);
        break;
    case 3:
        size = comp_sub_dist(P, m, jo_3[0][0], Is_3[0], IsC_3[0], near_pnt,
                             near_indx, lambda);
        break;
    case 4:
        size = comp_sub_dist(P, m, jo_4[0][0], Is_4[0], IsC_4[0], near_pnt,
                             near_indx, lambda);
        break;
}

return size;
}
```

```
/** RJR 05/26/93 *****/
*
* Function to compute the minimum distance between two convex polytopes in
* 3-space.
*
* On Entry:
*     P1 - table of 3-element points containing first polytope's vertices.
*     m1 - number of points in P1.
*     P2 - table of 3-element points containing second polytope's vertices.
*     m2 - number of points in P2.
*     VP - an empty array of size 3.
*     near_indx - a 4x2 matrix possibly containing indices of initialization
*                 points. The first column are indices into P1, and the second
*                 column are indices into P2.
*     lambda - an empty array of size 4.
*     m3 - a pointer to an int, which indicates how many initial points
*           to extract from near_indx. If 0, near_indx is ignored.
*
* On Exit:
*     Vp - vector difference of the two near points in P1 and P2.
*           The length of this vector is the minimum distance between P1
*           and P2.
*     near_indx - updated indices into P1 and P2 which indicate the affinely
*                 independent point sets from each polytope which can be used
*                 to compute along with lambda the near points in P1 and P2
*                 as in eq. (12). These indices can be used to re-initialize
*                 dist3d in the next iteration.
*     lambda - the lambda as in eqs. (11) & (12).
*     m3 - the updated number of indices for P1 and P2 in near_indx.
*
* Function Return : none.
*
*****/

void dist3d(P1, m1, P2, m2, VP, near_indx, lambda, m3)
double      P1[][3], P2[][3], VP[], lambda[];
int         m1, m2, near_indx[][2], *m3;
{
    Boolean      pass;
    int          set_size, I[4], i, j, i_tab[4], j_tab[4], P1_i, P2_i, k;
    double       Hs(), Pk[4][3], Pk_subset[4][3], Vk[3], neg_Vk[3], Cp[3],
                Gp;

    if ((*m3) == 0) {          /** if *m3 == 0 use single point initialization **/
        set_size = 1;
        VECSUB3(Pk[0], P1[0], P2[0]);          /** first elementary polytope **/
        i_tab[0] = j_tab[0] = 0;
    }
    else {                    /** else use indices from near_indx **/
        for (k = 0; k < (*m3); k++) {
            i = i_tab[k] = near_indx[k][0];
            j = j_tab[k] = near_indx[k][1];
            VECSUB3(Pk[k], P1[i], P2[j]);      /** first elementary polytope **/
        }
        set_size = *m3;
    }

    pass = FALSE;
    while (!pass) {

        /** compute Vk **/

```

```

    if (set_size == 1) {
        CPVECTOR3(Vk, Pk[0]);
        I[0] = 0;
    }
    else
        set_size = sub_dist(Pk, set_size, Vk, I, lambda);

    /** eq. (13) */

    CPVECTOR3(neg_Vk, Vk);          VECNEGATE3(neg_Vk);
    Gp = DOT3(Vk, Vk) + Hs(P1, m1, P2, m2, neg_Vk, Cp, &P1_i, &P2_i);

    /** keep track of indices for P1 and P2 */

    for (i = 0; i < set_size; i++) {
        j = I[i];
        i_tab[i] = i_tab[j];        /** j is value from member of some Is */
        j_tab[i] = j_tab[j];        /** j is value from member of some Is */
    }

    if (EQZ(Gp))                    /** Do we have a solution */
        pass = TRUE;
    else {
        for (i = 0; i < set_size; i++) {
            j = I[i];
            CPVECTOR3(Pk_subset[i], Pk[j]); /** extract affine subset of Pk */
        }
        for (i = 0; i < set_size; i++)
            CPVECTOR3(Pk[i], Pk_subset[i]); /** load into Pk+1 */

        CPVECTOR3(Pk[i], Cp);        /** Union of Pk+1 with Cp */
        i_tab[i] = P1_i;  j_tab[i] = P2_i;
        set_size++;
    }
}

CPVECTOR3(VP, Vk);                  /** load VP */
*m3 = set_size;
for(i = 0; i < set_size; i++) {
    near_idx[i][0] = i_tab[i];        /** set indices of near pnt. in P1 */
    near_idx[i][1] = j_tab[i];        /** set indices of near pnt. in P2 */
}
}

```

```

/**** RJR 05/26/93 *****/
*
*   Function to compute a proper separating plane between a pair of
*   polytopes.  The plane will be a support plane for polytope 1.
*
*   On Entry:
*       couple - couple structure for a pair of polytopes.
*
*   On Exit:
*       couple - containing new proper separating plane, if one was
*               found.
*
*   Function Return :
*       result of whether a separating plane exists, or not.
*
*****/

```

```
Boolean get_new_plane(couple)
Couple      couple;
{
    Polyhedron    polyhedron1, polyhedron2;
    Boolean       plane_exists;
    double        pnts1[MAX_VERTS][3], pnts2[MAX_VERTS][3], dist,
                  u[3], v[3], lambda[4], VP[3];
    int           i, k, m1, m2;

    plane_exists = FALSE;

    polyhedron1 = couple->polyhdrn1;    polyhedron2 = couple->polyhdrn2;

    /** Apply M1 to vertices of polytope 1 **/

    m1 = polyhedron1->m;
    for (i = 0; i < m1; i++) {
        CPVECTOR3(pnts1[i], polyhedron1->verts[i]);
        VECADD3(pnts1[i], pnts1[i], polyhedron1->trn);
    }

    /** Apply M2 to vertices of polytope 1 **/

    m2 = polyhedron2->m;
    for (i = 0; i < m2; i++) {
        CPVECTOR3(pnts2[i], polyhedron2->verts[i]);
        VECADD3(pnts2[i], pnts2[i], polyhedron2->trn);
    }

    /** solve eq. (1) for two polytopes **/

    dist3d(pnts1, m1, pnts2, m2, VP, couple->vert_indx, lambda, &couple->n);

    dist = sqrt(DOT3(VP,VP));    /** distance between polytopes **/

    if (!EQZ(dist)) {          /** Does a separating plane exist **/
        plane_exists = TRUE;
        u[0] = u[1] = u[2] = v[0] = v[1] = v[2] = 0.0;
        for (i = 0; i < couple->n; i++) {
            k = couple->vert_indx[i][0];
            VECADDS3(u, lambda[i], pnts1[k], u);    /** point in P1 **/
            k = couple->vert_indx[i][1];
            VECADDS3(v, lambda[i], pnts2[k], v);    /** point in P2 **/
        }

        /** Store separating plane in P1's local coordinates **/

        VECADD3(u, u, polyhedron1->itrn);
        VECADD3(v, v, polyhedron1->itrn);

        /** Place separating plane in couple data structure **/

        CPVECTOR3(couple->pln_pnt1, u);
        CPVECTOR3(couple->pln_pnt2, v);
    }
    return plane_exists;
}

/*** RJR 05/26/93 ****
*
```

```
*   Function to detect if two polyhedra are intersecting.
*
*   On Entry:
*       couple - couple structure for a pair of polytopes.
*
*   On Exit:
*
*   Function Return :
*       result of whether polyhedra are intersecting or not.
*
*****/
```

```
Boolean Collision(couple)
Couple      couple;
{
    Polyhedron    polyhedron1, polyhedron2;
    Boolean       collide, loop;
    double        u[3], v[3], norm[3], d;
    int           i, m;

    polyhedron1 = couple->polyhdrn1;    polyhedron2 = couple->polyhdrn2;
    collide = FALSE;

    if (couple->plane_exists) {

        /** Transform proper separating plane to P2 local coordinates.    **/
        /** This avoids the computational cost of applying the            **/
        /** transformation matrix to all the vertices of P2.              **/

        CPVECTOR3(u, couple->pln_pnt1);    CPVECTOR3(v, couple->pln_pnt2);
        VECADD3(u, u, polyhedron1->trn);    VECADD3(v, v, polyhedron1->trn);
        VECADD3(u, u, polyhedron2->itrn);    VECADD3(v, v, polyhedron2->itrn);
        VECSUB3(norm, v, u);

        m = polyhedron2->m;    i = 0; loop = TRUE;
        while ((i < m) && (loop)) {

            /** Evaluate plane equation **/

            VECSUB3(v, polyhedron2->verts[i], u);
            d = DOT3(v, norm);

            if (d <= 0.0) {                                /** is P2 in opposite half-space **/
                loop = FALSE;
                if (!get_new_plane(couple)) {
                    collide = TRUE;                        /** Collision **/
                    couple->plane_exists = FALSE;
                }
            }
            i++;
        }
    }
    else
        if (get_new_plane(couple)) {
            couple->plane_exists = TRUE;                    /** No Collision **/
        }
        else
            collide = TRUE;                                /** Collision **/

    return collide;
}
```

```
/** RJR 05/26/93 *****/
*
* Function to initialize a polyhedron.
*
* On Entry:
* polyhedron - pointer to a polyhedron structure.
*     verts - verts to load.
*     m - number of verts.
*     tx - x translation.
*     ty - y translation.
*     tz - z translation.
*
* On Exit:
*     polyhedron - an initialized polyhedron.
*
* Function Return : none.
*
*****/

void init_polyhedron(polyhedron, verts, m, tx, ty, tz)
Polyhedron    polyhedron;
double        *verts, tx, ty, tz;
int           m;
{
    int         i;
    double      *p;

    polyhedron->trn[0] = tx; polyhedron->trn[1] = ty;
    polyhedron->trn[2] = tz;

    polyhedron->itrn[0] = -tx; polyhedron->itrn[1] = -ty;
    polyhedron->itrn[2] = -tz;

    polyhedron->m = m;

    p = verts;
    for (i = 0; i < m; i++) {
        CPVECTOR3(polyhedron->verts[i], p);
        p += 3;
    }
}

/** RJR 05/26/93 *****/
*
* Function to move a polyhedron.
*
* On Entry:
* polyhedron - pointer to a polyhedron.
*     tx - x translation.
*     ty - y translation.
*     tz - z translation.
*
* On Exit:
*     polyhedron - an updated polyhedron.
*
* Function Return : none.
*
*****/
```



```
void move_polyhedron(polyhedron, tx, ty, tz)
Polyhedron    polyhedron;
double        tx, ty, tz;
{
    polyhedron->trn[0] += tx;  polyhedron->trn[1] += ty;
    polyhedron->trn[2] += tz;

    polyhedron->itrn[0] -= tx;  polyhedron->itrn[1] -= ty;
    polyhedron->itrn[2] -= tz;
}

/**** RJR 05/26/93 *****/
*
*   This is the Main Program for the Collision Detection example. This test
*   program creates the vertices of three polyhedra: a sphere, a box, and a
*   cylinder. The three polyhedra oscillate back and forth along the x-axis.
*   A collision test is done after each movement on each pair of polyhedra.
*   This test program was run on an SGI Onyx/4 and an SGI 4D/80. A total of
*   30,000 collision detection tests were performed. There were 3,160
*   collisions detected. The dist3d function was called in 14% of the
*   collision tests. The average number of iterations in dist3d was 1.7.
*   The above functions are designed to compute accurate solutions when
*   the polyhedra are simple and convex. The functions will work on
*   concave polyhedra, but the solutions are computed using the convex hulls
*   of the concave polyhedra. In this case when the algorithm returns a
*   disjoint result it is exact, but when it returns an intersection result
*   it is approximate.
*
*****/
main()
{
    Polyhedron    Polyhedron1, Polyhedron2, Polyhedron3;
    Couple        Couple1, Couple2, Couple3;
    double        xstp1, xstp2, xstp3;
    int           i, steps;
    long          hits = 0;

    /**** Initialize the 3 test polyhedra ****/

    mak_box(box);
    mak_cyl(cyl);
    mak_sph(sphere);

    Polyhedron1 = (Polyhedron)malloc(sizeof(struct polyhedron));
    init_polyhedron(Polyhedron1, sphere, 342, 0.0, 0.0, 0.0);

    Polyhedron2 = (Polyhedron)malloc(sizeof(struct polyhedron));
    init_polyhedron(Polyhedron2, box, 8, 50.0, 0.0, 0.0);

    Polyhedron3 = (Polyhedron)malloc(sizeof(struct polyhedron));
    init_polyhedron(Polyhedron3, cyl, 36, -50.0, 0.0, 0.0);

    Couple1 = (Couple)malloc(sizeof(struct couple));
    Couple1->polyhdrn1 = Polyhedron1;  Couple1->polyhdrn2 = Polyhedron2;
    Couple1->n = 0;
    Couple1->plane_exists = FALSE;

    Couple2 = (Couple)malloc(sizeof(struct couple));
    Couple2->polyhdrn1 = Polyhedron1;  Couple2->polyhdrn2 = Polyhedron3;
    Couple2->n = 0;
    Couple2->plane_exists = FALSE;
```

```
Couple3 = (Couple)malloc(sizeof(struct couple));
Couple3->polyhdrn1 = Polyhedron3;  Couple3->polyhdrn2 = Polyhedron2;
Couple3->n = 0;
Couple3->plane_exists = FALSE;

/** Perform Collision Tests **/

xstp1 = 1.0;  xstp2 = 5.0; xstp3 = 10.0;  steps = 10000;

for (i = 0; i < steps; i++) {
    move_polyhedron(Polyhedron1, xstp1, 0.0, 0.0);
    move_polyhedron(Polyhedron2, xstp2, 0.0, 0.0);
    move_polyhedron(Polyhedron3, xstp3, 0.0, 0.0);

    if (Collision(Couple1))
        hits++;
    if (Collision(Couple2))
        hits++;
    if (Collision(Couple3))
        hits++;

    if (ABS(Polyhedron1->trn[0]) > 100.0)
        xstp1 = -xstp1;
    if (ABS(Polyhedron2->trn[0]) > 100.0)
        xstp2 = -xstp2;
    if (ABS(Polyhedron3->trn[0]) > 100.0)
        xstp3 = -xstp3;
}
printf("number of tests = %d\n", (steps * 3));
printf("number of hits = %ld\n", hits);
}
```

```
/*
Matrix Orthogonalization
Eric Raible
from "Graphics Gems", Academic Press, 1990
*/

/*
 * Reorthogonalize matrix R - that is find an orthogonal matrix that is
 * "close" to R by computing an approximation to the orthogonal matrix
 *
 * 
$$RC = R(R^T R)^{-1/2}$$

 * [RC is orthogonal because  $(RC)^T = (RC)$  ]
 *
 * To compute C, we evaluate the Taylor expansion of  $F(x) = (I + x)^{-1/2}$ 
 * (where  $x = C - I$ ) about  $x=0$ .
 * This gives  $C = I - (1/2)x + (3/8)x^2 - (5/16)x^3 + \dots$ 
 */

#include "GraphicsGems.h"

static float coef[10] = /* From mathematica */
{ 1, -1/2., 3/8., -5/16., 35/128., -63/256.,
  231/1024., -429/2048., 6435/32768., -12155/65536. };

MATRIX_reorthogonalize (R, limit)
    Matrix4 R;
{
    Matrix4 I, Temp, X, X_power, Sum;
    int power;

    limit = MAX(limit, 10);

    MATRIX_transpose (R, Temp); /* Rt */
    MATRIX_multiply (Temp, R, Temp); /* RtR */
    MATRIX_identify (I);
    MATRIX_subtract (Temp, I, X); /* RtR - I */
    MATRIX_identify (X_power); /* X^0 */
    MATRIX_identify (Sum); /* coef[0] * X^0 */

    for (power = 1; power < limit; ++power)
    {
        MATRIX_multiply (X_power, X, X_power);
        MATRIX_constant_multiply (coef[power], X_power, Temp);
        MATRIX_add (Sum, Temp, Sum);
    }

    MATRIX_multiply (R, Sum, R);
}
```

```
/*
An Efficient Bounding Sphere
by Jack Ritter
from "Graphics Gems", Academic Press, 1990
*/

/* Routine to calculate near-optimal bounding sphere over      */
/* a set of points in 3D */
/* This contains the routine find_bounding_sphere(), */
/* the struct definition, and the globals used for parameters. */
/* The abs() of all coordinates must be < BIGNUMBER */
/* Code written by Jack Ritter and Lyle Rains. */

#include <stdio.h>
#include <math.h>
#include "GraphicsGems.h"

#define BIGNUMBER 100000000.0          /* hundred million */

/* GLOBALS. These are used as input and output parameters. */

struct Point3Struct caller_p,cen;
double rad;
int NUM_POINTS;

/* Call with no parameters. Caller must set NUM_POINTS */
/* before calling. */
/* Caller must supply the routine GET_iTH_POINT(i), which loads his */
/* ith point into the global struct caller_p. (0 <= i < NUM_POINTS). */
/* The calling order of the points is irrelevant. */
/* The final bounding sphere will be put into the globals */
/* cen and rad. */

find_bounding_sphere()
{
register int i;
register double dx,dy,dz;
register double rad_sq,xspan,yspan,zspan,maxspan;
double old_to_p,old_to_p_sq,old_to_new;
struct Point3Struct xmin,xmax,ymin,ymax,zmin,zmax,dia1,dia2;

/* FIRST PASS: find 6 minima/maxima points */
xmin.x=ymin.y=zmin.z= BIGNUMBER; /* initialize for min/max compare */
xmax.x=ymax.y=zmax.z= -BIGNUMBER;
for (i=0;i<NUM_POINTS;i++)
{
GET_iTH_POINT(i); /* load global struct caller_p with */
/* his ith point. */

if (caller_p.x<xmin.x)
xmin = caller_p; /* New xminimum point */
if (caller_p.x>xmax.x)
xmax = caller_p;
if (caller_p.y<ymin.y)
ymin = caller_p;
if (caller_p.y>ymax.y)
ymax = caller_p;
if (caller_p.z<zmin.z)
zmin = caller_p;
if (caller_p.z>zmax.z)
```

```
        zmax = caller_p;
    }
/* Set xspan = distance between the 2 points xmin & xmax (squared) */
dx = xmax.x - xmin.x;
dy = xmax.y - xmin.y;
dz = xmax.z - xmin.z;
xspan = dx*dx + dy*dy + dz*dz;

/* Same for y & z spans */
dx = ymax.x - ymin.x;
dy = ymax.y - ymin.y;
dz = ymax.z - ymin.z;
yspan = dx*dx + dy*dy + dz*dz;

dx = zmax.x - zmin.x;
dy = zmax.y - zmin.y;
dz = zmax.z - zmin.z;
zspan = dx*dx + dy*dy + dz*dz;

/* Set points dial & dia2 to the maximally separated pair */
dial = xmin; dia2 = xmax; /* assume xspan biggest */
maxspan = xspan;
if (yspan>maxspan)
{
    maxspan = yspan;
    dial = ymin; dia2 = ymax;
}
if (zspan>maxspan)
{
    dial = zmin; dia2 = zmax;
}

/* dial,dia2 is a diameter of initial sphere */
/* calc initial center */
cen.x = (dial.x+dia2.x)/2.0;
cen.y = (dial.y+dia2.y)/2.0;
cen.z = (dial.z+dia2.z)/2.0;
/* calculate initial radius**2 and radius */
dx = dia2.x-cen.x; /* x component of radius vector */
dy = dia2.y-cen.y; /* y component of radius vector */
dz = dia2.z-cen.z; /* z component of radius vector */
rad_sq = dx*dx + dy*dy + dz*dz;
rad = sqrt(rad_sq);

/* SECOND PASS: increment current sphere */

for (i=0;i<NUM_POINTS;i++)
{
    GET_iTH_POINT(i); /* load global struct caller_p */
                        /* with his ith point. */
    dx = caller_p.x-cen.x;
    dy = caller_p.y-cen.y;
    dz = caller_p.z-cen.z;
    old_to_p_sq = dx*dx + dy*dy + dz*dz;
    if (old_to_p_sq > rad_sq) /* do r**2 test first */
    {
        /* this point is outside of current sphere */
        old_to_p = sqrt(old_to_p_sq);
        /* calc radius of new sphere */
        rad = (rad + old_to_p) / 2.0;
        rad_sq = rad*rad; /* for next r**2 compare */
    }
}
```

```
old_to_new = old_to_p - rad;
/* calc center of new sphere */
cen.x = (rad*cen.x + old_to_new*caller_p.x) / old_to_p;
cen.y = (rad*cen.y + old_to_new*caller_p.y) / old_to_p;
cen.z = (rad*cen.z + old_to_new*caller_p.z) / old_to_p;
/* Suppress if desired */
printf("\n New sphere: cen,rad = %f %f %f    %f",
        cen.x,cen.y,cen.z, rad);
}
```

```
}
```

```
}
```

```
/* end of find_bounding_sphere() */
```

```
/* The following code implements the four basic interval arithmetic
operations +, -, * and / within the C++ class interval as provided
by the AT & T C++ Release 2.0 for the Sparc architecture. There is
no checking that the intervals are valid intervals, i.e. if the
interval is [lo.hi] then it is valid iff lo <= hi.
```

It is assumed that a mechanism exists to direct the result of a floating point operation so that it is chopped. This corresponds to "tozero" in ANSI/IEEE Std. 754-1985 and it is performed by the routine

```
ieee_flags
```

in this code.

The first executable statement of any program using this code must therefore be

```
setmode(1);
```

or a corresponding call in in the architecture used in order that the rounding mode will be the chopped mode which is assumed in the class. Changes have to be made to the routine help\_ if another rounding mode is used.

```
*/
```

```
#define min(a, b)          (((a) < (b)) ? (a) : (b))
#define MIN(a, b, c, d)    min(min(min(a, b), c), d)
#define max(a, b)          (((a) > (b)) ? (a) : (b))
#define MAX(a, b, c, d)    max(max(max(a, b), c), d)
```

```
#include <stdio.h>
#include <stream.h>
#include <sys/ieee.h>
```

```
extern "C" {
    int ieee_flags(char *, char *, char *in, char **);
}
```

```
void setmode_(int *mode)
```

```
{
    char *out;

    ieee_flags("set", "direction", *mode ? "tozero" : "nearest", &out);
    ieee_flags("get", "direction", NULL, &out);
}
```

```
/*
```

This routine adds 1 to the last digit of a double data item which is here assumed to have 8 bytes where

```
seeeeeee eeeemmmm mmmmmmmm mmmmmmmm mmmmmmmm mmmmmmmm mmmmmmmm
```

s=sign of number

e=exponent 0111111111 is an exponent of 0

m=mantissa bits with assumed leading 1

The algorithm works by adding one to the last two bytes of the mantissa. If the result of this addition is zero then it overflows to the next two bytes. This means that the next two bytes must be checked for overflow until the mantissa bits are exhausted. It should then be noted that the exponent will be automatically incremented and the mantissa will have

been set to all zeros.

No check is made to see if the exponent overflows. Negative numbers will turn into the next larger negative number.

This routine assumes that the number of bytes in a double is an integer multiple of the number of bytes in an unsigned integer.

\*/

```
void helpx_(double *a)
{
    unsigned char *b;
    unsigned int *c,i;

    b=(unsigned char *)a;
    i=sizeof(double)-sizeof(unsigned int);
    do {
        c=(unsigned int *) (b+i);
        (*c)++;
        i-=sizeof(unsigned int);
    } while ((*c)==0 && i>=0);
}

/* A printing utility for printing the octal contents of a double
data item */
```

```
void print_bin(double a)
{
    char *b;
    int i,j,k;

    k=0;
    b=(char *)(&a);
    printf("%f ",a);
    for (i=0;i<sizeof(double);i++) {
        for(j=0;j<8;j++) {
            k++;
            if (k==2 || k==13) putchar(' ');
            printf("%d", (b[i]&0x80)==128);
            b[i]<=1;
        }
    }
    printf("\n");
}
```

/\*

The basic arithmetic operations are embedded in the class interval. In each case the pair of double data items representing the interval [lo,hi] is checked via the routine help\_. This routine accesses helpx\_ whenever

lo<0

or

hi>0.

In both cases the next double representable item is calculated in helpx\_ and the result is returned to the calling operator. In this manner it is guaranteed that the machine interval result will contain the interval result that would have been computed using infinite (real) interval arithmetic.

\*/

```
class interval {
```



```
double lo,hi;
```

```
public:
```

```
interval(double lo = 0, double hi = 0) {  
    this->lo = lo;  
    this->hi = hi;  
};
```

```
/* An interval printing utility */
```

```
void print() {  
    printf("[ %3lx, %3lx ]\n", lo, hi);  
};
```

```
friend interval help_(interval a) {  
    if(a.lo < 0 ) {  
        helpx_(&a.lo);  
    }  
    if(a.hi > 0 ){  
        helpx_(&a.hi);  
    }  
    return(a);  
}
```

```
friend interval operator+(interval a, interval b){  
    return(help_(interval(a.lo+b.lo,a.hi+b.hi)));  
};
```

```
friend interval operator-(interval a, interval b){  
    return(help_(interval(a.lo-b.hi,a.hi-b.lo)));  
};
```

```
friend interval operator*(interval a, interval b){  
    double ac,ad,bc,bd;  
    ac=a.lo * b.lo;  
    ad=a.lo * b.hi;  
    bc=a.hi * b.lo;  
    bd=a.hi * b.hi;  
    return(help_(interval(MIN(ac,ad,bc,bd),  
                          MAX(ac,ad,bc,bd))));  
};
```

```
friend interval operator/(interval a, interval b){  
    if( b.lo == 0.0 || b.hi == 0.0 ){  
        cout << form("\n bad interval for division");  
    }  
    return( a * help_(interval(1.0 / b.hi, 1.0 / b.lo)) );  
};
```

```
friend double lower_bound(interval a) {  
    return(a.lo);  
};
```

```
friend double upper_bound(interval a){  
    return(a.hi);  
};
```

```
};
```

```
main()
```

```
{  
    int mode = 1;  
    interval x(1.0, 2.0);  
    interval y(4.0, 5.0);
```

```
interval a(-1.0,1.0);
interval b,z;

    setmode_(&mode);
    a.print();
    print_bin(lower_bound(a));
    print_bin(upper_bound(a));
    b=help_(a);

    print_bin(lower_bound(b));
    print_bin(upper_bound(b));
    print_bin(-lower_bound(b));
    b.print();
    z = x + y;
    z.print();
    z = x - y;
    z.print();
    z = x * y;
    z.print();
    z = x / y;
    z.print();
}
```

```
#include <stdio.h>

#define OFFS 4 /* first & last segment min size */
#define SMAX 20 /* object min half size */

#define DIRECT 0
#define INVERSE 1

#define LEFT 0x00
#define RIGHT 0x01
#define BOTTOM 0x02
#define TOP 0x03

#define EQSIGN(x1, x2, x3, x4) ( ( x2 - x1 ) * ( x3 - x4 ) > 0 )

#define CODE(x, y, w, h, px, py, c) { \
    c = 0x00; \
    if ( px < x ) c = 0x01; \
    else if ( px >= x + w ) c = 0x02; \
    if ( py < y ) c |= 0x04; \
    else if ( py >= y + h ) c |= 0x08; \
}

typedef struct {
    int direction;
    int x, y;
} Pad;

typedef struct {
    int x, y, w, h;
} Obj;

/*****
Computes & corrects the connection path from an object Out pad to an object In pad.

Entry:
    startObj - the object to start the connection from
    endObj - the object to which the connection ends
    outPad - the Out pad to be connected
    inPad - the In pad to be connected

Exit:
    nump - the number of connection points
    pts - the connection point list
*****/

ComputeCon ( startObj, endObj, outPad, inPad, nump, pts )
    Obj *startObj;
    Obj *endObj;
    Pad *outPad;
    Pad *inPad;
    int nump;
    int pts[][2];
{
    nump = Connect(outPad->x, outPad->y, outPad->direction,
```

```
        inPad->x, inPad->y, inPad->direction, pts);
Correct(startObj->x, startObj->y, startObj->w, startObj->h, nump, pts);
Correct(endObj->x, endObj->y, endObj->w, endObj->h, nump, pts);
}
```

\*\*\*\*\*  
Computes the connection path from an object Out pad to an object In pad using  
the orthogonal path algorithm.

Entry:

```
xo    - Out pad x coordinate
yo    - Out pad y coordinate
diro  - Out pad direction
xi    - In pad x coordinate
yi    - In pad y coordinate
diri  - In pad direction
```

Exit:

```
p - the connection point list
```

Return:

```
the number of connection points
```

\*\*\*\*\*/

```
int Connect ( xo, yo, diro, xi, yi, diri, p )
int      xo, yo, diro;
int      xi, yi, diri;
int      p[1][2];
{
    static int      directs[4][4] = {
                                { TOP,    BOTTOM, LEFT,    RIGHT },
                                { BOTTOM, TOP,    RIGHT,  LEFT },
                                { RIGHT,  LEFT,   TOP,    BOTTOM },
                                { LEFT,   RIGHT,  BOTTOM,  TOP }
                                };
    int      i, np, txi, tyi;

    Transform(DIRECT, diro, xo, yo, xi, yi, &txi, &tyi);
    switch ( directs[diro][diri] ) { /* In pad direction
*/
        case LEFT:
            if ( tyi > 0 ) { /* T
*/
                if ( txi > 0 ) { /*
34 */
                    Shape3P(tx, ty, p);
                    np = 3;
                }
                else {
/* 12 */
                    Shape5PB(tx, ty, -1, p);
                    np = 5;
                }
            }
            else {
/* B */
                Shape5PB(tx, ty, -1, p);
                np = 5;
            }
            break;
        case RIGHT:
            if ( tyi > 0 ) { /* T
```

```

*/
                                if ( txi > 0 ) {                                /*
34      */
                                Shape5PB(txi, tyi, 1, p);
                                np = 5;
                                }
                                else {
/*      12      */
                                Shape3P(txi, tyi, p);
                                np = 3;
                                }
                                }
                                else {
/*      B      */
                                Shape5PB(txi, tyi, 1, p);
                                np = 5;
                                }
                                break;
                                case BOTTOM:
                                if ( tyi > 0 ) {                                /*      T
*/
                                Shape4PD(txi, tyi, p);
                                np = 4;
                                }
                                else {
/*      B      */
                                Shape6P(txi, tyi, -1, p);
                                np = 6;
                                }
                                break;
                                case TOP:
                                if ( txi < - SMAX || txi > SMAX ) {      /*      14      */
                                Shape4PU(txi, tyi, p);
                                np = 4;
                                }
                                else {
/*      23      */
                                Shape6R(txi, tyi, 1, p);
                                np = 6;
                                }
                                break;
                                }
                                for ( i = 0 ; i < np ; i++ )
                                Transform(INVERSE, diro, xo, yo, p[i][0], p[i][1], &p[i][0], &p[i][1]);
                                return (np);
                                }

```

/\*\*\*\*\*  
 Translates & rotates a point according to the Out pad direction.

Entry:

mode - transformation mode (DIRECT or INVERSE)  
 dout - Out pad direction  
 xout - Out pad x coordinate  
 yout - Out pad y coordinate  
 x, y - point to be transformed

Exit:

tx, ty - transformed point

\*\*\*\*\*/

```
Transform ( mode, dout, xout, yout, x, y, tx, ty )
{
    int     mode;
    int     dout, xout, yout;
    int     x, y;
    int     *tx, *ty;

    int     ox, oy;

    ox = x;
    oy = y;
    switch ( mode ) {
        case DIRECT:
            switch ( dout ) {
                case LEFT:
                    *tx = oy - yout;
                    *ty = xout - ox;
                    break;
                case RIGHT:
                    *tx = yout - oy;
                    *ty = ox - xout;
                    break;
                case BOTTOM:
                    *tx = xout - ox;
                    *ty = yout - oy;
                    break;
                case TOP:
                    *tx = ox - xout;
                    *ty = oy - yout;
                    break;
            }
            break;
        case INVERSE:
            switch ( dout ) {
                case LEFT:
                    *tx = xout - oy;
                    *ty = yout + ox;
                    break;
                case RIGHT:
                    *tx = xout + oy;
                    *ty = yout - ox;
                    break;
                case BOTTOM:
                    *tx = xout - ox;
                    *ty = yout - oy;
                    break;
                case TOP:
                    *tx = xout + ox;
                    *ty = yout + oy;
                    break;
            }
            break;
    }
}

/*****
Corrects the connection path moving the overcrossing segments out of the
crossed object.

Entry:
    x, y - object lower left corner
```

w, h - object width & height  
np - number of connection point  
p - original path points list

Exit:

p - eventually corrected path points list

\*\*\*\*\*/

Correct ( x, y, w, h, np, p )

int x, y, w, h;  
int np;  
int p[][2];

{

int i, j, c, op[2], od[2];

op[0] = x;

op[1] = y;

od[0] = w;

od[1] = h;

```
for ( j = 1, i = 2 ; i < np - 1 ; i++, j++ ) {
    if ( Overcross(x, y, w, h, p[j][0], p[j][1], p[i][0], p[i][1]) ) {
        c = ( p[j][0] == p[i][0] ) ? 0 : 1;
        if ( EQSIGN(p[j-1][c], p[j][c], p[i][c], p[i+1][c]) ) {
            if ( p[j][c] < p[j-1][c] )
                p[j][c] = p[i][c] = op[c] - OFFS + 2;
            else p[j][c] = p[i][c] = op[c] + od[c] - 1 + OFFS - 2;
        }
        else {
            if ( p[i][c] < p[i+1][c] )
                p[j][c] = p[i][c] = op[c] - OFFS + 2;
            else p[j][c] = p[i][c] = op[c] + od[c] - 1 + OFFS - 2;
        }
    }
}
```

\*\*\*\*\*

Implementation of line-rectangle intersection test as the first test in Cohen-Sutherland clipping algorithm.

Entry:

x, y - rectangle lower left corner  
w, h - rectangle width & height  
x1, y1 - line start point  
x2, y2 - line end point

Return:

TRUE if the line overcrosses the rectangle, FALSE otherwise

\*\*\*\*\*/

int Overcross ( x, y, w, h, x1, y1, x2, y2 )

int x, y, w, h;  
int x1, y1, x2, y2;

{

int c1, c2;

CODE(x, y, w, h, x1, y1, c1);

CODE(x, y, w, h, x2, y2, c2);

return(! ( c1 & c2));

}

```

/*****
Shape functions.

```

Entry:

```

    xi, yi - In pad coordinate
    [sign] - sign flag

```

Exit:

```

    xy - the computed path points list

```

```

*****/

```

```

Shape3P ( xi, yi, xy )
    int      xi, yi;
    int      xy[][2];
{
    xy[0][0] = 0;
    xy[0][1] = 0;
    xy[1][0] = 0;
    xy[1][1] = yi;
    xy[2][0] = xi;
    xy[2][1] = yi;
}

```

```

Shape4PD ( xi, yi, xy )
    int      xi, yi;
    int      xy[][2];
{
    xy[0][0] = 0;
    xy[0][1] = 0;
    xy[1][0] = 0;
    xy[1][1] = OFFS;
    xy[2][0] = xi;
    xy[2][1] = xy[1][1];
    xy[3][0] = xi;
    xy[3][1] = yi;
}

```

```

Shape4PU ( xi, yi, xy )
    int      xi, yi;
    int      xy[][2];
{
    xy[0][0] = 0;
    xy[0][1] = 0;
    xy[1][0] = 0;
    xy[1][1] = ( ( yi <= 0 ) ? 0 : yi ) + OFFS;
    xy[2][0] = xi;
    xy[2][1] = xy[1][1];
    xy[3][0] = xi;
    xy[3][1] = yi;
}

```

```

Shape5PB ( xi, yi, sign, xy )
    int      xi, yi;
    int      sign;
    int      xy[][2];
{
    xy[0][0] = 0;
    xy[0][1] = 0;
    xy[1][0] = 0;

```



```
xy[1][1] = OFFS;
if ( sign * xi < - SMAX ) xy[2][0] = xi + sign * OFFS;
else xy[2][0] = ( ( sign * xi < 0 ) ? 0 : xi ) + sign * OFFS;
xy[2][1] = xy[1][1];
xy[3][0] = xy[2][0];
xy[3][1] = yi;
xy[4][0] = xi;
xy[4][1] = yi;
```

```
}
```

```
Shape5PT ( xi, yi, sign, xy )
```

```
int      xi, yi;
int      sign;
int      xy[][2];

{
    xy[0][0] = 0;
    xy[0][1] = 0;
    xy[1][0] = 0;
    xy[1][1] = OFFS;
    xy[2][0] = xi + sign * OFFS;
    xy[2][1] = xy[1][1];
    xy[3][0] = xy[2][0];
    xy[3][1] = yi;
    xy[4][0] = xi;
    xy[4][1] = yi;
}
```

```
Shape6P ( xi, yi, sign, xy )
```

```
int      xi, yi;
int      sign;
int      xy[][2];

{
    int      xm;

    xm      = xi / 2;
    xy[0][0] = 0;
    xy[0][1] = 0;
    xy[1][0] = 0;
    xy[1][1] = OFFS;
    if ( yi < 0 ) {
        if ( xi > 0 ) xy[2][0] = ( xm > SMAX ) ? xm : SMAX;
        else xy[2][0] = ( xm > SMAX ) ? xm : - SMAX;
    }
    else {
        if ( xi > 0 ) xy[2][0] = ( xm > SMAX ) ? xm : - SMAX;
        else xy[2][0] = ( xm > SMAX ) ? xm : SMAX;
    }
    xy[2][1] = xy[1][1];
    xy[3][0] = xy[2][0];
    xy[3][1] = yi + sign * OFFS;
    xy[4][0] = xi;
    xy[4][1] = xy[3][1];
    xy[5][0] = xi;
    xy[5][1] = yi;
}
```

```
/*  
GROUPING NEARLY COPLANAR POLYGONS INTO COPLANAR SETS
```

David Salesin  
Filippo Tampieri

Cornell University  
\*/

```
/*  
    This code partitions a given set of arbitrary 3D polygons into  
    subsets of coplanar polygons.  
    The input polygons are organized in a linked list and the  
    resulting subsets of coplanar polygons are returned in the form  
    of a binary tree; each node of the binary tree contains a  
    different subset (once again stored as a linked list) and its  
    plane equation. An inorder traversal of the binary tree will  
    return the sets of coplanar polygons sorted by plane equations  
    according to the total ordering imposed by the relation  
    implemented by the routine 'comparePlaneEqs'.  
*/
```

```
#include <stdio.h>  
#include <math.h>
```

```
#define X    0  
#define Y    1  
#define Z    2  
#define D    3
```

```
#define VZERO(v)    (v[X] = v[Y] = v[Z] = 0.0)  
#define VNORM(v)    (sqrt(v[X] * v[X] + v[Y] * v[Y] + v[Z] * v[Z]))  
#define VDOT(u, v)  (u[0] * v[0] + u[1] * v[1] + u[2] * v[2])  
#define VINCR(u, v) (u[X] += v[X], u[Y] += v[Y], u[Z] += v[Z])
```

```
typedef float Vector[3];  
typedef Vector Point;  
typedef Vector Normal;  
typedef float Plane[4];
```

```
/*  
    Polygon--a polygon is stored as an array 'vertices' of size  
    'numVertices'. Pointer 'next' is used to implement sets of  
    polygons through linked lists.  
*/
```

```
typedef struct polygon {  
    Point *vertices;  
    int numVertices;  
    struct polygon *next;  
} Polygon;
```

```
/*
```

Node--each node stores a set of coplanar polygons. The set is implemented as a linked list pointed to by 'plist', and the plane equation of the set is stored in 'plane'. Pointers 'left' and 'right' point to the subtrees containing sets of coplanar polygons with plane equations respectively less than and greater than that stored in 'plane'.

```
*/
typedef struct node {
    Plane plane;
    Polygon *plist;
    struct node *left, *right;
} Node;

static float linEps; /* tolerances used by 'comparePlaneEq' to */
static float cosEps; /* account for numerical errors */

/* declare local routines */
static void computePlaneEq() ;
static Node *insertPlaneEq() ;
static int comparePlaneEqs() ;

/*
coplanarSets--takes as input a linked list of polygons 'plist',
and two tolerances 'linearEps' and 'angularEps' and returns a
binary tree of sets of coplanar polygons.
The tolerances are used to deal with floating-point arithmetic
and numerical errors when comparing plane equations; two plane
equations are considered equal if the angle between their
respective normals is less than or equal to angularEps (in
degrees) and the distance between the two planes is less than
or equal to linearEps.
*/
Node *coplanarSets(plist, linearEps, angularEps)
Polygon *plist;
float linearEps, angularEps;
{
    Node *tree;
    Plane plane;
    void computePlaneEq();
    Node *pset, *insertPlaneEq();
    Polygon *polygon, *nextPolygon;

    /* initialize the tolerances used by comparePlaneEqs() */
    linEps = linearEps;
    cosEps = cos(angularEps * M_PI / 180.0);

    /* place each input polygon in the appropriate set
       of coplanar polygons
    */
    tree = NULL;
    polygon = plist;
    while(polygon != NULL) {
        nextPolygon = polygon->next;
        /* first, compute the plane equation of the polygon */
        computePlaneEq(polygon, plane);
        /* then, find the set of coplanar polygons with this plane
           equation or create a new, empty one if none exist
        */
        tree = insertPlaneEq(tree, plane, &pset);
        /* finally, add the polygon to the selected set of
           coplanar polygons
        */
    }
}
```

```
    */
    polygon->next = pset->plist;
    pset->plist = polygon;
    /* go to the next polygon in the input list. Note that the
       'next' field in the polygon structure is reused to
       assemble lists of coplanar polygons; thus the necessity
       for 'nextPolygon'
    */
    polygon = nextPolygon;
}

return(tree);
}

/*
computePlaneEq--takes as input a pointer 'polygon' to an
arbitrary 3D polygon and returns in 'plane' the normalized
(unit normal) plane equation of the polygon.
Newell's method (see "Newell's Method for Computing the Plane
Equation of a Polygon" in this volume) is used for the
computation.
*/
static void computePlaneEq(polygon, plane)
Polygon *polygon;
Plane plane;
{
    int i;
    Point refpt;
    Normal normal;
    float *u, *v, len;

    /* first, compute the normal of the input polygon */
    VZERO(normal);
    VZERO(refpt);
    for(i = 0; i < polygon->numVertices; i++) {
        u = polygon->vertices[i];
        v = polygon->vertices[(i + 1) % polygon->numVertices];
        normal[X] += (u[Y] - v[Y]) * (u[Z] + v[Z]);
        normal[Y] += (u[Z] - v[Z]) * (u[X] + v[X]);
        normal[Z] += (u[X] - v[X]) * (u[Y] + v[Y]);
        VINCR(refpt, u);
    }
    /* then, compute the normalized plane equation using the
       arithmetic average of the vertices of the input polygon to
       determine its last coefficient. Note that, for efficiency,
       'refpt' stores the sum of the vertices rather than the
       actual average; the division by 'polygon->numVertices' is
       carried out together with the normalization when computing
       'plane[D]'.
    */
    len = VNORM(normal);
    plane[X] = normal[X] / len;
    plane[Y] = normal[Y] / len;
    plane[Z] = normal[Z] / len;
    len *= polygon->numVertices;
    plane[D] = -VDOT(refpt, normal) / len;
}

/*
insertPlaneEq--takes as input a binary tree 'tree' of sets of
coplanar polygons, and a plane equation 'plane', and returns
```

```
    a pointer 'pset' to a set of coplanar polygons with the given
    plane equation. A new, empty set is created if none is found.
*/

static Node *insertPlaneEq(tree, plane, pset)
Node *tree, **pset;
Plane plane;
{
    int i, c, comparePlaneEqs();

    if(tree == NULL) {
        /* the input plane is not in the tree.
           Create a new set for it
        */
        tree = (Node *)malloc(sizeof(Node));
        for(i = 0; i < 4; i++)
            tree->plane[i] = plane[i];
        tree->plist = NULL; /* no polygons in this set for now */
        tree->left = tree->right = NULL;
        *pset = tree;
    } else {
        /* compare the input plane equation with that
           of the visited node
        */
        c = comparePlaneEqs(plane, tree->plane);
        if(c < 0)
            tree->left = insertPlaneEq(tree->left, plane, pset);
        else if(c > 0)
            tree->right = insertPlaneEq(tree->right, plane, pset);
        else
            /* the input plane is approximately equal to that
               of this node
            */
            *pset = tree;
    }

    return(tree);
}

/*
comparePlaneEqs--compares two plane equations, 'p1' and 'p2',
and returns an integer less than, equal to, or greater than
zero, depending on whether 'p1' is less than, equal to, or
greater than 'p2'. The two global variables, 'linEps' and
'cosEps' are tolerances used to account for numerical errors.
*/
static int comparePlaneEqs(p1, p2)
Plane p1, p2;
{
    int ret;
    float cosTheta;

    /* compute the cosine of the angle between the normals
       of the two planes
    */
    cosTheta = VDOT(p1, p2);

    if(cosTheta < 0.0)
        ret = -1;
    else if(cosTheta < cosEps)
        ret = 1;
```

```
else {
    /* the two planes are parallel and oriented in
       the same direction
    */
    if(p1[3] + linEps < p2[3])
        ret = -1;
    else if(p2[3] + linEps < p1[3])
        ret = 1;
    else
        /* the two planes are equal */
        ret = 0;
}

return ret;
}
```

```
/*
 * ANSI C code from the article
 * "Fast Embossing Effects on Raster Image Data"
 * by John Schlag, jfs@kerner.com
 * in "Graphics Gems IV", Academic Press, 1994
 *
 *
 * Emboss - shade 24-bit pixels using a single distant light source.
 * Normals are obtained by differentiating a monochrome 'bump' image.
 * The unary case ('texture' == NULL) uses the shading result as output.
 * The binary case multiplies the optional 'texture' image by the shade.
 * Images are in row major order with interleaved color components (rgrgb...).
 * E.g., component c of pixel x,y of 'dst' is dst[3*(y*xSize + x) + c].
 *
 * To compile a test program on an SGI:
 *      cc -DMAIN emboss.c -lgl_s -lm -o emboss
 * The code for the Emboss routine itself is portable to most machines.
 */

#include <math.h>
#include <sys/types.h>

void
Emboss(
    double azimuth, double elevation, /* light source direction */
    u_short width45,                /* filter width */
    u_char *bump,                    /* monochrome bump image */
    u_char *texture, u_char *dst,    /* texture & output images */
    u_short xSize, u_short ySize    /* image size */
)
{
    long Nx, Ny, Nz, Lx, Ly, Lz, Nz2, NzLz, NdotL;
    register u_char *s1, *s2, *s3, shade, background;
    register u_short x, y;

#define pixelScale 255.9

    /*
     * compute the light vector from the input parameters.
     * normalize the length to pixelScale for fast shading calculation.
     */
    Lx = cos(azimuth) * cos(elevation) * pixelScale;
    Ly = sin(azimuth) * cos(elevation) * pixelScale;
    Lz = sin(elevation) * pixelScale;

    /*
     * constant z component of image surface normal - this depends on the
     * image slope we wish to associate with an angle of 45 degrees, which
     * depends on the width of the filter used to produce the source image.
     */
    Nz = (6 * 255) / width45;
    Nz2 = Nz * Nz;
    NzLz = Nz * Lz;

    /* optimization for vertical normals: L.[0 0 1] */
    background = Lz;

    /* mung pixels, avoiding edge pixels */
    dst += xSize*3;
    if (texture) texture += xSize*3;
    for (y = 1; y < ySize-1; y++, bump += xSize, dst += 3)
```

```
{
    s1 = bump + 1;
    s2 = s1 + xSize;
    s3 = s2 + xSize;
    dst += 3;
    if (texture) texture += 3;
    for (x = 1; x < xSize-1; x++, s1++, s2++, s3++)
    {
        /*
         * compute the normal from the bump map. the type of the expression
         * before the cast is compiler dependent. in some cases the sum is
         * unsigned, in others it is signed. ergo, cast to signed.
         */
        Nx = (int)(s1[-1] + s2[-1] + s3[-1] - s1[1] - s2[1] - s3[1]);
        Ny = (int)(s3[-1] + s3[0] + s3[1] - s1[-1] - s1[0] - s1[1]);

        /* shade with distant light source */
        if ( Nx == 0 && Ny == 0 )
            shade = background;
        else if ( (NdotL = Nx*Lx + Ny*Ly + NzLz) < 0 )
            shade = 0;
        else
            shade = NdotL / sqrt(Nx*Nx + Ny*Ny + Nz2);

        /* do something with the shading result */
        if ( texture ) {
            *dst++ = (*texture++ * shade) >> 8;
            *dst++ = (*texture++ * shade) >> 8;
            *dst++ = (*texture++ * shade) >> 8;
        }
        else {
            *dst++ = shade;
            *dst++ = shade;
            *dst++ = shade;
        }
    }
    if (texture) texture += 3;
}

#ifdef MAIN

#define TEXTURE 1

main()
{
    #define xSize 200
    #define ySize 200
    u_char bump[ySize][xSize];
    u_char texture[ySize][xSize][3], *txt = 0;
    u_char dst[ySize][xSize][3];
    int i, j;

    /* make a simple input image */
    memset(bump, 0, sizeof(bump));
    for(i = xSize/4; i < 3*xSize/4; i++)
        for(j = ySize/4; j < 3*ySize/4; j++)
            bump[j][i] = 128;

    #if TEXTURE
        /* make a texture */
    #endif
}
```



```
for(i = 0; i < xSize; i++)
    for(j = 0; j < ySize; j++) {
        texture[j][i][0] = random() >> (31 - 8);
        texture[j][i][1] = random() >> (31 - 8);
        texture[j][i][2] = random() >> (31 - 8);
    }
txt = (u_char *)texture;
#endif







/* emboss it */
memset(dst, 0, sizeof(dst));
#define dToR(d) ((d)*(M_PI/180))
Emboss(dToR(30), dToR(30), 3, (u_char *)bump, txt, (u_char *)dst,
    xSize, ySize);

/* display it (sgi component order is ABGR) */
prefsize(xSize, ySize);
winopen("emboss");
RGBmode();
gconfig();
for(i = 0; i < ySize; i++) {
    u_long line[xSize];
    u_char *lp = (u_char *)line;
    for(j = 0; j < xSize; j++) {
        *lp++ = 255;
        *lp++ = dst[i][j][2];
        *lp++ = dst[i][j][1];
        *lp++ = dst[i][j][0];
    }
    lrectwrite(0, ySize-1 - i, xSize-1, ySize-1 - i, line);
}
sleep(30);
}

#endif
```

# Index of

## /pubs/tog/GraphicsGems/gemsiv/dyn\_range/

Name	Last modified	Size	Description
 <a href="#">_Parent Directory</a>			
 <a href="#">_Makefile</a>	29-Jun-00 08:19	1K	
 <a href="#">_README</a>	29-Jun-00 08:19	1K	
 <a href="#">_hdp.c</a>	29-Jun-00 08:19	2K	
 <a href="#">_hdp.h</a>	29-Jun-00 08:19	1K	
 <a href="#">_test_hdp.c</a>	29-Jun-00 08:19	1K	

```
CC = gcc
```

```
test_hdp: test_hdp.o hdp.o
    $(CC) -o test_hdp test_hdp.o hdp.o
```

```
clean:
    rm -f *.o test_hdp
```

C code from the article

"High Dynamic Range Pixels"

by Christophe Schlick, [schlick@labri.u-bordeaux.fr](mailto:schlick@labri.u-bordeaux.fr)

in "Graphics Gems IV", Academic Press, 1994

files:

Makefile

hdp.c - C source file

hdp.h - header file

test\_hdp.c - test program

```
/*
** HDP.C : Encoding and Decoding of High Dynamic Range Pixels
*/

#include <stdio.h>
void *malloc(unsigned int);
#include "hdp.h"

/*
** Encoding and Decoding Look-Up Tables
*/
byte *EncodeLut;
real *DecodeLut;
real LutScale;

/*
** Construction of the Encoding Look-Up Table
**
** Input :
**     LoVal = Less significant (ie lowest non zero) value of the incoming range
**     HiVal = Most significant (ie highest) value of the incoming range
**     NbVal = Number of elements in the encoding LUT
**
** Output :
**     The function returns 0 if the allocation failed
*/
int init_HDP_encode (real LoVal, real HiVal, int NbVal)
{
    real t, r;
    int n;

    EncodeLut = (byte *) malloc (NbVal * sizeof (byte));
    if (! EncodeLut) return (NULL);

    NbVal--;

    /* Scaling factor = ratio between the encoding LUT and the incoming range */
    LutScale = NbVal / HiVal;

    /* Bias factor = ratio between the outcoming and the incoming range */
    r = 256.0 * LoVal / HiVal;

    for (n = 0; n <= NbVal; n++) {
        t = (float) n / NbVal;
        EncodeLut[n] = 255.0 * t / (t-r*t+r) + 0.5;
    }
    return (! NULL);
}

/*
** Destruction of the Encoding Look-Up Table
*/
void exit_HDP_encode (void)
{
    free (EncodeLut);
}

/*
** Construction of the Decoding Look-Up Table
**
** Input :
```

```
**      LoVal   = Less significant (ie lowest non zero) value of the incoming range
**      HiVal   = Most significant (ie highest) value of the incoming range
**      Bright  = Brightness factor in the range (-1,1)
**              (Bright < 0 : Image is darkened, Bright > 0 : Image is lightened)
**
** Output :
**      The function returns 0 if the allocation failed
**/
int init_HDP_decode (real LoVal, real HiVal, real Bright)
{
    float t, r;
    int n;

    DecodeLut = (real *) malloc (256 * sizeof (real));
    if (! DecodeLut) return (NULL);

/* Change Bright from (-1,1) into a scaling coefficient (0,infinity) */
    Bright = Bright < 0.0 ? Bright+1.0 : 1.0 / (1.0-Bright);

/* Bias factor = ratio of incoming and outcoming range * brightness factor */
    r = Bright * HiVal / LoVal / 256.0;

    for (n = 0; n <= 256; n++) {
        t = (float) n / 255.0;
        DecodeLut[n] = t / (t-t*r+r) * HiVal;
    }
    return (! NULL);
}

/*
** Destruction of the Decoding Look-Up Table
**/
void exit_HDP_decode (void)
{
    free (DecodeLut);
}
```

```
/*
** HDP.H : Encoding and Decoding of High Dynamic Range Pixels
*/

/*
** Encoding and Decoding Types
*/
typedef unsigned char byte;
typedef float real; /* change to "double" if you work with double precision */

typedef byte bytecolor[3];
typedef real realcolor[3];

/*
** Encoding and Decoding Tables
*/
extern byte *EncodeLut;
extern real *DecodeLut;
extern real LutScale;

/*
** Encoding and Decoding Functions
*/
extern int init_HDP_encode (real,real,int);
extern int init_HDP_decode (real,real,real);
extern void exit_HDP_encode (void);
extern void exit_HDP_decode (void);

/*
** Encoding and Decoding Macros
*/
#define HDP_ENCODE(RealColor,ByteColor) ( \
    ByteColor[0] = EncodeLut [(int) (RealColor[0] * LutScale + 0.5)], \
    ByteColor[1] = EncodeLut [(int) (RealColor[1] * LutScale + 0.5)], \
    ByteColor[2] = EncodeLut [(int) (RealColor[2] * LutScale + 0.5)])

#define HDP_DECODE(ByteColor,RealColor) ( \
    RealColor[0] = DecodeLut [ByteColor[0]], \
    RealColor[1] = DecodeLut [ByteColor[1]], \
    RealColor[2] = DecodeLut [ByteColor[2]])
```

```
/*
** TEST_HDP.C : Simple testing program for the HDP routines
*/

#include "hdp.h"

void main (void)
{
    realcolor RealColor;
    bytecolor ByteColor;
    real LoVal, HiVal, Bright;
    int Index, NbTst, NbVal;

/* Dynamic range of 4000 (try also larger or smaller values) */
    LoVal = 0.25;
    HiVal = 1000.0;

/* Memory for encoding LUT = 8 Kbytes */
    NbVal = 8192;

/* Construction of the encoding LUT */
    init_HDP_encode (LoVal, HiVal, NbVal);

/* No brightness modification (try also negative and positive values) */
    Bright = 0.0;

/* Construction of the decoding LUT */
    init_HDP_decode (LoVal, HiVal, Bright);







/*
    Test NbTst sample values (ranging from 0 to HiVal)
    before and after a coding/decoding sequence
*/
    NbTst = HiVal / LoVal;
    for (Index = 0; Index <= NbTst; Index++) {
        RealColor[0] = RealColor[1] = RealColor[2] = Index * HiVal / NbTst;
        printf ("Before = %.2f\t", RealColor[0]);
        HDP_ENCODE (RealColor, ByteColor);
        printf ("Coded value = %d\t", ByteColor[0]);
        HDP_DECODE (ByteColor, RealColor);
        printf ("After = %.2f\n", RealColor[0]);
    }

/* Destruction of the look-up tables */
    exit_HDP_encode ();
    exit_HDP_decode ();
}
```



# Index of

## /pubs/tog/GraphicsGems/gemsv/ch5-2/

Name	Last modified	Size	Description
 <a href="#">_Parent Directory</a>			
 <a href="#">_README</a>	29-Jun-00 08:23	1K	
 <a href="#">_quad.c</a>	29-Jun-00 08:23	4K	
 <a href="#">_quad.h</a>	29-Jun-00 08:23	1K	
 <a href="#">_quad_gg.c</a>	29-Jun-00 08:23	4K	
 <a href="#">_quad_gg.h</a>	29-Jun-00 08:23	1K	

NOTE: These routines borrow the include files found in

- ../ch7-7/mactbox/tool.h
- ../ch7-7/mactbox/real.h
- ../ch7-7/mactbox/vec2.h
- ../ch7-7/mactbox/vec3.h

```
/* ----- */
QUAD.C :

by Christophe Schlick and Gilles Subrenat (15 May 1994)

"Ray Intersection of Tessellated Surfaces : Quadrangles versus Triangles"
in Graphics Gems V (edited by A. Paeth), Academic Press
/* ----- */

#include "quad.h"

/*
** Macro definitions
*/
#define MY_TOL ((real) 0.0001)

#define LARGEST_COMPONENT(A) (ABS((A).x) > ABS((A).y) ? \
                             (ABS((A).x) > ABS((A).z) ? 'x' : 'z') : \
                             (ABS((A).y) > ABS((A).z) ? 'y' : 'z'))

/*
** Check if the point is in the quadrangle
*/
static bool point_in_quad (QUAD *Quad, HIT *Hit)
{
    char          LargestComponent;          /* of the normal vector          */
    realvec2      A, B, C, D;                /* Projected vertices          */
    realvec2      M;                          /* Projected intersection point */
    realvec2      AB, BC, CD, AD, AM, AE;     /* Miscellaneous 3D-vectors    */
    real          u, v;                       /* Parametric coordinates      */
    real          a, b, c, SqrtDelta;         /* Quadratic equation          */
    bool          Intersection = FALSE;        /* Intersection flag            */
    realvec2      Vector;                     /* Temporary 2D-vector         */

    /*
    ** Projection on the plane that is most parallel to the facet
    */
    LargestComponent = LARGEST_COMPONENT(Quad->Normal);

    if (LargestComponent == 'x') {
        A.x = Quad->A.y; B.x = Quad->B.y; C.x = Quad->C.y; D.x = Quad->D.y;
        M.x = Hit->Point.y;
    }
    else {
        A.x = Quad->A.x; B.x = Quad->B.x; C.x = Quad->C.x; D.x = Quad->D.x;
        M.x = Hit->Point.x;
    }

    if (LargestComponent == 'z') {
        A.y = Quad->A.y; B.y = Quad->B.y; C.y = Quad->C.y; D.y = Quad->D.y;
        M.y = Hit->Point.y;
    }
    else {
        A.y = Quad->A.z; B.y = Quad->B.z; C.y = Quad->C.z; D.y = Quad->D.z;
        M.y = Hit->Point.z;
    }

    SUB_VEC2 (AB, B, A); SUB_VEC2 (BC, C, B);
    SUB_VEC2 (CD, D, C); SUB_VEC2 (AD, D, A);
    ADD_VEC2 (AE, CD, AB); NEG_VEC2 (AE, AE); SUB_VEC2 (AM, M, A);
```

```
if (ZERO_TOL (DELTA_VEC2(AB, CD), MY_TOL))          /* case AB // CD */
{
    SUB_VEC2 (Vector, AB, CD);
    v = DELTA_VEC2(AM, Vector) / DELTA_VEC2(AD, Vector);
    if ((v >= 0.0) && (v <= 1.0)) {
        b = DELTA_VEC2(AB, AD) - DELTA_VEC2(AM, AE);
        c = DELTA_VEC2 (AM, AD);
        u = ZERO_TOL(b, MY_TOL) ? -1.0 : c/b;
        Intersection = ((u >= 0.0) && (u <= 1.0));
    }
}
else if (ZERO_TOL(DELTA_VEC2(BC, AD), MY_TOL))      /* case AD // BC */
{
    ADD_VEC2 (Vector, AD, BC);
    u = DELTA_VEC2(AM, Vector) / DELTA_VEC2(AB, Vector);
    if ((u >= 0.0) && (u <= 1.0)) {
        b = DELTA_VEC2(AD, AB) - DELTA_VEC2(AM, AE);
        c = DELTA_VEC2 (AM, AB);
        v = ZERO_TOL(b, MY_TOL) ? -1.0 : c/b;
        Intersection = ((v >= 0.0) && (v <= 1.0));
    }
}
else                                                /* general case */
{
    a = DELTA_VEC2(AB, AE); c = - DELTA_VEC2 (AM,AD);
    b = DELTA_VEC2(AB, AD) - DELTA_VEC2(AM, AE);
    a = -0.5/a; b *= a; c *= (a + a); SqrtDelta = b*b + c;
    if (SqrtDelta >= 0.0) {
        SqrtDelta = sqrt(SqrtDelta);
        u = b - SqrtDelta;
        if ((u < 0.0) || (u > 1.0))          /* we want u between 0 and 1 */
            u = b + SqrtDelta;
        if ((u >= 0.0) && (u <= 1.0)) {
            v = AD.x + u * AE.x;
            if (ZERO_TOL(v, MY_TOL))
                v = (AM.y - u * AB.y) / (AD.y + u * AE.y);
            else
                v = (AM.x - u * AB.x) / v;
            Intersection = ((v >= 0.0) && (v <= 1.0));
        }
    }
}

if (Intersection) {
    Hit->u = u;
    Hit->v = v;
}
return (Intersection);
}

/*
** Search for an intersection between a quadrangle and a ray
*/
bool hit_ray_quad (RAY *Ray, QUAD *Quad, HIT *Hit)
{
    realvec3      Point;

    /* if the ray is parallel to the quadrangle, there is no intersection */
    Hit->Distance = DOT_VEC3 (Ray->Vector, Quad->Normal);
    if (ZERO_TOL(Hit->Distance, MY_TOL)) return (FALSE);
```

```
/* compute ray intersection with the plane of the quadrangle */
SUB_VEC3 (Point, Quad->A, Ray->Point);
Hit->Distance = DOT_VEC3 (Point, Quad->Normal) / Hit->Distance;
MULS_VEC3 (Hit->Point, Ray->Vector, Hit->Distance);
INC_VEC3 (Hit->Point, Ray->Point);

/* is the point in the quadrangle ? */
return (point_in_quad(Quad, Hit));
}

/* ----- */
```

```
/* ----- */
QUAD.H :

by Christophe Schlick and Gilles Subrenat (15 May 1994)

"Ray Intersection of Tessellated Surfaces : Quadrangles versus Triangles"
in Graphics Gems V (edited by A. Paeth), Academic Press
/* ----- */

#ifndef _QUAD_
#define _QUAD_

#include <math.h>
#include "../ch7-7/mactbox/tool.h"
#include "../ch7-7/mactbox/real.h"
#include "../ch7-7/mactbox/vec2.h"
#include "../ch7-7/mactbox/vec3.h"

/*
** Type definitions
*/
typedef struct {
    realvec3  A,B,C,D;    /* Vertices in counter clockwise order */
    realvec3  Normal;     /* Normal vector pointing outwards */
} QUAD;

typedef struct {
    realvec3  Point;      /* Ray origin */
    realvec3  Vector;     /* Ray direction */
} RAY;

typedef struct {
    realvec3  Point;      /* Intersection point */
    real      Distance;   /* Distance from ray origin to intersection point */
    real      u, v;       /* Parametric coordinates of the intersection point */
} HIT;

/*
** External declarations
*/
extern bool ray_hit_quad (RAY *, QUAD *, HIT *);

#endif

/* ----- */
```

```
/* ----- */
QUAD_GG.C :

by Christophe Schlick and Gilles Subrenat (15 May 1994)

"Ray Intersection of Tessellated Surfaces : Quadrangles versus Triangles"
in Graphics Gems V (edited by A. Paeth), Academic Press
/* ----- */

#include "quad_gg.h"

/*****
 * macro definitions *
 *****/
#define EPSILON ((REAL) 0.0001)

#define DETERMINANT(A,B) ((REAL) ((A).x * (B).y - (A).y * (B).x))

#define V3_LIN(R,A,k,B) ((R).x = (A).x + k * (B).x, \
                        (R).y = (A).y + k * (B).y, \
                        (R).z = (A).z + k * (B).z)

#define LARGEST_COMPONENT(A) (ABS((A).x) > ABS((A).y) ? \
                             (ABS((A).x) > ABS((A).z) ? 'x' : 'z') : \
                             (ABS((A).y) > ABS((A).z) ? 'y' : 'z'))

/*
** Compute uv-coordinates of the point
** Return TRUE if the point is in the quadrangle
*/
static boolean point_in_quad (QUAD *Quad, HIT *Hit)
{
    char          LargestComponent;          /* of the normal vector          */
    Point2        A, B, C, D;               /* Projected vertices          */
    Point2        M;                       /* Projected intersection point */
    Vector2        AB, BC, CD, AD, AM, AE;   /* AE = DC - AB = DA - CB      */
    REAL          u, v;                   /* Parametric coordinates      */
    REAL          a, b, c, SqrtDelta;        /* for the quadratic equation   */
    boolean        IsInside = FALSE;         /* if u and v are inside        */
    Vector2        Vector;                 /* temporary 2D-vector         */

    /*
    ** Projection on the plane that is most parallel to the facet
    */
    LargestComponent = LARGEST_COMPONENT(Quad->Normal);

    if (LargestComponent == 'x') {
        A.x = Quad->A.y; B.x = Quad->B.y; C.x = Quad->C.y; D.x = Quad->D.y;
        M.x = Hit->Point.y;
    }
    else {
        A.x = Quad->A.x; B.x = Quad->B.x; C.x = Quad->C.x; D.x = Quad->D.x;
        M.x = Hit->Point.x;
    }

    if (LargestComponent == 'z') {
        A.y = Quad->A.y; B.y = Quad->B.y; C.y = Quad->C.y; D.y = Quad->D.y;
        M.y = Hit->Point.y;
    }
    else {
```

```
A.y = Quad->A.z; B.y = Quad->B.z; C.y = Quad->C.z; D.y = Quad->D.z;
M.y = Hit->Point.z;
}

V2Sub (&B, &A, &AB); V2Sub (&C, &B, &BC);
V2Sub (&D, &C, &CD); V2Sub (&D, &A, &AD);
V2Add (&CD, &AB, &AE); V2Negate (&AE); V2Sub (&M, &A, &AM);

if (fabs(DETERMINANT(AB, CD)) < EPSILON) /* case AB // CD */
{
    V2Sub (&AB, &CD, &Vector);
    v = DETERMINANT(AM, Vector) / DETERMINANT(AD, Vector);
    if ((v >= 0.0) && (v <= 1.0)) {
        b = DETERMINANT(AB, AD) - DETERMINANT(AM, AE);
        c = DETERMINANT (AM, AD);
        u = ABS(b) < EPSILON ? -1 : c/b;
        IsInside = ((u >= 0.0) && (u <= 1.0));
    }
}
else if (fabs(DETERMINANT(BC, AD)) < EPSILON) /* case AD // BC */
{
    V2Add (&AD, &BC, &Vector);
    u = DETERMINANT(AM, Vector) / DETERMINANT(AB, Vector);
    if ((u >= 0.0) && (u <= 1.0)) {
        b = DETERMINANT(AD, AB) - DETERMINANT(AM, AE);
        c = DETERMINANT (AM, AB);
        v = ABS(b) < EPSILON ? -1 : c/b;
        IsInside = ((v >= 0.0) && (v <= 1.0));
    }
}
else /* general case */
{
    a = DETERMINANT(AB, AE); c = - DETERMINANT (AM,AD);
    b = DETERMINANT(AB, AD) - DETERMINANT(AM, AE);
    a = -0.5/a; b *= a; c *= (a + a); SqrtDelta = b*b + c;
    if (SqrtDelta >= 0.0) {
        SqrtDelta = sqrt(SqrtDelta);
        u = b - SqrtDelta;
        if ((u < 0.0) || (u > 1.0)) /* to choose u between 0 and 1 */
            u = b + SqrtDelta;
        if ((u >= 0.0) && (u <= 1.0)) {
            v = AD.x + u * AE.x;
            if (ABS(v) < EPSILON)
                v = (AM.y - u * AB.y) / (AD.y + u * AE.y);
            else
                v = (AM.x - u * AB.x) / v;
            IsInside = ((v >= 0.0) && (v <= 1.0));
        }
    }
}

if (IsInside) {
    Hit->u = u;
    Hit->v = v;
}
return (IsInside);
}

/*
** Search for an intersection between a facet and a ray
**/
```



```
boolean hit_ray_quad (RAY *Ray, QUAD *Quad, HIT *Hit)
{
    Point3      Point;

    /* if the ray is parallel to the facet, there is no intersection */
    Hit->Distance = V3Dot (&(Ray->Vector), &(Quad->Normal));
    if (ABS(Hit->Distance) < EPSILON) return (FALSE);

    /* compute ray intersection with the plane of the facet */
    V3Sub (&(Quad->A), &(Ray->Point), &Point);
    Hit->Distance = V3Dot (&Point, &(Quad->Normal)) / Hit->Distance;
    V3_LIN (Hit->Point, Ray->Point, Hit->Distance, Ray->Vector);

    /* is the intersection point inside the facet */
    return (point_in_quad(Quad, Hit));
}

/* ----- */
```

```
/* ----- */
QUAD_GG.H :

by Christophe Schlick and Gilles Subrenat (15 May 1994)

"Ray Intersection of Tessellated Surfaces : Quadrangles versus Triangles"
in Graphics Gems V (edited by A. Paeth), Academic Press
/* ----- */

#ifndef _QUAD_GG_
#define _QUAD_GG_

#include <math.h>
#include "../ch7-7/GG4D/GGems.h"

/*
** Type definitions
*/
typedef double    REAL;

typedef struct {
    Point3    A,B,C,D;    /* Vertices in counter clockwise order */
    Vector3    Normal;    /* Normal vector pointing outwards */
} QUAD;

typedef struct {
    Point3    Point;    /* Ray origin */
    Vector3    Vector;    /* Ray direction */
} RAY;

typedef struct {
    Point3    Point;    /* Intersection point */
    REAL    Distance;    /* Distance from ray origin to intersection point */
    REAL    u, v;    /* Parametric coordinates of the intersection point */
} HIT;




/*
** External declarations
*/
extern boolean ray_hit_quad (RAY *, QUAD *, HIT *);

#endif

/* ----- */
```

# Index of

## /pubs/tog/GraphicsGems/gemsv/ch7-1/

Name	Last modified	Size	Description
 <a href="#">_Parent Directory</a>			
 <a href="#">_wave.c</a>	29-Jun-00 08:24	3K	
 <a href="#">_wave.h</a>	29-Jun-00 08:24	1K	

```
/* ----- */
WAVE.C :

This package provides 3 routines for generating rectangular-like,
triangular-like and sine-like waves including specific features.

by Christophe Schlick (10 September 1993)

"Wave Generators for Procedural Techniques in Computer Graphics"
in Graphics Gems V (edited by A. Paeth), Academic Press
/* ----- */

#include <math.h>
#include "wave.h"

/*
** Macro functions
*/

#define ABS(a)          ((a) < 0 ? -(a) : (a))
#define FLOOR(a)        ((a) < 0 ? (int) ((a)-1.0) : (int) (a))
#define MAX(a,b)        ((a) > (b) ? (a) : (b))
#define MIN(a,b)        ((a) < (b) ? (a) : (b))

/*
** rnd : Random function (adapted from Greg Ward in Graphics Gems II)
*/

#define FRND(a)          rnd (17*(a))
#define ARND(a)          rnd (97*(a))

static double rnd (register long s)
{
    s = s << 13 ^ s;
    return (((s*(s*s*15731+789221)+1376312589) & 0X7FFFFFFF) / 2147483648.0);
}

/*
** Rwave : Rectangular-like monodimensional wave
**
** Input : t = Wave function parameter
**         s = Shape parameter (in [-1,1])
**         f = Frequency variance (in [0,1])
**         a = Amplitude variance (in [0,1])
*/

double Rwave (register double t, double s, double f, double a)
{
    register int    i, j;
    register double b;

    i = j = FLOOR (t); t -= i; j++;

    if (f) {
        a = (FRND (i) - 0.5) * f;
        b = (FRND (j) - 0.5) * f + 1.0;
        t = (t-a) / (b-a);
    }

    if (i & 1) {i++; j--; t = 1.0-t;}
    t = (s < 0.0) ? (t+s*t) / (1.0+s*t) : (s > 0.0) ? t / (1.0-s+t) : t;
}
```

```
t = t < 0.5 ? 0.0 : 1.0;

if (a) {
    a = ARND (i) * a * 0.5;
    b = ARND (j) * a * 0.5;
    t = a + t * (1.0-a-b);
}
return (t);
}

/*
** Twave : Triangular-like monodimensional wave
**
** Input : t = Wave function parameter
**          s = Shape parameter (in [-1,1])
**          f = Frequency variance (in [0,1])
**          a = Amplitude variance (in [0,1])
**
*/

double Twave (register double t, double s, double f, double a) {
    register int    i, j;
    register double b;

    i = j = FLOOR (t); t -= i; j++;

    if (f) {
        a = (FRND (i) - 0.5) * f;
        b = (FRND (j) - 0.5) * f + 1.0;
        if (t < a) {
            i--; j--; t++; a++;
            b = a; a = (FRND (i) - 0.5) * f;
        } else if (t > b) {
            i++; j++; t--; b--;
            a = b; b = (FRND (j) - 0.5) * f + 1.0;
        }
        t = (t-a) / (b-a);
    }

    if (i & 1) {i++; j--; t = 1.0-t;}
    t = (s < 0.0) ? (t+s*t) / (1.0+s*t) : (s > 0.0) ? t / (1.0-s+s*t) : t;

    if (a) {
        a = ARND(i) * a * 0.5;
        b = ARND(j) * a * 0.5;
        t = a + t * (1.0-a-b);
    }
    return (t);
}

/*
** Swave : sinusoidal-like monodimensional wave
**
** Input : t = Wave function parameter
**          s = Shape parameter (in [-1,1])
**          f = Frequency variance (in [0,1])
**          a = Amplitude variance (in [0,1])
**
*/

double Swave (register double t, double s, double f, double a)
{
    register int    i, j;
```

```
register double b;
```

```
i = j = FLOOR (t); t -= i; j++;
```

```
if (f) {  
    a = (FRND (i) - 0.5) * f;  
    b = (FRND (j) - 0.5) * f + 1.0;  
    if (t < a) {  
        i--; j--; t++; a++;  
        b = a; a = (FRND (i) - 0.5) * f;  
    } else if (t > b) {  
        i++; j++; t--; b--;  
        a = b; b = (FRND (j) - 0.5) * f + 1.0;  
    }  
    t = (t-a) / (b-a);  
}
```

```
if (i & 1) {i++; j--; t = 1.0-t;}  
t = (s < 0.0) ? (t+s*t) / (1.0+s*t) : (s > 0.0) ? t / (1.0-s+s*t) : t;  
t *= t * (3.0-t-t);
```

```
if (a) {  
    a = ARND(i) * a * 0.5;  
    b = ARND(j) * a * 0.5;  
    t = a + t * (1.0-a-b);  
}  
return (t);  
}
```

```
/* ----- */
```

```
/* ----- */
WAVE.H :

This package provides 3 routines for generating rectangular-like,
triangular-like and sine-like waves including specific features.

by Christophe Schlick (10 September 1993)

"Wave Generators for Procedural Techniques in Computer Graphics"
in Graphics Gems V (edited by A. Paeth), Academic Press
/* ----- */

#ifndef _WAVE_
#define _WAVE_

extern double Rwave (register double t, double s, double Fvar, double Avar);
extern double Twave (register double t, double s, double Fvar, double Avar);
extern double Swave (register double t, double s, double Fvar, double Avar);

#endif

/* ----- */
```

```
/*
Solving the Nearest Point-on-Curve Problem
and
A Bezier Curve-Based Root-Finder
by Philip J. Schneider
from "Graphics Gems", Academic Press, 1990
*/

/*      point_on_curve.c      */

#include <stdio.h>
#include <malloc.h>
#include <math.h>
#include "GraphicsGems.h"

#define TESTMODE

/*
 * Forward declarations
 */
Point2 NearestPointOnCurve();
static int FindRoots();
static Point2 *ConvertToBezierForm();
static double ComputeXIntercept();
static int ControlPolygonFlatEnough();
static int CrossingCount();
static Point2 Bezier();
static Vector2 V2ScaleII();

int MAXDEPTH = 64; /* Maximum depth for recursion */

#define EPSILON (ldexp(1.0,-MAXDEPTH-1)) /*Flatness control value */
#define DEGREE 3 /* Cubic Bezier curve */
#define W_DEGREE 5 /* Degree of eqn to find roots of */

#ifdef TESTMODE
/*
 * main :
 * Given a cubic Bezier curve (i.e., its control points), and some
 * arbitrary point in the plane, find the point on the curve
 * closest to that arbitrary point.
 */
main()
{

    static Point2 bezCurve[4] = { /* A cubic Bezier curve */
        { 0.0, 0.0 },
        { 1.0, 2.0 },
        { 3.0, 3.0 },
        { 4.0, 2.0 },
    };
    static Point2 arbPoint = { 3.5, 2.0 }; /*Some arbitrary point*/
    Point2 pointOnCurve; /* Nearest point on the curve */

    /* Find the closest point */
    pointOnCurve = NearestPointOnCurve(arbPoint, bezCurve);
    printf("pointOnCurve : (%4.4f, %4.4f)\n", pointOnCurve.x,
        pointOnCurve.y);
}
#endif /* TESTMODE */
```



```
/*
 * NearestPointOnCurve :
 *     Compute the parameter value of the point on a Bezier
 *     curve segment closest to some arbitrary, user-input point.
 *     Return the point on the curve at that parameter value.
 */
Point2 NearestPointOnCurve(P, V)
    Point2    P;                                /* The user-supplied point */
    Point2    *V;                                /* Control points of cubic Bezier */
{
    Point2    *w;                                /* Ctl pts for 5th-degree eqn */
    double    t_candidate[W_DEGREE];           /* Possible roots */
    int       n_solutions;                       /* Number of roots found */
    double    t;                                /* Parameter value of closest pt*/

    /* Convert problem to 5th-degree Bezier form */
    w = ConvertToBezierForm(P, V);

    /* Find all possible roots of 5th-degree equation */
    n_solutions = FindRoots(w, W_DEGREE, t_candidate, 0);
    free((char *)w);

    /* Compare distances of P to all candidates, and to t=0, and t=1 */
    {
        double  dist, new_dist;
        Point2  p;
        Vector2  v;
        int      i;

        /* Check distance to beginning of curve, where t = 0 */
        dist = V2SquaredLength(V2Sub(&P, &V[0], &v));
        t = 0.0;

        /* Find distances for candidate points */
        for (i = 0; i < n_solutions; i++) {
            p = Bezier(V, DEGREE, t_candidate[i],
                       (Point2 *)NULL, (Point2 *)NULL);
            new_dist = V2SquaredLength(V2Sub(&P, &p, &v));
            if (new_dist < dist) {
                dist = new_dist;
                t = t_candidate[i];
            }
        }

        /* Finally, look at distance to end point, where t = 1.0 */
        new_dist = V2SquaredLength(V2Sub(&P, &V[DEGREE], &v));
        if (new_dist < dist) {
            dist = new_dist;
            t = 1.0;
        }
    }

    /* Return the point on the curve at parameter value t */
    printf("t : %4.12f\n", t);
    return (Bezier(V, DEGREE, t, (Point2 *)NULL, (Point2 *)NULL));
}
```

```

/*
 * ConvertToBezierForm :
 *      Given a point and a Bezier curve, generate a 5th-degree
 *      Bezier-format equation whose solution finds the point on the
 *      curve nearest the user-defined point.
 */
static Point2 *ConvertToBezierForm(P, V)
    Point2      P;                      /* The point to find t for      */
    Point2      *V;                     /* The control points         */
{
    int         i, j, k, m, n, ub, lb;
    int         row, column;            /* Table indices              */
    Vector2      c[DEGREE+1];           /* V(i)'s - P                 */
    Vector2      d[DEGREE];              /* V(i+1) - V(i)              */
    Point2      *w;                     /* Ctl pts of 5th-degree curve */
    double       cdTable[3][4];          /* Dot product of c, d         */
    static double z[3][4] = {           /* Precomputed "z" for cubics  */
        {1.0, 0.6, 0.3, 0.1},
        {0.4, 0.6, 0.6, 0.4},
        {0.1, 0.3, 0.6, 1.0},
    };

    /* Determine the c's -- these are vectors created by subtracting */
    /* point P from each of the control points                        */
    for (i = 0; i <= DEGREE; i++) {
        V2Sub(&V[i], &P, &c[i]);
    }
    /* Determine the d's -- these are vectors created by subtracting */
    /* each control point from the next                               */
    for (i = 0; i <= DEGREE - 1; i++) {
        d[i] = V2ScaleII(V2Sub(&V[i+1], &V[i], &d[i]), 3.0);
    }

    /* Create the c,d table -- this is a table of dot products of the */
    /* c's and d's                                                       */
    for (row = 0; row <= DEGREE - 1; row++) {
        for (column = 0; column <= DEGREE; column++) {
            cdTable[row][column] = V2Dot(&d[row], &c[column]);
        }
    }

    /* Now, apply the z's to the dot products, on the skew diagonal */
    /* Also, set up the x-values, making these "points"              */
    w = (Point2 *)malloc((unsigned)(W_DEGREE+1) * sizeof(Point2));
    for (i = 0; i <= W_DEGREE; i++) {
        w[i].y = 0.0;
        w[i].x = (double)(i) / W_DEGREE;
    }

    n = DEGREE;
    m = DEGREE-1;
    for (k = 0; k <= n + m; k++) {
        lb = MAX(0, k - m);
        ub = MIN(k, n);
        for (i = lb; i <= ub; i++) {
            j = k - i;
            w[i+j].y += cdTable[j][i] * z[j][i];
        }
    }
}

```

```
    return (w);
}

/*
 * FindRoots :
 *     Given a 5th-degree equation in Bernstein-Bezier form, find
 *     all of the roots in the interval [0, 1].  Return the number
 *     of roots found.
 */
static int FindRoots(w, degree, t, depth)
    Point2      *w;                /* The control points          */
    int          degree;           /* The degree of the polynomial */
    double       *t;               /* RETURN candidate t-values   */
    int          depth;            /* The depth of the recursion   */
{
    int          i;
    Point2       Left[W_DEGREE+1], /* New left and right          */
             Right[W_DEGREE+1];    /* control polygons            */
    int          left_count,        /* Solution count from         */
             right_count;          /* children                    */
    double       left_t[W_DEGREE+1], /* Solutions from kids         */
             right_t[W_DEGREE+1];

    switch (CrossingCount(w, degree)) {
        case 0 : { /* No solutions here */
            return 0;
        }
        case 1 : { /* Unique solution */
            /* Stop recursion when the tree is deep enough */
            /* if deep enough, return 1 solution at midpoint */
            if (depth >= MAXDEPTH) {
                t[0] = (w[0].x + w[W_DEGREE].x) / 2.0;
                return 1;
            }
            if (ControlPolygonFlatEnough(w, degree)) {
                t[0] = ComputeXIntercept(w, degree);
                return 1;
            }
            break;
        }
    }

    /* Otherwise, solve recursively after subdividing control polygon */
    Bezier(w, degree, 0.5, Left, Right);
    left_count = FindRoots(Left, degree, left_t, depth+1);
    right_count = FindRoots(Right, degree, right_t, depth+1);

    /* Gather solutions together */
    for (i = 0; i < left_count; i++) {
        t[i] = left_t[i];
    }
    for (i = 0; i < right_count; i++) {
        t[i+left_count] = right_t[i];
    }

    /* Send back total number of solutions */
    return (left_count+right_count);
}
```

```
/*
 * CrossingCount :
 *     Count the number of times a Bezier control polygon
 *     crosses the 0-axis. This number is >= the number of roots.
 */
static int CrossingCount(V, degree)
    Point2      *V;                /* Control pts of Bezier curve */
    int          degree;            /* Degree of Bezier curve */
{
    int          i;
    int          n_crossings = 0;    /* Number of zero-crossings */
    int          sign, old_sign;     /* Sign of coefficients */

    sign = old_sign = SGN(V[0].y);
    for (i = 1; i <= degree; i++) {
        sign = SGN(V[i].y);
        if (sign != old_sign) n_crossings++;
        old_sign = sign;
    }
    return n_crossings;
}

/*
 * ControlPolygonFlatEnough :
 *     Check if the control polygon of a Bezier curve is flat enough
 *     for recursive subdivision to bottom out.
 */
static int ControlPolygonFlatEnough(V, degree)
    Point2      *V;                /* Control points */
    int          degree;            /* Degree of polynomial */
{
    int          i;                /* Index variable */
    double       *distance;         /* Distances from pts to line */
    double       max_distance_above; /* maximum of these */
    double       max_distance_below;
    double       error;             /* Precision of root */
    double       intercept_1,
                intercept_2,
                left_intercept,
                right_intercept;
    double       a, b, c;           /* Coefficients of implicit */
                                    /* eqn for line from V[0]-V[deg] */

    /* Find the perpendicular distance
     * from each interior control point to
     * line connecting V[0] and V[degree]
     */
    distance = (double *)malloc((unsigned)(degree + 1) *
sizeof(double));
    {
        double   abSquared;

        /* Derive the implicit equation for line connecting first
         * and last control points */
        a = V[0].y - V[degree].y;
        b = V[degree].x - V[0].x;
    }
}
```

```
c = V[0].x * V[degree].y - V[degree].x * V[0].y;

abSquared = (a * a) + (b * b);

for (i = 1; i < degree; i++) {
    /* Compute distance from each of the points to that line */
    distance[i] = a * V[i].x + b * V[i].y + c;
    if (distance[i] > 0.0) {
        distance[i] = (distance[i] * distance[i]) / abSquared;
    }
    if (distance[i] < 0.0) {
        distance[i] = -((distance[i] * distance[i]) /
abSquared);
    }
}

/* Find the largest distance */
max_distance_above = 0.0;
max_distance_below = 0.0;
for (i = 1; i < degree; i++) {
    if (distance[i] < 0.0) {
        max_distance_below = MIN(max_distance_below, distance[i]);
    };
    if (distance[i] > 0.0) {
        max_distance_above = MAX(max_distance_above, distance[i]);
    }
}
free((char *)distance);

{
    double det, dInv;
    double a1, b1, c1, a2, b2, c2;

    /* Implicit equation for zero line */
    a1 = 0.0;
    b1 = 1.0;
    c1 = 0.0;

    /* Implicit equation for "above" line */
    a2 = a;
    b2 = b;
    c2 = c + max_distance_above;

    det = a1 * b2 - a2 * b1;
    dInv = 1.0/det;

    intercept_1 = (b1 * c2 - b2 * c1) * dInv;

    /* Implicit equation for "below" line */
    a2 = a;
    b2 = b;
    c2 = c + max_distance_below;

    det = a1 * b2 - a2 * b1;
    dInv = 1.0/det;

    intercept_2 = (b1 * c2 - b2 * c1) * dInv;
}
```

```
/* Compute intercepts of bounding box */
left_intercept = MIN(intercept_1, intercept_2);
right_intercept = MAX(intercept_1, intercept_2);

error = 0.5 * (right_intercept-left_intercept);
if (error < EPSILON) {
    return 1;
}
else {
    return 0;
}
}

/*
 * ComputeXIntercept :
 *     Compute intersection of chord from first control point to last
 *     with 0-axis.
 */
/* NOTE: "T" and "Y" do not have to be computed, and there are many useless
 * operations in the following (e.g. "0.0 - 0.0").
 */
static double ComputeXIntercept(V, degree)
    Point2      *V;                /* Control points */
    int          degree;           /* Degree of curve */
{
    double      XLK, YLK, XNM, YNM, XMK, YMK;
    double      det, detInv;
    double      S, T;
    double      X, Y;

    XLK = 1.0 - 0.0;
    YLK = 0.0 - 0.0;
    XNM = V[degree].x - V[0].x;
    YNM = V[degree].y - V[0].y;
    XMK = V[0].x - 0.0;
    YMK = V[0].y - 0.0;

    det = XNM*YLK - YNM*XLK;
    detInv = 1.0/det;

    S = (XNM*YMK - YNM*XMK) * detInv;
/* T = (XLK*YMK - YLK*XMK) * detInv; */

    X = 0.0 + XLK * S;
/* Y = 0.0 + YLK * S; */

    return X;
}

/*
 * Bezier :
 *     Evaluate a Bezier curve at a particular parameter value
 *     Fill in control points for resulting sub-curves if "Left" and
 *     "Right" are non-null.
 */
static Point2 Bezier(V, degree, t, Left, Right)
```

```
int      degree;          /* Degree of bezier curve          */
Point2   *V;              /* Control pts                */
double   t;               /* Parameter value            */
Point2   *Left;           /* RETURN left half ctl pts   */
Point2   *Right;          /* RETURN right half ctl pts  */

{
    int      i, j;         /* Index variables           */
    Point2   Vtemp[W_DEGREE+1][W_DEGREE+1];

    /* Copy control points */
    for (j = 0; j <= degree; j++) {
        Vtemp[0][j] = V[j];
    }

    /* Triangle computation */
    for (i = 1; i <= degree; i++) {
        for (j = 0; j <= degree - i; j++) {
            Vtemp[i][j].x =
                (1.0 - t) * Vtemp[i-1][j].x + t * Vtemp[i-1][j+1].x;
            Vtemp[i][j].y =
                (1.0 - t) * Vtemp[i-1][j].y + t * Vtemp[i-1][j+1].y;
        }
    }

    if (Left != NULL) {
        for (j = 0; j <= degree; j++) {
            Left[j] = Vtemp[j][0];
        }
    }
    if (Right != NULL) {
        for (j = 0; j <= degree; j++) {
            Right[j] = Vtemp[degree-j][j];
        }
    }

    return (Vtemp[degree][0]);
}

static Vector2 V2ScaleII(v, s)
    Vector2   *v;
    double    s;
{
    Vector2 result;

    result.x = v->x * s; result.y = v->y * s;
    return (result);
}
```

```
/*
An Algorithm for Automatically Fitting Digitized Curves
by Philip J. Schneider
from "Graphics Gems", Academic Press, 1990
*/

#define TESTMODE

/*  fit_cubic.c */
/*      Piecewise cubic fitting code      */

#include "GraphicsGems.h"
#include <stdio.h>
#include <malloc.h>
#include <math.h>

typedef Point2 *BezierCurve;

/* Forward declarations */
void      FitCurve();
static void      FitCubic();
static double    *Reparameterize();
static double    NewtonRaphsonRootFind();
static Point2    BezierII();
static double    B0(), B1(), B2(), B3();
static Vector2   ComputeLeftTangent();
static Vector2   ComputeRightTangent();
static Vector2   ComputeCenterTangent();
static double    ComputeMaxError();
static double    *ChordLengthParameterize();
static BezierCurve GenerateBezier();
static Vector2   V2AddII();
static Vector2   V2ScaleIII();
static Vector2   V2SubII();

#define MAXPOINTS      1000          /* The most points you can have */

#ifdef TESTMODE

DrawBezierCurve(n, curve)
int n;
BezierCurve curve;
{
    /* You'll have to write this yourself. */
}

/*
 *  main:
 *      Example of how to use the curve-fitting code.  Given an array
 *      of points and a tolerance (squared error between points and
 *      fitted curve), the algorithm will generate a piecewise
 *      cubic Bezier representation that approximates the points.
 *      When a cubic is generated, the routine "DrawBezierCurve"
 *      is called, which outputs the Bezier curve just created
 *      (arguments are the degree and the control points, respectively).
 *      Users will have to implement this function themselves
 *      ascii output, etc.
 */
main()
{
```



```
static Point2 d[7] = {          /* Digitized points */
    { 0.0, 0.0 },
    { 0.0, 0.5 },
    { 1.1, 1.4 },
    { 2.1, 1.6 },
    { 3.2, 1.1 },
    { 4.0, 0.2 },
    { 4.0, 0.0 },
};
double      error = 4.0;          /* Squared error */
FitCurve(d, 7, error);           /* Fit the Bezier curves */
}
#endif                            /* TESTMODE */

/*
 * FitCurve :
 *     Fit a Bezier curve to a set of digitized points
 */
void FitCurve(d, nPts, error)
    Point2      *d;                /* Array of digitized points */
    int          nPts;             /* Number of digitized points */
    double       error;            /* User-defined error squared */
{
    Vector2      tHat1, tHat2;     /* Unit tangent vectors at endpoints */

    tHat1 = ComputeLeftTangent(d, 0);
    tHat2 = ComputeRightTangent(d, nPts - 1);
    FitCubic(d, 0, nPts - 1, tHat1, tHat2, error);
}

/*
 * FitCubic :
 *     Fit a Bezier curve to a (sub)set of digitized points
 */
static void FitCubic(d, first, last, tHat1, tHat2, error)
    Point2      *d;                /* Array of digitized points */
    int          first, last;       /* Indices of first and last pts in region */
    Vector2      tHat1, tHat2;     /* Unit tangent vectors at endpoints */
    double       error;            /* User-defined error squared */
{
    BezierCurve bezCurve; /*Control points of fitted Bezier curve*/
    double       *u;              /* Parameter values for point */
    double       *uPrime;         /* Improved parameter values */
    double       maxError;        /* Maximum fitting error */
    int          splitPoint;       /* Point to split point set at */
    int          nPts;            /* Number of points in subset */
    double       iterationError; /*Error below which you try iterating */
    int          maxIterations = 4; /* Max times to try iterating */
    Vector2      tHatCenter;       /* Unit tangent vector at splitPoint */
    int          i;

    iterationError = error * error;
    nPts = last - first + 1;

    /* Use heuristic if region only has two points in it */
    if (nPts == 2) {
        double dist = V2DistanceBetween2Points(&d[last], &d[first]) / 3.0;

        bezCurve = (Point2 *)malloc(4 * sizeof(Point2));
    }
}
```

```
        bezCurve[0] = d[first];
        bezCurve[3] = d[last];
        V2Add(&bezCurve[0], V2Scale(&tHat1, dist), &bezCurve[1]);
        V2Add(&bezCurve[3], V2Scale(&tHat2, dist), &bezCurve[2]);
        DrawBezierCurve(3, bezCurve);
        free((void *)bezCurve);
        return;
    }

    /* Parameterize points, and attempt to fit curve */
    u = ChordLengthParameterize(d, first, last);
    bezCurve = GenerateBezier(d, first, last, u, tHat1, tHat2);

    /* Find max deviation of points to fitted curve */
    maxError = ComputeMaxError(d, first, last, bezCurve, u, &splitPoint);
    if (maxError < error) {
        DrawBezierCurve(3, bezCurve);
        free((void *)u);
        free((void *)bezCurve);
        return;
    }

    /* If error not too large, try some reparameterization */
    /* and iteration */
    if (maxError < iterationError) {
        for (i = 0; i < maxIterations; i++) {
            uPrime = Reparameterize(d, first, last, u, bezCurve);
            bezCurve = GenerateBezier(d, first, last, uPrime, tHat1, tHat2);
            maxError = ComputeMaxError(d, first, last,
                                      bezCurve, uPrime, &splitPoint);
            if (maxError < error) {
                DrawBezierCurve(3, bezCurve);
                free((void *)u);
                free((void *)bezCurve);
                return;
            }
        }
        free((void *)u);
        u = uPrime;
    }
}

/* Fitting failed -- split at max error point and fit recursively */
free((void *)u);
free((void *)bezCurve);
tHatCenter = ComputeCenterTangent(d, splitPoint);
FitCubic(d, first, splitPoint, tHat1, tHatCenter, error);
V2Negate(&tHatCenter);
FitCubic(d, splitPoint, last, tHatCenter, tHat2, error);
}

/*
 * GenerateBezier :
 * Use least-squares method to find Bezier control points for region.
 */
static BezierCurve GenerateBezier(d, first, last, uPrime, tHat1, tHat2)
    Point2      *d;                /* Array of digitized points */
    int         first, last;       /* Indices defining region */
    double      *uPrime;           /* Parameter values for region */

```

```
Vector2      tHat1, tHat2;    /* Unit tangents at endpoints */
{
    int        i;
    Vector2    A[MAXPOINTS][2]; /* Precomputed rhs for eqn      */
    int        nPts;           /* Number of pts in sub-curve */
    double     C[2][2];        /* Matrix C                    */
    double     X[2];           /* Matrix X                    */
    double     det_C0_C1,      /* Determinants of matrices    */
              det_C0_X,
              det_X_C1;

    double     alpha_l,        /* Alpha values, left and right */
              alpha_r;

    Vector2    tmp;            /* Utility variable            */
    BezierCurve bezCurve;      /* RETURN bezier curve ctl pts */

    bezCurve = (Point2 *)malloc(4 * sizeof(Point2));
    nPts = last - first + 1;

    /* Compute the A's */
    for (i = 0; i < nPts; i++) {
        Vector2    v1, v2;
        v1 = tHat1;
        v2 = tHat2;
        V2Scale(&v1, B1(uPrime[i]));
        V2Scale(&v2, B2(uPrime[i]));
        A[i][0] = v1;
        A[i][1] = v2;
    }

    /* Create the C and X matrices */
    C[0][0] = 0.0;
    C[0][1] = 0.0;
    C[1][0] = 0.0;
    C[1][1] = 0.0;
    X[0] = 0.0;
    X[1] = 0.0;

    for (i = 0; i < nPts; i++) {
        C[0][0] += V2Dot(&A[i][0], &A[i][0]);
        C[0][1] += V2Dot(&A[i][0], &A[i][1]);
        C[1][0] += V2Dot(&A[i][1], &A[i][0]);
        C[1][1] += V2Dot(&A[i][1], &A[i][1]);

        tmp = V2SubII(d[first + i],
            V2AddII(
                V2ScaleIII(d[first], B0(uPrime[i])),
                V2AddII(
                    V2ScaleIII(d[first], B1(uPrime[i])),
                    V2AddII(
                        V2ScaleIII(d[last], B2(uPrime[i])),
                        V2ScaleIII(d[last], B3(uPrime[i])))))));

        X[0] += V2Dot(&A[i][0], &tmp);
        X[1] += V2Dot(&A[i][1], &tmp);
    }

    /* Compute the determinants of C and X */
    det_C0_C1 = C[0][0] * C[1][1] - C[1][0] * C[0][1];
```

```
det_C0_X  = C[0][0] * X[1]      - C[0][1] * X[0];
det_X_C1  = X[0]      * C[1][1] - X[1]      * C[0][1];

/* Finally, derive alpha values      */
if (det_C0_C1 == 0.0) {
    det_C0_C1 = (C[0][0] * C[1][1]) * 10e-12;
}
alpha_l = det_X_C1 / det_C0_C1;
alpha_r = det_C0_X / det_C0_C1;

/* If alpha negative, use the Wu/Barasky heuristic (see text) */
/* (if alpha is 0, you get coincident control points that lead to
   * divide by zero in any subsequent NewtonRaphsonRootFind() call. */
if (alpha_l < 1.0e-6 || alpha_r < 1.0e-6) {
    double dist = V2DistanceBetween2Points(&d[last], &d[first]) /
        3.0;

    bezCurve[0] = d[first];
    bezCurve[3] = d[last];
    V2Add(&bezCurve[0], V2Scale(&tHat1, dist), &bezCurve[1]);
    V2Add(&bezCurve[3], V2Scale(&tHat2, dist), &bezCurve[2]);
    return (bezCurve);
}

/* First and last control points of the Bezier curve are */
/* positioned exactly at the first and last data points */
/* Control points 1 and 2 are positioned an alpha distance out */
/* on the tangent vectors, left and right, respectively */
bezCurve[0] = d[first];
bezCurve[3] = d[last];
V2Add(&bezCurve[0], V2Scale(&tHat1, alpha_l), &bezCurve[1]);
V2Add(&bezCurve[3], V2Scale(&tHat2, alpha_r), &bezCurve[2]);
return (bezCurve);
}
```

```
/*
 * Reparameterize:
 *   Given set of points and their parameterization, try to find
 *   a better parameterization.
 */
static double *Reparameterize(d, first, last, u, bezCurve)
    Point2      *d;                /* Array of digitized points */
    int          first, last;       /* Indices defining region */
    double       *u;               /* Current parameter values */
    BezierCurve bezCurve;          /* Current fitted curve */
{
    int          nPts = last-first+1;
    int          i;
    double       *uPrime;          /* New parameter values */

    uPrime = (double *)malloc(nPts * sizeof(double));
    for (i = first; i <= last; i++) {
        uPrime[i-first] = NewtonRaphsonRootFind(bezCurve, d[i], u[i-
            first]);
    }
    return (uPrime);
}
```

```
/*
 * NewtonRaphsonRootFind :
 *     Use Newton-Raphson iteration to find better root.
 */
static double NewtonRaphsonRootFind(Q, P, u)
    BezierCurve Q;                /* Current fitted curve */
    Point2      P;                /* Digitized point */
    double      u;                /* Parameter value for "P" */
{
    double      numerator, denominator;
    Point2      Q1[3], Q2[2];    /* Q' and Q'' */
    Point2      Q_u, Q1_u, Q2_u; /* u evaluated at Q, Q', & Q'' */
    double      uPrime;          /* Improved u */
    int         i;

    /* Compute Q(u) */
    Q_u = BezierII(3, Q, u);

    /* Generate control vertices for Q' */
    for (i = 0; i <= 2; i++) {
        Q1[i].x = (Q[i+1].x - Q[i].x) * 3.0;
        Q1[i].y = (Q[i+1].y - Q[i].y) * 3.0;
    }

    /* Generate control vertices for Q'' */
    for (i = 0; i <= 1; i++) {
        Q2[i].x = (Q1[i+1].x - Q1[i].x) * 2.0;
        Q2[i].y = (Q1[i+1].y - Q1[i].y) * 2.0;
    }

    /* Compute Q'(u) and Q''(u) */
    Q1_u = BezierII(2, Q1, u);
    Q2_u = BezierII(1, Q2, u);

    /* Compute f(u)/f'(u) */
    numerator = (Q_u.x - P.x) * (Q1_u.x) + (Q_u.y - P.y) * (Q1_u.y);
    denominator = (Q1_u.x) * (Q1_u.x) + (Q1_u.y) * (Q1_u.y) +
        (Q_u.x - P.x) * (Q2_u.x) + (Q_u.y - P.y) * (Q2_u.y);

    /* u = u - f(u)/f'(u) */
    uPrime = u - (numerator/denominator);
    return (uPrime);
}

/*
 * Bezier :
 *     Evaluate a Bezier curve at a particular parameter value
 */
static Point2 BezierII(degree, V, t)
    int         degree;          /* The degree of the bezier curve */
    Point2      *V;              /* Array of control points */
    double      t;               /* Parametric value to find point for */
{
    int         i, j;
    Point2      Q;               /* Point on curve at parameter t */
    Point2      *Vtemp;          /* Local copy of control points */

```

```
/* Copy array */
Vtemp = (Point2 *)malloc((unsigned)((degree+1)
                               * sizeof (Point2)));
for (i = 0; i <= degree; i++) {
    Vtemp[i] = V[i];
}

/* Triangle computation */
for (i = 1; i <= degree; i++) {
    for (j = 0; j <= degree-i; j++) {
        Vtemp[j].x = (1.0 - t) * Vtemp[j].x + t * Vtemp[j+1].x;
        Vtemp[j].y = (1.0 - t) * Vtemp[j].y + t * Vtemp[j+1].y;
    }
}

Q = Vtemp[0];
free((void *)Vtemp);
return Q;
}

/*
 * B0, B1, B2, B3 :
 * Bezier multipliers
 */
static double B0(u)
double u;
{
    double tmp = 1.0 - u;
    return (tmp * tmp * tmp);
}

static double B1(u)
double u;
{
    double tmp = 1.0 - u;
    return (3 * u * (tmp * tmp));
}

static double B2(u)
double u;
{
    double tmp = 1.0 - u;
    return (3 * u * u * tmp);
}

static double B3(u)
double u;
{
    return (u * u * u);
}

/*
 * ComputeLeftTangent, ComputeRightTangent, ComputeCenterTangent :
 * Approximate unit tangents at endpoints and "center" of digitized curve
 */
static Vector2 ComputeLeftTangent(d, end)
```

```
    Point2      *d;                      /* Digitized points*/
    int          end;                    /* Index to "left" end of region */
{
    Vector2      tHat1;
    tHat1 = V2SubII(d[end+1], d[end]);
    tHat1 = *V2Normalize(&tHat1);
    return tHat1;
}

static Vector2 ComputeRightTangent(d, end)
    Point2      *d;                      /* Digitized points          */
    int          end;                    /* Index to "right" end of region */
{
    Vector2      tHat2;
    tHat2 = V2SubII(d[end-1], d[end]);
    tHat2 = *V2Normalize(&tHat2);
    return tHat2;
}

static Vector2 ComputeCenterTangent(d, center)
    Point2      *d;                      /* Digitized points          */
    int          center;                  /* Index to point inside region */
{
    Vector2      V1, V2, tHatCenter;

    V1 = V2SubII(d[center-1], d[center]);
    V2 = V2SubII(d[center], d[center+1]);
    tHatCenter.x = (V1.x + V2.x)/2.0;
    tHatCenter.y = (V1.y + V2.y)/2.0;
    tHatCenter = *V2Normalize(&tHatCenter);
    return tHatCenter;
}

/*
 * ChordLengthParameterize :
 *     Assign parameter values to digitized points
 *     using relative distances between points.
 */
static double *ChordLengthParameterize(d, first, last)
    Point2      *d;                      /* Array of digitized points */
    int          first, last;              /* Indices defining region    */
{
    int          i;
    double       *u;                      /* Parameterization          */

    u = (double *)malloc((unsigned)(last-first+1) * sizeof(double));

    u[0] = 0.0;
    for (i = first+1; i <= last; i++) {
        u[i-first] = u[i-first-1] +
            V2DistanceBetween2Points(&d[i], &d[i-1]);
    }

    for (i = first + 1; i <= last; i++) {
        u[i-first] = u[i-first] / u[last-first];
    }

    return(u);
}
```

```
/*
 * ComputeMaxError :
 *     Find the maximum squared distance of digitized points
 *     to fitted curve.
 */
static double ComputeMaxError(d, first, last, bezCurve, u, splitPoint)
    Point2      *d;                /* Array of digitized points */
    int          first, last;       /* Indices defining region */
    BezierCurve bezCurve;          /* Fitted Bezier curve */
    double       *u;                /* Parameterization of points */
    int          *splitPoint;       /* Point of maximum error */
{
    int          i;
    double       maxDist;           /* Maximum error */
    double       dist;              /* Current error */
    Point2       P;                 /* Point on curve */
    Vector2      v;                 /* Vector from point to curve */

    *splitPoint = (last - first + 1)/2;
    maxDist = 0.0;
    for (i = first + 1; i < last; i++) {
        P = BezierII(3, bezCurve, u[i-first]);
        v = V2SubII(P, d[i]);
        dist = V2SquaredLength(&v);
        if (dist >= maxDist) {
            maxDist = dist;
            *splitPoint = i;
        }
    }
    return (maxDist);
}

static Vector2 V2AddII(a, b)
    Vector2 a, b;
{
    Vector2 c;
    c.x = a.x + b.x; c.y = a.y + b.y;
    return (c);
}

static Vector2 V2ScaleIII(v, s)
    Vector2 v;
    double s;
{
    Vector2 result;
    result.x = v.x * s; result.y = v.y * s;
    return (result);
}

static Vector2 V2SubII(a, b)
    Vector2 a, b;
{
    Vector2 c;
    c.x = a.x - b.x; c.y = a.y - b.y;
    return (c);
}
```



```
/*
 *
 *          Filtered Image Rescaling
 *
 *          by Dale Schumacher
 */

#include <stdio.h>
#include <string.h>
#include <malloc.h>
#include <math.h>
#include "GraphicsGems.h"

static char    _Program[] = "fzoom";
static char    _Version[] = "0.20";
static char    _Copyright[] = "Public Domain 1991 by Dale Schumacher";

#ifndef EXIT_SUCCESS
#define EXIT_SUCCESS    (0)
#define EXIT_FAILURE    (1)
#endif

typedef unsigned char    Pixel;

typedef struct {
    int        xsize;        /* horizontal size of the image in Pixels */
    int        ysize;        /* vertical size of the image in Pixels */
    Pixel * data;            /* pointer to first scanline of image */
    int        span;        /* byte offset between two scanlines */
} Image;

#define WHITE_PIXEL    (255)
#define BLACK_PIXEL    (0)

/*
 *
 *      generic image access and i/o support routines
 */

static char *
next_token(f)
FILE *f;
{
    static char delim[] = " \t\r\n";
    static char *t = NULL;
    static char lnbuf[256];
    char *p;

    while(t == NULL) {
        /* nothing in the buffer */
        if(fgets(lnbuf, sizeof(lnbuf), f)) {    /* read a line */
            if(p = strchr(lnbuf, '#')) {    /* clip any comment */
                *p = '\0';
            }
            t = strtok(lnbuf, delim);    /* get first token */
        } else {
            return(NULL);
        }
    }
    p = t;
    t = strtok((char *)NULL, delim);    /* get next token */
    return(p);
}
```

```
Image *
new_image(xsize, ysize) /* create a blank image */
int xsize, ysize;
{
    Image *image;

    if((image = (Image *)malloc(sizeof(Image)))
    && (image->data = (Pixel *)calloc(ysize, xsize))) {
        image->xsize = xsize;
        image->ysize = ysize;
        image->span = xsize;
    }
    return(image);
}

void
free_image(image)
Image *image;
{
    free(image->data);
    free(image);
}

Image *
load_image(f)          /* read image from file */
FILE *f;
{
    char *p;
    int width, height;
    Image *image;

    if(((p = next_token(f)) && (strcmp(p, "Bm") == 0))
    && ((p = next_token(f)) && ((width = atoi(p)) > 0))
    && ((p = next_token(f)) && ((height = atoi(p)) > 0))
    && ((p = next_token(f)) && (strcmp(p, "8") == 0))
    && (image = new_image(width, height))
    && (fread(image->data, width, height, f) == height)) {
        return(image);          /* load successful */
    } else {
        return(NULL);          /* load failed */
    }
}

int
save_image(f, image)    /* write image to file */
FILE *f;
Image *image;
{
    fprintf(f, "Bm # PXM 8-bit greyscale image\n");
    fprintf(f, "%d %d 8 # width height depth\n",
        image->xsize, image->ysize);
    if(fwrite(image->data, image->xsize, image->ysize, f) == image->ysize) {
        return(0);          /* save successful */
    } else {
        return(-1);          /* save failed */
    }
}

Pixel
get_pixel(image, x, y)
Image *image;
```

```
int x, y;
{
    static Image *im = NULL;
    static int yy = -1;
    static Pixel *p = NULL;

    if((x < 0) || (x >= image->xsize) || (y < 0) || (y >= image->ysize)) {
        return(0);
    }
    if((im != image) || (yy != y)) {
        im = image;
        yy = y;
        p = image->data + (y * image->span);
    }
    return(p[x]);
}

void
get_row(row, image, y)
Pixel *row;
Image *image;
int y;
{
    if((y < 0) || (y >= image->ysize)) {
        return;
    }
    memcpy(row,
           image->data + (y * image->span),
           (sizeof(Pixel) * image->xsize));
}

void
get_column(column, image, x)
Pixel *column;
Image *image;
int x;
{
    int i, d;
    Pixel *p;

    if((x < 0) || (x >= image->xsize)) {
        return;
    }
    d = image->span;
    for(i = image->ysize, p = image->data + x; i-- > 0; p += d) {
        *column++ = *p;
    }
}

Pixel
put_pixel(image, x, y, data)
Image *image;
int x, y;
Pixel data;
{
    static Image *im = NULL;
    static int yy = -1;
    static Pixel *p = NULL;

    if((x < 0) || (x >= image->xsize) || (y < 0) || (y >= image->ysize)) {
        return(0);
    }
}
```

```
    }
    if((im != image) || (yy != y)) {
        im = image;
        yy = y;
        p = image->data + (y * image->span);
    }
    return(p[x] = data);
}

/*
 *   filter function definitions
 */

#define filter_support          (1.0)

double
filter(t)
double t;
{
    /* f(t) = 2|t|^3 - 3|t|^2 + 1, -1 <= t <= 1 */
    if(t < 0.0) t = -t;
    if(t < 1.0) return((2.0 * t - 3.0) * t * t + 1.0);
    return(0.0);
}

#define box_support             (0.5)

double
box_filter(t)
double t;
{
    if((t > -0.5) && (t <= 0.5)) return(1.0);
    return(0.0);
}

#define triangle_support        (1.0)

double
triangle_filter(t)
double t;
{
    if(t < 0.0) t = -t;
    if(t < 1.0) return(1.0 - t);
    return(0.0);
}

#define bell_support            (1.5)

double
bell_filter(t)          /* box (*) box (*) box */
double t;
{
    if(t < 0) t = -t;
    if(t < .5) return(.75 - (t * t));
    if(t < 1.5) {
        t = (t - 1.5);
        return(.5 * (t * t));
    }
    return(0.0);
}
```

```
#define B_spline_support          (2.0)

double
B_spline_filter(t)          /* box (*) box (*) box (*) box */
double t;
{
    double tt;

    if(t < 0) t = -t;
    if(t < 1) {
        tt = t * t;
        return((.5 * tt * t) - tt + (2.0 / 3.0));
    } else if(t < 2) {
        t = 2 - t;
        return((1.0 / 6.0) * (t * t * t));
    }
    return(0.0);
}

double
sinc(x)
double x;
{
    x *= M_PI;
    if(x != 0) return(sin(x) / x);
    return(1.0);
}

#define Lanczos3_support          (3.0)

double
Lanczos3_filter(t)
double t;
{
    if(t < 0) t = -t;
    if(t < 3.0) return(sinc(t) * sinc(t/3.0));
    return(0.0);
}

#define Mitchell_support          (2.0)

#define B          (1.0 / 3.0)
#define C          (1.0 / 3.0)

double
Mitchell_filter(t)
double t;
{
    double tt;

    tt = t * t;
    if(t < 0) t = -t;
    if(t < 1.0) {
        t = ((12.0 - 9.0 * B - 6.0 * C) * (t * tt))
            + ((-18.0 + 12.0 * B + 6.0 * C) * tt)
            + (6.0 - 2 * B));
        return(t / 6.0);
    } else if(t < 2.0) {
        t = (((-1.0 * B - 6.0 * C) * (t * tt))
            + ((6.0 * B + 30.0 * C) * tt))
    }
}
```

```
        + ((-12.0 * B - 48.0 * C) * t)
        + (8.0 * B + 24 * C));
    return(t / 6.0);
}
return(0.0);
}

/*
 *   image rescaling routine
 */

typedef struct {
    int      pixel;
    double   weight;
} CONTRIB;

typedef struct {
    int      n;                /* number of contributors */
    CONTRIB *p;               /* pointer to list of contributions */
} CLIST;

CLIST      *contrib;          /* array of contribution lists */

void
zoom(dst, src, filterf, fwidth)
Image *dst;                  /* destination image structure */
Image *src;                  /* source image structure */
double (*filterf)();         /* filter function */
double fwidth;               /* filter width (support) */
{
    Image *tmp;               /* intermediate image */
    double xscale, yscale;    /* zoom scale factors */
    int i, j, k;              /* loop variables */
    int n;                    /* pixel number */
    double center, left, right; /* filter calculation variables */
    double width, fscale, weight; /* filter calculation variables */
    Pixel *raster;            /* a row or column of pixels */

    /* create intermediate image to hold horizontal zoom */
    tmp = new_image(dst->xsize, src->ysize);
    xscale = (double) dst->xsize / (double) src->xsize;
    yscale = (double) dst->ysize / (double) src->ysize;

    /* pre-calculate filter contributions for a row */
    contrib = (CLIST *)calloc(dst->xsize, sizeof(CLIST));
    if(xscale < 1.0) {
        width = fwidth / xscale;
        fscale = 1.0 / xscale;
        for(i = 0; i < dst->xsize; ++i) {
            contrib[i].n = 0;
            contrib[i].p = (CONTRIB *)calloc((int) (width * 2 + 1),
                                                sizeof(CONTRIB));
            center = (double) i / xscale;
            left = ceil(center - width);
            right = floor(center + width);
            for(j = left; j <= right; ++j) {
                weight = center - (double) j;
                weight = (*filterf)(weight / fscale) / fscale;
                if(j < 0) {
                    n = -j;
                } else if(j >= src->xsize) {

```

```

        n = (src->xsize - j) + src->xsize - 1;
    } else {
        n = j;
    }
    k = contrib[i].n++;
    contrib[i].p[k].pixel = n;
    contrib[i].p[k].weight = weight;
}

} else {
    for(i = 0; i < dst->xsize; ++i) {
        contrib[i].n = 0;
        contrib[i].p = (CONTRIB *)calloc((int) (fwidth * 2 + 1),
                                           sizeof(CONTRIB));
        center = (double) i / xscale;
        left = ceil(center - fwidth);
        right = floor(center + fwidth);
        for(j = left; j <= right; ++j) {
            weight = center - (double) j;
            weight = (*filterf)(weight);
            if(j < 0) {
                n = -j;
            } else if(j >= src->xsize) {
                n = (src->xsize - j) + src->xsize - 1;
            } else {
                n = j;
            }
            k = contrib[i].n++;
            contrib[i].p[k].pixel = n;
            contrib[i].p[k].weight = weight;
        }
    }
}

/* apply filter to zoom horizontally from src to tmp */
raster = (Pixel *)calloc(src->xsize, sizeof(Pixel));
for(k = 0; k < tmp->ysize; ++k) {
    get_row(raster, src, k);
    for(i = 0; i < tmp->xsize; ++i) {
        weight = 0.0;
        for(j = 0; j < contrib[i].n; ++j) {
            weight += raster[contrib[i].p[j].pixel]
                      * contrib[i].p[j].weight;
        }
        put_pixel(tmp, i, k,
                  (Pixel)CLAMP(weight, BLACK_PIXEL, WHITE_PIXEL));
    }
}
free(raster);

/* free the memory allocated for horizontal filter weights */
for(i = 0; i < tmp->xsize; ++i) {
    free(contrib[i].p);
}
free(contrib);

/* pre-calculate filter contributions for a column */
contrib = (CLIST *)calloc(dst->ysize, sizeof(CLIST));
if(yscale < 1.0) {
    width = fwidth / yscale;
    fscale = 1.0 / yscale;
```

```
        for(i = 0; i < dst->ysize; ++i) {
            contrib[i].n = 0;
            contrib[i].p = (CONTRIB *)calloc((int) (width * 2 + 1),
                                              sizeof(CONTRIB));
            center = (double) i / yscale;
            left = ceil(center - width);
            right = floor(center + width);
            for(j = left; j <= right; ++j) {
                weight = center - (double) j;
                weight = (*filterf)(weight / fscale) / fscale;
                if(j < 0) {
                    n = -j;
                } else if(j >= tmp->ysize) {
                    n = (tmp->ysize - j) + tmp->ysize - 1;
                } else {
                    n = j;
                }
                k = contrib[i].n++;
                contrib[i].p[k].pixel = n;
                contrib[i].p[k].weight = weight;
            }
        }
    } else {
        for(i = 0; i < dst->ysize; ++i) {
            contrib[i].n = 0;
            contrib[i].p = (CONTRIB *)calloc((int) (fwidth * 2 + 1),
                                              sizeof(CONTRIB));
            center = (double) i / yscale;
            left = ceil(center - fwidth);
            right = floor(center + fwidth);
            for(j = left; j <= right; ++j) {
                weight = center - (double) j;
                weight = (*filterf)(weight);
                if(j < 0) {
                    n = -j;
                } else if(j >= tmp->ysize) {
                    n = (tmp->ysize - j) + tmp->ysize - 1;
                } else {
                    n = j;
                }
                k = contrib[i].n++;
                contrib[i].p[k].pixel = n;
                contrib[i].p[k].weight = weight;
            }
        }
    }

    /* apply filter to zoom vertically from tmp to dst */
    raster = (Pixel *)calloc(tmp->ysize, sizeof(Pixel));
    for(k = 0; k < dst->xsize; ++k) {
        get_column(raster, tmp, k);
        for(i = 0; i < dst->ysize; ++i) {
            weight = 0.0;
            for(j = 0; j < contrib[i].n; ++j) {
                weight += raster[contrib[i].p[j].pixel]
                    * contrib[i].p[j].weight;
            }
            put_pixel(dst, k, i,
                    (Pixel)CLAMP(weight, BLACK_PIXEL, WHITE_PIXEL));
        }
    }
}
```



```
    free(raster);

    /* free the memory allocated for vertical filter weights */
    for(i = 0; i < dst->ysize; ++i) {
        free(contrib[i].p);
    }
    free(contrib);

    free_image(tmp);
}

/*
 *   command line interface
 */

void
usage()
{
    fprintf(stderr, "usage: %s [-options] input.bm output.bm\n", _Program);
    fprintf(stderr, "\n
options:\n\
    -x xsize          output x size\n\
    -y ysize          output y size\n\
    -f filter          filter type\n\
{b=box, t=triangle, q=bell, B=B-spline, h=hermite, l=Lanczos3, m=Mitchell}\n\
");
    exit(1);
}

void
banner()
{
    printf("%s v%s -- %s\n", _Program, _Version, _Copyright);
}

main(argc, argv)
int argc;
char *argv[];
{
    register int c;
    extern int optind;
    extern char *optarg;
    int xsize = 0, ysize = 0;
    double (*f)() = filter;
    double s = filter_support;
    char *dstfile, *srcfile;
    Image *dst, *src;
    FILE *fp;

    while((c = getopt(argc, argv, "x:y:f:V")) != EOF) {
        switch(c) {
            case 'x': xsize = atoi(optarg); break;
            case 'y': ysize = atoi(optarg); break;
            case 'f':
                switch(*optarg) {
                    case 'b': f=box_filter; s=box_support; break;
                    case 't': f=triangle_filter; s=triangle_support; break;
                    case 'q': f=bell_filter; s=bell_support; break;
                    case 'B': f=B_spline_filter; s=B_spline_support; break;
                    case 'h': f=filter; s=filter_support; break;
                    case 'l': f=Lanczos3_filter; s=Lanczos3_support; break;
                }
            }
        }
    }
}
```

```
        case 'm': f=Mitchell_filter; s=Mitchell_support; break;
        default: usage();
    }
    break;
case 'V': banner(); exit(EXIT_SUCCESS);
case '?': usage();
default:  usage();
}
}
if((argc - optind) != 2) usage();
srcfile = argv[optind];
dstfile = argv[optind + 1];
if(((fp = fopen(srcfile, "r")) == NULL)
|| ((src = load_image(fp)) == NULL)) {
    fprintf(stderr, "%s: can't load source image '%s'\n",
            _Program, srcfile);
    exit(EXIT_FAILURE);
}
fclose(fp);
if(xsize <= 0) xsize = src->xsize;
if(ysize <= 0) ysize = src->ysize;
dst = new_image(xsize, ysize);
zoom(dst, src, f, s);
if(((fp = fopen(dstfile, "w")) == NULL)
|| (save_image(fp, dst) != 0)) {
    fprintf(stderr, "%s: can't save destination image '%s'\n",
            _Program, dstfile);
    exit(EXIT_FAILURE);
}
fclose(fp);
exit(EXIT_SUCCESS);
}
```

```
/*
 *
 *      Filtered Image Rescaling
 *
 *      by Dale Schumacher
 *
 */

/*
Additional changes by Ray Gardener, Daylon Graphics Ltd.
December 4, 1999

Summary:

- Horizontal filter contributions are calculated on the fly,
  as each column is mapped from src to dst image. This lets
  us omit having to allocate a temporary full horizontal stretch
  of the src image.

- If none of the src pixels within a sampling region differ,
  then the output pixel is forced to equal (any of) the source pixel.
  This ensures that filters do not corrupt areas of constant color.

- Filter weight contribution results, after summing, are
  rounded to the nearest pixel color value instead of
  being casted to Pixel (usually an int or char). Otherwise,
  artifacting occurs.

- All memory allocations checked for failure; zoom() returns
  error code. new_image() returns NULL if unable to allocate
  pixel storage, even if Image struct can be allocated.
  Some assertions added.

- load_image(), save_image() take filenames, not file handles.

- TGA bitmap format available. If you want to add a filetype,
  extend the gImageHandlers array, and factor in your load_image_xxx()
  and save_image_xxx() functions. Search for string 'add your'
  to find appropriate code locations.

- The 'input' and 'output' command-line arguments do not have
  to specify .bm files; any supported filetype is okay.

- Added implementation of getopt() if compiling under Windows.
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <malloc.h>
#include <math.h>
#include "GraphicsGems.h"

static char    _Program[] = "fzoom";
static char    _Version[] = "0.30";
static char    _Copyright[] = "Public Domain 1991 by Dale Schumacher. Mods by Ray
Gardener";

#ifndef EXIT_SUCCESS
#define EXIT_SUCCESS    (0)
#define EXIT_FAILURE    (1)
#endif
```

```
/* M_PI was not in gems header ?? */
#ifndef M_PI
#define M_PI      3.14159265359
#endif

#ifdef WIN32

/* getopt() - Parses command line (argv) looking for switch chars
Returns option flag char if option found.
Returns EOF if end of arglist or non-flag arg reached.
Returns 0 if option arg doesn't match any switch.

rcg: No idea if this fully implements getopt()'s API;
just provided so main() would compile.
*/
int optind;
char *optarg;

int
getopt(argc, argv, pszSwitches)
int argc;
char* argv[];
char* pszSwitches;
{
    static int n = 1;
    char* pszToken;
    char* arg = "- ";
    char* szSwitches;

    if(n >= argc || argv[n][0] != '-')
    {
        optind = n;
        n = 1;
        return EOF;
    }
    szSwitches = _strdup(pszSwitches);
    if(szSwitches == NULL)
        return EOF;

    /* Compare argv[n] with acceptable switches */
    pszToken = strtok(szSwitches, ":");
    if(pszToken == NULL)
    {
        free(szSwitches);
        return EOF;
    }

    while(pszToken)
    {
        if(strlen(argv[n]) > 1 && argv[n][0] == '-')
        {
            arg[1] = pszToken[0];
            if(memcmp(arg, argv[n], 2) == 0)
            {
                optind = n++;
                optarg = argv[n++];
                free(szSwitches);
                return arg[1];
            }
        }
    }
}
```

```
    }
    pszToken = strtok(NULL, ":");
}
n++;
free(szSwitches);
return 0;
}

#endif // WIN32

typedef unsigned char    Pixel;
/* Note: if you define Pixel to something bigger than char,
    you may need to add more support in bitmap file I/O functions.
*/

typedef struct
{
    int        xsize;           /* horizontal size of the image in Pixels */
    int        ysize;           /* vertical size of the image in Pixels */
    Pixel * data;               /* pointer to first scanline of image */
    int        span;            /* byte offset between two scanlines */
} Image;

#define WHITE_PIXEL      (255)
#define BLACK_PIXEL      (0)

/*
 *      generic image access and i/o support routines
 */

static char *
next_token(f)
FILE *f;
{
    static char delim[] = " \t\r\n";
    static char *t = NULL;
    static char lnbuf[256];
    char *p;

    while(t == NULL) {
        /* nothing in the buffer */
        if(fgets(lnbuf, sizeof(lnbuf), f)) { /* read a line */
            if(p = strchr(lnbuf, '#')) { /* clip any comment */
                *p = '\0';
            }
            t = strtok(lnbuf, delim); /* get first token */
        } else {
            return(NULL);
        }
    }
    p = t;
    t = strtok((char *)NULL, delim); /* get next token */
    return(p);
}

Pixel
get_pixel(image, x, y)
Image *image;
int x, y;
{
```

```
static Image *im = NULL;
static int yy = -1;
static Pixel *p = NULL;

if((x < 0) || (x >= image->xsize) || (y < 0) || (y >= image->ysize)) {
    return(0);
}
if((im != image) || (yy != y)) {
    im = image;
    yy = y;
    p = image->data + (y * image->span);
}
return(p[x]);
}
```

```
void
get_row(row, image, y)
Pixel *row;
Image *image;
int y;
{
    if((y < 0) || (y >= image->ysize)) {
        return;
    }
    memcpy(row,
           image->data + (y * image->span),
           (sizeof(Pixel) * image->xsize));
}
```

```
void
get_column(column, image, x)
Pixel *column;
Image *image;
int x;
{
    int i, d;
    Pixel *p;

    if((x < 0) || (x >= image->xsize)) {
        return;
    }
    d = image->span;
    for(i = image->ysize, p = image->data + x; i-- > 0; p += d) {
        *column++ = *p;
    }
}
```

```
Pixel
put_pixel(image, x, y, data)
Image *image;
int x, y;
Pixel data;
{
    static Image *im = NULL;
    static int yy = -1;
    static Pixel *p = NULL;

    if((x < 0) || (x >= image->xsize) || (y < 0) || (y >= image->ysize)) {
        return(0);
    }
    if((im != image) || (yy != y)) {
```

```
        im = image;
        yy = y;
        p = image->data + (y * image->span);
    }
    return(p[x] = data);
}

Image *
new_image(xsize, ysize) /* create a blank image */
int xsize, ysize;
{
    Image *image;
    ASSERT(xsize > 0 && ysize > 0);

    if(image = (Image *)malloc(sizeof(Image)))
    {
        if(image->data = (Pixel *)calloc(ysize, xsize))
        {
            image->xsize = xsize;
            image->ysize = ysize;
            image->span = xsize;
        }
        else
        {
            free(image);
            image = NULL;
        }
    }
    return image;
}

void
free_image(image)
Image *image;
{
    ASSERT(image && image->data);
    free(image->data);
    free(image);
}

/*
    fileextension()

    Returns type of file based on its name.
    Returns the empty string if filename has no extension
*/
char*
fileextension(f)
char* f;
{
    /* Get a filename's extension string. */
    int i;
    for(i = strlen(f) - 1; i > 0; i--)
    {
        if(f[i] == '.')
            return f + i + 1;
    }
    /* No extension. Return the end of the string,
       which effectively returns a null string. */
    return f + strlen(f);
}
```

```
}

/* add your filetype loaders and savers here */

/* -- PXM bitmap support----- */

Image *
load_image_bm(f)          /* read image from bm file */
FILE *f;
{
    char *p;
    int width, height;
    Image *image;

    if(((p = next_token(f)) && (strcmp(p, "Bm") == 0))
    && ((p = next_token(f)) && ((width = atoi(p)) > 0))
    && ((p = next_token(f)) && ((height = atoi(p)) > 0))
    && ((p = next_token(f)) && (strcmp(p, "8") == 0))
    && (image = new_image(width, height))
    && (fread(image->data, (size_t)width, (size_t)height, f) == (size_t)height)) {
        return(image);          /* load successful */
    } else {
        return(NULL);          /* load failed */
    }
}

int
save_image_bm(f, image) /* write image to bm file */
FILE *f;
Image *image;
{
    fprintf(f, "Bm # PXM 8-bit greyscale image\x0A");
    fprintf(f, "%d %d 8 # width height depth\x0A",
            image->xsize, image->ysize);
    if(fwrite(image->data, (size_t)image->xsize, (size_t)image->ysize, f) ==
(size_t)image->ysize) {
        return(0);          /* save successful */
    } else {
        return(-1);          /* save failed */
    }
}

/* -- TGA bitmap support----- */

typedef struct
{
    unsigned char    lsb, msb;
} DoubleByte;

#define INT_TO_DB(n, db)      { (db).lsb = (unsigned char)((n) & 0xFF);\
                                (db).msb = (unsigned char)((n)
>> 8); }

#define DB_TO_INT(db)        (((int)((db).msb) << 8) + (db).lsb)

typedef struct
```



```
{
    unsigned char    IDfieldLen;
    unsigned char    colorMapType;
    unsigned char    imageType;
    DoubleByte       ColorMapOrigin;
    DoubleByte       ColorMapLen;
    unsigned char    ColorMapEntrySize; /* no. of bits */
    DoubleByte       Xorigin;
    DoubleByte       Yorigin;
    DoubleByte       Width;
    DoubleByte       Height;
    unsigned char    bpp;
    unsigned char    descriptor;
} TGAheader;

int
save_image_tga(f, image)          /* save image to TGA file */
FILE* f;
Image* image;
{
    int x, y;
    int n, j;
    TGAheader    header;
    boolean bOK = TRUE; /* assume success */

    /* Grayscale images only. */
    ASSERT(sizeof(Pixel) == 1);

    /* Write header */
    memset(&header, 0, sizeof(header));
    header.colorMapType = 1;
    header.imageType = 1;
    INT_TO_DB(256, header.ColorMapLen);
    header.ColorMapEntrySize = 24;
    INT_TO_DB(image->xsize, header.Width);
    INT_TO_DB(image->ysize, header.Height);
    header.bpp = 8;
    header.descriptor = 0x20; /* top down */

    bOK = (fwrite(&header, sizeof(header), 1, f) == 1);

    /* Write palette. */
    for(n = 0; n < 256 && bOK; n++)
    {
        for(j = 0; j < 3 && bOK; j++)
            bOK = fputc(n, f) != EOF;
    }

    /* Write pixel index values. */
    for(y = 0; y < image->ysize && bOK; y++)
    {
        for(x = 0; x < image->xsize && bOK; x++)
            bOK = (EOF != fputc(get_pixel(image, x, y), f));
    }
    return bOK ? 0 : -1;
} /* save_image_tga */
```

Image \*

```
load_image_tga(f)          /* read image from TGA file */
FILE *f;
{
    Image* image = NULL;
    boolean bTopDown;
    boolean bOK = FALSE;
    TGAheader header;
    int width, height, c, x, y, yy;

    /* Grayscale images only. */
    ASSERT(sizeof(Pixel) == 1);

    if(fread(&header, sizeof(header), 1, f) == 1 && header.bpp == 8)
    {
        bTopDown = (header.descriptor & 0x20);
        width = DB_TO_INT(header.Width);
        height = DB_TO_INT(header.Height);

        image = new_image(width, height);
        if(image != NULL)
        {
            /* Skip palette */
            bOK = 0 == fseek(f, DB_TO_INT(header.ColorMapLen) *
header.ColorMapEntrySize/8, SEEK_CUR);

            /* Read pixels */
            for(y = 0; y < height && bOK; y++)
            {
                yy = bTopDown ? y : (height-1) - y;

                for(x = 0; x < width && bOK; x++)
                {
                    bOK = ((c = fgetc(f)) != EOF);
                    if(bOK)
                        put_pixel(image, x, yy, (Pixel)c);
                }
            }
        }
        if(!bOK && image != NULL)
        {
            free_image(image);
            image = NULL;
        }

        return image;
    }
}

/* -- End TGA bitmap support ----- */

/*
    ImageHandler stuff.
    An ImageHandler stores an image file's extension,
    and pointers to functions that read and write the image format.
*/
typedef struct
{
    char* filetype;
    Image* (*reader)();
    int (*writer)();
}
```

```
} ImageHandler;

ImageHandler gImageHandlers[] =
{
    { "bm", load_image_bm, save_image_bm },
    { "tga", load_image_tga, save_image_tga }
    /* add your image handlers here */
};

/*
    find_imagehandler()

    Given a filename, return an image handler for it.
    Return NULL if no handler available.
*/
ImageHandler*
find_imagehandler(f)
char* f;
{
    int i;
    for(i = 0; i < sizeof(gImageHandlers) / sizeof(gImageHandlers[0]); i++)
    {
        if(stricmp(gImageHandlers[i].filetype, fileextension(f)) == 0)
            return &gImageHandlers[i];
    }
    return NULL;
}

/*
    load_image()

    Given a filename, hands off to appropriate image loader.
    Returns pointer to loaded Image, NULL if it can't.
*/
Image *
load_image(f)          /* read image from file */
char *f;
{
    Image *image;
    FILE* fp;
    ImageHandler* handler;

    ASSERT(f);

    fp = fopen(f, "rb");
    if(fp == NULL)
        return NULL;

    if(handler = find_imagehandler(f))
        image = handler->reader(fp);

    fclose(fp);
    return image;
}

/*
    save_image()
```

Given a filename and an Image, hands off to appropriate image saver.  
Returns -1 on error, 0 on success.

```
*/
int
save_image(f, image)
char *f;
Image* image;
{
    FILE* fp;
    ImageHandler* handler;
    int nRet = -1; /* assume failure */

    ASSERT(f && image);

    fp = fopen(f, "wb");
    if(fp != NULL)
    {
        if(handler = find_imagehandler(f))
            nRet = handler->writer(fp, image);

        fclose(fp);
    }
    return nRet;
}

/*
 *   filter function definitions
 */

#define filter_support          (1.0)

double
filter(t)
double t;
{
    /*  $f(t) = 2|t|^3 - 3|t|^2 + 1$ ,  $-1 \leq t \leq 1$  */
    if(t < 0.0) t = -t;
    if(t < 1.0) return((2.0 * t - 3.0) * t * t + 1.0);
    return(0.0);
}

#define box_support            (0.5)

double
box_filter(t)
double t;
{
    if((t > -0.5) && (t <= 0.5)) return(1.0);
    return(0.0);
}

#define triangle_support       (1.0)

double
triangle_filter(t)
double t;
{
    if(t < 0.0) t = -t;
```

```
    if(t < 1.0) return(1.0 - t);
    return(0.0);
}
```

```
#define bell_support          (1.5)
```

```
double
bell_filter(t)              /* box (*) box (*) box */
double t;
{
    if(t < 0) t = -t;
    if(t < .5) return(.75 - (t * t));
    if(t < 1.5) {
        t = (t - 1.5);
        return(.5 * (t * t));
    }
    return(0.0);
}
```

```
#define B_spline_support     (2.0)
```

```
double
B_spline_filter(t)         /* box (*) box (*) box (*) box */
double t;
{
    double tt;

    if(t < 0) t = -t;
    if(t < 1) {
        tt = t * t;
        return((.5 * tt * t) - tt + (2.0 / 3.0));
    } else if(t < 2) {
        t = 2 - t;
        return((1.0 / 6.0) * (t * t * t));
    }
    return(0.0);
}
```

```
double
sinc(x)
double x;
{
    x *= M_PI;
    if(x != 0) return(sin(x) / x);
    return(1.0);
}
```

```
#define Lanczos3_support     (3.0)
```

```
double
Lanczos3_filter(t)
double t;
{
    if(t < 0) t = -t;
    if(t < 3.0) return(sinc(t) * sinc(t/3.0));
    return(0.0);
}
```

```
#define Mitchell_support     (2.0)
```

```
#define B          (1.0 / 3.0)
```

```
#define C          (1.0 / 3.0)

double
Mitchell_filter(t)
double t;
{
    double tt;

    tt = t * t;
    if(t < 0) t = -t;
    if(t < 1.0) {
        t = (((12.0 - 9.0 * B - 6.0 * C) * (t * tt))
            + ((-18.0 + 12.0 * B + 6.0 * C) * tt)
            + (6.0 - 2 * B));
        return(t / 6.0);
    } else if(t < 2.0) {
        t = (((-1.0 * B - 6.0 * C) * (t * tt))
            + ((6.0 * B + 30.0 * C) * tt)
            + ((-12.0 * B - 48.0 * C) * t)
            + (8.0 * B + 24 * C));
        return(t / 6.0);
    }
    return(0.0);
}

/*
 *   image rescaling routine
 */

typedef struct {
    int      pixel;
    double   weight;
} CONTRIB;

typedef struct {
    int      n;                /* number of contributors */
    CONTRIB *p;               /* pointer to list of contributions */
} CLIST;

CLIST  *contrib;              /* array of contribution lists */

/*
    roundcloser()

    Round an FP value to its closest int representation.
    General routine; ideally belongs in general math lib file.
*/
int roundcloser(double d)
{
    /* Untested potential one-liner, but smacks of call overhead */
    /* return fabs(ceil(d)-d) <= 0.5 ? ceil(d) : floor(d); */

    /* Untested potential optimized ceil() usage */
    double cd = ceil(d);
    int ncd = (int)cd;
    if(fabs(cd - d) > 0.5)
        ncd--;
    return ncd;
}
*/
```

```
/* Version that uses no function calls at all. */
```

```
int n = (int) d;  
double diff = d - (double)n;  
if(diff < 0)  
    diff = -diff;  
if(diff >= 0.5)  
{  
    if(d < 0)  
        n--;  
    else  
        n++;  
}  
return n;
```

```
} /* roundcloser */
```

```
/*
```

```
    calc_x_contrib()
```

```
Calculates the filter weights for a single target column.  
contribX->p must be freed afterwards.
```

```
Returns -1 if error, 0 otherwise.
```

```
*/
```

```
int calc_x_contrib(contribX, xscale, fwidth, dstwidth, srcwidth, filterf, i)  
CLIST* contribX; /* Receiver of contrib info */  
double xscale; /* Horizontal zooming scale */  
double fwidth; /* Filter sampling width */  
int dstwidth; /* Target bitmap width */  
int srcwidth; /* Source bitmap width */  
double (*filterf)(double); /* Filter proc */  
int i; /* Pixel column in source bitmap being  
processed */  
{  
    double width;  
    double fscale;  
    double center, left, right;  
    double weight;  
    int j, k, n;  
  
    if(xscale < 1.0)  
    {  
        /* Shrinking image */  
        width = fwidth / xscale;  
        fscale = 1.0 / xscale;  
  
        contribX->n = 0;  
        contribX->p = (CONTRIB *)calloc((int) (width * 2 + 1),  
                                         sizeof(CONTRIB));  
        if(contribX->p == NULL)  
            return -1;  
  
        center = (double) i / xscale;  
        left = ceil(center - width);  
        right = floor(center + width);  
        for(j = (int)left; j <= right; ++j)  
        {  
            weight = center - (double) j;  
            weight = (*filterf)(weight / fscale) / fscale;  
            if(j < 0)  
                n = -j;
```

```
        else if(j >= srcwidth)
            n = (srcwidth - j) + srcwidth - 1;
        else
            n = j;

        k = contribX->n++;
        contribX->p[k].pixel = n;
        contribX->p[k].weight = weight;
    }

}
else
{
    /* Expanding image */
    contribX->n = 0;
    contribX->p = (CONTRIB *)calloc((int) (fwidth * 2 + 1),
                                    sizeof(CONTRIB));
    if(contribX->p == NULL)
        return -1;
    center = (double) i / xscale;
    left = ceil(center - fwidth);
    right = floor(center + fwidth);

    for(j = (int)left; j <= right; ++j)
    {
        weight = center - (double) j;
        weight = (*filterf)(weight);
        if(j < 0) {
            n = -j;
        } else if(j >= srcwidth) {
            n = (srcwidth - j) + srcwidth - 1;
        } else {
            n = j;
        }
        k = contribX->n++;
        contribX->p[k].pixel = n;
        contribX->p[k].weight = weight;
    }
}
return 0;
} /* calc_x_contrib */

/*
    zoom()

    Resizes bitmaps while resampling them.
    Returns -1 if error, 0 if success.
*/
int
zoom(dst, src, filterf, fwidth)
Image* dst;
Image* src;
double (*filterf)(double);
double fwidth;
{
    Pixel* tmp;
    double xscale, yscale;          /* zoom scale factors */
    int xx;
    int i, j, k;                   /* loop variables */
    int n;                          /* pixel number */
}
```



```
double center, left, right;      /* filter calculation variables */
double width, fscale, weight;    /* filter calculation variables */
Pixel pel, pel2;
int bPelDelta;
CLIST *contribY;                 /* array of contribution lists */
CLIST contribX;
int nRet = -1;

/* create intermediate column to hold horizontal dst column zoom */
tmp = (Pixel*)malloc(src->ysize * sizeof(Pixel));
if(tmp == NULL)
    return 0;

xscale = (double) dst->xsize / (double) src->xsize;

/* Build y weights */
/* pre-calculate filter contributions for a column */
contribY = (CLIST *)calloc(dst->ysize, sizeof(CLIST));
if(contribY == NULL)
{
    free(tmp);
    return -1;
}

yscale = (double) dst->ysize / (double) src->ysize;

if(yscale < 1.0)
{
    width = fwidth / yscale;
    fscale = 1.0 / yscale;
    for(i = 0; i < dst->ysize; ++i)
    {
        contribY[i].n = 0;
        contribY[i].p = (CONTRIB *)calloc((int) (width * 2 + 1),
                                           sizeof(CONTRIB));
        if(contribY[i].p == NULL)
        {
            free(tmp);
            free(contribY);
            return -1;
        }
        center = (double) i / yscale;
        left = ceil(center - width);
        right = floor(center + width);
        for(j = (int)left; j <= right; ++j) {
            weight = center - (double) j;
            weight = (*filterf)(weight / fscale) / fscale;
            if(j < 0) {
                n = -j;
            } else if(j >= src->ysize) {
                n = (src->ysize - j) + src->ysize - 1;
            } else {
                n = j;
            }
            k = contribY[i].n++;
            contribY[i].p[k].pixel = n;
            contribY[i].p[k].weight = weight;
        }
    }
} else {
    for(i = 0; i < dst->ysize; ++i) {
```

```
        contribY[i].n = 0;
        contribY[i].p = (CONTRIB *)calloc((int) (fwidth * 2 + 1),
                                           sizeof(CONTRIB));
        if(contribY[i].p == NULL)
        {
            free(tmp);
            free(contribY);
            return -1;
        }
        center = (double) i / yscale;
        left = ceil(center - fwidth);
        right = floor(center + fwidth);
        for(j = (int)left; j <= right; ++j) {
            weight = center - (double) j;
            weight = (*filterf)(weight);
            if(j < 0) {
                n = -j;
            } else if(j >= src->ysize) {
                n = (src->ysize - j) + src->ysize - 1;
            } else {
                n = j;
            }
            k = contribY[i].n++;
            contribY[i].p[k].pixel = n;
            contribY[i].p[k].weight = weight;
        }
    }

for(xx = 0; xx < dst->xsize; xx++)
{
    if(0 != calc_x_contrib(&contribX, xscale, fwidth,
                          dst->xsize, src->xsize,
filterf, xx))
    {
        goto __zoom_cleanup;
    }
    /* Apply horz filter to make dst column in tmp. */
    for(k = 0; k < src->ysize; ++k)
    {
        weight = 0.0;
        bPelDelta = FALSE;
        pel = get_pixel(src, contribX.p[0].pixel, k);
        for(j = 0; j < contribX.n; ++j)
        {
            pel2 = get_pixel(src, contribX.p[j].pixel, k);
            if(pel2 != pel)
                bPelDelta = TRUE;
            weight += pel2 * contribX.p[j].weight;
        }
        weight = bPelDelta ? roundcloser(weight) : pel;

        tmp[k] = (Pixel)CLAMP(weight, BLACK_PIXEL, WHITE_PIXEL);
    } /* next row in temp column */

    free(contribX.p);

    /* The temp column has been built. Now stretch it
       vertically into dst column. */
    for(i = 0; i < dst->ysize; ++i)
```

```
    {
        weight = 0.0;
        bPelDelta = FALSE;
        pel = tmp[contribY[i].p[0].pixel];

        for(j = 0; j < contribY[i].n; ++j)
        {
            pel2 = tmp[contribY[i].p[j].pixel];
            if(pel2 != pel)
                bPelDelta = TRUE;
            weight += pel2 * contribY[i].p[j].weight;
        }
        weight = bPelDelta ? roundcloser(weight) : pel;
        put_pixel(dst, xx, i,
            (Pixel)CLAMP(weight, BLACK_PIXEL, WHITE_PIXEL));
    } /* next dst row */
} /* next dst column */
nRet = 0; /* success */
```

```
__zoom_cleanup:
    free(tmp);

    /* free the memory allocated for vertical filter weights */
    for(i = 0; i < dst->ysize; ++i)
        free(contribY[i].p);
    free(contribY);

    return nRet;
} /* zoom */
```

```
/*
 *      command line interface
 */
```

```
void
usage()
{
    fprintf(stderr, "usage: %s [-options] input output\n", _Program);
    fprintf(stderr, "\n
options:\n\
    -x xsize          output x size\n\
    -y ysize          output y size\n\
    -f filter          filter type\n\
{b=box, t=triangle, q=bell, B=B-spline, h=hermite, l=Lanczos3, m=Mitchell}\n\
    input, output    files to read/write. Use BM or TGA extension.\n\
");
    exit(1);
}
```

```
void
banner()
{
    printf("%s v%s -- %s\n", _Program, _Version, _Copyright);
}
```

```
int
main(argc, argv)
int argc;
```

```
char *argv[];
{
    register int c;
#ifdef WIN32
    extern int optind;
    extern char *optarg;
#endif
    int xsize = 0, ysize = 0;
    double (*f)() = filter;
    double s = filter_support;
    char *dstfile, *srcfile;
    Image *dst, *src;

    while((c = getopt(argc, argv, "x:y:f:V")) != EOF) {
        switch(c) {
            case 'x': xsize = atoi(optarg); break;
            case 'y': ysize = atoi(optarg); break;
            case 'f':
                switch(*optarg) {
                    case 'b': f=box_filter; s=box_support; break;
                    case 't': f=triangle_filter; s=triangle_support; break;
                    case 'q': f=bell_filter; s=bell_support; break;
                    case 'B': f=B_spline_filter; s=B_spline_support; break;
                    case 'h': f=filter; s=filter_support; break;
                    case 'l': f=Lanczos3_filter; s=Lanczos3_support; break;
                    case 'm': f=Mitchell_filter; s=Mitchell_support; break;
                    default: usage();
                }
                break;
            case 'V': banner(); exit(EXIT_SUCCESS);
            case '?': usage();
            default: usage();
        }
    }

    if((argc - optind) != 2) usage();
    srcfile = argv[optind];
    dstfile = argv[optind + 1];

    if((src = load_image(srcfile)) == NULL)
    {
        fprintf(stderr, "%s: can't load source image '%s'\n",
            _Program, srcfile);
        exit(EXIT_FAILURE);
    }

    if(xsize <= 0) xsize = src->xsize;
    if(ysize <= 0) ysize = src->ysize;

    dst = new_image(xsize, ysize);

    if(zoom(dst, src, f, s) != 0)
    {
        fprintf(stderr, "%s: can't process image '%s'\n",
            _Program, srcfile);
        exit(EXIT_FAILURE);
    }
    else
    {
        if(save_image(dstfile, dst) != 0)
        {

```

```
        fprintf(stderr, "%s: can't save destination image '%s'\n",
                _Program, dstfile);
        exit(EXIT_FAILURE);
    }
}
exit(EXIT_SUCCESS);
return 0;
}
```

/\* \*\*\*\* \*/

Optimized Bitmap Scaling Routines  
by Dale Schumacher

/\* \*\*\*\* \*/

```
typedef struct {
    int      xsize;          /* length of each scanline in pixels */
    int      ysize;          /* number of scanlines in the image */
    int      yspan;          /* byte offset between scanlines */
    unsigned char * data;    /* pointer to bitmap data */
} Bitmap;

static unsigned char
redux_6_of_8[256] = {
0x00,0x01,0x02,0x03,0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x04,0x05,0x06,0x07,
0x08,0x09,0x0a,0x0b,0x08,0x09,0x0a,0x0b,0x0c,0x0d,0x0e,0x0f,0x0c,0x0d,0x0e,0x0f,
0x10,0x11,0x12,0x13,0x10,0x11,0x12,0x13,0x14,0x15,0x16,0x17,0x14,0x15,0x16,0x17,
0x18,0x19,0x1a,0x1b,0x18,0x19,0x1a,0x1b,0x1c,0x1d,0x1e,0x1f,0x1c,0x1d,0x1e,0x1f,
0x00,0x01,0x02,0x03,0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x04,0x05,0x06,0x07,
0x08,0x09,0x0a,0x0b,0x08,0x09,0x0a,0x0b,0x0c,0x0d,0x0e,0x0f,0x0c,0x0d,0x0e,0x0f,
0x10,0x11,0x12,0x13,0x10,0x11,0x12,0x13,0x14,0x15,0x16,0x17,0x14,0x15,0x16,0x17,
0x18,0x19,0x1a,0x1b,0x18,0x19,0x1a,0x1b,0x1c,0x1d,0x1e,0x1f,0x1c,0x1d,0x1e,0x1f,
0x20,0x21,0x22,0x23,0x20,0x21,0x22,0x23,0x24,0x25,0x26,0x27,0x24,0x25,0x26,0x27,
0x28,0x29,0x2a,0x2b,0x28,0x29,0x2a,0x2b,0x2c,0x2d,0x2e,0x2f,0x2c,0x2d,0x2e,0x2f,
0x30,0x31,0x32,0x33,0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x34,0x35,0x36,0x37,
0x38,0x39,0x3a,0x3b,0x38,0x39,0x3a,0x3b,0x3c,0x3d,0x3e,0x3f,0x3c,0x3d,0x3e,0x3f,
0x20,0x21,0x22,0x23,0x20,0x21,0x22,0x23,0x24,0x25,0x26,0x27,0x24,0x25,0x26,0x27,
0x28,0x29,0x2a,0x2b,0x28,0x29,0x2a,0x2b,0x2c,0x2d,0x2e,0x2f,0x2c,0x2d,0x2e,0x2f,
0x30,0x31,0x32,0x33,0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x34,0x35,0x36,0x37,
0x38,0x39,0x3a,0x3b,0x38,0x39,0x3a,0x3b,0x3c,0x3d,0x3e,0x3f,0x3c,0x3d,0x3e,0x3f
};

void
scale_bitmap_6_of_8(dst, src)
Bitmap *dst, *src;
{
    register int bitbuf = 0, bits;
    register int x, y, j, xbytes;
    unsigned char *p, *pp, *q, *qq;

    xbytes = src->xsize / 8;
    qq = src->data;
    pp = dst->data;
    for(y = 0; y < src->ysize; ++y) {
        bits = 0;
        p = pp;
        q = qq;
        qq += src->yspan;
        j = y & 0x03;
        if(j == 2) {
            continue;          /* skip scanline */
        }
        pp += dst->yspan;
        for(x = 0; x < xbytes; ++x) {
            bitbuf <= 6;
            bitbuf |= redux_6_of_8[*q++];
            bits += 6;
            if(bits >= 8) {
                *p++ = bitbuf >> (bits - 8);
            }
        }
    }
}
```

```
        bits -= 8;
    }
}

static unsigned char
redux_3_of_8[256] = {
0x00,0x00,0x00,0x01,0x00,0x01,0x01,0x01,0x00,0x01,0x00,0x01,0x02,0x03,0x03,0x03,
0x00,0x00,0x00,0x01,0x02,0x03,0x03,0x03,0x02,0x03,0x02,0x03,0x02,0x03,0x03,0x03,
0x00,0x00,0x00,0x01,0x00,0x01,0x01,0x01,0x02,0x03,0x02,0x03,0x02,0x03,0x03,0x03,
0x02,0x02,0x02,0x03,0x02,0x03,0x03,0x03,0x02,0x03,0x02,0x03,0x02,0x03,0x03,0x03,
0x00,0x00,0x00,0x01,0x00,0x01,0x01,0x01,0x00,0x01,0x00,0x01,0x02,0x03,0x03,0x03,
0x04,0x04,0x04,0x05,0x06,0x07,0x07,0x07,0x06,0x07,0x06,0x07,0x06,0x07,0x07,0x07,
0x04,0x04,0x04,0x05,0x04,0x05,0x05,0x05,0x06,0x07,0x06,0x07,0x06,0x07,0x07,0x07,
0x06,0x06,0x06,0x07,0x06,0x07,0x07,0x07,0x06,0x07,0x06,0x07,0x06,0x07,0x07,0x07,
0x00,0x00,0x00,0x01,0x00,0x01,0x01,0x01,0x00,0x01,0x00,0x01,0x02,0x03,0x03,0x03,
0x00,0x00,0x00,0x01,0x02,0x03,0x03,0x03,0x02,0x03,0x02,0x03,0x02,0x03,0x03,0x03,
0x04,0x04,0x04,0x05,0x04,0x05,0x05,0x05,0x06,0x07,0x06,0x07,0x06,0x07,0x07,0x07,
0x06,0x06,0x06,0x07,0x06,0x07,0x07,0x07,0x06,0x07,0x06,0x07,0x06,0x07,0x07,0x07,
0x04,0x04,0x04,0x05,0x04,0x05,0x05,0x05,0x04,0x05,0x04,0x05,0x06,0x07,0x07,0x07,
0x04,0x04,0x04,0x05,0x06,0x07,0x07,0x07,0x06,0x07,0x06,0x07,0x06,0x07,0x07,0x07,
0x04,0x04,0x04,0x05,0x04,0x05,0x05,0x05,0x06,0x07,0x06,0x07,0x06,0x07,0x07,0x07,
0x06,0x06,0x06,0x07,0x06,0x07,0x07,0x07,0x06,0x07,0x06,0x07,0x06,0x07,0x07,0x07
};

void
scale_bitmap_3_of_8(dst, src)
Bitmap *dst, *src;
{
    register int bitbuf = 0, bits;
    register int x, y, j, xbytes;
    unsigned char *p, *pp, *q, *qq;

    xbytes = src->xsize / 8;
    qq = src->data;
    pp = dst->data;
    for(y = 0; y < src->ysize; ++y) {
        bits = 0;
        p = pp;
        q = qq;
        qq += src->yspan;
        j = y & 7;
        if(!((j == 1) || (j == 4) || (j == 7))) {
            continue; /* skip scanline */
        }
        pp += dst->yspan;
        for(x = 0; x < xbytes; ++x) {
            bitbuf <= 3;
            bitbuf |= redux_3_of_8[*q++];
            bits += 3;
            if(bits >= 8) {
                *p++ = bitbuf >> (bits - 8);
                bits -= 8;
            }
        }
    }
}

static unsigned short redux_7_of_16[0x10000];
```

```
void
scale_bitmap_7_of_16(dst, src)
Bitmap *dst, *src;
{
    static init_flag = 0;
    register long bitbuf = 0;
    register int bits;
    register int x, y, i, j, xwords;
    unsigned char *pp, *qq;
    unsigned short *dp, *dq;

    if(init_flag == 0) { /* compute table at run time, but only once! */
#define BIT(b) ((i >> (b)) & 1) /* extract bit 'b' from int 'i' */
        for(i = 0; i < 0x10000; ++i) {
            redux_7_of_16[i] = (BIT(14) << 6)
                               | (BIT(12) << 5)
                               | (BIT(10) << 4)
                               | ((BIT(8) | BIT(7)) << 3)
                               | (BIT(5) << 2)
                               | (BIT(3) << 1)
                               | (BIT(1) | BIT(0));
        }
        init_flag = 1;
#undef BIT
    }
    xwords = src->xsize / 16;
    qq = src->data;
    pp = dst->data;
    for(y = 0; y < src->ysize; ++y) {
        bits = 0;
        dp = (unsigned short *)pp;
        dq = (unsigned short *)qq;
        qq += src->yspan;
        j = y & 0xF;
        if(!(j==1||j==3||j==5||j==8||j==10||j==12||j==14)) {
            continue; /* skip scanline */
        }
        pp += dst->yspan;
        for(x = 0; x < xwords; ++x) {
            bitbuf <= 7;
            bitbuf |= redux_7_of_16[*dq++];
            bits += 7;
            if(bits >= 16) {
                *dp++ = bitbuf >> (bits - 16);
                bits -= 16;
            }
        }
    }
}

static unsigned char
bit_reverse_byte[256] = {
0x00,0x80,0x40,0xc0,0x20,0xa0,0x60,0xe0,0x10,0x90,0x50,0xd0,0x30,0xb0,0x70,0xf0,
0x08,0x88,0x48,0xc8,0x28,0xa8,0x68,0xe8,0x18,0x98,0x58,0xd8,0x38,0xb8,0x78,0xf8,
0x04,0x84,0x44,0xc4,0x24,0xa4,0x64,0xe4,0x14,0x94,0x54,0xd4,0x34,0xb4,0x74,0xf4,
0x0c,0x8c,0x4c,0xcc,0x2c,0xac,0x6c,0xec,0x1c,0x9c,0x5c,0xdc,0x3c,0xbc,0x7c,0xfc,
0x02,0x82,0x42,0xc2,0x22,0xa2,0x62,0xe2,0x12,0x92,0x52,0xd2,0x32,0xb2,0x72,0xf2,
0x0a,0x8a,0x4a,0xca,0x2a,0xaa,0x6a,0xea,0x1a,0x9a,0x5a,0xda,0x3a,0xba,0x7a,0xfa,
0x06,0x86,0x46,0xc6,0x26,0xa6,0x66,0xe6,0x16,0x96,0x56,0xd6,0x36,0xb6,0x76,0xf6,
0x0e,0x8e,0x4e,0xce,0x2e,0xae,0x6e,0xee,0x1e,0x9e,0x5e,0xde,0x3e,0xbe,0x7e,0xfe,
0x01,0x81,0x41,0xc1,0x21,0xa1,0x61,0xe1,0x11,0x91,0x51,0xd1,0x31,0xb1,0x71,0xf1,
```



```
0x09,0x89,0x49,0xc9,0x29,0xa9,0x69,0xe9,0x19,0x99,0x59,0xd9,0x39,0xb9,0x79,0xf9,
0x05,0x85,0x45,0xc5,0x25,0xa5,0x65,0xe5,0x15,0x95,0x55,0xd5,0x35,0xb5,0x75,0xf5,
0x0d,0x8d,0x4d,0xcd,0x2d,0xad,0x6d,0xed,0x1d,0x9d,0x5d,0xdd,0x3d,0xbd,0x7d,0xfd,
0x03,0x83,0x43,0xc3,0x23,0xa3,0x63,0xe3,0x13,0x93,0x53,0xd3,0x33,0xb3,0x73,0xf3,
0x0b,0x8b,0x4b,0xcb,0x2b,0xab,0x6b,0xeb,0x1b,0x9b,0x5b,0xdb,0x3b,0xbb,0x7b,0xfb,
0x07,0x87,0x47,0xc7,0x27,0xa7,0x67,0xe7,0x17,0x97,0x57,0xd7,0x37,0xb7,0x77,0xf7,
0x0f,0x8f,0x4f,0xcf,0x2f,0xaf,0x6f,0xef,0x1f,0x9f,0x5f,0xdf,0x3f,0xbf,0x7f,0xff
};
```

```
void
rotate_bitmap_180(dst, src)
Bitmap *dst, *src;
{
    register int x, y, xbytes;
    unsigned char *p, *pp, *q, *qq;

    xbytes = src->xsize / 8;
    qq = dst->data;
    pp = src->data;
    pp += (src->>yspan * src->ysize);
    for(y = 0; y < src->ysize; ++y) {
        q = qq;
        qq += dst->yspan;
        pp -= src->yspan;
        p = pp + xbytes;
        for(x = 0; x < xbytes; ++x) {
            *q++ = bit_reverse_byte[*--p];
        }
    }
}
```

```
/*
 * Roots3And4.c
 *
 * Utility functions to find cubic and quartic roots,
 * coefficients are passed like this:
 *
 *      c[0] + c[1]*x + c[2]*x^2 + c[3]*x^3 + c[4]*x^4 = 0
 *
 * The functions return the number of non-complex roots and
 * put the values into the s array.
 *
 * Author:          Jochen Schwarze (schwarze@isa.de)
 *
 * Jan 26, 1990      Version for Graphics Gems
 * Oct 11, 1990      Fixed sign problem for negative q's in SolveQuartic
 *                   (reported by Mark Podlipec),
 *                   Old-style function definitions,
 *                   IsZero() as a macro
 * Nov 23, 1990      Some systems do not declare acos() and cbrt() in
 *                   <math.h>, though the functions exist in the library.
 *                   If large coefficients are used, EQN_EPS should be
 *                   reduced considerably (e.g. to 1E-30), results will be
 *                   correct but multiple roots might be reported more
 *                   than once.
 */

#include <math.h>
#ifndef M_PI
#define M_PI          3.14159265358979323846
#endif
extern double  sqrt(), cbrt(), cos(), acos();

/* epsilon surrounding for near zero values */

#define EQN_EPS      1e-9
#define IsZero(x)    ((x) > -EQN_EPS && (x) < EQN_EPS)

#ifdef NOCBRT
#define cbrt(x)      ((x) > 0.0 ? pow((double)(x), 1.0/3.0) : \
                     ((x) < 0.0 ? -pow((double)-(x), 1.0/3.0) : 0.0))
#endif

int SolveQuadric(c, s)
    double c[ 3 ];
    double s[ 2 ];
{
    double p, q, D;

    /* normal form: x^2 + px + q = 0 */

    p = c[ 1 ] / (2 * c[ 2 ]);
    q = c[ 0 ] / c[ 2 ];

    D = p * p - q;

    if (IsZero(D))
    {
        s[ 0 ] = - p;
        return 1;
    }
    else if (D < 0)
```

```
{
    return 0;
}
else if (D > 0)
{
    double sqrt_D = sqrt(D);

    s[ 0 ] = sqrt_D - p;
    s[ 1 ] = - sqrt_D - p;
    return 2;
}
}

int SolveCubic(c, s)
double c[ 4 ];
double s[ 3 ];
{
    int i, num;
    double sub;
    double A, B, C;
    double sq_A, p, q;
    double cb_p, D;

    /* normal form: x^3 + Ax^2 + Bx + C = 0 */

    A = c[ 2 ] / c[ 3 ];
    B = c[ 1 ] / c[ 3 ];
    C = c[ 0 ] / c[ 3 ];

    /* substitute x = y - A/3 to eliminate quadric term:
       x^3 + px + q = 0 */

    sq_A = A * A;
    p = 1.0/3 * (- 1.0/3 * sq_A + B);
    q = 1.0/2 * (2.0/27 * A * sq_A - 1.0/3 * A * B + C);

    /* use Cardano's formula */

    cb_p = p * p * p;
    D = q * q + cb_p;

    if (IsZero(D))
    {
        if (IsZero(q)) /* one triple solution */
        {
            s[ 0 ] = 0;
            num = 1;
        }
        else /* one single and one double solution */
        {
            double u = cbrt(-q);
            s[ 0 ] = 2 * u;
            s[ 1 ] = - u;
            num = 2;
        }
    }
    else if (D < 0) /* Casus irreducibilis: three real solutions */
    {
        double phi = 1.0/3 * acos(-q / sqrt(-cb_p));
        double t = 2 * sqrt(-p);
```

```
s[ 0 ] = t * cos(phi);
s[ 1 ] = - t * cos(phi + M_PI / 3);
s[ 2 ] = - t * cos(phi - M_PI / 3);
num = 3;
}
else /* one real solution */
{
    double sqrt_D = sqrt(D);
    double u = cbrt(sqrt_D - q);
    double v = - cbrt(sqrt_D + q);

    s[ 0 ] = u + v;
    num = 1;
}

/* resubstitute */

sub = 1.0/3 * A;

for (i = 0; i < num; ++i)
    s[ i ] -= sub;

return num;
}

int SolveQuartic(c, s)
double c[ 5 ];
double s[ 4 ];
{
    double coeffs[ 4 ];
    double z, u, v, sub;
    double A, B, C, D;
    double sq_A, p, q, r;
    int i, num;

    /* normal form: x^4 + Ax^3 + Bx^2 + Cx + D = 0 */

    A = c[ 3 ] / c[ 4 ];
    B = c[ 2 ] / c[ 4 ];
    C = c[ 1 ] / c[ 4 ];
    D = c[ 0 ] / c[ 4 ];

    /* substitute x = y - A/4 to eliminate cubic term:
       x^4 + px^2 + qx + r = 0 */

    sq_A = A * A;
    p = - 3.0/8 * sq_A + B;
    q = 1.0/8 * sq_A * A - 1.0/2 * A * B + C;
    r = - 3.0/256*sq_A*sq_A + 1.0/16*sq_A*B - 1.0/4*A*C + D;

    if (IsZero(r))
    {
        /* no absolute term: y(y^3 + py + q) = 0 */

        coeffs[ 0 ] = q;
        coeffs[ 1 ] = p;
        coeffs[ 2 ] = 0;
        coeffs[ 3 ] = 1;
    }
}
```

```
    num = SolveCubic(coeffs, s);

    s[ num++ ] = 0;
}
else
{
    /* solve the resolvent cubic ... */

    coeffs[ 0 ] = 1.0/2 * r * p - 1.0/8 * q * q;
    coeffs[ 1 ] = - r;
    coeffs[ 2 ] = - 1.0/2 * p;
    coeffs[ 3 ] = 1;

    (void) SolveCubic(coeffs, s);

    /* ... and take the one real solution ... */

    z = s[ 0 ];

    /* ... to build two quadric equations */

    u = z * z - r;
    v = 2 * z - p;

    if (IsZero(u))
        u = 0;
    else if (u > 0)
        u = sqrt(u);
    else
        return 0;

    if (IsZero(v))
        v = 0;
    else if (v > 0)
        v = sqrt(v);
    else
        return 0;

    coeffs[ 0 ] = z - u;
    coeffs[ 1 ] = q < 0 ? -v : v;
    coeffs[ 2 ] = 1;

    num = SolveQuadric(coeffs, s);

    coeffs[ 0 ] = z + u;
    coeffs[ 1 ] = q < 0 ? v : -v;
    coeffs[ 2 ] = 1;

    num += SolveQuadric(coeffs, s + num);
}

/* resubstitute */

sub = 1.0/4 * A;

for (i = 0; i < num; ++i)
    s[ i ] -= sub;

return num;
}
```

```
/*
Fast Circle-Rectangle Intersection Checking
by Clifford A. Shaffer
from "Graphics Gems", Academic Press, 1990
*/

#include "GraphicsGems.h"

boolean Check_Intersect(R, C, Rad)

/* Return TRUE iff rectangle R intersects circle with centerpoint C and
   radius Rad. */
Box2 *R;
Point2 *C;
double Rad;
{
    double Rad2;

    Rad2 = Rad * Rad;
    /* Translate coordinates, placing C at the origin. */
    R->max.x -= C->x;  R->max.y -= C->y;
    R->min.x -= C->x;  R->min.y -= C->y;

    if (R->max.x < 0) /* R to left of circle center */
        if (R->max.y < 0) /* R in lower left corner */
            return ((R->max.x * R->max.x + R->max.y * R->max.y) < Rad2);
        else if (R->min.y > 0) /* R in upper left corner */
            return ((R->max.x * R->max.x + R->min.y * R->min.y) < Rad2);
        else /* R due West of circle */
            return (ABS(R->max.x) < Rad);
    else if (R->min.x > 0) /* R to right of circle center */
        if (R->max.y < 0) /* R in lower right corner */
            return ((R->min.x * R->min.x + R->max.y * R->max.y) < Rad2);
        else if (R->min.y > 0) /* R in upper right corner */
            return ((R->min.x * R->min.x + R->min.y + R->min.y) < Rad2);
        else /* R due East of circle */
            return (R->min.x < Rad);
    else /* R on circle vertical centerline */
        if (R->max.y < 0) /* R due South of circle */
            return (ABS(R->max.y) < Rad);
        else if (R->min.y > 0) /* R due North of circle */
            return (R->min.y < Rad);
        else /* R contains circle centerpoint */
            return(TRUE);
}
```

```
/*
Bit Interleaving for Quad- or Octrees
by Clifford A. Shaffer
from "Graphics Gems", Academic Press, 1990
*/

#include "GraphicsGems.h"
#define B_MAX_DEPTH 14 /* maximum depth allowed */

/* byteval is the lookup table for coordinate interleaving. Given a
   4 bit portion of the (x, y) coordinates, return the bit interleaving.
   Notice that this table looks like the order in which the pixels of
   a 16 X 16 pixel image would be visited. */
int byteval[16][16] =
{
    0,  1,  4,  5, 16, 17, 20, 21, 64, 65, 68, 69, 80, 81, 84, 85,
    2,  3,  6,  7, 18, 19, 22, 23, 66, 67, 70, 71, 82, 83, 86, 87,
    8,  9, 12, 13, 24, 25, 28, 29, 72, 73, 76, 77, 88, 89, 92, 93,
   10, 11, 14, 15, 26, 27, 30, 31, 74, 75, 78, 79, 90, 91, 94, 95,
   32, 33, 36, 37, 48, 49, 52, 53, 96, 97, 100, 101, 112, 113, 116, 117,
   34, 35, 38, 39, 50, 51, 54, 55, 98, 99, 102, 103, 114, 115, 118, 119,
   40, 41, 44, 45, 56, 57, 60, 61, 104, 105, 108, 109, 120, 121, 124, 125,
   42, 43, 46, 47, 58, 59, 62, 63, 106, 107, 110, 111, 122, 123, 126, 127,
  128, 129, 132, 133, 144, 145, 148, 149, 192, 193, 196, 197, 208, 209, 212, 213,
  130, 131, 134, 135, 146, 147, 150, 151, 194, 195, 198, 199, 210, 211, 214, 215,
  136, 137, 140, 141, 152, 153, 156, 157, 200, 201, 204, 205, 216, 217, 220, 221,
  138, 139, 142, 143, 154, 155, 158, 159, 202, 203, 206, 207, 218, 219, 222, 223,
  160, 161, 164, 165, 176, 177, 180, 181, 224, 225, 228, 229, 240, 241, 244, 245,
  162, 163, 166, 167, 178, 179, 182, 183, 226, 227, 230, 231, 242, 243, 246, 247,
  168, 169, 172, 173, 184, 185, 188, 189, 232, 233, 236, 237, 248, 249, 252, 253,
  170, 171, 174, 175, 186, 187, 190, 191, 234, 235, 238, 239, 250, 251, 254, 255};

/* bytemask is the mask for byte interleaving - masks out the
   non-significant bit positions. This is determined by the
   depth of the node. For example, a node of depth 0 is at the root.
   Thus, there are no branches and no bits are significant.
   The bottom 4 bits (the depth) are always retained.
   Values are in octal notation. */
int bytemask[B_MAX_DEPTH + 1] = {017,
    030000000017, 036000000017, 037400000017, 037700000017,
    037760000017, 037774000017, 037777000017, 037777600017,
    037777740017, 037777770017, 037777776017, 037777777417,
    037777777717, 037777777777};

long *interleave(addr, x, y, depth, max_depth)
/* Return the interleaved code for a quadtree node at depth depth
   whose upper left hand corner has coordinates (x, y) in a tree with maximum
   depth max_depth. This function receives and returns a
   pointer to addr, which is either a long integer or (more typically)
   an array of long integers whose first integer contains the
   interleaved code. */
long *addr;
int max_depth, depth;
int x, y;
{

/* Scale x, y values to be consistent with maximum coord size */
/* and depth of tree */
x <= (B_MAX_DEPTH - max_depth);
y <= (B_MAX_DEPTH - max_depth);
```

```
/* calculate the bit interleaving of the x, y values that have now
   been appropriately shifted, and place this interleave in the address
   portion of addr. Note that the binary representations of x and y are
   being processed from right to left */

*addr = depth;
*addr |= byteval[y & 03][x & 03] << 4;
*addr |= byteval[(y >> 2) & 017][(x >> 2) & 017] << 8;
*addr |= byteval[(y >> 6) & 017][(x >> 6) & 017] << 16;
*addr |= byteval[(y >> 10) & 017][(x >> 10) & 017] << 24;
*addr &= bytemask[depth];

/* if there were unused portions of the x and y addresses then */
/* the address was too large for the depth values given. */
/* Return address built */
return (addr);
}
```

```
/* The next two arrays are used in calculating the (x, y) coordinates
   of the upper left-hand corner of a node from its bit interleaved
   address. Given an 8 bit number, the arrays return the effect of
   removing every other bit (the y bits precede the x bits). */
```

```
int xval[256] = { 0, 1, 0, 1, 2, 3, 2, 3, 0, 1, 0, 1, 2, 3, 2, 3,
                  4, 5, 4, 5, 6, 7, 6, 7, 4, 5, 4, 5, 6, 7, 6, 7,
                  0, 1, 0, 1, 2, 3, 2, 3, 0, 1, 0, 1, 2, 3, 2, 3,
                  4, 5, 4, 5, 6, 7, 6, 7, 4, 5, 4, 5, 6, 7, 6, 7,
                  8, 9, 8, 9, 10, 11, 10, 11, 8, 9, 8, 9, 10, 11, 10, 11,
                  12, 13, 12, 13, 14, 15, 14, 15, 12, 13, 12, 13, 14, 15, 14, 15,
                  8, 9, 8, 9, 10, 11, 10, 11, 8, 9, 8, 9, 10, 11, 10, 11,
                  12, 13, 12, 13, 14, 15, 14, 15, 12, 13, 12, 13, 14, 15, 14, 15,
                  0, 1, 0, 1, 2, 3, 2, 3, 0, 1, 0, 1, 2, 3, 2, 3,
                  4, 5, 4, 5, 6, 7, 6, 7, 4, 5, 4, 5, 6, 7, 6, 7,
                  0, 1, 0, 1, 2, 3, 2, 3, 0, 1, 0, 1, 2, 3, 2, 3,
                  4, 5, 4, 5, 6, 7, 6, 7, 4, 5, 4, 5, 6, 7, 6, 7,
                  8, 9, 8, 9, 10, 11, 10, 11, 8, 9, 8, 9, 10, 11, 10, 11,
                  12, 13, 12, 13, 14, 15, 14, 15, 12, 13, 12, 13, 14, 15, 14, 15,
                  8, 9, 8, 9, 10, 11, 10, 11, 8, 9, 8, 9, 10, 11, 10, 11,
                  12, 13, 12, 13, 14, 15, 14, 15, 12, 13, 12, 13, 14, 15, 14, 15};
```

```
int yval[256] = { 0, 0, 1, 1, 0, 0, 1, 1, 2, 2, 3, 3, 2, 2, 3, 3,
                  0, 0, 1, 1, 0, 0, 1, 1, 2, 2, 3, 3, 2, 2, 3, 3,
                  4, 4, 5, 5, 4, 4, 5, 5, 6, 6, 7, 7, 6, 6, 7, 7,
                  4, 4, 5, 5, 4, 4, 5, 5, 6, 6, 7, 7, 6, 6, 7, 7,
                  0, 0, 1, 1, 0, 0, 1, 1, 2, 2, 3, 3, 2, 2, 3, 3,
                  0, 0, 1, 1, 0, 0, 1, 1, 2, 2, 3, 3, 2, 2, 3, 3,
                  4, 4, 5, 5, 4, 4, 5, 5, 6, 6, 7, 7, 6, 6, 7, 7,
                  4, 4, 5, 5, 4, 4, 5, 5, 6, 6, 7, 7, 6, 6, 7, 7,
                  8, 8, 9, 9, 8, 8, 9, 9, 10, 10, 11, 11, 10, 10, 11, 11,
                  8, 8, 9, 9, 8, 8, 9, 9, 10, 10, 11, 11, 10, 10, 11, 11,
                  12, 12, 13, 13, 12, 12, 13, 13, 14, 14, 15, 15, 14, 14, 15, 15,
                  12, 12, 13, 13, 12, 12, 13, 13, 14, 14, 15, 15, 14, 14, 15, 15,
                  8, 8, 9, 9, 8, 8, 9, 9, 10, 10, 11, 11, 10, 10, 11, 11,
                  8, 8, 9, 9, 8, 8, 9, 9, 10, 10, 11, 11, 10, 10, 11, 11,
                  12, 12, 13, 13, 12, 12, 13, 13, 14, 14, 15, 15, 14, 14, 15, 15,
                  12, 12, 13, 13, 12, 12, 13, 13, 14, 14, 15, 15, 14, 14, 15, 15};
```



```
int getx(addr, max_depth)
/* Return the x coordinate of the upper left hand corner of addr for a
   tree with maximum depth max_depth. */
long *addr;
int max_depth;
{
    register x;

    x = xval[(*addr >> 4) & 017];
    x |= xval[(*addr >> 8) & 0377] << 2;
    x |= xval[(*addr >> 16) & 0377] << 6;
    x |= xval[(*addr >> 24) & 0377] << 10;
    x >>= B_MAX_DEPTH - max_depth;
    return (x);
}
```

```
int QKy(addr, max_depth)
/* Return the y coordinate of the upper left hand corner of addr for a
   tree with maximum depth max_depth. */

long *addr;
int max_depth;
{
    register y;

    y = yval[(*addr >> 4) & 017];
    y |= yval[(*addr >> 8) & 0377] << 2;
    y |= yval[(*addr >> 16) & 0377] << 6;
    y |= yval[(*addr >> 24) & 0377] << 10;
    y >>= B_MAX_DEPTH - max_depth;
    return (y);
}
```

```
int getdepth(addr)
/* Return the depth of the node.  Simply return the bottom 4 bits. */

long *addr;
{
    return(*addr & 017);
}
```

```
/*
Fast Line-Edge Intersections on a Uniform Grid
by Andrew Shapira
from "Graphics Gems", Academic Press, 1990
*/

#include "GraphicsGems.h"

#define OCTANT(f1, f2, f3, f4, f5, i1, s1, r1, r2) \
    for (f1, f2, f3, nr = 0; f4; f5) { \
        if (nr < liconst) { \
            if (i1) \
                r1(&C); \
            else \
                vertex(&C); \
        } \
        else { \
            s1; \
            if (nr -= liconst) { \
                r2(&C); \
                r1(&C); \
            } \
            else \
                vertex(&C); \
        } \
    }

find_intersections(Pptr, Qptr)
IntPoint2 *Pptr, *Qptr; /* P and Q as described in gem text */
{
    IntPoint2 P, Q; /* P and Q, dereferenced for speed */
    IntPoint2 C; /* current grid point */
    int nr; /* remainder */
    int deltax, deltay; /* Q.x - P.x, Q.y - P.y */
    int liconst; /* loop-invariant constant */

    P.x = Pptr->x;
    P.y = Pptr->y;
    Q.x = Qptr->x;
    Q.y = Qptr->y;
    deltax = Q.x - P.x;
    deltay = Q.y - P.y;

    /* for reference purposes, let theta be the angle from P to Q */

    if ((deltax >= 0) && (deltay >= 0) && (deltay < deltax))
        /* 0 <= theta < 45 */
        OCTANT(C.x = P.x + 1, C.y = P.y, liconst = deltax - deltay,
                C.x < Q.x, C.x++, nr += deltay, C.y++, up, left)
    else if ((deltax > 0) && (deltay >= 0) && (deltay >= deltax))
        /* 45 <= theta < 90 */
        OCTANT(C.y = P.y + 1, C.x = P.x, liconst = deltay - deltax,
                C.y < Q.y, C.y++, nr += deltax, C.x++, right, down)
    else if ((deltax <= 0) && (deltay >= 0) && (deltay > -deltax))
        /* 90 <= theta < 135 */
        OCTANT(C.y = P.y + 1, C.x = P.x, liconst = deltay + deltax,
                C.y < Q.y, C.y++, nr -= deltax, C.x--, left, down)
    else if ((deltax <= 0) && (deltay > 0) && (deltay <= -deltax))
        /* 135 <= theta < 180 */
        OCTANT(C.x = P.x - 1, C.y = P.y, liconst = -deltax - deltay,
```

```
        C.x > Q.x, C.x--, nr += deltay, C.y++, up, right)
else if ((deltax <= 0) && (deltay <= 0) && (deltay > deltax))
    /* 180 <= theta < 225 */
    OCTANT(C.x = P.x - 1, C.y = P.y, liconst = -deltax + deltay,
        C.x > Q.x, C.x--, nr -= deltay, C.y--, down, right)
else if ((deltax < 0) && (deltay <= 0) && (deltay <= deltax))
    /* 225 <= theta < 270 */
    OCTANT(C.y = P.y - 1, C.x = P.x, liconst = -deltay + deltax,
        C.y > Q.y, C.y--, nr -= deltax, C.x--, left, up)
else if ((deltax >= 0) && (deltay <= 0) && (-deltay > deltax))
    /* 270 <= theta < 315 */
    OCTANT(C.y = P.y - 1, C.x = P.x, liconst = -deltay - deltax,
        C.y > Q.y, C.y--, nr += deltax, C.x++, right, up)
else if ((deltax >= 0) && (deltay < 0) && (-deltay <= deltax))
    /* 315 <= theta < 360 */
    OCTANT(C.x = P.x + 1, C.y = P.y, liconst = deltax + deltay,
        C.x < Q.x, C.x++, nr -= deltay, C.y--, down, left)
else {}
    /* P = Q */
}

vertex(I)
IntPoint2    *I;
{
    /* Note: replace printf with code to process vertex, if desired */
    (void) printf("vertex at %d %d\n", I->x, I->y);
}

left(I)
IntPoint2    *I;
{
    /* Note: replace printf with code to process leftward */
    /* intersection, if desired */
    (void) printf("left from %d %d\n", I->x, I->y);
}

up(I)
IntPoint2    *I;
{
    /* Note: replace printf with code to process upward */
    /* intersection, if desired */
    (void) printf("up from %d %d\n", I->x, I->y);
}

right(I)
IntPoint2    *I;
{
    /* Note: replace printf with code to process rightward */
    /* intersection, if desired */
    (void) printf("right from %d %d\n", I->x, I->y);
}

down(I)
IntPoint2    *I;
{
    /* Note: replace printf with code to process downward */
    /* intersection, if desired */
    (void) printf("down from %d %d\n", I->x, I->y);
}
```

```

/*****
The efficiency of the following c-code can be increased by defining macros at
appropriate places. For example, the Leaf() function can be replaced by a
corresponding macros. Another way to increase the code efficiency is by passing
the address of structures instead of the structures themselves.
*****/

/*****
Supporting Data Structures
*****/

#include <stdio.h>
#include "GraphicsGems.h"

typedef struct {
    Point3    min, max;          /* extent of the primitive */

    /* definition for different primitives */

} GeomObj, *GeomObjPtr;

typedef struct {

    /* Link list of primitives */

    int    length;               /* Length of the link list */
} GeomObjList;

typedef struct {
    Point3    origin;           /* ray origin */
    Point3    direction;        /* unit vector, indicating ray direction */
} Ray;

typedef struct BinNode {
    Point3    min, max;         /* extent of node */
    GeomObjList members;        /* list of enclosed primitives */
    struct BinNode *child[2];   /* pointers to children nodes, if any */

    /* distance to the plane which subdivides the children */
    double    (*DistanceToDivisionPlane)();

    /* children near/far ordering relative to a input point */
    void      (*GetChildren)();

} BinNode, *BinNodePtr;

typedef struct {
    Point3    min, max;         /* extent of the entire bin tree */
    GeomObjList members;        /* list of all of the primitives */
    int        MaxDepth;         /* max allowed depth of the tree */
    int        MaxListLength;    /* max primitive allowed in a leaf node */
    BinNodePtr root;             /* root of the entire bin tree */
} BinTree;

/*****
Data structure for a simple stack. This is necessary for implementing an
efficient linear BSP tree walking. A Stack size of 50 means it is possible
to support a BSP tree of up to depth 49 and NO MORE!!! It should be enough for
the next decade or so.
*****/
```

```
#define STACKSIZE 50

typedef struct {
    BinNodePtr node;
    double      min, max;
} StackElem;

typedef struct {
    int      stackPtr;
    StackElem stack[STACKSIZE];
} Stack, *StackPtr;

/*****
Stack operations.
*****/

void InitStack(stack)
StackPtr stack;
{
    stack->stack[0].node = NULL;
    stack->stackPtr = 1;
}

void push(stack, node, min, max)
StackPtr    stack;
BinNodePtr  node;
double min, max;
{
    stack->stack[stack->stackPtr].node = node;
    stack->stack[stack->stackPtr].min = min;
    stack->stack[stack->stackPtr].max = max;
    (stack->stackPtr)++;
}

void pop(stack, node, min, max)
StackPtr    stack;
BinNodePtr  *node;
double      *min, *max;
{
    (stack->stackPtr)--;
    *node = stack->stack[stack->stackPtr].node;
    *min = stack->stack[stack->stackPtr].min;
    *max = stack->stack[stack->stackPtr].max;
}

/*****/
Returns the distance between origin and plane, measured along the input
direction. direction is a unit vector.

Entry:
    plane      - subdivision plane of current node
    origin     - origin of the ray
    direction  - direction of the ray, must be a unit vector

Exit:
    returns the distance between the origin and plane measured along
    the direction

Note:
    there is a function for each of the three subdivision planes
```

\*\*\*\*\*/

```
double DistanceToXPlane(plane, ray)
Point3 plane;
Ray    ray;
{
    return ( (plane.x - ray.origin.x) / ray.direction.x);
}
```

```
double DistanceToYPlane(plane, ray)
Point3 plane;
Ray    ray;
{
    return ( (plane.y - ray.origin.y) / ray.direction.y);
}
```

```
double DistanceToZPlane(plane, ray)
Point3 plane;
Ray    ray;
{
    return ( (plane.z - ray.origin.z) / ray.direction.z);
}
```

/\*  
Determines which of the half space of the two children contains origin, return  
that child as near, the other as far.

Entry:  
    currentNode - node currently working on  
    origin      - origin of the ray

Exit:  
    near - node whose half plane contains the origin  
    far  - node whose half plane does not contain the origin

Note:  
    there is a function for each of the three subdivision planes

\*\*\*\*\*/

```
void GetXChildren(currentNode, origin, near, far)
BinNodePtr currentNode, *near, *far;
Point3 origin;
{
    /* remember that child[0]->max or child[1]->min is the subdivision plane */

    if ( currentNode->child[0]->max.x >= origin.x ) {
        *near = currentNode->child[0];
        *far  = currentNode->child[1];
    } else {
        *far  = currentNode->child[0];
        *near = currentNode->child[1];
    }
}
```

```
void GetYChildren(currentNode, origin, near, far)
BinNodePtr currentNode, *near, *far;
Point3 origin;
{
    /* remember that child[0]->max or child[1]->min is the subdivision plane */
```

```
    if ( currentNode->child[0]->max.y >= origin.y ) {
        *near = currentNode->child[0];
        *far = currentNode->child[1];
    } else {
        *far = currentNode->child[0];
        *near = currentNode->child[1];
    }
}

void GetZChildren(currentNode, origin, near, far)
BinNodePtr currentNode, *near, *far;
Point3 origin;
{
    /* remember that child[0]->max or child[1]->min is the subdivision plane */

    if ( currentNode->child[0]->max.z >= origin.z ) {
        *near = currentNode->child[0];
        *far = currentNode->child[1];
    } else {
        *far = currentNode->child[0];
        *near = currentNode->child[1];
    }
}

/*****
Some miscellaneous supporting functions.
*****/

boolean Leaf(node)
BinNodePtr node;
{
    return (node->child[0] == NULL);
}

boolean PointInNode(node, pt)
BinNodePtr node;
Point3 pt;
{
    return ((pt.x >= node->min.x ) && (pt.y >= node->min.y ) &&
            (pt.z >= node->min.z ) && (pt.x <= node->max.x ) &&
            (pt.y <= node->max.y ) && (pt.z <= node->max.z ));
}

boolean GeomInNode(node, obj)
BinNodePtr node;
GeomObjPtr obj;
{
    if (node->min.x > obj->max.x || node->max.x < obj->min.x) return FALSE;
    if (node->min.y > obj->max.y || node->max.y < obj->min.y) return FALSE;
    if (node->min.z > obj->max.z || node->max.z < obj->min.z) return FALSE;
    return TRUE;
}

void PointAtDistance(ray, distance, pt)
Ray ray;
double distance;
Point3 *pt;
{
```

```
    pt->x = ray.origin.x + distance * ray.direction.x;
    pt->y = ray.origin.y + distance * ray.direction.y;
    pt->z = ray.origin.z + distance * ray.direction.z;
}

boolean RayBoxIntersect(ray, min, max, returnMin, returnMax)
Ray ray;
Point3 min, max;
double *returnMin, *returnMax;
{
    /*
     * This routine intersects the ray with the box
     * defined by min and max, returns the intersection
     * status. If ray successfully intersects the box,
     * then this routine also returns the distances
     * (from the ray origin) to the two points that the
     * ray intersects the box on.
     *
     * For example, refer to Graphics Gems I, pp. 395 (736)
     */
}

boolean RayObjIntersect(ray, objList, obj, distance)
Ray ray;
GeomObjList objList;
GeomObj *obj;
double *distance;
{
    /*
     * This routine intersects ray with all of the objects
     * in the objList and returns the closest intersection
     * distance and the interesting object, if there is one.
     */
}

/*****
 * Traverses ray through BSPTree and intersects ray with all of the objects along
 * the way. Returns the closest intersection distance and the intersecting object
 * if there is one.
 */

Entry:
    ray      - the ray being traced
    BSPTree - the BSP tree enclosing the entire environment

Exit:
    obj      - the first object that intersects the ray
    distance - distance to the intersecting object
*****/
boolean RayTreeIntersect(ray, BSPTree, obj, distance)
Ray ray;
BinTree BSPTree;
GeomObj *obj;
double *distance;
{
    StackPtr stack;
    BinNodePtr currentNode, nearChild, farChild;
    double dist, min, max;
    Point3 p;
```



```
/* test if the whole BSP tree is missed by the input ray */

if (!RayBoxIntersect(ray, BSPTree.min, BSPTree.max, &min, &max))
    return FALSE;

stack = (StackPtr)malloc(sizeof(Stack));
InitStack(stack);

currentNode = BSPTree.root;

while (currentNode != NULL) {
    while ( !(Leaf(currentNode)) ) {
        dist = currentNode->DistanceToDivisionPlane(
            currentNode->child[0]->max, ray);
        currentNode->GetChildren(
            currentNode, ray.origin, nearChild, farChild);

        if ( (dist>max) || (dist<0) ) {
            currentNode = nearChild;
        } else if (dist<min) {
            currentNode = farChild;
        } else {
            push(stack, farChild, dist, max);
            currentNode = nearChild;
            max = dist;
        }
    }

    if ( RayObjIntersect(ray, currentNode->members, obj, distance) ) {
        PointAtDistance(ray, distance, &p);
        if (PointInNode(currentNode, p))
            return TRUE;
    }
    pop(stack, &currentNode, &min, &max);
}
return FALSE;
}

/*****
Builds the BSP tree by subdividing along the center of x, y, or z bounds, one
each time this function is called. This function calls itself recursively until
either the tree is deeper than MaxDepth or all of the tree leaves contains less
than MaxListLength of objects.

Entry:
    node          - node currently working on
    depth         - current tree depth
    MaxDepth      - Max allowed tree depth
    MaxListLength - Max allowed object list length of a leave node
    axis         - subdivision axis for the children of node
                  (0-x, 1-y, 2-z)
*****/
void Subdivide(node, depth, MaxDepth, MaxListLength, axis)
BinNodePtr node;
int depth,          /* current tree depth */
    MaxDepth,      /* the specified max allowed depth */
    MaxListLength, /* the specified max allowed list length */
```

```
axis;          /* current subdivision plane, 1-x, 2-y, 3-z */
{
    int i,      nextAxis;
    GeomObjPtr ObjPtr;

    node->child[0] = node->child[1] = NULL;

    if ((node->members.length > MaxListLength) && (depth < MaxDepth)) {

        for (i = 0; i < 2; i++) {

            node->child[i] = (BinNodePtr)malloc(sizeof(BinNode));
            node->child[i]->min.x = node->min.x;
            node->child[i]->min.y = node->min.y;
            node->child[i]->min.z = node->min.z;
            node->child[i]->max.x = node->max.x;
            node->child[i]->max.y = node->max.y;
            node->child[i]->max.z = node->max.z;

            if (axis == 1) {

                /* current subdivision plane is x */
                node->child[i]->min.x =
                    node->min.x + 0.5 * i * (node->max.x - node->min.x);
                node->child[i]->max.x =
                    node->min.x + 0.5 * (i+1) * (node->max.x - node->min.x);

                /* child subdivision plane will be y */
                nextAxis = 2;
                node->child[i]->DistanceToDivisionPlane = DistanceToYPlane;
                node->child[i]->GetChildren = GetYChildren;

            } else if (axis == 2) {

                /* current subdivision plane is y */

                node->child[i]->min.y =
                    node->min.y + 0.5 * i * (node->max.y - node->min.y);
                node->child[i]->max.y =
                    node->min.y + 0.5 * (i+1) * (node->max.y - node->min.y);

                /* child subdivision plane will be z */
                nextAxis = 3;
                node->child[i]->DistanceToDivisionPlane = DistanceToZPlane;
                node->child[i]->GetChildren = GetZChildren;

            } else {

                /* current subdivision plane is z */
                node->child[i]->min.z =
                    node->min.z + 0.5 * i * (node->max.z - node->min.z);
                node->child[i]->max.z =
                    node->min.z + 0.5 * (i+1) * (node->max.z - node->min.z);

                /* child subdivision plane will be x */
                nextAxis = 1;
                node->child[i]->DistanceToDivisionPlane = DistanceToXPlane;
                node->child[i]->GetChildren = GetXChildren;

            }

        }

        ObjPtr = FirstOfLinkList(node->members);
    }
}
```

```
    while (ObjPtr != NULL) {
        if (GeomInNode(node->child[i], ObjPtr))
            AddToLinkList(node->child[i]->members, ObjPtr);
        ObjPtr = NextOfLinkList(node->members);
    }
    Subdivide(node->child[i], depth+1, MaxDepth, MaxListLength, nextAxis);
}
```

```
}
}
```

```

/*****
Initialize and start the building of BSPTree.
*****/
```

Entry:

    BSPTree           - The BSPTree enclosing the entire scene

```
*****/
```

```
void InitBinTree(BSPTree)
```

```
BinTree *BSPTree;
```

```
{
```

```
    CalculateTheExtentOfTheBinTree(&(BSPTree->min), &(BSPTree->max));
```

```
    /* BSPTree->members = ObjectsWithinTheExtent(BSPTree->min, BSPTree->max); */
```

```
    BSPTree->MaxDepth = GetMaxAllowedDepth();
```

```
    BSPTree->MaxListLength = GetMaxAllowedListLength();
```

```
/* Start building the BSPTree by subdividing along the x axis first */
```

```
    BSPTree->root = (BinNodePtr)malloc(sizeof(BinNode));
```

```
    BSPTree->root->min = BSPTree->min;
```

```
    BSPTree->root->max = BSPTree->max;
```

```
    BSPTree->root->DistanceToDivisionPlane = DistanceToXPlane;
```

```
    BSPTree->root->GetChildren = GetXChildren;
```

```
    DuplicateLinkList(BSPTree->root->members, BSPTree->members);
```

```
    Subdivide(BSPTree->root, 0, BSPTree->MaxDepth, BSPTree->MaxListLength, 1);
```

```
}
```

```
/* urot.c */
/* Generates a uniform random rotation */
/* Ken Shoemake, September 1991 */

#include <stdlib.h>
#include <math.h>
#include "GraphicsGems.h"

/* Define an INT32 value to be a 32 bit signed integer */
typedef int INT32;

typedef struct {float x, y, z, w;} Quat;
enum QuatPart {X, Y, Z, W, QuatLen, V=0};

/* * * * * * Utility for quaternion conversion * * * * * */

/** Qt_ToMatrix
 * Construct rotation matrix from quaternion (unit or not).
 * Assumes matrix is used to multiply row vector on the right:
 * vnew = vold mat. Works correctly for right-handed coordinate system
 * and right-handed rotations. For column vectors or for left-handed
 * coordinate systems, transpose the matrix.
 */
void Qt_ToMatrix(Quat q, Matrix3 *out)
{
    double norm = q.x*q.x + q.y*q.y + q.z*q.z + q.w*q.w;
    double s = (norm > 0.0) ? 2.0/norm : 0.0;
    double xs = q.x*s,      ys = q.y*s,      zs = q.z*s;
    double wx = q.w*xs,     wy = q.w*ys,     wz = q.w*zs,
           xx = q.x*xs,     xy = q.x*ys,     xz = q.x*zs,
           yy = q.y*ys,     yz = q.y*zs,     zz = q.z*zs;
    double (*mat)[3] = out->element;
    mat[X][X] = 1.0 - (yy + zz); mat[X][Y] = xy + wz; mat[X][Z] = xz - wy;
    mat[Y][X] = xy - wz; mat[Y][Y] = 1.0 - (xx + zz); mat[Y][Z] = yz + wx;
    mat[Z][X] = xz + wy; mat[Z][Y] = yz - wx; mat[Z][Z] = 1.0 - (xx + yy);
} /* Qt_ToMatrix */

/* * * * * * How to do it using gaussians * * * * * */

/** Qt_RandomG
 * Generate uniform random unit quaternion from random seed.
 */
Quat Qt_RandomG(INT32 *argseed)
{
    /* This algorithm generates a gaussian deviate for each coordinate, so
     * the total effect is to generate a symmetric 4-D gaussian distribution,
     * by separability. Projecting onto the surface of the hypersphere gives
     * a uniform distribution.
     */
    Quat q;
    /* uurand generates doubles uniformly distributed between -1 and +1 */
    /* This linear congruential generator is inline to exploit signed ints */
    register INT32 seed = *argseed;
#define uurand() ((seed = (seed+1)*69069)/2147483648.0)
    register double x = uurand(), y = uurand();
    register double z = uurand(), w = uurand();
    register double s1, s2;
    double num1, num2, root1, root2, r;
    while ((s1 = x*x+y*y) > 1.0) {x = uurand(); y = uurand();}
    while ((s2 = z*z+w*w) > 1.0) {z = uurand(); w = uurand();}
```

```
/* Now the point (x,y) is uniformly distributed in the unit disk */
/* So is the point (z,w), independently */
num1 = -2*log(s1); num2 = -2*log(s2);
/* Now x*sqrt(num1/s1) is gaussian distributed, using polar method */
/* Similarly for y, z, and w, and all are independent */
r = num1 + num2; /* Sum of squares of four gaussians */
root1 = sqrt((num1/s1)/r); root2 = sqrt((num2/s2)/r);
/* Normalizing onto unit sphere gives uniform unit quaternion */
q.x = x*root1; q.y = y*root1; q.z = z*root2; q.w = w*root2;
*argseed = seed;
```

```
#undef uurand
```

```
return (q);
```

```
} /* Qt_RandomG */
```

```
/** M3_RandomRotG
```

```
* Generate uniform random rotation matrix from random seed.
```

```
*/
```

```
void M3_RandomRotG(INT32 *seed, Matrix3 *m)
```

```
{
```

```
Qt_ToMatrix(Qt_RandomG(seed), m);
```

```
} /* M3_RandomRotG */
```

```
/* * * * * * How to do it using subgroup algorithm * * * * * */
```

```
/** Qt_Random
```

```
* Generate uniform random unit quaternion from uniform deviates.
```

```
* Each x[i] should vary between 0 and 1.
```

```
*/
```

```
Quat Qt_Random(double x[3])
```

```
{
```

```
/* The subgroup algorithm can be condensed to this efficient form.
```

```
* Use rotations around z as a subgroup, with coset representatives
```

```
* the rotations pointing the z axis in different directions.
```

```
*/
```

```
Quat q;
```

```
register double r1 = sqrt(1.0 - x[0]), r2 = sqrt(x[0]);
```

```
register double t1 = PITIMES2*x[1], t2 = PITIMES2*x[2];
```

```
register double c1 = cos(t1), s1 = sin(t1);
```

```
register double c2 = cos(t2), s2 = sin(t2);
```

```
q.x = s1*r1; q.y = c1*r1; q.z = s2*r2; q.w = c2*r2;
```

```
return (q);
```

```
} /* Qt_Random */
```

```
/** M3_RandomRot
```

```
* Generate uniform random rotation matrix from uniform deviates.
```

```
*/
```

```
void M3_RandomRot(double x[3], Matrix3 *m)
```

```
{
```













```
Qt_ToMatrix(Qt_Random(x), m);
```

```
} /* M3_RandomRot */
```

```
/* End of urot.c */
```

# Index of

## /pubs/tog/GraphicsGems/gemsiv/arcball/

Name	Last modified	Size	Description
 <a href="#">_Parent Directory</a>			
 <a href="#">_Ball.c</a>	29-Jun-00 08:19	5K	
 <a href="#">_Ball.h</a>	29-Jun-00 08:19	1K	
 <a href="#">_BallAux.c</a>	29-Jun-00 08:19	3K	
 <a href="#">_BallAux.h</a>	29-Jun-00 08:19	1K	
 <a href="#">_BallMath.c</a>	29-Jun-00 08:19	2K	
 <a href="#">_BallMath.h</a>	29-Jun-00 08:19	1K	
 <a href="#">_Body.c</a>	29-Jun-00 08:19	2K	
 <a href="#">_Body.h</a>	29-Jun-00 08:19	1K	
 <a href="#">_Demo.c</a>	29-Jun-00 08:19	3K	
 <a href="#">_Makefile</a>	29-Jun-00 08:19	1K	
 <a href="#">_README</a>	29-Jun-00 08:19	1K	

```
/* ***** Ball.c ***** */
/* Ken Shoemake, 1993 */
#include <gl/gl.h>
#include "Ball.h"
#include "BallMath.h"

#define LG_NSEGS 4
#define NSEGS (1<<LG_NSEGS)
#define RIMCOLOR()    RGBcolor(255, 255, 255)
#define FARCOLOR()    RGBcolor(195, 127, 31)
#define NEARCOLOR()   RGBcolor(255, 255, 63)
#define DRAGCOLOR()   RGBcolor(127, 255, 255)
#define RESCOLOR()    RGBcolor(195, 31, 31)

HMatrix mId = {{1,0,0,0},{0,1,0,0},{0,0,1,0},{0,0,0,1}};
float otherAxis[][4] = {{-0.48, 0.80, 0.36, 1}};

/* Establish reasonable initial values for controller. */
void Ball_Init(BallData *ball)
{
    int i;
    ball->center = qOne;
    ball->radius = 1.0;
    ball->vDown = ball->vNow = qOne;
    ball->qDown = ball->qNow = qOne;
    for (i=15; i>=0; i--)
        ((float *)ball->mNow)[i] = ((float *)ball->mDown)[i] = ((float *)mId)[i];
    ball->showResult = ball->dragging = FALSE;
    ball->axisSet = NoAxes;
    ball->sets[CameraAxes] = mId[X]; ball->setSizes[CameraAxes] = 3;
    ball->sets[BodyAxes] = ball->mDown[X]; ball->setSizes[BodyAxes] = 3;
    ball->sets[OtherAxes] = otherAxis[X]; ball->setSizes[OtherAxes] = 1;
}

/* Set the center and size of the controller. */
void Ball_Place(BallData *ball, HVect center, double radius)
{
    ball->center = center;
    ball->radius = radius;
}

/* Incorporate new mouse position. */
void Ball_Mouse(BallData *ball, HVect vNow)
{
    ball->vNow = vNow;
}

/* Choose a constraint set, or none. */
void Ball_UseSet(BallData *ball, AxisSet axisSet)
{
    if (!ball->dragging) ball->axisSet = axisSet;
}

/* Begin drawing arc for all drags combined. */
void Ball_ShowResult(BallData *ball)
{
    ball->showResult = TRUE;
}

/* Stop drawing arc for all drags combined. */
void Ball_HideResult(BallData *ball)
```

```
{
    ball->showResult = FALSE;
}

/* Using vDown, vNow, dragging, and axisSet, compute rotation etc. */
void Ball_Update(BallData *ball)
{
    int i, setSize = ball->setSizes[ball->axisSet];
    HVect *set = (HVect *) (ball->sets[ball->axisSet]);
    ball->vFrom = MouseOnSphere(ball->vDown, ball->center, ball->radius);
    ball->vTo = MouseOnSphere(ball->vNow, ball->center, ball->radius);
    if (ball->dragging) {
        if (ball->axisSet != NoAxes) {
            ball->vFrom = ConstrainToAxis(ball->vFrom, set[ball->axisIndex]);
            ball->vTo = ConstrainToAxis(ball->vTo, set[ball->axisIndex]);
        }
        ball->qDrag = Qt_FromBallPoints(ball->vFrom, ball->vTo);
        ball->qNow = Qt_Mul(ball->qDrag, ball->qDown);
    } else {
        if (ball->axisSet != NoAxes) {
            ball->axisIndex = NearestConstraintAxis(ball->vTo, set, setSize);
        }
    }
    Qt_ToBallPoints(ball->qDown, &ball->vrFrom, &ball->vrTo);
    Qt_ToMatrix(Qt_Conj(ball->qNow), ball->mNow); /* Gives transpose for GL. */
}

/* Return rotation matrix defined by controller use. */
void Ball_Value(BallData *ball, HMatrix mNow)
{
    int i;
    for (i=15; i>=0; i--) ((float *)mNow)[i] = ((float *)ball->mNow)[i];
}

/* Begin drag sequence. */
void Ball_BeginDrag(BallData *ball)
{
    ball->dragging = TRUE;
    ball->vDown = ball->vNow;
}

/* Stop drag sequence. */
void Ball_EndDrag(BallData *ball)
{
    int i;
    ball->dragging = FALSE;
    ball->qDown = ball->qNow;
    for (i=15; i>=0; i--)
        ((float *)ball->mDown)[i] = ((float *)ball->mNow)[i];
}

/* Draw the controller with all its arcs. */
void Ball_Draw(BallData *ball)
{
    float r = ball->radius;
    pushmatrix();
    loadmatrix(mId);
    ortho2(-1.0, 1.0, -1.0, 1.0);
    RIMCOLOR();
    scale(r, r, r);
}
```



```
    circ(0.0, 0.0, 1.0);
    Ball_DrawResultArc(ball);
    Ball_DrawConstraints(ball);
    Ball_DrawDragArc(ball);
    popmatrix();
}

/* Draw an arc defined by its ends. */
void DrawAnyArc(HVect vFrom, HVect vTo)
{
    int i;
    HVect pts[NSEGS+1];
    double dot;
    pts[0] = vFrom;
    pts[1] = pts[NSEGS] = vTo;
    for (i=0; i<LG_NSEGS; i++) pts[1] = V3_Bisect(pts[0], pts[1]);
    dot = 2.0*V3_Dot(pts[0], pts[1]);
    for (i=2; i<NSEGS; i++) {
        pts[i] = V3_Sub(V3_Scale(pts[i-1], dot), pts[i-2]);
    }
    bgnline();
    for (i=0; i<=NSEGS; i++)
        v3f((float *)&pts[i]);
    endlne();
}

/* Draw the arc of a semi-circle defined by its axis. */
void DrawHalfArc(HVect n)
{
    HVect p, m;
    p.z = 0;
    if (n.z != 1.0) {
        p.x = n.y; p.y = -n.x;
        p = V3_Unit(p);
    } else {
        p.x = 0; p.y = 1;
    }
    m = V3_Cross(p, n);
    DrawAnyArc(p, m);
    DrawAnyArc(m, V3_Negate(p));
}

/* Draw all constraint arcs. */
void Ball_DrawConstraints(BallData *ball)
{
    ConstraintSet set;
    HVect axis;
    int j, axisI, setSize = ball->setSizes[ball->axisSet];
    if (ball->axisSet==NoAxes) return;
    set = ball->sets[ball->axisSet];
    for (axisI=0; axisI<setSize; axisI++) {
        if (ball->axisIndex!=axisI) {
            if (ball->dragging) continue;
            FARCOLOR();
        } else NEARCOLOR();
        axis = *(HVect *)&set[4*axisI];
        if (axis.z==1.0) {
            circ(0.0, 0.0, 1.0);
        } else {
            DrawHalfArc(axis);
        }
    }
}
```

```
    }  
}  
  
/* Draw "rubber band" arc during dragging. */  
void Ball_DrawDragArc(BallData *ball)  
{  
    DRAGCOLOR();  
    if (ball->dragging) DrawAnyArc(ball->vFrom, ball->vTo);  
}  
  
/* Draw arc for result of all drags. */  
void Ball_DrawResultArc(BallData *ball)  
{  
    RESCOLOR();  
    if (ball->showResult) DrawAnyArc(ball->vrFrom, ball->vrTo);  
}
```

```
/* ***** Ball.h ***** */
#ifndef _H_Ball
#define _H_Ball
#include "BallAux.h"

typedef enum AxisSet{NoAxes, CameraAxes, BodyAxes, OtherAxes, NSets} AxisSet;
typedef float *ConstraintSet;
typedef struct {
    HVect center;
    double radius;
    Quat qNow, qDown, qDrag;
    HVect vNow, vDown, vFrom, vTo, vrFrom, vrTo;
    HMatrix mNow, mDown;
    Bool showResult, dragging;
    ConstraintSet sets[NSets];
    int setSizes[NSets];
    AxisSet axisSet;
    int axisIndex;
} BallData;

/* Public routines */
void Ball_Init(BallData *ball);
void Ball_Place(BallData *ball, HVect center, double radius);
void Ball_Mouse(BallData *ball, HVect vNow);
void Ball_UseSet(BallData *ball, AxisSet axisSet);
void Ball_ShowResult(BallData *ball);
void Ball_HideResult(BallData *ball);
void Ball_Update(BallData *ball);
void Ball_Value(BallData *ball, HMatrix mNow);
void Ball_BeginDrag(BallData *ball);
void Ball_EndDrag(BallData *ball);
void Ball_Draw(BallData *ball);
/* Private routines */
void DrawAnyArc(HVect vFrom, HVect vTo);
void DrawHalfArc(HVect n);
void Ball_DrawConstraints(BallData *ball);
void Ball_DrawDragArc(BallData *ball);
void Ball_DrawResultArc(BallData *ball);
#endif
```

```
/* ***** BallAux.c ***** */
#include <math.h>
#include "BallAux.h"

Quat qOne = {0, 0, 0, 1};

/* Return quaternion product qL * qR. Note: order is important!
 * To combine rotations, use the product Mul(qSecond, qFirst),
 * which gives the effect of rotating by qFirst then qSecond. */
Quat Qt_Mul(Quat qL, Quat qR)
{
    Quat qq;
    qq.w = qL.w*qR.w - qL.x*qR.x - qL.y*qR.y - qL.z*qR.z;
    qq.x = qL.w*qR.x + qL.x*qR.w + qL.y*qR.z - qL.z*qR.y;
    qq.y = qL.w*qR.y + qL.y*qR.w + qL.z*qR.x - qL.x*qR.z;
    qq.z = qL.w*qR.z + qL.z*qR.w + qL.x*qR.y - qL.y*qR.x;
    return (qq);
}

/* Construct rotation matrix from (possibly non-unit) quaternion.
 * Assumes matrix is used to multiply column vector on the left:
 * vnew = mat vold. Works correctly for right-handed coordinate system
 * and right-handed rotations. */
HMatrix *Qt_ToMatrix(Quat q, HMatrix out)
{
    double Nq = q.x*q.x + q.y*q.y + q.z*q.z + q.w*q.w;
    double s = (Nq > 0.0) ? (2.0 / Nq) : 0.0;
    double xs = q.x*s,      ys = q.y*s,      zs = q.z*s;
    double wx = q.w*xs,     wy = q.w*ys,     wz = q.w*zs;
    double xx = q.x*xs,     xy = q.x*ys,     xz = q.x*zs;
    double yy = q.y*ys,     yz = q.y*zs,     zz = q.z*zs;
    out[X][X] = 1.0 - (yy + zz); out[Y][X] = xy + wz; out[Z][X] = xz - wy;
    out[X][Y] = xy - wz; out[Y][Y] = 1.0 - (xx + zz); out[Z][Y] = yz + wx;
    out[X][Z] = xz + wy; out[Y][Z] = yz - wx; out[Z][Z] = 1.0 - (xx + yy);
    out[X][W] = out[Y][W] = out[Z][W] = out[W][X] = out[W][Y] = out[W][Z] = 0.0;
    out[W][W] = 1.0;
    return ((HMatrix *)&out);
}

/* Return conjugate of quaternion. */
Quat Qt_Conj(Quat q)
{
    Quat qq;
    qq.x = -q.x; qq.y = -q.y; qq.z = -q.z; qq.w = q.w;
    return (qq);
}

/* Return vector formed from components */
HVect V3_(float x, float y, float z)
{
    HVect v;
    v.x = x; v.y = y; v.z = z; v.w = 0;
    return (v);
}

/* Return norm of v, defined as sum of squares of components */
float V3_Norm(HVect v)
{
    return ( v.x*v.x + v.y*v.y + v.z*v.z );
}
```

```
/* Return unit magnitude vector in direction of v */
HVect V3_Unit(HVect v)
{
    static HVect u = {0, 0, 0, 0};
    float vlen = sqrt(V3_Norm(v));
    if (vlen != 0.0) {
        u.x = v.x/vlen; u.y = v.y/vlen; u.z = v.z/vlen;
    }
    return (u);
}

/* Return version of v scaled by s */
HVect V3_Scale(HVect v, float s)
{
    HVect u;
    u.x = s*v.x; u.y = s*v.y; u.z = s*v.z; u.w = v.w;
    return (u);
}

/* Return negative of v */
HVect V3_Negate(HVect v)
{
    static HVect u = {0, 0, 0, 0};
    u.x = -v.x; u.y = -v.y; u.z = -v.z;
    return (u);
}

/* Return sum of v1 and v2 */
HVect V3_Add(HVect v1, HVect v2)
{
    static HVect v = {0, 0, 0, 0};
    v.x = v1.x+v2.x; v.y = v1.y+v2.y; v.z = v1.z+v2.z;
    return (v);
}

/* Return difference of v1 minus v2 */
HVect V3_Sub(HVect v1, HVect v2)
{
    static HVect v = {0, 0, 0, 0};
    v.x = v1.x-v2.x; v.y = v1.y-v2.y; v.z = v1.z-v2.z;
    return (v);
}

/* Halve arc between unit vectors v0 and v1. */
HVect V3_Bisect(HVect v0, HVect v1)
{
    HVect v = {0, 0, 0, 0};
    float Nv;
    v = V3_Add(v0, v1);
    Nv = V3_Norm(v);
    if (Nv < 1.0e-5) {
        v = V3_(0, 0, 1);
    } else {
        v = V3_Scale(v, 1/sqrt(Nv));
    }
    return (v);
}

/* Return dot product of v1 and v2 */
float V3_Dot(HVect v1, HVect v2)
```

```
{  
    return (v1.x*v2.x + v1.y*v2.y + v1.z*v2.z);  
}
```

```
/* Return cross product, v1 x v2 */  
HVect V3_Cross(HVect v1, HVect v2)  
{  
    static HVect v = {0, 0, 0, 0};  
    v.x = v1.y*v2.z-v1.z*v2.y;  
    v.y = v1.z*v2.x-v1.x*v2.z;  
    v.z = v1.x*v2.y-v1.y*v2.x;  
    return (v);  
}
```

```
/****** BallAux.h - Vector and quaternion routines for Arcball. *****/
#ifndef _H_BallAux
#define _H_BallAux

typedef int Bool;
typedef struct {float x, y, z, w;} Quat;
enum QuatPart {X, Y, Z, W, QuatLen};
typedef Quat HVect;
typedef float HMatrix[QuatLen][QuatLen];

extern Quat qOne;
HMatrix *Qt_ToMatrix(Quat q, HMatrix out);
Quat Qt_Conj(Quat q);
Quat Qt_Mul(Quat qL, Quat qR);
HVect V3_(float x, float y, float z);
float V3_Norm(HVect v);
HVect V3_Unit(HVect v);
HVect V3_Scale(HVect v, float s);
HVect V3_Negate(HVect v);
HVect V3_Sub(HVect v1, HVect v2);
float V3_Dot(HVect v1, HVect v2);
HVect V3_Cross(HVect v1, HVect v2);
HVect V3_Bisect(HVect v0, HVect v1);
#endif
```

```
/**** BallMath.c - Essential routines for ArcBall.  ****/
#include <math.h>
#include "BallMath.h"
#include "BallAux.h"

/* Convert window coordinates to sphere coordinates. */
HVect MouseOnSphere(HVect mouse, HVect ballCenter, double ballRadius)
{
    HVect ballMouse;
    register double mag;
    ballMouse.x = (mouse.x - ballCenter.x) / ballRadius;
    ballMouse.y = (mouse.y - ballCenter.y) / ballRadius;
    mag = ballMouse.x*ballMouse.x + ballMouse.y*ballMouse.y;
    if (mag > 1.0) {
        register double scale = 1.0/sqrt(mag);
        ballMouse.x *= scale; ballMouse.y *= scale;
        ballMouse.z = 0.0;
    } else {
        ballMouse.z = sqrt(1 - mag);
    }
    ballMouse.w = 0.0;
    return (ballMouse);
}

/* Construct a unit quaternion from two points on unit sphere */
Quat Qt_FromBallPoints(HVect from, HVect to)
{
    Quat qu;
    qu.x = from.y*to.z - from.z*to.y;
    qu.y = from.z*to.x - from.x*to.z;
    qu.z = from.x*to.y - from.y*to.x;
    qu.w = from.x*to.x + from.y*to.y + from.z*to.z;
    return (qu);
}

/* Convert a unit quaternion to two points on unit sphere */
void Qt_ToBallPoints(Quat q, HVect *arcFrom, HVect *arcTo)
{
    double s;
    s = sqrt(q.x*q.x + q.y*q.y);
    if (s == 0.0) {
        *arcFrom = V3_(0.0, 1.0, 0.0);
    } else {
        *arcFrom = V3_(-q.y/s, q.x/s, 0.0);
    }
    arcTo->x = q.w*arcFrom->x - q.z*arcFrom->y;
    arcTo->y = q.w*arcFrom->y + q.z*arcFrom->x;
    arcTo->z = q.x*arcFrom->y - q.y*arcFrom->x;
    if (q.w < 0.0) *arcFrom = V3_(-arcFrom->x, -arcFrom->y, 0.0);
}

/* Force sphere point to be perpendicular to axis. */
HVect ConstrainToAxis(HVect loose, HVect axis)
{
    HVect onPlane;
    register float norm;
    onPlane = V3_Sub(loose, V3_Scale(axis, V3_Dot(axis, loose)));
    norm = V3_Norm(onPlane);
    if (norm > 0.0) {
        if (onPlane.z < 0.0) onPlane = V3_Negate(onPlane);
        return ( V3_Scale(onPlane, 1/sqrt(norm)) );
    }
}
```



```
    } /* else drop through */
    if (axis.z == 1) {
        onPlane = V3_(1.0, 0.0, 0.0);
    } else {
        onPlane = V3_Unit(V3_(-axis.y, axis.x, 0.0));
    }
    return (onPlane);
}

/* Find the index of nearest arc of axis set. */
int NearestConstraintAxis(HVect loose, HVect *axes, int nAxes)
{
    HVect onPlane;
    register float max, dot;
    register int i, nearest;
    max = -1; nearest = 0;
    for (i=0; i<nAxes; i++) {
        onPlane = ConstrainToAxis(loose, axes[i]);
        dot = V3_Dot(onPlane, loose);
        if (dot>max) {
            max = dot; nearest = i;
        }
    }
    return (nearest);
}
```

```
/****** BallMath.h - Essential routines for Arcball.  *****/
#ifndef _H_BallMath
#define _H_BallMath
#include "BallAux.h"

HVect MouseOnSphere(HVect mouse, HVect ballCenter, double ballRadius);
HVect ConstrainToAxis(HVect loose, HVect axis);
int NearestConstraintAxis(HVect loose, HVect *axes, int nAxes);
Quat Qt_FromBallPoints(HVect from, HVect to);
void Qt_ToBallPoints(Quat q, HVect *arcFrom, HVect *arcTo);
#endif
```

```
/* ***** Body.c ***** */
#include <gl/gl.h>
#include "Body.h"

enum QuatPart {X, Y, Z, W};
int bodyNPoints = 8;
int bodyNFaces = 7;

float theBodyRadius = 3.0;
float thePoints[][4] = {
    { 3.0, 0.0, 0.0, 1},
    {-1.0, 1.0, 0.0, 1},
    {-1.0, -1.0, 0.0, 1},
    {-0.75, 0.0, -0.25, 1},
    { 1.0, 0.0, 0.0, 1},
    {-0.75, 0.0, 0.75, 1},
    {-0.5, -0.125, 0.0, 1},
    {-0.5, 0.125, 0.0, 1}
};
int theFaceVertices[][4] = {
    {3, 0, 1, 2},
    {3, 4, 5, 6},
    {3, 4, 7, 5},
    {3, 5, 7, 6},
    {3, 0, 2, 3},
    {3, 0, 3, 1},
    {3, 1, 3, 2},
};
float theFaceNormals[][4] = {
    {0., 0., 1., 0},
    {0.08152896377979659767, -0.978347565357559172, 0.1902342488195253946, 0},
    {0.08152896377979659767, 0.978347565357559172, 0.1902342488195253946, 0},
    {-0.9486832980505137996, 0., -0.3162277660168379332, 0},
    {0.06428243465332250222, -0.2571297386132900089, -0.9642365197998375334, 0},
    {0.06428243465332250222, 0.2571297386132900089, -0.9642365197998375334, 0},
    {-0.7071067811865475244, 0., -0.7071067811865475244, 0},
};
short theFaceColors[][3] = {
    {102, 204, 255},
    { 0, 153, 204},
    { 0, 153, 204},
    {204, 51, 157},
    { 51, 102, 157},
    { 51, 102, 157},
    {102, 102, 172},
};

/* Transform body normals, draw front */
void drawbody(Matrix Rot)
{
    double bodyScale = 1.0/theBodyRadius;
    register int i, j, k, n;

    pushmatrix();
    scale(bodyScale, bodyScale, bodyScale);
    for (j=0; j<bodyNFaces; j++) {
        double dot = Rot[X][Z]*theFaceNormals[j][X]
                    +Rot[Y][Z]*theFaceNormals[j][Y]
                    +Rot[Z][Z]*theFaceNormals[j][Z];
        if (dot>0.0) { /* Front-facing polygon, so draw it */
            short shadedColor[3];

```

```
        dot += 0.4; if (dot>1.0) dot = 1.0;
        shadedColor[0] = dot*theFaceColors[j][0];
        shadedColor[1] = dot*theFaceColors[j][1];
        shadedColor[2] = dot*theFaceColors[j][2];
        n = theFaceVertices[j][0];
        RGBcolor(shadedColor[0], shadedColor[1], shadedColor[2]);
        bgnpolygon();
        for (k=1; k<=n; k++) {
            i = theFaceVertices[j][k];
            v4f(thePoints[i]);
        }
        endpolygon();
    }
}
popmatrix();
}
```

```
/**** Body.h ****/  
#ifndef _H_Body  
#define _H_Body  
#include <gl/gl.h>  
void drawbody(Matrix Rot);  
#endif
```

```
/****** Demo.c *****/
/* Ken Shoemake, 1993 */
#include <gl/gl.h>
#include <gl/device.h>

#include "BallAux.h"
#include "Body.h"
#include "Ball.h"

typedef struct {long x, y;} Place;

#define RADIUS      (0.75)
#define CNTRLDN     1
#define SHIFTDN     2

void main(void)
{
    int gid;
    short active;      /* TRUE if window is attached */
    Device dev;
    short val;
    Place winsize, winorig;
    Place mouseNow, mouseDown;
    int keysDown;
    HVect vNow;
    BallData ball;

    keepaspect(1, 1);
    prefposition(50, 950, 50, 950);
    gid = winopen("Arcball Demo");
    doublebuffer();
    RGBmode();
    gconfig();
    qdevice(WINSHUT);
    qdevice(MOUSEX); qdevice(MOUSEY); qdevice(LEFTMOUSE);
    qdevice(LEFTCTRLKEY); qdevice(RIGHTCTRLKEY);
    qdevice(LEFTSHIFTKEY); qdevice(RIGHTSHIFTKEY);
    qdevice(CAPSLCKKEY);
    /* perspective(400, 1.0, 0.001, 100000.0); */
    ortho(-1.0, 1.0, -1.0, 1.0, 0.001, 100000.0);
    translate(0.0, 0.0, -3.0);
    active = 0;

    getsize(&winsize.x, &winsize.y);
    getorigin(&winorig.x, &winorig.y);
    keysDown = 0;
    Ball_Init(&ball);
    Ball_Place(&ball, qOne, RADIUS);

    while (TRUE) {
        while (qtest()) {          /* process queued events */
            dev = qread(&val);
            switch (dev) {
                case WINSHUT:      /* exit program */
                    gexit();
                    exit(0);
                    break;
                case INPUTCHANGE:
                    active = val;
                    break;
                case REDRAW:

```

```
        reshapeviewport();
        getsize(&winSize.x, &winSize.y);
        getorigin(&winorig.x, &winorig.y);
        break;
    case MOUSEX:
        mouseNow.x = val;
        vNow.x = 2.0*(mouseNow.x - winorig.x)/winSize.x - 1.0;
        break;
    case MOUSEY:
        mouseNow.y = val;
        vNow.y = 2.0*(mouseNow.y - winorig.y)/winSize.y - 1.0;
        break;
    case LEFTMOUSE:
        if (val) Ball_BeginDrag(&ball);
        else    Ball_EndDrag(&ball);
        break;
    case LEFTCTRLKEY: case RIGHTCTRLKEY:
        keysDown = (keysDown&~CNTRLDN)|(val? CNTRLDN : 0);
        break;
    case LEFTSHIFTKEY: case RIGHTSHIFTKEY:
        keysDown = (keysDown&~SHIFTDN)|(val? SHIFTDN : 0);
        break;
    case CAPSLOCKKEY:
        if (val) Ball_ShowResult(&ball);
        else    Ball_HideResult(&ball);
        break;
    default:
        break;
}
/* end of switch */
Ball_Mouse(&ball, vNow);
switch (keysDown) {
    case CNTRLDN+SHIFTDN: Ball_UseSet(&ball, OtherAxes); break;
    case CNTRLDN:         Ball_UseSet(&ball, BodyAxes);  break;
    case SHIFTDN:         Ball_UseSet(&ball, CameraAxes); break;
    default:              Ball_UseSet(&ball, NoAxes);    break;
}
}
/* end of while on QTest */
Ball_Update(&ball);
scene_Draw(&ball); /* draw into the back buffer */
swapbuffers();     /* and show it in the front buffer */
}
/* NOT REACHED */
}
```

/\* Draw whole window, including controller. \*/

void scene\_Draw(BallData \*ball)

```
{
    RGBcolor(0, 0, 0);
    clear();
    body_Draw(ball);
    Ball_Draw(ball);
}
```

/\* Draw the object being controlled. \*/

```
void body_Draw(BallData *ball)
{
    HMatrix mNow;
    Ball_Value(ball, mNow);
    pushmatrix();
    multmatrix(mNow);
    scale(RADIUS, RADIUS, RADIUS);
}
```

```
    drawbody(mNow);  
    popmatrix();  
}
```



```
Demo: Demo.o Body.o Ball.o BallMath.o BallAux.o
      cc -o Demo Demo.o Body.o Ball.o BallMath.o BallAux.o -lgl_s -lm

clean:
      rm -f *.o Demo
```

ANSI C code from the article

"Arcball Rotation Control"

by Ken Shoemake, [shoemake@graphics.cis.upenn.edu](mailto:shoemake@graphics.cis.upenn.edu)






in "Graphics Gems IV", Academic Press, 1994

"Demo.c" is an interactive SGI program for Arcball.

Much of the rest of the code is portable to other machines.

# Index of

## /pubs/tog/GraphicsGems/gemsiv/polar\_decomp/

Name	Last modified	Size	Description
 <a href="#">_Parent Directory</a>			
 <a href="#">_Decompose.c</a>	29-Jun-00 08:20	15K	
 <a href="#">_Decompose.h</a>	29-Jun-00 08:20	1K	
 <a href="#">_Makefile</a>	29-Jun-00 08:20	1K	
 <a href="#">_README</a>	29-Jun-00 08:20	1K	

```
/***** Decompose.c *****/
/* Ken Shoemake, 1993 */
#include <math.h>
#include "Decompose.h"

/***** Matrix Preliminaries *****/

/** Fill out 3x3 matrix to 4x4 **/
#define mat_pad(A) (A[W][X]=A[X][W]=A[W][Y]=A[Y][W]=A[W][Z]=A[Z][W]=0,A[W][W]=1)

/** Copy nxn matrix A to C using "gets" for assignment **/
#define mat_copy(C,gets,A,n) {int i,j; for(i=0;i<n;i++) for(j=0;j<n;j++)\
    C[i][j] gets (A[i][j]);}

/** Copy transpose of nxn matrix A to C using "gets" for assignment **/
#define mat_tpose(AT,gets,A,n) {int i,j; for(i=0;i<n;i++) for(j=0;j<n;j++)\
    AT[i][j] gets (A[j][i]);}

/** Assign nxn matrix C the element-wise combination of A and B using "op" **/
#define mat_binop(C,gets,A,op,B,n) {int i,j; for(i=0;i<n;i++) for(j=0;j<n;j++)\
    C[i][j] gets (A[i][j]) op (B[i][j]);}

/** Multiply the upper left 3x3 parts of A and B to get AB **/
void mat_mult(HMatrix A, HMatrix B, HMatrix AB)
{
    int i, j;
    for (i=0; i<3; i++) for (j=0; j<3; j++)
        AB[i][j] = A[i][0]*B[0][j] + A[i][1]*B[1][j] + A[i][2]*B[2][j];
}

/** Return dot product of length 3 vectors va and vb **/
float vdot(float *va, float *vb)
{
    return (va[0]*vb[0] + va[1]*vb[1] + va[2]*vb[2]);
}

/** Set v to cross product of length 3 vectors va and vb **/
void vcross(float *va, float *vb, float *v)
{
    v[0] = va[1]*vb[2] - va[2]*vb[1];
    v[1] = va[2]*vb[0] - va[0]*vb[2];
    v[2] = va[0]*vb[1] - va[1]*vb[0];
}

/** Set MadjT to transpose of inverse of M times determinant of M **/
void adjoint_transpose(HMatrix M, HMatrix MadjT)
{
    vcross(M[1], M[2], MadjT[0]);
    vcross(M[2], M[0], MadjT[1]);
    vcross(M[0], M[1], MadjT[2]);
}

/***** Quaternion Preliminaries *****/

/* Construct a (possibly non-unit) quaternion from real components. */
Quat Qt_(float x, float y, float z, float w)
{
    Quat qq;
    qq.x = x; qq.y = y; qq.z = z; qq.w = w;
    return (qq);
}
```

```
/* Return conjugate of quaternion. */
Quat Qt_Conj(Quat q)
{
    Quat qq;
    qq.x = -q.x; qq.y = -q.y; qq.z = -q.z; qq.w = q.w;
    return (qq);
}

/* Return quaternion product qL * qR. Note: order is important!
 * To combine rotations, use the product Mul(qSecond, qFirst),
 * which gives the effect of rotating by qFirst then qSecond. */
Quat Qt_Mul(Quat qL, Quat qR)
{
    Quat qq;
    qq.w = qL.w*qR.w - qL.x*qR.x - qL.y*qR.y - qL.z*qR.z;
    qq.x = qL.w*qR.x + qL.x*qR.w + qL.y*qR.z - qL.z*qR.y;
    qq.y = qL.w*qR.y + qL.y*qR.w + qL.z*qR.x - qL.x*qR.z;
    qq.z = qL.w*qR.z + qL.z*qR.w + qL.x*qR.y - qL.y*qR.x;
    return (qq);
}

/* Return product of quaternion q by scalar w. */
Quat Qt_Scale(Quat q, float w)
{
    Quat qq;
    qq.w = q.w*w; qq.x = q.x*w; qq.y = q.y*w; qq.z = q.z*w;
    return (qq);
}

/* Construct a unit quaternion from rotation matrix. Assumes matrix is
 * used to multiply column vector on the left: vnew = mat vold. Works
 * correctly for right-handed coordinate system and right-handed rotations.
 * Translation and perspective components ignored. */
Quat Qt_FromMatrix(HMatrix mat)
{
    /* This algorithm avoids near-zero divides by looking for a large component
     * - first w, then x, y, or z. When the trace is greater than zero,
     * |w| is greater than 1/2, which is as small as a largest component can be.
     * Otherwise, the largest diagonal entry corresponds to the largest of |x|,
     * |y|, or |z|, one of which must be larger than |w|, and at least 1/2. */
    Quat qu;
    register double tr, s;

    tr = mat[X][X] + mat[Y][Y] + mat[Z][Z];
    if (tr >= 0.0) {
        s = sqrt(tr + mat[W][W]);
        qu.w = s*0.5;
        s = 0.5 / s;
        qu.x = (mat[Z][Y] - mat[Y][Z]) * s;
        qu.y = (mat[X][Z] - mat[Z][X]) * s;
        qu.z = (mat[Y][X] - mat[X][Y]) * s;
    } else {
        int h = X;
        if (mat[Y][Y] > mat[X][X]) h = Y;
        if (mat[Z][Z] > mat[h][h]) h = Z;
        switch (h) {
#define caseMacro(i,j,k,I,J,K) \
            case I:\
                s = sqrt( (mat[I][I] - (mat[J][J]+mat[K][K])) + mat[W][W] );\
                qu.i = s*0.5;\

```

```

        s = 0.5 / s;\
        qu.j = (mat[I][J] + mat[J][I]) * s;\
        qu.k = (mat[K][I] + mat[I][K]) * s;\
        qu.w = (mat[K][J] - mat[J][K]) * s;\
        break
    caseMacro(x,y,z,X,Y,Z);
    caseMacro(y,z,x,Y,Z,X);
    caseMacro(z,x,y,Z,X,Y);
    }
}
if (mat[W][W] != 1.0) qu = Qt_Scale(qu, 1/sqrt(mat[W][W]));
return (qu);
}
/***** Decomp Auxiliaries *****/

static HMatrix mat_id = {{1,0,0,0},{0,1,0,0},{0,0,1,0},{0,0,0,1}};

/** Compute either the 1 or infinity norm of M, depending on tpose */
float mat_norm(HMatrix M, int tpose)
{
    int i;
    float sum, max;
    max = 0.0;
    for (i=0; i<3; i++) {
        if (tpose) sum = fabs(M[0][i])+fabs(M[1][i])+fabs(M[2][i]);
        else      sum = fabs(M[i][0])+fabs(M[i][1])+fabs(M[i][2]);
        if (max<sum) max = sum;
    }
    return max;
}

float norm_inf(HMatrix M) {return mat_norm(M, 0);}
float norm_one(HMatrix M) {return mat_norm(M, 1);}

/** Return index of column of M containing maximum abs entry, or -1 if M=0 */
int find_max_col(HMatrix M)
{
    float abs, max;
    int i, j, col;
    max = 0.0; col = -1;
    for (i=0; i<3; i++) for (j=0; j<3; j++) {
        abs = M[i][j]; if (abs<0.0) abs = -abs;
        if (abs>max) {max = abs; col = j;}
    }
    return col;
}

/** Setup u for Household reflection to zero all v components but first */
void make_reflector(float *v, float *u)
{
    float s = sqrt(vdot(v, v));
    u[0] = v[0]; u[1] = v[1];
    u[2] = v[2] + ((v[2]<0.0) ? -s : s);
    s = sqrt(2.0/vdot(u, u));
    u[0] = u[0]*s; u[1] = u[1]*s; u[2] = u[2]*s;
}

/** Apply Householder reflection represented by u to column vectors of M */
void reflect_cols(HMatrix M, float *u)
{
    int i, j;

```

```
    for (i=0; i<3; i++) {
        float s = u[0]*M[0][i] + u[1]*M[1][i] + u[2]*M[2][i];
        for (j=0; j<3; j++) M[j][i] -= u[j]*s;
    }
}

/** Apply Householder reflection represented by u to row vectors of M */
void reflect_rows(HMatrix M, float *u)
{
    int i, j;
    for (i=0; i<3; i++) {
        float s = vdot(u, M[i]);
        for (j=0; j<3; j++) M[i][j] -= u[j]*s;
    }
}

/** Find orthogonal factor Q of rank 1 (or less) M */
void do_rank1(HMatrix M, HMatrix Q)
{
    float v1[3], v2[3], s;
    int col;
    mat_copy(Q, mat_id, 4);
    /* If rank(M) is 1, we should find a non-zero column in M */
    col = find_max_col(M);
    if (col<0) return; /* Rank is 0 */
    v1[0] = M[0][col]; v1[1] = M[1][col]; v1[2] = M[2][col];
    make_reflector(v1, v1); reflect_cols(M, v1);
    v2[0] = M[2][0]; v2[1] = M[2][1]; v2[2] = M[2][2];
    make_reflector(v2, v2); reflect_rows(M, v2);
    s = M[2][2];
    if (s<0.0) Q[2][2] = -1.0;
    reflect_cols(Q, v1); reflect_rows(Q, v2);
}

/** Find orthogonal factor Q of rank 2 (or less) M using adjoint transpose */
void do_rank2(HMatrix M, HMatrix MadjT, HMatrix Q)
{
    float v1[3], v2[3];
    float w, x, y, z, c, s, d;
    int col;
    /* If rank(M) is 2, we should find a non-zero column in MadjT */
    col = find_max_col(MadjT);
    if (col<0) {do_rank1(M, Q); return;} /* Rank<2 */
    v1[0] = MadjT[0][col]; v1[1] = MadjT[1][col]; v1[2] = MadjT[2][col];
    make_reflector(v1, v1); reflect_cols(M, v1);
    vcross(M[0], M[1], v2);
    make_reflector(v2, v2); reflect_rows(M, v2);
    w = M[0][0]; x = M[0][1]; y = M[1][0]; z = M[1][1];
    if (w*z>x*y) {
        c = z+w; s = y-x; d = sqrt(c*c+s*s); c = c/d; s = s/d;
        Q[0][0] = Q[1][1] = c; Q[0][1] = -(Q[1][0] = s);
    } else {
        c = z-w; s = y+x; d = sqrt(c*c+s*s); c = c/d; s = s/d;
        Q[0][0] = -(Q[1][1] = c); Q[0][1] = Q[1][0] = s;
    }
    Q[0][2] = Q[2][0] = Q[1][2] = Q[2][1] = 0.0; Q[2][2] = 1.0;
    reflect_cols(Q, v1); reflect_rows(Q, v2);
}

/***** Polar Decomposition *****/
```

```
/* Polar Decomposition of 3x3 matrix in 4x4,
 * M = QS. See Nicholas Higham and Robert S. Schreiber,
 * Fast Polar Decomposition of An Arbitrary Matrix,
 * Technical Report 88-942, October 1988,
 * Department of Computer Science, Cornell University.
 */
float polar_decomp(HMatrix M, HMatrix Q, HMatrix S)
{
#define TOL 1.0e-6
    HMatrix Mk, MadjTk, Ek;
    float det, M_one, M_inf, MadjT_one, MadjT_inf, E_one, gamma, g1, g2;
    int i, j;
    mat_tpose(Mk,=,M,3);
    M_one = norm_one(Mk); M_inf = norm_inf(Mk);
    do {
        adjoint_transpose(Mk, MadjTk);
        det = vdot(Mk[0], MadjTk[0]);
        if (det==0.0) {do_rank2(Mk, MadjTk, Mk); break;}
        MadjT_one = norm_one(MadjTk); MadjT_inf = norm_inf(MadjTk);
        gamma = sqrt(sqrt((MadjT_one*MadjT_inf)/(M_one*M_inf)))/fabs(det);
        g1 = gamma*0.5;
        g2 = 0.5/(gamma*det);
        mat_copy(Ek,=,Mk,3);
        mat_binop(Mk,=,g1*Mk,+,g2*MadjTk,3);
        mat_copy(Ek,-=,Mk,3);
        E_one = norm_one(Ek);
        M_one = norm_one(Mk); M_inf = norm_inf(Mk);
    } while (E_one>(M_one*TOL));
    mat_tpose(Q,=,Mk,3); mat_pad(Q);
    mat_mult(Mk, M, S); mat_pad(S);
    for (i=0; i<3; i++) for (j=i; j<3; j++)
        S[i][j] = S[j][i] = 0.5*(S[i][j]+S[j][i]);
    return (det);
}
```

```
/****** Spectral Decomposition *****/
```

```
/* Compute the spectral decomposition of symmetric positive semi-definite S.
 * Returns rotation in U and scale factors in result, so that if K is a diagonal
 * matrix of the scale factors, then S = U K (U transpose). Uses Jacobi method.
 * See Gene H. Golub and Charles F. Van Loan. Matrix Computations. Hopkins 1983.
 */
HVect spect_decomp(HMatrix S, HMatrix U)
{
    HVect kv;
```



```

double Diag[3],OffD[3]; /* OffD is off-diag (by omitted index) */
double g,h,fabsh,fabsOffDi,t,theta,c,s,tau,ta,OffDq,a,b;
static char nxt[] = {Y,Z,X};
int sweep, i, j;
mat_copy(U,mat_id,4);
Diag[X] = S[X][X]; Diag[Y] = S[Y][Y]; Diag[Z] = S[Z][Z];
OffD[X] = S[Y][Z]; OffD[Y] = S[Z][X]; OffD[Z] = S[X][Y];
for (sweep=20; sweep>0; sweep--) {
    float sm = fabs(OffD[X])+fabs(OffD[Y])+fabs(OffD[Z]);
    if (sm==0.0) break;
    for (i=Z; i>=X; i--) {
        int p = nxt[i]; int q = nxt[p];
        fabsOffDi = fabs(OffD[i]);
        g = 100.0*fabsOffDi;
        if (fabsOffDi>0.0) {
            h = Diag[q] - Diag[p];
            fabsh = fabs(h);
            if (fabsh+g==fabsh) {
                t = OffD[i]/h;
            } else {
                theta = 0.5*h/OffD[i];
                t = 1.0/(fabs(theta)+sqrt(theta*theta+1.0));
                if (theta<0.0) t = -t;
            }
            c = 1.0/sqrt(t*t+1.0); s = t*c;
            tau = s/(c+1.0);
            ta = t*OffD[i]; OffD[i] = 0.0;
            Diag[p] -= ta; Diag[q] += ta;
            OffDq = OffD[q];
            OffD[q] -= s*(OffD[p] + tau*OffDq);
            OffD[p] += s*(OffDq - tau*OffD[p]);
            for (j=Z; j>=X; j--) {
                a = U[j][p]; b = U[j][q];
                U[j][p] -= s*(b + tau*a);
                U[j][q] += s*(a - tau*b);
            }
        }
    }
}
kv.x = Diag[X]; kv.y = Diag[Y]; kv.z = Diag[Z]; kv.w = 1.0;
return (kv);
}

/***** Spectral Axis Adjustment *****/

/* Given a unit quaternion, q, and a scale vector, k, find a unit quaternion, p,
 * which permutes the axes and turns freely in the plane of duplicate scale
 * factors, such that q p has the largest possible w component, i.e. the
 * smallest possible angle. Permutes k's components to go with q p instead of q.
 * See Ken Shoemake and Tom Duff. Matrix Animation and Polar Decomposition.
 * Proceedings of Graphics Interface 1992. Details on p. 262-263.
 */
Quat snuggle(Quat q, HVect *k)
{
#define SQRTHALF (0.7071067811865475244)
#define sgn(n,v) ((n)?-(v):(v))
#define swap(a,i,j) {a[3]=a[i]; a[i]=a[j]; a[j]=a[3];}
#define cycle(a,p) if (p) {a[3]=a[0]; a[0]=a[1]; a[1]=a[2]; a[2]=a[3];}\
                    else {a[3]=a[2]; a[2]=a[1]; a[1]=a[0]; a[0]=a[3];}

    Quat p;
    float ka[4];

```

```

int i, turn = -1;
ka[X] = k->x; ka[Y] = k->y; ka[Z] = k->z;
if (ka[X]==ka[Y]) {if (ka[X]==ka[Z]) turn = W; else turn = Z;}
else {if (ka[X]==ka[Z]) turn = Y; else if (ka[Y]==ka[Z]) turn = X;}
if (turn>=0) {
    Quat qtoz, qp;
    unsigned neg[3], win;
    double mag[3], t;
    static Quat qxtoz = {0,SQRTHALF,0,SQRTHALF};
    static Quat qytoz = {SQRTHALF,0,0,SQRTHALF};
    static Quat qppmm = { 0.5, 0.5,-0.5,-0.5};
    static Quat qpppp = { 0.5, 0.5, 0.5, 0.5};
    static Quat qmpmm = {-0.5, 0.5,-0.5,-0.5};
    static Quat qpppm = { 0.5, 0.5, 0.5,-0.5};
    static Quat q0001 = { 0.0, 0.0, 0.0, 1.0};
    static Quat q1000 = { 1.0, 0.0, 0.0, 0.0};
    switch (turn) {
    default: return (Qt_Conj(q));
    case X: q = Qt_Mul(q, qtoz = qxtoz); swap(ka,X,Z) break;
    case Y: q = Qt_Mul(q, qtoz = qytoz); swap(ka,Y,Z) break;
    case Z: qtoz = q0001; break;
    }
    q = Qt_Conj(q);
    mag[0] = (double)q.z*q.z+(double)q.w*q.w-0.5;
    mag[1] = (double)q.x*q.z-(double)q.y*q.w;
    mag[2] = (double)q.y*q.z+(double)q.x*q.w;
    for (i=0; i<3; i++) if (neg[i] = (mag[i]<0.0)) mag[i] = -mag[i];
    if (mag[0]>mag[1]) {if (mag[0]>mag[2]) win = 0; else win = 2;}
    else {if (mag[1]>mag[2]) win = 1; else win = 2;}
    switch (win) {
    case 0: if (neg[0]) p = q1000; else p = q0001; break;
    case 1: if (neg[1]) p = qppmm; else p = qpppp; cycle(ka,0) break;
    case 2: if (neg[2]) p = qmpmm; else p = qpppm; cycle(ka,1) break;
    }
    qp = Qt_Mul(q, p);
    t = sqrt(mag[win]+0.5);
    p = Qt_Mul(p, Qt_(0.0,0.0,-qp.z/t,qp.w/t));
    p = Qt_Mul(qtoz, Qt_Conj(p));
} else {
    float qa[4], pa[4];
    unsigned lo, hi, neg[4], par = 0;
    double all, big, two;
    qa[0] = q.x; qa[1] = q.y; qa[2] = q.z; qa[3] = q.w;
    for (i=0; i<4; i++) {
        pa[i] = 0.0;
        if (neg[i] = (qa[i]<0.0)) qa[i] = -qa[i];
        par ^= neg[i];
    }
    /* Find two largest components, indices in hi and lo */
    if (qa[0]>qa[1]) lo = 0; else lo = 1;
    if (qa[2]>qa[3]) hi = 2; else hi = 3;
    if (qa[lo]>qa[hi]) {
        if (qa[lo^1]>qa[hi]) {hi = lo; lo ^= 1;}
        else {hi ^= lo; lo ^= hi; hi ^= lo;}
    } else {if (qa[hi^1]>qa[lo]) lo = hi^1;}
    all = (qa[0]+qa[1]+qa[2]+qa[3])*0.5;
    two = (qa[hi]+qa[lo])*SQRTHALF;
    big = qa[hi];
    if (all>two) {
        if (all>big) {/*all*/
            {int i; for (i=0; i<4; i++) pa[i] = sgn(neg[i], 0.5);}

```

```
        cycle(ka,par)
    } else { /*big*/ pa[hi] = sgn(neg[hi],1.0);}
} else {
    if (two>big) { /*two*/
        pa[hi] = sgn(neg[hi],SQRTHALF); pa[lo] = sgn(neg[lo], SQRTHALF);
        if (lo>hi) {hi ^= lo; lo ^= hi; hi ^= lo;}
        if (hi==W) {hi = "\001\002\000"[lo]; lo = 3-hi-lo;}
        swap(ka,hi,lo)
    } else { /*big*/ pa[hi] = sgn(neg[hi],1.0);}
}
p.x = -pa[0]; p.y = -pa[1]; p.z = -pa[2]; p.w = pa[3];
}
k->x = ka[X]; k->y = ka[Y]; k->z = ka[Z];
return (p);
}
```

/\*\*\*\*\*\* Decompose Affine Matrix \*\*\*\*\*/

/\* Decompose 4x4 affine matrix A as TFRUK(U transpose), where t contains the  
\* translation components, q contains the rotation R, u contains U, k contains  
\* scale factors, and f contains the sign of the determinant.  
\* Assumes A transforms column vectors in right-handed coordinates.  
\* See Ken Shoemake and Tom Duff. Matrix Animation and Polar Decomposition.  
\* Proceedings of Graphics Interface 1992.  
\*/

void decomp\_affine(HMatrix A, AffineParts \*parts)

```
{
    HMatrix Q, S, U;
    Quat p;
    float det;
    parts->t = Qt_(A[X][W], A[Y][W], A[Z][W], 0);
    det = polar_decomp(A, Q, S);
    if (det<0.0) {
        mat_copy(Q,=-Q,3);
        parts->f = -1;
    } else parts->f = 1;
    parts->q = Qt_FromMatrix(Q);
    parts->k = spect_decomp(S, U);
    parts->u = Qt_FromMatrix(U);
    p = snuggle(parts->u, &parts->k);
    parts->u = Qt_Mul(parts->u, p);
}
```

/\*\*\*\*\*\* Invert Affine Decomposition \*\*\*\*\*/

/\* Compute inverse of affine decomposition.

\*/  
void invert\_affine(AffineParts \*parts, AffineParts \*inverse)

```
{
    Quat t, p;
    inverse->f = parts->f;
```

```
inverse->q = Qt_Conj(parts->q);
inverse->u = Qt_Mul(parts->q, parts->u);
inverse->k.x = (parts->k.x==0.0) ? 0.0 : 1.0/parts->k.x;
inverse->k.y = (parts->k.y==0.0) ? 0.0 : 1.0/parts->k.y;
inverse->k.z = (parts->k.z==0.0) ? 0.0 : 1.0/parts->k.z;
inverse->k.w = parts->k.w;
t = Qt_(-parts->t.x, -parts->t.y, -parts->t.z, 0);
t = Qt_Mul(Qt_Conj(inverse->u), Qt_Mul(t, inverse->u));
t = Qt_(inverse->k.x*t.x, inverse->k.y*t.y, inverse->k.z*t.z, 0);
p = Qt_Mul(inverse->q, inverse->u);
t = Qt_Mul(p, Qt_Mul(t, Qt_Conj(p)));
inverse->t = (inverse->f>0.0) ? t : Qt_(-t.x, -t.y, -t.z, 0);
}
```

```
/**** Decompose.h - Basic declarations ****/
#ifndef _H_Decompose
#define _H_Decompose
typedef struct {float x, y, z, w;} Quat; /* Quaternion */
enum QuatPart {X, Y, Z, W};
typedef Quat HVect; /* Homogeneous 3D vector */
typedef float HMatrix[4][4]; /* Right-handed, for column vectors */
typedef struct {
    HVect t; /* Translation components */
    Quat q; /* Essential rotation */
    Quat u; /* Stretch rotation */
    HVect k; /* Stretch factors */
    float f; /* Sign of determinant */
} AffineParts;
float polar_decomp(HMatrix M, HMatrix Q, HMatrix S);
HVect spect_decomp(HMatrix S, HMatrix U);
Quat snuggle(Quat q, HVect *k);
void decomp_affine(HMatrix A, AffineParts *parts);
void invert_affine(AffineParts *parts, AffineParts *inverse);
#endif
```

CC = gcc

Decompose.o: Decompose.h

[http://www.acm.org/pubs/tog/GraphicsGems/gemsiv/polar\\_decomp/README](http://www.acm.org/pubs/tog/GraphicsGems/gemsiv/polar_decomp/README)








ANSI C code from the article

"Polar Matrix Decomposition"

by Ken Shoemake, [shoemake@graphics.cis.upenn.edu](mailto:shoemake@graphics.cis.upenn.edu)

in "Graphics Gems IV", Academic Press, 1994

# Index of /pubs/tog/GraphicsGems/gemsiv/euler\_angle/

Name	Last modified	Size	Description
 <a href="#">_Parent Directory</a>			
 <a href="#">_EulerAngles.c</a>	29-Jun-00 08:19	3K	
 <a href="#">_EulerAngles.h</a>	29-Jun-00 08:19	3K	
 <a href="#">_EulerSample.c</a>	29-Jun-00 08:19	1K	
 <a href="#">_Makefile</a>	29-Jun-00 08:19	1K	
 <a href="#">_QuatTypes.h</a>	29-Jun-00 08:19	1K	
 <a href="#">_README</a>	29-Jun-00 08:19	1K	



```
/***** EulerAngles.c - Convert Euler angles to/from matrix or quat *****/
/* Ken Shoemake, 1993 */
#include <math.h>
#include <float.h>
#include "EulerAngles.h"

EulerAngles Eul_(float ai, float aj, float ah, int order)
{
    EulerAngles ea;
    ea.x = ai; ea.y = aj; ea.z = ah;
    ea.w = order;
    return (ea);
}

/* Construct quaternion from Euler angles (in radians). */
Quat Eul_ToQuat(EulerAngles ea)
{
    Quat qu;
    double a[3], ti, tj, th, ci, cj, ch, si, sj, sh, cc, cs, sc, ss;
    int i,j,k,h,n,s,f;
    EulGetOrd(ea.w,i,j,k,h,n,s,f);
    if (f==EulFrmR) {float t = ea.x; ea.x = ea.z; ea.z = t;}
    if (n==EulParOdd) ea.y = -ea.y;
    ti = ea.x*0.5; tj = ea.y*0.5; th = ea.z*0.5;
    ci = cos(ti);  cj = cos(tj);  ch = cos(th);
    si = sin(ti);  sj = sin(tj);  sh = sin(th);
    cc = ci*ch; cs = ci*sh; sc = si*ch; ss = si*sh;
    if (s==EulRepYes) {
        a[i] = cj*(cs + sc);    /* Could speed up with */
        a[j] = sj*(cc + ss);    /* trig identities. */
        a[k] = sj*(cs - sc);
        qu.w = cj*(cc - ss);
    } else {
        a[i] = cj*sc - sj*cs;
        a[j] = cj*ss + sj*cc;
        a[k] = cj*cs - sj*sc;
        qu.w = cj*cc + sj*ss;
    }
    if (n==EulParOdd) a[j] = -a[j];
    qu.x = a[X]; qu.y = a[Y]; qu.z = a[Z];
    return (qu);
}

/* Construct matrix from Euler angles (in radians). */
void Eul_ToHMatrix(EulerAngles ea, HMatrix M)
{
    double ti, tj, th, ci, cj, ch, si, sj, sh, cc, cs, sc, ss;
    int i,j,k,h,n,s,f;
    EulGetOrd(ea.w,i,j,k,h,n,s,f);
    if (f==EulFrmR) {float t = ea.x; ea.x = ea.z; ea.z = t;}
    if (n==EulParOdd) {ea.x = -ea.x; ea.y = -ea.y; ea.z = -ea.z;}
    ti = ea.x;  tj = ea.y;  th = ea.z;
    ci = cos(ti); cj = cos(tj); ch = cos(th);
    si = sin(ti); sj = sin(tj); sh = sin(th);
    cc = ci*ch; cs = ci*sh; sc = si*ch; ss = si*sh;
    if (s==EulRepYes) {
        M[i][i] = cj;      M[i][j] = sj*si;      M[i][k] = sj*ci;
        M[j][i] = sj*sh;   M[j][j] = -cj*ss+cc;   M[j][k] = -cj*cs-sc;
        M[k][i] = -sj*ch;  M[k][j] =  cj*sc+cs;   M[k][k] =  cj*cc-ss;
    } else {
        M[i][i] = cj*ch;   M[i][j] = sj*sc-cs;   M[i][k] = sj*cc+ss;
        M[j][i] = cj*sh;   M[j][j] = sj*ss+cc;   M[j][k] = sj*cs-sc;
    }
}
```

```
    M[k][i] = -sj;    M[k][j] = cj*si;    M[k][k] = cj*ci;
}
M[W][X]=M[W][Y]=M[W][Z]=M[X][W]=M[Y][W]=M[Z][W]=0.0; M[W][W]=1.0;
}

/* Convert matrix to Euler angles (in radians). */
EulerAngles Eul_FromHMatrix(HMatrix M, int order)
{
    EulerAngles ea;
    int i,j,k,h,n,s,f;
    EulGetOrd(order,i,j,k,h,n,s,f);
    if (s==EulRepYes) {
        double sy = sqrt(M[i][j]*M[i][j] + M[i][k]*M[i][k]);
        if (sy > 16*FLT_EPSILON) {
            ea.x = atan2(M[i][j], M[i][k]);
            ea.y = atan2(sy, M[i][i]);
            ea.z = atan2(M[j][i], -M[k][i]);
        } else {
            ea.x = atan2(-M[j][k], M[j][j]);
            ea.y = atan2(sy, M[i][i]);
            ea.z = 0;
        }
    } else {
        double cy = sqrt(M[i][i]*M[i][i] + M[j][i]*M[j][i]);
        if (cy > 16*FLT_EPSILON) {
            ea.x = atan2(M[k][j], M[k][k]);
            ea.y = atan2(-M[k][i], cy);
            ea.z = atan2(M[j][i], M[i][i]);
        } else {
            ea.x = atan2(-M[j][k], M[j][j]);
            ea.y = atan2(-M[k][i], cy);
            ea.z = 0;
        }
    }
    if (n==EulParOdd) {ea.x = -ea.x; ea.y = - ea.y; ea.z = -ea.z;}
    if (f==EulFrmR) {float t = ea.x; ea.x = ea.z; ea.z = t;}
    ea.w = order;
    return (ea);
}

/* Convert quaternion to Euler angles (in radians). */
EulerAngles Eul_FromQuat(Quat q, int order)
{
    HMatrix M;
    double Nq = q.x*q.x+q.y*q.y+q.z*q.z+q.w*q.w;
    double s = (Nq > 0.0) ? (2.0 / Nq) : 0.0;
    double xs = q.x*s,    ys = q.y*s,    zs = q.z*s;
    double wx = q.w*xs,    wy = q.w*ys,    wz = q.w*zs;
    double xx = q.x*xs,    xy = q.x*ys,    xz = q.x*zs;
    double yy = q.y*ys,    yz = q.y*zs,    zz = q.z*zs;
    M[X][X] = 1.0 - (yy + zz); M[X][Y] = xy - wz; M[X][Z] = xz + wy;
    M[Y][X] = xy + wz; M[Y][Y] = 1.0 - (xx + zz); M[Y][Z] = yz - wx;
    M[Z][X] = xz - wy; M[Z][Y] = yz + wx; M[Z][Z] = 1.0 - (xx + yy);
    M[W][X]=M[W][Y]=M[W][Z]=M[X][W]=M[Y][W]=M[Z][W]=0.0; M[W][W]=1.0;
    return (Eul_FromHMatrix(M, order));
}
```

```
/***** EulerAngles.h - Support for 24 angle schemes *****/
/* Ken Shoemake, 1993 */
#ifndef _H_EulerAngles
#define _H_EulerAngles
#include "QuatTypes.h"
/**** Order type constants, constructors, extractors ****/
/* There are 24 possible conventions, designated by: */
/*   o EulAxI = axis used initially */
/*   o EulPar = parity of axis permutation */
/*   o EulRep = repetition of initial axis as last */
/*   o EulFrm = frame from which axes are taken */
/* Axes I,J,K will be a permutation of X,Y,Z. */
/* Axis H will be either I or K, depending on EulRep. */
/* Frame S takes axes from initial static frame. */
/* If ord = (AxI=X, Par=Even, Rep=No, Frm=S), then */
/* {a,b,c,ord} means Rz(c)Ry(b)Rx(a), where Rz(c)v */
/* rotates v around Z by c radians. */
#define EulFrmS 0
#define EulFrmR 1
#define EulFrm(ord) ((unsigned)(ord)&1)
#define EulRepNo 0
#define EulRepYes 1
#define EulRep(ord) (((unsigned)(ord)>>1)&1)
#define EulParEven 0
#define EulParOdd 1
#define EulPar(ord) (((unsigned)(ord)>>2)&1)
#define EulSafe "\000\001\002\000"
#define EulNext "\001\002\000\001"
#define EulAxI(ord) ((int)(EulSafe[(((unsigned)(ord)>>3)&3)]))
#define EulAxJ(ord) ((int)(EulNext[EulAxI(ord)+(EulPar(ord)==EulParOdd)]))
#define EulAxK(ord) ((int)(EulNext[EulAxI(ord)+(EulPar(ord)!=EulParOdd)]))
#define EulAxH(ord) ((EulRep(ord)==EulRepNo)?EulAxK(ord):EulAxI(ord))
/* EulGetOrd unpacks all useful information about order simultaneously. */
#define EulGetOrd(ord,i,j,k,h,n,s,f) {unsigned o=ord;f=o&1;o>>=1;s=o&1;o>>=1;\
n=o&1;o>>=1;i=EulSafe[o&3];j=EulNext[i+n];k=EulNext[i+1-n];h=s?k:i;}
/* EulOrd creates an order value between 0 and 23 from 4-tuple choices. */
#define EulOrd(i,p,r,f) ((((((i)<<1)+(p))<<1)+(r))<<1)+(f))
/* Static axes */
#define EulOrdXYZs EulOrd(X,EulParEven,EulRepNo,EulFrmS)
#define EulOrdXYXs EulOrd(X,EulParEven,EulRepYes,EulFrmS)
#define EulOrdXZYs EulOrd(X,EulParOdd,EulRepNo,EulFrmS)
#define EulOrdZXZs EulOrd(X,EulParOdd,EulRepYes,EulFrmS)
#define EulOrdYZXs EulOrd(Y,EulParEven,EulRepNo,EulFrmS)
#define EulOrdYZYs EulOrd(Y,EulParEven,EulRepYes,EulFrmS)
#define EulOrdYXZs EulOrd(Y,EulParOdd,EulRepNo,EulFrmS)
#define EulOrdYXYs EulOrd(Y,EulParOdd,EulRepYes,EulFrmS)
#define EulOrdZXYs EulOrd(Z,EulParEven,EulRepNo,EulFrmS)
#define EulOrdZXZs EulOrd(Z,EulParEven,EulRepYes,EulFrmS)
#define EulOrdZYXs EulOrd(Z,EulParOdd,EulRepNo,EulFrmS)
#define EulOrdZYZs EulOrd(Z,EulParOdd,EulRepYes,EulFrmS)
/* Rotating axes */
#define EulOrdZYXR EulOrd(X,EulParEven,EulRepNo,EulFrmR)
#define EulOrdXYXR EulOrd(X,EulParEven,EulRepYes,EulFrmR)
#define EulOrdYZXR EulOrd(X,EulParOdd,EulRepNo,EulFrmR)
#define EulOrdZXXR EulOrd(X,EulParOdd,EulRepYes,EulFrmR)
#define EulOrdXZYR EulOrd(Y,EulParEven,EulRepNo,EulFrmR)
#define EulOrdYZYR EulOrd(Y,EulParEven,EulRepYes,EulFrmR)
#define EulOrdYXZR EulOrd(Y,EulParOdd,EulRepNo,EulFrmR)
#define EulOrdYXYR EulOrd(Y,EulParOdd,EulRepYes,EulFrmR)
#define EulOrdZXYR EulOrd(Z,EulParEven,EulRepNo,EulFrmR)
#define EulOrdZXZR EulOrd(Z,EulParEven,EulRepYes,EulFrmR)
```

```
#define EulOrdXYZr      EulOrd(Z,EulParOdd,EulRepNo,EulFrmR)
#define EulOrdZYZr      EulOrd(Z,EulParOdd,EulRepYes,EulFrmR)

EulerAngles Eul_(float ai, float aj, float ah, int order);
Quat Eul_ToQuat(EulerAngles ea);
void Eul_ToHMatrix(EulerAngles ea, HMatrix M);
EulerAngles Eul_FromHMatrix(HMatrix M, int order);
EulerAngles Eul_FromQuat(Quat q, int order);
#endif
```

```
/* EulerSample.c - Read angles as quantum mechanics, write as aerospace */
#include <stdio.h>
#include "EulerAngles.h"
void main(void)
{
    EulerAngles outAngs, inAngs = {0,0,0,EulOrdXYXr};
    HMatrix R;
    printf("Phi Theta Psi (radians): ");
    scanf("%f %f %f",&inAngs.x,&inAngs.y,&inAngs.z);
    Eul_ToHMatrix(inAngs, R);
    outAngs = Eul_FromHMatrix(R, EulOrdXYZs);
    printf(" Roll   Pitch   Yaw   (radians)\n");
    printf("%6.3f %6.3f %6.3f\n", outAngs.x, outAngs.y, outAngs.z);
}
```

```
CC = gcc
```

```
EulerSample: EulerSample.o EulerAngles.o  
    $(CC) -o EulerSample EulerSample.o EulerAngles.o -lm
```

```
clean:  
    rm -f *.o EulerSample
```

```
EulerAngles.o: EulerAngles.h QuatTypes.h  
EulerSample.o: EulerAngles.h QuatTypes.h
```

```
/**** QuatTypes.h - Basic type declarations ****/
#ifndef _H_QuatTypes
#define _H_QuatTypes
/*** Definitions ***/
typedef struct {float x, y, z, w;} Quat; /* Quaternion */
enum QuatPart {X, Y, Z, W};
typedef float HMatrix[4][4]; /* Right-handed, for column vectors */
typedef Quat EulerAngles; /* (x,y,z)=ang 1,2,3, w=order code */
#endif
```

[http://www.acm.org/pubs/tog/GraphicsGems/gemsiv/euler\\_angle/README](http://www.acm.org/pubs/tog/GraphicsGems/gemsiv/euler_angle/README)

ANSI C code from the article

"Euler Angle Conversion"




by Ken Shoemake, [shoemake@graphics.cis.upenn.edu](mailto:shoemake@graphics.cis.upenn.edu)

in "Graphics Gems IV", Academic Press, 1994



# Index of

## /pubs/tog/GraphicsGems/gemsv/ch1-4/

Name	Last modified	Size	Description
 <a href="#">_Parent Directory</a>			
 <a href="#">_rat.c</a>	03-Jan-01 10:50	6K	
 <a href="#">_rat.h</a>	29-Jun-00 08:22	1K	

```
/* ***** rat.c ***** */
/* Ken Shoemake, 1994 */

#include <math.h>
#include "rat.h"

static void Mul32(UINT32 x, UINT32 y, UINT32 *hi, UINT32 *lo)
{
    UINT32 xlo = x&0xFFFF, xhi = (x>>16)&0xFFFF;
    UINT32 ylo = y&0xFFFF, yhi = (y>>16)&0xFFFF;
    UINT32 t1, t2, t3;
    UINT32 lololo, lohi, t1lo, t1hi, t2lo, t2hi, carry;
    *lo = xlo * ylo; *hi = xhi * yhi;
    t1 = xhi * ylo; t2 = xlo * yhi;
    lololo = *lo&0xFFFF; lohi = (*lo>>16)&0xFFFF;
    t1lo = t1&0xFFFF; t1hi = (t1>>16)&0xFFFF;
    t2lo = t2&0xFFFF; t2hi = (t2>>16)&0xFFFF;
    t3 = lohi + t1lo + t2lo;
    carry = (t3>>16)&0xFFFF; lohi = t3&0xFFFF;
    *hi += t1hi + t2hi + carry; *lo = (lohi<<16) + lololo;
}

/* ratapprox(x,n) returns the best rational approximation to x whose numerator
   and denominator are less than or equal to n in absolute value. The denominator
   will be positive, and the numerator and denominator will be in lowest terms.
   IEEE 32-bit floating point and 32-bit integers are required.
   All best rational approximations of a real x may be obtained from x's
   continued fraction representation,  $x = c_0 + 1/(c_1 + 1/(c_2 + 1/(...)))$ 
   by truncation to k terms and possibly "interpolation" of the last term.
   The continued fraction expansion itself is obtained by a variant of the
   standard GCD algorithm, which is folded into the recursions generating
   successive numerators and denominators. These recursions both have the
   same form:  $f[k] = c[k]*f[k-1] + f[k-2]$ . For further information, see
   Fundamentals of Mathematics, Volume I, MIT Press, 1983.
*/
Rational ratapprox(float x, INT32 limit)
{
    float tooLargeToFix = ldexp(1.0, BITS); /* 0x4f000000=2147483648.0 */
    float tooSmallToFix = ldexp(1.0, -BITS); /* 0x30000000=4.6566e-10 */
    float halfTooSmallToFix = ldexp(1.0, -BITS-1); /* 0x2f800000=2.3283e-10 */
    int expForInt = 24; /* This exponent in float makes mantissa an INT32 */
    static Rational ratZero = {0, 1};
    INT32 sign = 1;
    BOOL flip = FALSE; /* If TRUE, nk and dk are swapped */
    int scale; /* Power of 2 to get x into integer domain */
    UINT32 ak2, ak1, ak; /* GCD arguments, initially 1 and x */
    UINT32 ck, climit; /* ck is GCD quotient and c.f. term k */
    INT32 nk, dk; /* Result num. and den., recursively found */
    INT32 nk1 = 0, dk2 = 0; /* History terms for recursion */
    INT32 nk2 = 1, dk1 = 1;
    BOOL hard = FALSE;
    Rational val;

    if (limit <= 0) return (ratZero); /* Insist limit > 0 */
    if (x < 0.0) {x = -x; sign = -1;}
    val.numer = sign; val.denom = limit;
    /* Handle first non-zero integer term of continued fraction,
       rest prepared for integer GCD, sure to fit.
    */
    if (x >= 1.0) { /* First continued fraction term is non-zero */
        float rest;
```

```
    if (x >= tooLargeToFix || (ck = x) >= limit)
        {val.number = sign*limit; val.denom = 1; return (val);}
    flip = TRUE;          /* Keep denominator larger, for fast loop test */
    nk = 1; dk = ck;      /* Make new numerator and denominator */
    rest = x - ck;
    frexp(1.0,&scale);
    scale = expForInt - scale;
    ak = ldexp(rest, scale);
    ak1 = ldexp(1.0, scale);
} else { /* First continued fraction term is zero */
    int n;
    UINT32 num = 1;
    if (x <= tooSmallToFix) { /* Is x too tiny to be 1/INT32 ? */
        if (x <= halfTooSmallToFix) return (ratZero);
        if (limit > (UINT32)(0.5/x)) return (val);
        else return (ratZero);
    }
    /* Treating 1.0 and x as integers, divide 1/x in a peculiar way
       to get accurate remainder
       */
    frexp(x,&scale);
    scale = expForInt - scale;
    ak1 = ldexp(x, scale);
    n = (scale<BITS)?scale:BITS; /* Stay within UINT32 arithmetic */
    num <= n;
    ck = num/ak1; /* First attempt at 1/x */
    ak = num%ak1; /* First attempt at remainder */
    while ((scale -= n) > 0) { /* Shift quotient, remainder until done */
        n = (scale<8)?scale:8; /* The 8 is 24 bits of x in 32 bits */
        num = ak<<n;
        ck = (ck<<n) + (num/ak1);
        ck = ck<<n + num/ak1;
        ak = num%ak1; /* Reduce remainder */
    }
    /* All done with divide */
    if (ck >= limit) { /* Is x too tiny to be 1/limit ? */
        if (2*limit > ck)
            return (val);
        else return (ratZero);
    }
    nk = 1; dk = ck; /* Make new numer and denom */
}
while (ak != 0) { /* If possible, quit when have exact result */
    ak2 = ak1; ak1 = ak; /* Prepare for next term */
    nk2 = nk1; nk1 = nk; /* (This loop does almost all the work) */
    dk2 = dk1; dk1 = dk;
    ck = ak2/ak1; /* Get next term of continued fraction */
    ak = ak2 - ck*ak1; /* Get remainder (GCD step) */
    climt = (limit - dk2)/dk1; /* Anticipate result of recursion on denom */
    if (climt <= ck) {hard = TRUE; break;} /* Do not let denom exceed limit */
    nk = ck*nk1 + nk2; /* Make new result numer and denom */
    dk = ck*dk1 + dk2;
}
if (hard) {
    UINT32 twoClimt = 2*climt;
    if (twoClimt >= ck) { /* If climt < ck/2 no improvement possible */
        nk = climt*nk1 + nk2; /* Make limited numerator and denominator */
        dk = climt*dk1 + dk2;
        if (twoClimt == ck) { /* If climt == ck improvement not sure */
            /* Using climt is better only when dk2/dk1 > ak/ak1 */
            /* For full precision, test dk2*ak1 > dk1*ak */

```

```
        UINT32 dk2ak1Hi, dk2ak1Lo, dk1akHi, dk1akLo;
        Mul32(flip?nk2:dk2, ak1, &dk2ak1Hi, &dk2ak1Lo);
        Mul32(flip?nk1:dk1, ak, &dk1akHi, &dk1akLo);
        if ((dk2ak1Hi < dk1akHi)
            || ((dk2ak1Hi == dk1akHi) && (dk2ak1Lo <= dk1akLo)))
            { nk = nk1; dk = dk1; } /* Not an improvement, so undo step */
    }
}
if (flip) {val.numer = sign*dk; val.denom = nk;}
else      {val.numer = sign*nk; val.denom = dk;}
return (val);
}
```

```
/****** rat.h *****/
/* Ken Shoemake, 1994 */





#ifndef _H_rat
#define _H_rat

#include <limits.h>
typedef int BOOL;
#define TRUE 1
#define FALSE 0
#define BITS (32-1)
#if (INT_MAX>=2147483647)
    typedef int INT32;
    typedef unsigned int UINT32;
#else
    typedef long INT32;
    typedef unsigned long UINT32;
#endif
typedef struct {INT32 numer,denom;} Rational;

Rational ratapprox(float x, INT32 limit);
#endif
```

# Index of

## /pubs/tog/GraphicsGems/gemsv/ch4-9/

Name	Last modified	Size	Description
 <a href="#">_Parent Directory</a>			
 <a href="#">_lincrv.c</a>	29-Jun-00 08:23	2K	
 <a href="#">_lincrv.h</a>	29-Jun-00 08:23	1K	
 <a href="#">_lincrvte.c</a>	29-Jun-00 08:23	1K	

```
/****** lincrv.c *****/
/* Ken Shoemake, 1994 */

#include "lincrv.h"

/* Perform a generic vector unary operation. */
#define V_Op(vdst,gets,op,vsrc,n) {register int V_i;\
    for(V_i=(n)-1;V_i>=0;V_i--) (vdst)[V_i] gets op ((vsrc)[V_i]);}

static void lerp(Knot t, Knot a0, Knot a1, Vect p0, Vect p1, int m, Vect p)
{
    register Knot t0=(a1-t)/(a1-a0), t1=1-t0;
    register int i;
    for (i=m-1; i>=0; i--) p[i] = t0*p0[i] + t1*p1[i];
}

/* DialASpline(t,a,p,m,n,work,Cn,interp,val) computes a point val at parameter
   t on a spline with knot values a and control points p. The curve will have
   Cn continuity, and if interp is TRUE it will interpolate the control points.
   Possibilities include Langrange interpolants, Bezier curves, Catmull-Rom
   interpolating splines, and B-spline curves. Points have m coordinates, and
   n+1 of them are provided. The work array must have room for n+1 points.
   */
int DialASpline(Knot t, Knot a[], Vect p[], int m, int n, Vect work[],
                unsigned int Cn, Bool interp, Vect val)
{
    register int i, j, k, h, lo, hi;

    if (Cn>n-1) Cn = n-1;          /* Anything greater gives one polynomial */
    for (k=0; t>a[k]; k++);         /* Find enclosing knot interval */
    for (h=k; t==a[k]; k++);        /* May want to use fewer legs */
    if (k>n) {k = n; if (h>k) h = k;}
    h = 1+Cn - (k-h); k--;
    lo = k-Cn; hi = k+1+Cn;

    if (interp) {                  /* Lagrange interpolation steps */
        int drop=0;
        if (lo<0) {lo = 0; drop += Cn-k;
            if (hi-lo<Cn) {drop += Cn-hi; hi = Cn;}}
        if (hi>n) {hi = n; drop += k+1+Cn-n;
            if (hi-lo<Cn) {drop += lo-(n-Cn); lo = n-Cn;}}
        for (i=lo; i<=hi; i++) V_Op(work[i],,,p[i],m);
        for (j=1; j<=Cn; j++) {
            for (i=lo; i<=hi-j; i++) {
                lerp(t,a[i],a[i+j],work[i],work[i+1],m,work[i]);
            }
        }
        h = 1+Cn-drop;
    } else {                        /* Prepare for B-spline steps */
        if (lo<0) {h += lo; lo = 0;}
        for (i=lo; i<=lo+h; i++) V_Op(work[i],,,p[i],m);
        if (h<0) h = 0;
    }
    for (j=0; j<h; j++) {
        int tmp = 1+Cn-j;
        for (i=h-1; i>=j; i--) {
            lerp(t,a[lo+i],a[lo+i+tmp],work[lo+i],work[lo+i+1],m,work[lo+i+1]);
        }
    }
    V_Op(val,==,work[lo+h],m);
    return (k);
}
```

}



```
/****** lincrv.h *****/
/* Ken Shoemake, 1994 */

#define MAXDIM 2
typedef float Vect[MAXDIM];
typedef float Knot;
typedef int Bool;

int DialASpline(Knot t, Knot a[], Vect p[], int m, int n, Vect work[],
                unsigned int Cn, Bool interp, Vect val);
```

```
/** lincrvtest.c **/
```

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "lincrv.h"
```

```
#define TRUE 1
#define FALSE 0
#define BIG (1.0e12)
```

```
static Vect work[4];
static Vect ctlPts[] = { {0,0}, {1,1}, {1,0}, {0,0}, };
static float *knots;
static float bezKts[] = {0, 0, 0, 1, 1, 1, BIG};
static float lagKts[] = {0.00, 0.25, 0.75, 1.00, BIG};
static float catKts[] = {-1, 0, 1, 2, BIG};
static float bspKts[] = {-2, -1, 0, 1, 2, 3, BIG};
static int m = MAXDIM;
static int n = 3;
static int Cn = 1;
static Bool interp = FALSE;
static Vect val = {0.84375, 0.0};
static float t = 0;
static int eh = 0;
```

```
enum Flavor{PLY, LAG, BEZ, CAT, BSP, NFLAVORS};
char fnames[][4] = {"PLY", "LAG", "BEZ", "CAT", "BSP"};
```

```
void main(void)
{
    int i;
    int flavor = PLY;

    for (flavor=0; flavor<NFLAVORS; flavor++) {
        switch (flavor) {
            case PLY:    knots = lagKts; interp = TRUE;  Cn = 0; break;
            case LAG:    knots = lagKts; interp = TRUE;  Cn = 2; break;
            case BEZ:    knots = bezKts; interp = FALSE; Cn = 2; break;
            case CAT:    knots = catKts; interp = TRUE;  Cn = 1; break;
            case BSP:    knots = bspKts; interp = FALSE; Cn = 2; break;
            default:     knots = bspKts; interp = FALSE; Cn = 0; break;
        }
        printf("Flavor %s: interp=%d, Cn=%d\n", fnames[flavor],interp,Cn);
        for (t=0.0; t<=1.0; t+=0.125) {
            eh = DialASpline(t, knots, ctlPts, m, n, work, Cn, interp, val);
            printf("(%.3f) ", t);
            for (i=0; i<MAXDIM; i++) printf("%.6f ",val[i]); printf("\n");
        }
    }
}
```

```

/*****
Plot a series of points along a  $\pi/2$ -radian arc of an ellipse.
The arc is specified in terms of a control polygon (a triangle)
with vertices P, Q and K. The arc begins at P, ends at Q, and is
completely contained within the control polygon. The draw_point
function plots a single pixel at display coordinates (x,y).
*****/
```

Entry:

xP, yP, xQ, yQ, xK, yK -- coordinates of P, Q and K. These  
are 32-bit fixed-point values with 16 bits of fraction.  
m -- nonnegative integer that controls spacing between points.  
The angular increment between points is  $1/2^m$  radians.

Exit:

The number of points plotted is  $1 + \text{floor}((\pi/2)*2^m)$ .

```

*****/
```

```

#define PIV2 102944      /* fixed point  $\pi/2$  */
#define TWOPI 411775     /* fixed point  $2\pi$  */
#define HALF 32768       /* fixed point  $1/2$  */
typedef long FIX;        /* 32-bit fixed point, 16-bit fraction */
```

```
qtr_elips(xP, yP, xQ, yQ, xK, yK, m)
```

```
FIX xP, yP, xQ, yQ, xK, yK;
```

```
int m;
```

```
{
```

```
    int i, x, y;
```

```
    FIX vx, ux, vy, uy, w, xJ, yJ;
```

```
    vx = xK - xQ;                /* displacements from center */
```

```
    ux = xK - xP;
```

```
    vy = yK - yQ;
```

```
    uy = yK - yP;
```

```
    xJ = xP - vx + HALF;         /* center of ellipse J */
```

```
    yJ = yP - vy + HALF;
```

```
    ux -= (w = ux >> (2*m + 3)); /* cancel 2nd-order error */
```

```
    ux -= (w >= (2*m + 4));      /* cancel 4th-order error */
```

```
    ux -= w >> (2*m + 3);       /* cancel 6th-order error */
```

```
    ux += vx >> (m + 1);        /* cancel 1st-order error */
```

```
    uy -= (w = uy >> (2*m + 3)); /* cancel 2nd-order error */
```

```
    uy -= (w >= (2*m + 4));      /* cancel 4th-order error */
```

```
    uy -= w >> (2*m + 3);       /* cancel 6th-order error */
```

```
    uy += vy >> (m + 1);        /* cancel 1st-order error */
```

```
    for (i = (PIV2 << m) >> 16; i >= 0; --i) {
```

```
        x = (xJ + vx) >> 16;
```

```
        y = (yJ + vy) >> 16;
```

```
        draw_point(x, y);
```

```
        ux -= vx >> m;
```

```
        vx += ux >> m;
```

```
        uy -= vy >> m;
```








```
        vy += uy >> m;
```

```
    }
```

```
}
```

# Index of

## /pubs/tog/GraphicsGems/gems/2DClip/

Name	Last modified	Size	Description
 <a href="#">_Parent Directory</a>			
 <a href="#">_Makefile</a>	29-Jun-00 08:12	1K	
 <a href="#">_bio.c</a>	29-Jun-00 08:12	3K	
 <a href="#">_box.h</a>	29-Jun-00 08:12	1K	
 <a href="#">_clip.c</a>	29-Jun-00 08:12	2K	
 <a href="#">_cross.c</a>	29-Jun-00 08:12	6K	
 <a href="#">_line.h</a>	29-Jun-00 08:12	1K	

```
LIBFILE = ../gemslib.a

CFLAGS = $(GENCFLAGS) -I..

OFILES = clip.o bio.o cross.o

$(LIBFILE): $(OFILES)
    ar rcs $(LIBFILE) $(OFILES)

clean:
    /bin/rm -f clip.o bio.o cross.o

$(OFILES): line.h ../GraphicsGems.h
```

```
/*
 * file  bio.c
 *      contains the basic operations
 *
 */
#include      <stdio.h>
#include      "GraphicsGems.h"
#include      "line.h"

/*
 * def_contour
 *
 * Purpose:
 * add a contour to the list
 * NOTE: coordinates must already be converted into longs!
 *
 * x      x coordinates of the end points of segments
 * y      y coordinates of the end points of segments
 * n      number of coordinate pairs
 * no     contour number (id), does not have to be unique!
 * type   type of clip operation desired CLIP_NORMAL means
 *        clip everything inside the contour
 */
def_contour(x, y, n, no, type)
long  x[], y[];
int   n, no, type;
{
    short  i;
    long   dx1, dx2, dy1, dy2;
    long   minx, miny, maxx, maxy;
    CONTOUR *cp;
    SEGMENT *sp, *spp;

    if((cp = CL) == (CONTOUR *)NULL) {
        cp = CL = NEWTYPE(CONTOUR);
    }
    else {
        while(cp->_next != (CONTOUR *)NULL)
            cp = cp->_next;
        i = cp->_no;
        cp = cp->_next = NEWTYPE(CONTOUR);
    }

    cp->_next = (CONTOUR *)NULL;
    cp->_no = no;
    SET_ON(cp);
    if(type == CLIP_NORMAL)
        SET_INVERSE(cp);
    else
        SET_NORMAL(cp);
    minx = miny = 1000000;
    maxx = maxy = -1;
    for(i=0; i<n; i++) {
        if(i == 0) {
            cp->_s = sp = NEWTYPE(SEGMENT);
            sp->_from._x = x[0];
            sp->_from._y = y[0];
            sp->_to._x   = x[1];
            sp->_to._y   = y[1];
        }
    }
}
```

```
    else {
    /*
    * if necessary stretch the contour
    * and skip the point
    */
        dx1 = sp->_to._x - sp->_from._x;
        dy1 = sp->_to._y - sp->_from._y;
        dx2 = x[(i == n-1) ? 0 : i+1] - sp->_to._x;
        dy2 = y[(i == n-1) ? 0 : i+1] - sp->_to._y;
        if(dy2*dx1 == dy1*dx2) {
            sp->_to._x = x[(i == n-1) ? 0 : i+1];
            sp->_to._y = y[(i == n-1) ? 0 : i+1];
        }
        else {
            spp = sp;
            sp = sp->_next = NEWTYPE(SEGMENT);
            sp->_prev = spp;
            sp->_from._x = x[i];
            sp->_from._y = y[i];
            sp->_to._x = x[(i == n-1) ? 0 : i+1];
            sp->_to._y = y[(i == n-1) ? 0 : i+1];
        }
    }
}
```

```
/*
* calculate the enclosing box
*/
    if(x[i] < minx)
        minx = x[i];
    if(x[i] > maxx)
        maxx = x[i];
    if(y[i] < miny)
        miny = y[i];
    if(y[i] > maxy)
        maxy = y[i];
}
cp->_minx = minx;
cp->_maxx = maxx;
cp->_miny = miny;
cp->_maxy = maxy;
sp->_next = cp->_s;
cp->_s->_prev = sp;
cp->_next = (CONTOUR *)NULL;
}
```

```
/*
* get_contour_ptr
*
* PURPOSE
* get the pointer to a contour given its id
* with multiple id's first fit algorithm is
* used. Returns NULL in case of error.
*
* no      id of contour
*/
CONTOUR *get_contour_ptr(no)
int      no;
{
    CONTOUR *cp;
```

```
    if((cp = CL) == (CONTOUR *)NULL)
        return((CONTOUR *)NULL);
    else {
        while(cp != (CONTOUR *)NULL) {
            if(cp->_no == no)
                return(cp);
            else
                cp = cp->_next;
        }
        return((CONTOUR *)NULL);
    }
}
```

```
/*
 * del_contour
 *
 * PURPOSE
 * delete contour(s) from the list with id
 * no
 */
del_contour(no)
int no;
{
    CONTOUR *cp, *cpp;
    CONTOUR *qp = (CONTOUR *)NULL;
    SEGMENT *sp, *spp;

    if((cp = CL) == (CONTOUR *)NULL)
        return;
    while(cp != (CONTOUR *)NULL) {
        if(cp->_no == no) {
            sp = cp->_s;
            do {
                spp = sp->_next;
                free(sp);
                sp = spp;
            } while(sp != cp->_s);
            cpp = cp->_next;
            free(cp);
            if(qp)
                qp->_next = cpp;
            else
                CL = cpp;
            cp = cpp;
        }
        else {
            qp = cp;
            cp = cp->_next;
        }
    }
}
```



```
/*
 * file box.h
 *      a short include file is better then no include file
 */
typedef struct    {                /* guess what this is      */
    long    _lowx;
    long    _lowy;
    long    _highx;
    long    _highy;
} BOX;
```

```
/*
 * file clip.c
 *      contains the actual clipping routines
 */
#include      <stdio.h>
#include      "GraphicsGems.h"
#include      "line.h"

/*
 * vis_vector
 *
 *      PURPOSE
 *      actual user interface. Draws a clipped line
 *      NOTE: coordinates are given in converted LONGS!
 *
 *      xf, yf  from coordinates of vector to be drawn
 *      xt, yt  to coordinates of vector to be drawn
 */
vis_vector(xf, yf, xt, yt)
long      xf, yf, xt, yt;
{
    SEGMENT l;

    if(xf == xt && yf == yt)
        return;
    l._from._x = xf;
    l._from._y = yf;
    l._to._x   = xt;
    l._to._y   = yt;

/*
 * start at top of list
 */
    clip(CL, &l);
}

/*
 * clip
 *
 *      PURPOSE
 *
 *      p      pointer to polygon
 *      l      pointer to line segment
 */
clip(p, l)
CONTOUR *p;
SEGMENT *l;
{
    SEGMENT ss;
    CLIST  *sol;
    POINT  pt;
    boolean up, delay, inside, p_inside(), disjunct();
    int     i;
    short   nsol, nsmax = 2;

/*
 * list exhausted do what you like
 * we want to plot
 */
    if(p == (CONTOUR *)NULL) {
        move((l->_from._x), (l->_from._y));
```

```
        cont((l->_to._x), (l->_to._y));
        return;
    }
/*
 * polygon is switched off
 * take next one
 */
    if(!IS_ON(p)) {
        clip(p->_next, l);
        return;
    }
/*
 * comparison on basis of the
 * enclosing rectangle
 */
    if(disjunct(p, l)) {
        if(!IS_NORMAL(p)) {
            clip(p->_next, l);
        }
        return;
    }
/*
 * calculate possible intersections
 */
    sol = (CLIST *) calloc(2, sizeof(CLIST));
    sol[0]._p._x = l->_from._x;
    sol[0]._p._y = l->_from._y;
    sol[0]._type = STD;
    sol[1]._p._x = l->_to._x;
    sol[1]._p._y = l->_to._y;
    sol[1]._type = STD;
    nsol = 2;
    cross_calc(p, l, &sol, &nsol, nsmax);
    pt._x = sol[0]._p._x;
    pt._y = sol[0]._p._y;
/*
 * determine status of first point
 */
    inside = p_inside(p, &pt);
    if((!inside && IS_NORMAL(p)) || (inside && !IS_NORMAL(p)))
        up = TRUE;
    else
        up = FALSE;
    delay = FALSE;
/*
 * process list of intersections
 */
    for(i=1; i<nsol; i++) {
        if(!up) {
            ss._from._x = sol[i-1]._p._x;
            ss._from._y = sol[i-1]._p._y;
            ss._to._x = sol[i]._p._x;
            ss._to._y = sol[i]._p._y;
            clip(p->_next, &ss);
        }
        if(!delay) {
            if(sol[i]._type != DELAY)
                up = (up) ? FALSE : TRUE;
            else
                delay = TRUE;
        }
    }
}
```

```
        }
        else {
            up = (up) ? FALSE : TRUE;
            delay = FALSE;
        }
    }
    free(sol);
}

/*
 * disjunct
 *
 * PURPOSE
 * determine if the box enclosing the polygon
 * stored in p and the box enclosing the line
 * segment stored in l are disjunct.
 * Return TRUE if disjunct else FALSE
 *
 * p      points to the polygon structure
 * l      points to the line segment structure
 */
boolean disjunct(p, l)
CONTOUR *p;
SEGMENT *l;

{
    if((MAX(l->_from._x, l->_to._x) < p->_minx) ||
        (MIN(l->_from._x, l->_to._x) > p->_maxx) ||
        (MAX(l->_from._y, l->_to._y) < p->_miny) ||
        (MIN(l->_from._y, l->_to._y) > p->_maxy) )
        return(TRUE);
    else
        return(FALSE);
}

#define DEBUG
#ifdef DEBUG
move(x, y)
long x, y;
{
    printf("(%d,%d) ->", x, y);
}

cont(x, y)
long x, y;
{
    printf("(%d,%d)\n", x, y);
}

#endif
```

```
/*
 * file cross.c:
 *     calculate the intersections
 */
#include      <math.h>
#include      "GraphicsGems.h"
#include      "line.h"
#include      "box.h"
/*
 * cross_calc:
 *
 *     PURPOSE
 *     calculate the intersections between the polygon
 *     stored in poly and the line segment stored in l
 *     and put the intersections into psol.
 *
 *     poly    pointer to the structure containing the polygon
 *     l       pointer to the structure containing the line segment
 *     psol    pointer to the pointer where intersections are stored
 *     nsol    current number of intersections stored
 *     nsmax   maximum storage in psol for intersections
 *             if nsol exceeds nsmax additional storage is allocated
 */
cross_calc(poly, l, psol, nsol, nsmax)
CONTOUR *poly;
SEGMENT *l;
CLIST **psol;
short *nsol, nsmax;
{
    SEGMENT *p;
    CLIST *sol;
    double s;
    long x, y, a, b, c;
    int psort(), type;

    sol = *psol;
    p = poly->s;
    do {
/*
 * calculate the a, b and c coefficients and determine the
 * type of intersection
 */

        a = (p->_to._y - p->_from._y)*(l->_to._x - l->_from._x) -
            (p->_to._x - p->_from._x)*(l->_to._y - l->_from._y);
        b = (p->_from._x - l->_from._x)*(l->_to._y - l->_from._y) -
            (p->_from._y - l->_from._y)*(l->_to._x - l->_from._x);
        c = (p->_from._x - l->_from._x)*(p->_to._y - p->_from._y) -
            (p->_from._y - l->_from._y)*(p->_to._x - p->_from._x);
        if(a == 0)
            type = (b == 0) ? COINCIDE : PARALLEL;
        else {
            if(a > 0) {
                if((b >= 0 && b <= a) &&
                    (c >= 0 && c <= a))
                    type = CROSS;
                else
                    type = NO_CROSS;
            }
        }
    }
}
```

```
        else {
            if((b <= 0 && b >= a) &&
               (c <= 0 && c >= a))
                type = CROSS;
            else
                type = NO_CROSS;
        }
    }

/*
 * process the intersection found
 */
    switch(type) {
        case NO_CROSS: case PARALLEL:
            break;

        case CROSS:
            if(b == a || c == a || c == 0)
                break;
            if(b == 0 &&
               p_where(&(p->_prev->_from), &(p->_to), 1) >= 0)
                break;
            s = (double)b/(double)a;
            if(l->_from._x == l->_to._x)
                x = l->_from._x;
            else
                x = p->_from._x +
                    (int)((p->_to._x - p->_from._x)*s);
            if(l->_from._y == l->_to._y)
                y = l->_from._y;
            else
                y = p->_from._y +
                    (int)((p->_to._y - p->_from._y)*s);

            if(*nsol == nsmax) {
                nsmax *= 2;
                *psol = sol = (CLIST *) realloc(sol,
nsmax*sizeof(CLIST));
            }
            sol[*nsol]._p._x = x;
            sol[*nsol]._p._y = y;
            sol[*nsol]._type = STD;
            *nsol += 1;
            break;

        case COINCIDE:
            if(p_where(&(p->_prev->_from),
                       &(p->_next->_to), 1) > 0)
                break;
            if(l->_from._x != l->_to._x) {
                if((MAX(l->_from._x, l->_to._x) <
                    MIN(p->_from._x, p->_to._x) ) ||
                   (MIN(l->_from._x, l->_to._x) >
                    MAX(p->_from._x, p->_to._x) ) )
                    break;
                if(MIN(l->_from._x, l->_to._x) <
                   MIN(p->_from._x, p->_to._x) ) {
                    if(*nsol == nsmax) {
                        nsmax *= 2;
                        *psol = sol = (CLIST *)
realloc(sol,
nsmax*sizeof(CLIST));
                    }
                }
            }
    }
}
```

```

    }
    sol[*nsol]._p._x = p->_from._x;
    sol[*nsol]._p._y = p->_from._y;
    sol[*nsol]._type = DELAY;
    *nsol += 1;
}
if(MAX(l->_from._x, l->_to._x) >
    MAX(p->_from._x, p->_to._x) ) {
    if(*nsol == nsmax) {
        nsmax *= 2;
        *psol = sol = (CLIST *)
realloc(sol,
nsmax*sizeof(CLIST));

    }
    sol[*nsol]._p._x = p->_to._x;
    sol[*nsol]._p._y = p->_to._y;
    sol[*nsol]._type = DELAY;
    *nsol += 1;
}
}
else {
    if((MAX(l->_from._y, l->_to._y) <
        MIN(p->_from._y, p->_to._y) ) ||
        (MIN(l->_from._y, l->_to._y) >
        MAX(p->_from._y, p->_to._y)) )
        break;
    if(MIN(l->_from._y, l->_to._y) <
        MIN(p->_from._y, p->_to._y) ) {
        if(*nsol == nsmax) {
            nsmax *= 2;
            *psol = sol = (CLIST *)
realloc(sol,
nsmax*sizeof(CLIST));

        }
        sol[*nsol]._p._x = p->_from._x;
        sol[*nsol]._p._y = p->_from._y;
        sol[*nsol]._type = DELAY;
        *nsol += 1;
    }
    if(MAX(l->_from._y, l->_to._y) >
        MAX(p->_from._y, p->_to._y) ) {
        if(*nsol == nsmax) {
            nsmax *= 2;
            *psol = sol = (CLIST *)
realloc(sol,
nsmax*sizeof(CLIST));

        }
        sol[*nsol]._p._x = p->_to._x;
        sol[*nsol]._p._y = p->_to._y;
        sol[*nsol]._type = DELAY;
        *nsol += 1;
    }
}
}
break;
}
p = p->_next;
} while(p != poly->_s);
qsort(sol, *nsol, sizeof(CLIST), psort);
}
```

```
/*
 * p_where
 *
 * PURPOSE
 * determine position of point p1 and p2 relative to
 * linesegment l.
 * Return value
 * < 0    p1 and p2 lie at different sides of l
 * = 0    one of both points lie on l
 * > 0    p1 and p2 lie at same side of l
 *
 * p1      pointer to coordinates of point
 * p2      pointer to coordinates of point
 * l       pointer to linesegment
 */
p_where(p1, p2, l)
POINT    *p1, *p2;
SEGMENT *l;
{
    long    dx, dy, dx1, dx2, dy1, dy2, p_1, p_2;

    dx  = l->_to._x - l->_from._x;
    dy  = l->_to._y - l->_from._y;
    dx1 = p1->_x - l->_from._x;
    dy1 = p1->_y - l->_from._y;
    dx2 = p2->_x - l->_to._x;
    dy2 = p2->_y - l->_to._y;
    p_1 = (dx*dy1 - dy*dx1);
    p_2 = (dx*dy2 - dy*dx2);
    if(p_1 == 0 || p_2 == 0)
        return(0);
    else {
        if((p_1 > 0 && p_2 < 0) || (p_1 < 0 && p_2 > 0))
            return(-1);
        else
            return(1);
    }
}
```

```
/*
 * p_inside
 *
 * PURPOSE
 * determine if the point stored in pt lies inside
 * the polygon stored in p
 * Return value:
 * FALSE    pt lies outside p
 * TRUE     pt lies inside  p
 *
 * p        pointer to the polygon
 * pt       pointer to the point
 */
boolean p_inside(p, pt)
CONTOUR *p;
POINT    *pt;
{
    SEGMENT l;
    CLIST    *sol;
```



```
    short    nsol = 0, nsmax = 2, reduce = 0, i;
    boolean on_contour(), odd;

    l._from._x = p->_minx-2;
    l._from._y = pt->_y;
    l._to._x   = pt->_x;
    l._to._y   = pt->_y;
    sol = (CLIST *) calloc(2, sizeof(CLIST));
    cross_calc(p, &l, &sol, &nsol, nsmax);
    for(i=0; i<nsol-1; i++)
        if(sol[i]._type == DELAY && sol[i+1]._type == DELAY)
            reduce++;

    free(sol);
    odd = (nsol - reduce) & 0x01;
    return(odd ? !on_contour(p, pt) : FALSE);
}
```

```
/*
 * function used for sorting
 */
```

```
pshort(p1, p2)
CLIST *p1, *p2;
{
    if(p1->_p._x != p2->_p._x)
        return(p1->_p._x - p2->_p._x);
    else
        return(p1->_p._y - p2->_p._y);
}
```

```
/*
 * on_contour
 *
 * PURPOSE
 * determine if the point pt lies on the
 * contour p.
 * Return value
 * TRUE    point lies on contour
 * FALSE   point lies not on contour
 *
 * p       pointer to the polygon structure
 * pt      pointer to the point
 */
```

```
boolean on_contour(p, pt)
CONTOUR *p;
POINT *pt;
{
    SEGMENT *sp;
    long    dx1, dy1, dx2, dy2;

    sp = p->_s;
    do {
        if((pt->_x >= MIN(sp->_from._x, sp->_to._x)) &&
            (pt->_x <= MAX(sp->_from._x, sp->_to._x)) ) {
            dx1 = pt->_x - sp->_from._x;
            dx2 = sp->_to._x - pt->_x;
            dy1 = pt->_y - sp->_from._y;
            dy2 = sp->_to._y - pt->_y;
            if(dy1*dx2 == dy2*dx1)
                return(TRUE);
        }
    }
```

```
        sp = sp->_next;
    } while(sp != p->_s);
    return(FALSE);
```

```
}
```

```
/*
Two-Dimensional Clipping: A Vector Based Approach
by Hans Spoelder and Fons Ullings
from "Graphics Gems", Academic Press, 1990
*/

/*
 * file line.h
 * contains major definitions for the clipping routines
 */

#define NFAC          10                /* discrete measure */

#define SCALE          (1 << NFAC)      /* 1024 points/cm */
#define TO_INT(X)      ((int)((X)*SCALE))
#define TO_FLT(X)      (((float)(X))/SCALE)

#define COINCIDE        1                /* what do the lines do */
#define PARALLEL        2
#define CROSS           3
#define NO_CROSS        4

#define STD              0                /* crossing types */
#define DELAY           1

#define CLIP_NORMAL     1

typedef struct {                        /* holds a point */
    long    _x;                        /* holds x coordinate */
    long    _y;                        /* holds y coordinate */
} POINT;

typedef struct {                        /* holds a cross point */
    POINT    _p;                        /* holds the solution */
    short    _type;                    /* more information */
} CLIST;

struct segment {                        /* holds a segment */
    POINT    _from;                    /* start coordinates */
    POINT    _to;                      /* stop coordinates */
    struct segment *_next;
    struct segment *_prev;
};

#define SEGMENT          struct segment

struct contour {                        /* holds a contour */
    short    _no;                      /* contour counter */
    short    _status;                  /* holds information */
    short    _cnt;                     /* number of elements */
    SEGMENT *_s;                       /* the segments */
    struct contour *_next; /* linked list */
    long     _minx;                    /* coordinates of box */
    long     _miny;
    long     _maxx;
    long     _maxy;
};
```

```
#define CONTOUR          struct contour

#define ACTIVE           01          /* polygon attributes   */
#define NORMAL           02

#define SET_ON(p)        ((p)->_status |= ACTIVE)
#define SET_NORMAL(p)    ((p)->_status |= NORMAL)

#define SET_OFF(p)       ((p)->_status &= ~ACTIVE)
#define SET_INVERSE(p)   ((p)->_status &= ~NORMAL)

#define IS_ON(p)         ((p)->_status & ACTIVE)
#define IS_NORMAL(p)     ((p)->_status & NORMAL)

extern  CONTOUR  *CL;

CONTOUR  *get_contour_ptr();

extern  short    C_COUNT;
```

```
#include <math.h>
#include <stdlib.h>                /* for qsort */

#define START 0
#define END 1

#define QUAD1  90.0
#define QUAD2 180.0
#define QUAD3 270.0
#define QUAD4 360.0
#define FACTOR 57.29577951

typedef struct {
    float angle;
    int type;
} intsct_st;

int compare(intsct_st *, intsct_st *); /* used by qsort */

/*
 * clip_circle:
 *   clips a circle with center (Xc,Yc) and radius R to the box
 *   given by clip_bnds.
 *   the function return value indicates the number of segments
 *   after clipping.
 *   the start and end angle values of the visible segments are
 *   returned in ang_st and ang_en.
 */

int clip_circle( float Xc, float Yc, /* center of the circle */
                 float R,          /* radius of the circle */
                 float clip_bnds[], /* clip boundary:
                                     [left, upper, right, lower] */
                 float ang_st[],    /* start angles for visible arcs */
                 float ang_en[] )   /* end angles for visible arcs */
{
    float alpha, beta, gamma, delta;
    float circle_bnds[4];
    int i, n, num_sector;
    int overlap;
    float d;
    intsct_st intsct[20];
    float prev;

    /*
     * find the bounding box of the circle
     */
    circle_bnds[0] = Xc + R;
    circle_bnds[1] = Yc + R;
    circle_bnds[2] = Xc - R;
    circle_bnds[3] = Yc - R;

    /*
     * do a bounding box check to see if the circle is completely
     * clipped out
     */
    if (circle_bnds[2] > clip_bnds[0] ||
        circle_bnds[0] < clip_bnds[2] ||
        circle_bnds[3] > clip_bnds[1] ||
        circle_bnds[1] < clip_bnds[3])
        return 0;
```

```
alpha=beta=gamma=delta=0;
n = 0;

if( circle_bnds[0] > clip_bnds[0] )
/*
 * the right boundary is crossed
 */
{
    d = (float) ( clip_bnds[0] - Xc );
    alpha = (int) FACTOR*acos(d/R);

    intsct[n].angle = 0; intsct[n++].type = START;
    intsct[n].angle = alpha; intsct[n++].type = END;
    intsct[n].angle = QUAD4-alpha; intsct[n++].type = START;
    intsct[n].angle = QUAD4; intsct[n++].type = END;
}

if( circle_bnds[1] > clip_bnds[1] )
/*
 * the upper boundary is crossed
 */
{
    d = (float) ( clip_bnds[1] - Yc);
    beta = (int) FACTOR*acos(d/R);

    if ( (QUAD1-beta) < 0 )
    {
        intsct[n].angle = QUAD4+QUAD1-beta; intsct[n++].type = START;
        intsct[n].angle = QUAD4; intsct[n++].type = END;
        intsct[n].angle = 0; intsct[n++].type = START;
    }
    else
    {
        intsct[n].angle = QUAD1-beta; intsct[n++].type = START;
    }
    intsct[n].angle = QUAD1+beta; intsct[n++].type = END;
}

if( circle_bnds[2] < clip_bnds[2] )
/*
 * the left boundary is crossed
 */
{
    d = (float) ( Xc - clip_bnds[2] );
    gamma = (int) FACTOR*acos(d/R);

    intsct[n].angle = QUAD2-gamma; intsct[n++].type = START;
    intsct[n].angle = QUAD2+gamma; intsct[n++].type = END;
}

if( circle_bnds[3] < clip_bnds[3] )
/*
 * the lower boundary is crossed
 */
{
    d = (float) ( Yc - clip_bnds[3] );
    delta = (int) FACTOR*acos(d/R);

    intsct[n].angle = QUAD3-delta; intsct[n++].type = START;
    if ( (QUAD3+delta) > QUAD4 )
```

```
{
    intsct[n].angle = QUAD4; intsct[n++].type = END;
    intsct[n].angle = 0; intsct[n++].type = START;
    intsct[n].angle = QUAD3+delta-QUAD4; intsct[n++].type = END;
}
else
{
    intsct[n].angle = QUAD3+delta; intsct[n++].type = END;
}
}
/*
 * the complete circle is visible if n = 0
 */
if (n == 0 )
{
    ang_st[0] = 0;
    ang_en[0] = QUAD4;
    return 1;
}
/*
 * Sort all events in increasing order of angles
 */
qsort ((void*)intsct, (size_t) n, sizeof(intsct_st), compare);

/*
 * Extract the visible sectors
 */
num_sector = 0; overlap = 0; prev = 0;

for (i=0; i<n; i++)
{
    if (overlap == 0)
        if (intsct[i].angle > prev)
        {
            ang_st[num_sector] = prev;
            ang_en[num_sector++] = intsct[i].angle;
        }

    if (intsct[i].type == START)
        overlap++;
    else
        overlap--;

    prev = intsct[i].angle;
}

if (prev < QUAD4)
{
    ang_st[num_sector] = prev;
    ang_en[num_sector++] = QUAD4;
}
return num_sector;
}

int compare(intsct_st *a, intsct_st *b)
{
    if (a->angle < b->angle)
        return -1;
    else if (a->angle == b->angle)
        return 0;
    else
```

```
    return 1;
```

```
}
```



# Index of

## /pubs/tog/GraphicsGems/gemsiv/graph\_layout/

Name	Last modified	Size	Description
 <a href="#">Parent Directory</a>			
 <a href="#">Makefile</a>	29-Jun-00 08:20	1K	
 <a href="#">README</a>	29-Jun-00 08:20	7K	
 <a href="#">defines.h</a>	29-Jun-00 08:19	1K	
 <a href="#">fileio.C</a>	29-Jun-00 08:19	5K	
 <a href="#">fileio.hxx</a>	29-Jun-00 08:19	1K	
 <a href="#">g.dat</a>	29-Jun-00 08:19	1K	
 <a href="#">g20.dat</a>	29-Jun-00 08:19	2K	
 <a href="#">graph.C</a>	29-Jun-00 08:19	26K	
 <a href="#">graph.hxx</a>	29-Jun-00 08:19	6K	
 <a href="#">graph.mak</a>	29-Jun-00 08:19	1K	
 <a href="#">layout.C</a>	29-Jun-00 08:20	9K	
 <a href="#">mswin.gen</a>	29-Jun-00 08:20	1K	
 <a href="#">mswindow.C</a>	29-Jun-00 08:20	7K	
 <a href="#">mswindow.hxx</a>	29-Jun-00 08:20	3K	
 <a href="#">vector.C</a>	29-Jun-00 08:20	2K	
 <a href="#">vector.hxx</a>	29-Jun-00 08:20	1K	
 <a href="#">window.C</a>	29-Jun-00 08:20	4K	
 <a href="#">window.hxx</a>	29-Jun-00 08:20	3K	

```
#
# makefile to build program graph layout
#

.C.o:
    $(CC) -c $(CFLAGS) $<

#
# Define Objects
#
OBJS= window.o graph.o layout.o vector.o fileio.o

#
# define build flags
#
CC = gcc
INCLUDES= -I/usr/include/X11R4
CFLAGS = -g $(INCLUDES)
LDFLAGS= -L/usr/lib/X11R4
LIBS= -lX11 -lm

#
# Dependencies
#
all : graph

window.o: window.C window.hxx

graph.o: graph.C window.hxx vector.hxx defines.h graph.hxx

layout.o: layout.C window.hxx vector.hxx defines.h graph.hxx

vector.o: vector.C vector.hxx

fileio.o: fileio.C window.hxx vector.hxx defines.h graph.hxx fileio.hxx

graph: $(OBJS)
    $(CC) $(LDFLAGS) $(OBJS) $(LIBS) -o $@
```

C++ code from the article  
"Dynamic Layout Algorithm to Display General Graphs"  
by Laszlo Szirmay-Kalos, szirmay@fsz.bme.hu  
in "Graphics Gems IV", Academic Press, 1994

Installation of the Dynamic Layout Program ( graph )  
=====

This program can be installed under UNIX/X-WINDOW or under  
MS-WINDOWS environment.

To install under UNIX/X-WINDOWS use the make utility,  
-----  
which generates the executable program called "graph" relying on the Makefile:

make -f Makefile

Makefile supposes:

- gcc (gnu C++ compiler)
- XLib header files in /usr/include/X11R4
- XLib library in /usr/lib/X11R4

If your system has the XLib header files and libraries in different  
locations please modify the INCLUDES and LDFLAGS flags in the  
Makefile. Since the program has been written to meet the AT&T C++  
standard, even if you have a C++ compiler other than gcc, you can  
expect to compile the program without difficulties after modifying the  
CC and CFLAGS. For the sake of maximum compatibility the program does  
not take advantage of advanced widget sets ( such as Motif ), neither  
does it require color screen.

To install under DOS/MS-WINDOWS 3.x ,  
-----  
a UNIX shell script (mswin.gen) should be invoked first,  
which converts the file names according to the requirements of DOS  
and Borland C++ compiler. This shell script expects a directory as command  
line argument and will place the generated file to the given directory:

mswin.gen mswin

will generate the files in mswin directory.

Having generated the files, they should be transferred to a  
DOS/MS-WINDOWS system and the make utility of the borland C++ 3.1  
compiler can be used to generate the executable file ( called graph ),  
which will run under MS-WINDOWS environment.

The information file of the make is called graph.mak here, which supposes:  
- borland C++ compiler, version 3.1

Running the graph program:  
=====

Under UNIX/X-WINDOW the program can be started by typing "graph" with a  
file name argument referring to the definition of a graph:

graph g20.dat

Under DOS/MS-WINDOWS you should also start MS-WINDOWS as for example:

```
win graph g20.dat
```

Two sample files are included in this package: g.dat, g20.dat. In addition to that, you can also define graphs in the format described below.

Having started the program, it will display the original arrangement of the graph on the screen, which can be altered by the layout algorithm by pressing the key <L>, or can be arranged randomly by pressing the key <R>. Key <Q> gets the program to quit, while saving the last arrangement in file ggg.dat.

Files of the program  
=====

The complete file set consists of the following files:

1. C++ Source files:

- layout.C = Dynamic layout and Initial Placement algorithms
- fileio.C = File I/O operations
- graph.C = Manipulation of Graph data structure and event handlers
- vector.C = 2D vector operations
- window.C = class library to interface XLib
- mswindow.C = class library to interface MS-WINDOWS

2. C++ Header files:

- defines.h
- fileio.hxx
- graph.hxx
- vector.hxx
- window.hxx
- mswindow.hxx

3. README files:

- README = you are reading this file!

4. Program generation files:

- Makefile = makefile for X-WINDOWS
- graph.mak = makefile for MS-WINDOWS
- mswin.gen = UNIX shell script to generate dos file names:

- layout.C -> layout.cpp
- fileio.C -> fileio.cpp
- graph.C -> graph.cpp
- vector.C -> vector.cpp
- mswindow.C -> mswindow.cpp
- defines.h -> defines.h
- fileio.hxx -> fileio.hxx
- graph.hxx -> graph.hxx
- vector.hxx -> vector.hxx
- mswindow.hxx -> mswindow.hxx
- graph.mak -> graph.mak
- \*.dat -> \*.dat

5. Sample Data files for the definition of graphs

- g.dat
- g20.dat

## Definitions of graph description files

=====

The layout program expects the input data in a text file and also generates output file in the same format during the termination. This file defines a weighted graph to be arranged in a easy-to-understand, programming language like way.

The definition language consists of

keywords: NAME, POSITION, TYPE, RELATIONS, OF, NODE, RELATION, RELATED, TO,  
----- WITH, INTENSITY, END, MOVEABLE, FIXED  
Keywords should be specified by capital letters.

operators:

-----  
= :

string variables:

-----  
Any ASCII character string, having maximal length of 10 and which does not contain characters "= : # space tab newline"

real variables:

-----  
For positions the range (0..1000.0, 0..1000.0) is allowed and for weights (relation intensity) the range (-10.0..10.0) is permitted.

comments:

-----  
The characters following a # character as far as the end of line are assumed to form a comment

Keywords, variables and comments should be separated by whitespace characters (space, tabulator, newline) or operators.

The basic structure of Graph definition File:

```
node definition
node definition
.
.
.
node definition

node relation definition
node relation definition
.
.
.
node relation definition
```

node definition ( position is optional ):

NAME = nodename [POSITION = position] TYPE = nodetype

nodename:

Any ASCII character string, having maximal length of 10 and which does not contain characters "= : # space newline". This name must be unique, that is only one node can have it.

position:

Two real values in the range of (0..1000.0)

nodetype:

FIXED or MOVEABLE

node relation definition:

RELATIONS OF nodename NODE

relation definition

relation definition

.

.

.

relation definition

END

nodename:

Name of already declared node in node definition

relation definition:

RELATION relation : RELATED TO node WITH INTENSITY intensity

relation:

Any ASCII character string, having maximal length of 10 and which does not contain characters "= : # space newline"

The name \* means that no name is specified for this relation.

node:

Name of already declared node in node definition

intensity:

A real value in the range of (-10.0 .. 10.0)

A relation between two nodes can be defined in the file several times. In this case the last definition is taken into consideration.

SAMPLE INPUT FILE

-----

#

# DEMO RELATION GROUP - I am a comment

#

#

# DECLARATIONS OF NODES

#

NAME = n1 POSITION = 764.2 216.0 TYPE = FIXED

NAME = n2 POSITION = 131.3 858.6 TYPE = FIXED

NAME = n3 TYPE = MOVEABLE # POSITION = x,y is optional

NAME = n4 TYPE = MOVEABLE

NAME = n5 TYPE = MOVEABLE

NAME = n6 TYPE = MOVEABLE

NAME = n7 TYPE = MOVEABLE

NAME = n8 TYPE = MOVEABLE

NAME = n9 POSITION = 688.0 587.1 TYPE = MOVEABLE

```
#
#      DECLARATIONS OF RELATIONS
#
RELATIONS OF n1 NODE
RELATION r12 : RELATED TO n2 WITH INTENSITY 3.000000
END

RELATIONS OF n2 NODE
RELATION r23 : RELATED TO n3 WITH INTENSITY 2.000000
END

RELATIONS OF n3 NODE
RELATION r31 : RELATED TO n1 WITH INTENSITY 6.000000
END

RELATIONS OF n4 NODE
RELATION r42 : RELATED TO n2 WITH INTENSITY 4.000000
END

RELATIONS OF n5 NODE
RELATION r53 : RELATED TO n3 WITH INTENSITY 3.000000
#
# * relation name means no name
#
RELATION * : RELATED TO n8 WITH INTENSITY 3.000000
END

RELATIONS OF n7 NODE          # empty relation is also allowed
END

RELATIONS OF n9 NODE
#
# * relation name means no name
#
RELATION * : RELATED TO n8 WITH INTENSITY 3.000000
END
```

```

/*****
**      TEST FILE FOR graph (Dynamic Layout Alg)
**
**      HEADER      - DEFINITION OF CONSTANTS
**
** Author: dr. Szirmay-Kalos Laszlo (szirmay@fsz.bme.hu)
**      Technical University of Budapest, Hungary
*****/
/*
*      DEFINITION OF OBJECT WINDOW PARAMETERS
*/
#define OVERWINDOW_X          1000.0
#define OVERWINDOW_Y          1000.0
#define WALL_MARGIN            (OVERWINDOW_X / 10.0)

/*
*      MAXIMAL ALLOWABLE RELATION OF TWO NODES
*/
#define MAXRELATION            10.0

/*
*      RETURN OF THE LAYOUT ALGORITHM
*/
#define STOPPED                0
#define INSTABLE                1
#define TOO_LONG                2

/*
*      TYPE OF NODES
*/
#define MOVEABLE_NODE          0
#define FIXED_NODE              1

#define ALL_NODES               -1

/*
*      LOOK OF NODES
*/
#define NODESIZE_X              30
#define NODESIZE_Y              30

/*
*      MAXIMAL SIZE OF STRINGS
*/
#define MAXNAME                  10
#define MAXSTRING                10

/*
*      RETURN OF SEARCH FUNCTIONS
*/
#define EMPTY_LIST              0
#define FIRST_FOUND              1
#define FOUND                    2
#define NOT_FOUND                3

```



```

/*****
**      TEST FILE FOR graph (Dynamic Layout Alg)
**
**      MODUL      - BUFFERED FILE INPUT WITH SYNTAX CHECK
**                  AND NOT BUFFERED FILE OUTPUT
**
** Author: dr. Szirmay-Kalos Laszlo (szirmay@fsz.bme.hu)
**          Technical University of Budapest, Hungary
*****/

#ifdef MSWINDOWS
#include "fileio.hxx"
#else
#include "fileio.hxx"
#endif

/*----- Get -----*/
/* Buffered character input from the opened file */
/* IN  : character address */
/* OUT : char is put to the address */
/*      ret = was it succesful */
/*-----*/
BOOL FileIO :: Get( char * pc )
{
    if ( buffpt == nbytes ) {
        if ( (nbytes = fread( buffer, 1, BUFFERSIZE, file)) <= 0) return FALSE;
        else
            buffpt = 0;
    }
    *pc = buffer[ buffpt++ ];
    return TRUE;
}

/*----- GetString -----*/
/* Gets a string from a opened file. String ends are space, tab */
/* EOL(\n) and operators( =, : ). Parts between # end EOL are */
/* ignored ( comments ) */
/* IN  : string buffer adress and maximal length */
/* OUT : was it succesful ? */
/*      string is put to the address s */
/*-----*/
BOOL FileIO :: GetString ( pchar s, int maxlength )
{
    int  ichar = 0;
    char last_char;
    BOOL iscomment = FALSE;

    for ( ; ; ) {
/*
*      GET CHARACTER
*/
        if ( !Get ( &last_char ) ) return FALSE;

        switch ( last_char ) {
/*
*      COMMENT
*/
            case '#':
                iscomment = TRUE;
                break;

/*
*      OPERATORS

```

```

*/
    case ':':
    case '=':
        if ( !iscomment ) {
            if (ichar > 0) UnGet();           // unget operator
            else          s[ichar++] = last_char;
            s[ichar] = '\\0';
            return TRUE;
        } else break;

/*
*   SEPARATORS
*/

    case '\\n':
        iscomment = FALSE;
        line_count++;
    case '\\t':
    case ' ':
        if ( !iscomment ) {
            if ( ichar > 0 ) {
                s[ichar] = '\\0';
                return TRUE;
            }
        }
        break;

/*
*   KEYWORDS AND VARIABLES
*/

    default:
        if ( !iscomment ) {
            if ( ichar == maxlength ) {
                return FALSE;
            } else {
                s[ichar++] = last_char;
                break;
            }
        } else break;
    }
}

/*-----      OpenFile      -----*/
/* Opens a TEXT file for the defined operation          */
/* IN  : file name                                     */
/* OUT : was it succesful ?                           */
/* SIDE EFFECT: - internal file descriptor is created end line */
/*               is initialized                         */
/*-----*/
BOOL FileIO :: OpenFile( pchar name )
{
    line_count = 1;
    buffpt = 0;
    nbytes = 0;

    if ( (file = fopen( name, operation )) == NULL ) {
        return FALSE;
    }
    return TRUE;
}

/*-----      CloseFile      -----*/
/* Closes the opened file          */

```

```
/*-----*/
void FileIO :: CloseFile ( )
{
    fclose ( file );
}

/*----- GetKeyword -----*/
/* Gets a string and compares with the pattern keyword */
/* IN : pattern keyword */
/* OUT : is the specified pattern received ? */
/*-----*/
BOOL FileIO :: GetKeyword ( pchar key )
{
    if ( !GetString( s, MAXSTRING ) ) return FALSE;
    return GetKeyAgain( key );
}

/*----- GetKeyAgain -----*/
/* Gets the last inspected string again and compares with the */
/* pattern keyword */
/* IN : pattern keyword */
/* OUT : is the specified pattern received ? */
/*-----*/
BOOL FileIO :: GetKeyAgain ( pchar key )
{
    if ( strcmp( key, s ) == 0 ) return TRUE;
    else {
        return FALSE;
    }
}

/*----- GetVariable -----*/
/* Gets a double variable and compares to the specified range */
/* IN : address of var and min, max of range */
/* OUT : was it succesful, and the received var */
/*-----*/
BOOL FileIO :: GetVariable ( double *pv, double minv, double maxv )
{
    if ( !GetString(s,MAXLINE) ) return FALSE;

    if ( sscanf(s,"%lf", pv ) != 1 ) {
        return FALSE;
    } else {
        if ( *pv < minv || *pv > maxv ) {
            return FALSE;
        }
        return TRUE;
    }
}

/*----- GetOperator -----*/
/* Gets a non-white character and compares with the specified */
/* operator */
/* IN : specified operator */
/* OUT : was it the specified operator */
/*-----*/
BOOL FileIO :: GetOperator ( char op )
{
    if ( !GetString( s, 1 ) ) return FALSE;
    if ( s[0] != op ) {
        return FALSE;
    }
}
```

```
    }
    return TRUE;
}

/*----- PutString -----*/
/* Writes the given string to the file without buffering */
/* IN : specified string */
/* OUT : was it succesful */
/*-----*/
BOOL FileIO :: PutString ( pchar s )
{
    if ( fwrite( s, 1, strlen(s), file ) == 0 ) return FALSE;
    else return TRUE;
}
```

```

/*****
**      TEST FILE FOR graph (Dynamic Layout Alg)
**
**      HEADER      - BUFFERED FILE INPUT WITH SYNTAX CHECK
**                   AND NOT BUFFERED FILE OUTPUT
**
** Author: dr. Szirmay-Kalos Laszlo (szirmay@fsz.bme.hu)
**          Technical University of Budapest, Hungary
*****/

#ifdef MSWINDOWS
#include "graph.hxx"
#else
#include "graph.hxx"
#endif

#define BUFFERSIZE      2048
#define MAXLINE         80

class FileIO {
    FILE *   file;                // file descriptor
    char     operation[4];        // operation for open
    int      line_count;          // number of inputed lines
    int      error;               // error code
    char     s[MAXLINE + 1];      // last string
    int      buffpt;              // buffer indes
    int      nbytes;              // number of valid bytes in buffer
    char     buffer[BUFFERSIZE];  // io buffer

protected:
    BOOL     Get( pchar );        // get char and advance
    void     UnGet( void ) { buffpt--; } // unget last char

public:
    FileIO( char * op ) { strcpy( operation, op ); }
    BOOL     OpenFile ( pchar );  // open file having that name
    BOOL     GetString ( pchar, int ); // get a string from the file
    BOOL     GetKeyWord ( pchar ); // get a string and compare to a key
    BOOL     GetKeyAgain ( pchar ); // get last string again and compare
    BOOL     GetVariable ( double *, double, double ); // get a double variable
    BOOL     GetOperator ( char ); // get a char and compare to the given char
    int      GetLineNum ( void ) { return line_count; }
    BOOL     PutString ( pchar );  // write string to the file
    void     CloseFile ( void );   // close file
};
```

```
#
#      DEMO RELATION GROUP - I am a comment
#

#
#      DECLARATIONS OF NODES
#
NAME = n1 POSITION = 764.2 216.0 TYPE = FIXED
NAME = n2 POSITION = 131.3 858.6 TYPE = FIXED
NAME = n3 TYPE = MOVEABLE # POSITION = x,y is optional
NAME = n4 TYPE = MOVEABLE
NAME = n5 TYPE = MOVEABLE
NAME = n6 TYPE = MOVEABLE
NAME = n7 TYPE = MOVEABLE
NAME = n8 TYPE = MOVEABLE
NAME = n9 POSITION = 688.0 587.1 TYPE = MOVEABLE

#
#      DECLARATIONS OF RELATIONS
#
RELATIONS OF n1 NODE
RELATION r17 : RELATED TO n7 WITH INTENSITY 3.000000
END

RELATIONS OF n2 NODE
RELATION r26 : RELATED TO n6 WITH INTENSITY 2.000000
END

RELATIONS OF n3 NODE
RELATION r31 : RELATED TO n1 WITH INTENSITY 6.000000
END

RELATIONS OF n4 NODE
RELATION r42 : RELATED TO n2 WITH INTENSITY 4.000000
RELATION r46 : RELATED TO n6 WITH INTENSITY 1.000000
END

RELATIONS OF n5 NODE
RELATION r53 : RELATED TO n3 WITH INTENSITY 3.000000
#
# * relation name means no name
#
RELATION * : RELATED TO n8 WITH INTENSITY 3.000000
RELATION * : RELATED TO n6 WITH INTENSITY 1.000000
RELATION * : RELATED TO n4 WITH INTENSITY 3.000000
END

RELATIONS OF n7 NODE # empty relation is also allowed
END

RELATIONS OF n9 NODE
#
# * relation name means no name
#
RELATION * : RELATED TO n8 WITH INTENSITY 3.000000
END
```

```
NAME = 0 TYPE = MOVEABLE
NAME = 1 TYPE = MOVEABLE
NAME = 2 TYPE = MOVEABLE
NAME = 3 TYPE = MOVEABLE
NAME = 4 TYPE = MOVEABLE
NAME = 5 TYPE = MOVEABLE
NAME = 6 TYPE = MOVEABLE
NAME = 7 TYPE = MOVEABLE
NAME = 8 TYPE = MOVEABLE
NAME = 9 TYPE = MOVEABLE
NAME = 10 TYPE = MOVEABLE
NAME = 11 TYPE = MOVEABLE
NAME = 12 TYPE = MOVEABLE
NAME = 13 TYPE = MOVEABLE
NAME = 14 TYPE = MOVEABLE
NAME = 15 TYPE = MOVEABLE
NAME = 16 TYPE = MOVEABLE
NAME = 17 TYPE = MOVEABLE
NAME = 18 TYPE = MOVEABLE
NAME = 19 TYPE = MOVEABLE
```

RELATIONS OF 0 NODE

```
RELATION * : RELATED TO 1 WITH INTENSITY 2.000000
RELATION * : RELATED TO 2 WITH INTENSITY 2.000000
RELATION * : RELATED TO 3 WITH INTENSITY 2.000000
RELATION * : RELATED TO 4 WITH INTENSITY 2.000000
RELATION * : RELATED TO 5 WITH INTENSITY 2.000000
RELATION * : RELATED TO 6 WITH INTENSITY 2.000000
RELATION * : RELATED TO 7 WITH INTENSITY 2.000000
RELATION * : RELATED TO 8 WITH INTENSITY 2.000000
RELATION * : RELATED TO 9 WITH INTENSITY 2.000000
RELATION * : RELATED TO 10 WITH INTENSITY 2.000000
RELATION * : RELATED TO 11 WITH INTENSITY 2.000000
RELATION * : RELATED TO 12 WITH INTENSITY 2.000000
RELATION * : RELATED TO 13 WITH INTENSITY 2.000000
RELATION * : RELATED TO 14 WITH INTENSITY 2.000000
RELATION * : RELATED TO 15 WITH INTENSITY 2.000000
RELATION * : RELATED TO 16 WITH INTENSITY 2.000000
RELATION * : RELATED TO 17 WITH INTENSITY 2.000000
RELATION * : RELATED TO 18 WITH INTENSITY 2.000000
RELATION * : RELATED TO 19 WITH INTENSITY 2.000000
END
```

RELATIONS OF 1 NODE

```
RELATION * : RELATED TO 2 WITH INTENSITY 8.000000
END
```

RELATIONS OF 2 NODE

```
RELATION * : RELATED TO 3 WITH INTENSITY 8.000000
END
```

RELATIONS OF 3 NODE

```
RELATION * : RELATED TO 4 WITH INTENSITY 8.000000
END
```

RELATIONS OF 4 NODE

```
RELATION * : RELATED TO 5 WITH INTENSITY 8.000000
END
```

RELATIONS OF 5 NODE

```
RELATION * : RELATED TO 6 WITH INTENSITY 8.000000
```

END

RELATIONS OF 6 NODE  
RELATION \* : RELATED TO 7 WITH INTENSITY 8.000000  
END

RELATIONS OF 7 NODE  
RELATION \* : RELATED TO 8 WITH INTENSITY 8.000000  
END

RELATIONS OF 8 NODE  
RELATION \* : RELATED TO 9 WITH INTENSITY 8.000000  
END

RELATIONS OF 9 NODE  
RELATION \* : RELATED TO 10 WITH INTENSITY 8.000000  
END

RELATIONS OF 10 NODE  
RELATION \* : RELATED TO 11 WITH INTENSITY 8.000000  
END

RELATIONS OF 11 NODE  
RELATION \* : RELATED TO 12 WITH INTENSITY 8.000000  
END

RELATIONS OF 12 NODE  
RELATION \* : RELATED TO 13 WITH INTENSITY 8.000000  
END

RELATIONS OF 13 NODE  
RELATION \* : RELATED TO 14 WITH INTENSITY 8.000000  
END

RELATIONS OF 14 NODE  
RELATION \* : RELATED TO 15 WITH INTENSITY 8.000000  
END

RELATIONS OF 15 NODE  
RELATION \* : RELATED TO 16 WITH INTENSITY 8.000000  
END

RELATIONS OF 16 NODE  
RELATION \* : RELATED TO 17 WITH INTENSITY 8.000000  
END

RELATIONS OF 17 NODE  
RELATION \* : RELATED TO 18 WITH INTENSITY 8.000000  
END

RELATIONS OF 18 NODE  
RELATION \* : RELATED TO 19 WITH INTENSITY 8.000000  
END

RELATIONS OF 19 NODE  
RELATION \* : RELATED TO 1 WITH INTENSITY 2.000000  
END



```

/*****
**      TEST FILE FOR graph (Dynamic Layout Alg)
**
**      MODUL - MANIPULATION OF THE GRAPH DATA STRUCTURE
**
** Author: dr. Szirmay-Kalos Laszlo (szirmay@fsz.bme.hu)
**      Technical University of Budapest, Hungary
*****/
#ifdef MSWINDOWS
#include "fileio.hxx"
#else
#include "fileio.hxx"
#endif

/*****
/*      NODE
*****/
/*****
/*----- node constructor -----*/
/* Constructs node object and initializes
/* IN : name and type (MOVEABLE or FIXED) of node
/* OUT : -
/*-----*/
Node :: Node(pchar newname, TYPE ntype)
{
    newname[MAXNAME] = '\0';
    strcpy(name,newname);
    type = ntype;
    force = vector(0.0, 0.0);
    pos = vector(0.0, 0.0);
    speed = vector(0.0, 0.0);
}

/*****
/*      RELATION
*****/
/*****
/*----- relation constructor -----*/
/* Constructs relation object and initializes
/* IN : name , related node, intensity,
/*-----*/
Relation :: Relation( pchar nname, Node * np, double r )
{
    SetRelation( nname, r );
    relation_to = np;
}

/*----- SetRelation -----*/
/* Change the name intensity and relation of a constructed rel
/* IN : name, intensity
/*-----*/
void Relation :: SetRelation( pchar nname, double r )
{
    strcpy( name, nname );
    intensity = r;
}

/*****
/*      NODE ELEM
*****/
/*****
/*----- NodeElem constructor -----*/
/* Constructs a node elem of a list object and initializes
*****/
```

```

/* IN : name and type (MOVEABLE or FIXED) of node */
/*-----*/
NodeElem :: NodeElem( pchar name, TYPE type )
           : Node( name, type )
{
    next_node = NULL;
    relation = NULL;
}

/*****
/*      RELATION ELEM                                */
*****/
/*----- RelationNode constructor -----*/
/* Constructs relation node of a list object and initializes */
/* IN : name , related node, intensity, */
/*-----*/
RelationElem :: RelationElem( pchar name, Node * p, double r )
                        : Relation( name, p, r )
{
    next_relation = NULL;
}

/*****
/*      GRAPH = NODE - RELATION DATA                                */
/*
/*      The Graph data structure is a dynamic structure.
/*
/*      The nodes are placed on a singly linked list, where fix nodes
/*      are on the beginning, and moveable nodes are on the end.
/*      The nodes are also identified by serial numbers, the moveable
/*      nodes are having positive while fixed nodes negative numbers.
/*      Control pointers : currnode - points to the actual node
/*                        relatenode - other node which forms a pair
/*                        with currnode for relation ops
/*                        start_node - the beginning of the list
/*                        last_node - the end of the list
/*
/*      The relations of a given node are stored on an other linked list
/*      connected to the node of the given node. The relation node
/*      contains name, type, intensity parameters and a pointer to the
/*      related node. The relation of two node is stored on the
/*      relation list of the node having smaller serial number!
/*      Control pointers : currelation -points to the actual relation node
/*                        prevrelation - points to the relation just
/*                        before currelation on the actual relation
/*                        list.
/*
/*      STRUCTURE OVERVIEW: P = Node, R = RelationNode
/*
/*      start_node                                lastnode
/*      P ----> P ----> P ----> P ----> P ----> P ----> NULL
/*      |               ^               |               ^               ^
/*      R -----|               R-----|               |
/*      |               |               |               |               |
/*      NULL            R-----|               |
/*                      |               |
/*                      NULL
*****/
/*----- Graph Constructor -----*/
/* Initializes Graph data structure */

```

```
/*-----*/
Graph :: Graph()
{
    start_node = NULL;
    last_node  = NULL;
    currnode   = NULL;
    relatenode = NULL;
    currrelation = NULL;
    prevrelation = NULL;
    nfixnode   = nmovnode = 0;
}

/*----- RestoreNodes -----*/
/* Restores the node-relation data structure from a file */
/* The file type is TEXT. */
/* IN : file name */
/* SIDE EFFECT: - node-relation data structure is destroyed */
/* then it is restored from the given file */
/*-----*/
void Graph :: RestoreNodes ( pchar file_name )
{
    char s[MAXLINE + 1];
    char node_name[MAXNAME + 1];
    char rel_node_name[MAXNAME + 1];
    char relation_name[MAXNAME + 1];
    double x, y;
    double relation;
    FileIO fi ( "r" );
    BOOL first_rel;

    if ( !fi.OpenFile ( file_name ) ) app.Error("Input file does not exists");

/*
 * RESTORE NODES
 */
    for ( ; ; ) {
/*
 * TRY TO INPUT NODE NAME
 */
        if ( !fi.GetKeyword ( "NAME" ) ) {
/*
 * FAILED -> END OF NODE LIST
 */
            first_rel = TRUE;
            break;
        } else {
            if ( !fi.GetOperator ( '=' ) ) app.Error( "= expected", fi.GetLineNum() );
            if ( !fi.GetString(node_name,MAXNAME) ) app.Error( "Name expected",
fi.GetLineNum() );
/*
 * TRY TO INPUT NODE POSITION
 */
            if ( !fi.GetKeyword("POSITION") ) {
/*
 * FAILED -> ASSUME NO POSITION, GENERATE RANDOMLY
 */
                x = (OVERWINDOW_X - WALL_MARGIN * 2.0) /
                    (double)RAND_MAX * (double)rand() + WALL_MARGIN;
                y = (OVERWINDOW_Y - WALL_MARGIN * 2.0) /
                    (double)RAND_MAX * (double)rand() + WALL_MARGIN;
                if ( !fi.GetKeyAgain("TYPE") ) app.Error( "TYPE expected",
fi.GetLineNum() );

```

```
        } else {
            if ( !fi.GetOperator ( '=' ) ) app.Error( "= expected", fi.GetLineNum()
);
                if ( !fi.GetVariable( &x, 0.0, OVERWINDOW_X ) || !fi.GetVariable( &y,
0.0, OVERWINDOW_Y ) )
                    app.Error( "Coordinate out of space", fi.GetLineNum() );
                if ( !fi.GetKeyWord("TYPE") ) app.Error( "TYPE expected",
fi.GetLineNum() );
            }
        }
/*
*    TRY TO INPUT TYPE PARAMETERS
*/
        if ( !fi.GetOperator ( '=' ) ) app.Error( "= expected", fi.GetLineNum() );
        if ( !fi.GetString( s, MAXLINE ) ) app.Error("Line too long", fi.GetLineNum()
);
/*
*    ADD NEW NODE TO THE DATA STRUCTURE AND CHECK THE NAME IF UNIQUE
*/
        if ( strcmp( s, "FIXED" ) == 0 ) {
            if ( !AddNode(node_name, FIXED_NODE) ) app.Error("Not unique node
name", fi.GetLineNum() );
        } else if ( strcmp( s, "MOVEABLE" ) == 0 ) {
            if ( !AddNode(node_name, MOVEABLE_NODE) ) app.Error("Not unique node
name", fi.GetLineNum() );
        } else
            app.Error("Invalid Node type",
fi.GetLineNum() );

        currnode -> Position( ) = vector( x, y );
    }
}
/*
*    RESTORE RELATIONS
*/
for( ; ; ) {
/*
*    TRY TO GET RELATION LIST HEAD, IF FAIL -> END OF INPUT
*/
    if ( first_rel ) {
        if ( !fi.GetKeyAgain ( "RELATIONS" ) ) break;
        first_rel = FALSE;
    } else {
        if ( !fi.GetKeyWord ( "RELATIONS" ) ) break;
    }
    if ( !fi.GetKeyWord("OF") ) app.Error("OF expected", fi.GetLineNum() );
    if ( !fi.GetString( node_name, MAXNAME ) ) app.Error("Name too long",
fi.GetLineNum() );
    if ( !fi.GetKeyWord ( "NODE" ) ) app.Error("NODE expected", fi.GetLineNum() );
/*
*    IDENTIFY NODE
*/
    if ( !SearchNode( node_name ) ) app.Error("Not declared Node", fi.GetLineNum() );
/*
*    TRY TO INPUT RELATION LIST
*/
    for( ; ; ) {        // get the whole relation list of this node
/*
*    TRY TO GET RELATION NAME
*/
        if ( !fi.GetKeyWord ( "RELATION" ) ) {
/*
*    FAILED -> CHECK END OF RELATION MARKER
```

```
*/
        if ( !fi.GetKeyAgain ( "END" ) ) app.Error("END expected",
fi.GetLineNum() );
        else break;
    }
    if ( !fi.GetString( relation_name, MAXNAME ) ) app.Error("Name too long",
fi.GetLineNum() );
/*
*    CHECK IF NO-NAME RELATION
*/
        if ( strcmp( relation_name, "*" ) == 0 ) relation_name[0] = '\\0';
        if ( !fi.GetOperator ( ':' ) ) app.Error(":" expected", fi.GetLineNum() );
/*
*    TRY TO GET RELATED NODE WITH RELATION PARAMETERS
*/
        if ( !fi.GetKeyWord("RELATED") )    app.Error("RELATED expected",
fi.GetLineNum() );
        if ( !fi.GetKeyWord("TO") )        app.Error("TO expected", fi.GetLineNum()
);
        if ( !fi.GetString( rel_node_name, MAXNAME ) ) app.Error("Name expected",
fi.GetLineNum() );
        if ( !fi.GetKeyWord("WITH") )      app.Error("WIDTH expected",
fi.GetLineNum() );
        if ( !fi.GetKeyWord("INTENSITY") ) app.Error("INTENSITY expected",
fi.GetLineNum() );

        if ( !fi.GetVariable( &relation, -MAXRELATION, MAXRELATION ) )
            app.Error("Relation is out of range", fi.GetLineNum() );
/*
*    BUILD THE NEW RELATION INTO THE DATA STRUCTURE
*/
        NodeElem * tmpnode = currnode;
        if ( !RelSearchNode( rel_node_name ) ) app.Error("Not declared Node",
fi.GetLineNum() );
        AddRelation( relation_name, relation );
        currnode = tmpnode;
    }
}

fi.CloseFile();
}

/*----- SaveNodes -----*/
/* Saves the node-relation data structure to a file name */
/* The file type is TEXT. */
/* IN : file name */
/*-----*/
BOOL Graph :: SaveNodes ( pchar file_name )
{
    char s[MAXLINE];
    FileIO fo ( "w" );

    if ( !fo.OpenFile (file_name) ) return FALSE;

/*
*    SAVE NODES
*/
    if ( !FirstNode() ) {
        fo.CloseFile( );
        return TRUE;
    }
}
```

```

do {
    sprintf(s,
        "NAME = %s POSITION =  %6.3lf %6.3lf TYPE = %s\n",
        currnode -> GetName(),
        currnode -> Position().X(),
        currnode -> Position().Y(),
        (currnode -> GetType() == FIXED_NODE ? "FIXED" : "MOVEABLE"));
    fo.PutString( s );
} while ( NextNode() );

/*
 *   SAVE RELATIONS
 */
FirstNode();
do {
    sprintf(s,
        "\nRELATIONS OF %s NODE\n",
        currnode -> GetName());
    fo.PutString( s );

    if ( !FirstRelation() ) {
        fo.PutString( "END\n" );          // END OF RELATION MARKER
        continue;
    }

    do {
        if ( strlen( currelation -> GetName() ) != 0 )
            sprintf(s, "RELATION %s : ", currelation -> GetName() );
        else
            sprintf(s, "RELATION * : ");
        fo.PutString( s );

        sprintf(s,
            "RELATED TO %s WITH INTENSITY %6.3lf \n",
            currelation -> GetOtherNode() -> GetName(),
            currelation -> GetRelation() );
        fo.PutString( s );
    } while ( NextRelation() );

    fo.PutString( "END\n" );          // END OF RELATION MARKER

} while ( NextNode() );

fo.CloseFile( );
return TRUE;
}

/*----- SetNodePos -----*/
/* Sets the position of a node */
/* IN : new position */
/*-----*/
void Graph :: SetNodePos ( vector p )
{
    if ( currnode != NULL ) currnode -> Position( ) = p;
}

/*----- GetRelation -----*/
/* Gets the relation intensity of the actual relation */
/* IN : - */
/* OUT : intensity ( -MAXRELATION -> MAXRELATION ) */

```

```
/*-----*/
double Graph :: GetRelation( )
{
    if (currelation != NULL) return currelation -> GetRelation( );
    else return 0.0;
}

/*----- GetRelationName -----*/
/* Gets the name of the actual relation */
/* OUT : name of NULL if no relation */
/*-----*/
pchar Graph :: GetRelationName( )
{
    if ( currelation != NULL ) return currelation -> GetName( );
    else return NULL;
}

/*----- AddNode -----*/
/* Checks if a node having the same name exist and if not new */
/* node is allocated and added to the beginning of the list if */
/* the node is FIXED or to the end if it is MOVEABLE */
/* IN : name and type of the new node */
/* OUT : is this name unique ? */
/* SIDE EFFECT: currnode is set to the new node. */
/* if FIXED node */
/* start_node adjusted, nfixnode incremented */
/* if MOVEABLE NODE */
/* last_node adjusted, nmovnode incremented */
/*-----*/
BOOL Graph :: AddNode ( pchar name, char type )
{
    /*
    * DECIDE IF THIS NAME IS UNIQUE, IF NOT RETURN ERROR
    */
    if ( SearchNode( name ) ) return FALSE;

    currnode = new NodeElem(name, type);

    if (start_node == NULL) {
        /*
        * IF THIS IS THE FIRST NODE
        */
        start_node = last_node = currnode;
    } else {
        if ( type == FIXED_NODE ) {
            /*
            * IF FIXED NODE -> ADD TO THE BEGINNING OF THE LIST
            */
            currnode -> SetNext( start_node );
            start_node = currnode;
        } else {
            /*
            * IF MOVEABLE NODE -> ADD TO THE END OF THE LIST
            */
            last_node -> SetNext( currnode );
            last_node = currnode;
        }
    }
    if ( type == FIXED_NODE ) {
        nfixnode++;
        currnode -> SetSerNum( -nfixnode );
    }
}
```

```
    } else {
        nmovnode++;
        currnode -> SetSerNum( nmovnode );
    }
    return TRUE;
}

/*----- SearchNode -----*/
/* Searches node by name */
/* IN : searched name */
/* OUT : is there node having this name ? */
/* SIDE EFFECT: currnode is set to the found node. */
/*-----*/
BOOL Graph :: SearchNode ( pchar name )
{
    if ( !FirstNode() ) return FALSE;
    do {
        if ( strcmp( currnode -> GetName(), name ) == 0 ) return TRUE;
    } while ( NextNode ( ) );

    return FALSE;
}

/*----- RelSearchNode -----*/
/* Searches relate node by name */
/* IN : picked position */
/* OUT : is there node in the pick aperture ? */
/* SIDE EFFECT: */
/* To ensure that relatenode has greater serial number than currnode: */
/* IF found node has smaller serial number than currnode */
/* relatenode is set to currnode */
/* currnode is set to the found node */
/* ELSE */
/* relatenode is set to the found node. */
/* Initializes currelation to the relation of currnode and relatenode. */
/*-----*/
BOOL Graph :: RelSearchNode ( pchar name )
{
    NodeElem * oldcurrnode = currnode;
    BOOL found = SearchNode( name );

    if ( found ) {
        relatenode = currnode;
        currnode = oldcurrnode;
        SwapRelation( );
        SearchRelation( );
    } else {
        currelation = NULL;
        currnode = oldcurrnode;
    }
    return found;
}

/*----- SearchRelation -----*/
/* Search for a relation between currnode and relatenode. */
/* If this relation doesnot exist currelation is NULL and */
/* prevrelation points to the end of the relation list of */
/* currnode. */
/* IN : - */
```



```
/* OUT : EMPTY_LIST - No relation list */
/* FIRST_FOUND - The first relation node found */
/* FOUND - Not the first node found */
/* NOT_FOUND - No such relation */
/* SIDE EFFECT: currrelation= searched relation or NULL */
/* prevrelation= the previous relation or the last */
/* node of the relation list or NULL */
/* if no node at all */
/*-----*/
int Graph :: SearchRelation ( )
{
    currrelation = currnode -> GetRelation( );
    prevrelation = currnode -> GetRelation( );
    if (currrelation == NULL) return EMPTY_LIST;
    if (currrelation -> GetOtherNode() == relatenode) return FIRST_FOUND;

    currrelation = prevrelation -> GetNext();

    for ( ; ; ) {
        if (currrelation == NULL) return NOT_FOUND;
        if (currrelation -> GetOtherNode() == relatenode) return FOUND;
        prevrelation = currrelation;
        currrelation = prevrelation -> GetNext();
    }
}

/*----- SwapRelation -----*/
/* To ensure that relatenode has greater serial number than */
/* currnode: */
/* IF relatenode node has smaller serial number than */
/* relatenode and currnode are swapped */
/* IN : - */
/* OUT : - */
/*-----*/
void Graph :: SwapRelation ( )
{
    NodeElem * tmpnode;

    if ( currnode == NULL || relatenode == NULL ) return;

    if ( currnode -> GetSerNum() > relatenode -> GetSerNum() ) {
        tmpnode = currnode;
        currnode = relatenode;
        relatenode = tmpnode;
    }
}

/*----- AddRelation -----*/
/* Adds new or changes the parameters of an existing relation. */
/* If this is a new relation RelationNode is allocated and */
/* placed on the end of relation list of currnode. */
/* The parameters are set according to the explicit parameters */
/* and the implicit relatenode par. */
/* IN : name,intensity, type */
/* OUT : - */
/* SIDE EFFECT: currrelation= new or changed relation */
/*-----*/
void Graph :: AddRelation ( pchar name, double rel )
{
    /*
    * CHECK IF THIS RELATION EXISTS OR FIND THE END OF RELATION LIST
    */
}
```

```
*/
switch ( SearchRelation( ) ) {

case FIRST_FOUND:          //      THIS RELATION HAS BEEN ALREADY DEFINED
case FOUND :
    currrelation -> SetRelation( name, rel );
    return;

case NOT_FOUND:            //      NOT FIRST ADD NEW RELATION TO THE END OF LIST
    currrelation = new RelationElem ( name, relatenode, rel );
    prevrelation -> SetNext ( currrelation );
    return;

case EMPTY_LIST:          //      THIS IS GOING TO BE THE FIRST
    currrelation = new RelationElem ( name, relatenode, rel );
    currnode -> SetRelation( currrelation );
    return;
}
}

/*----- FirstNode -----*/
/* Select currnode as start_node (beginning of the list      */
/* IN : -                                                    */
/* OUT : Are nodes on the list                                */
/*-----*/
BOOL Graph :: FirstNode ( )
{
    if ( (currnode = start_node) == NULL ) return FALSE;
    else return TRUE;
}

/*----- FirstMoveNode -----*/
/* Select currnode as first moveable node on the list      */
/* IN : -                                                    */
/* OUT : Are moveable nodes on the list                                */
/*-----*/
BOOL Graph :: FirstMoveNode ( )
{
    if ( (currnode = start_node) == NULL ) return FALSE;

    while ( currnode -> GetType() != MOVEABLE_NODE ) {
        currnode = currnode -> GetNext();
        if ( currnode == NULL ) return FALSE;
    }
    return TRUE;
}

/*----- NextNode -----*/
/* Let currnode be the next after currnode                  */
/* IN : maximal serial number to be considered of ALL_NODES */
/* OUT : Was it the last node?                                */
/* SIDE EFFECT: currnode = NULL if no more nodes            */
/*-----*/
BOOL Graph :: NextNode ( int maxsernum )
{
    if ( maxsernum == ALL_NODES ) {
        if ( ( currnode = currnode -> GetNext() ) == NULL ) return FALSE;
    } else {
        if ( ( currnode = currnode -> GetNext() ) == NULL ||
            currnode -> GetSerNum() > maxsernum ) return FALSE;
    }
}
```

```
        return TRUE ;
    }

/*----- FirstRelation -----*/
/* Select currrelation as first relation of the relation list */
/* of currnode. */
/* IN : - */
/* OUT : Has the currnode any relation? */
/*-----*/
BOOL Graph :: FirstRelation ( )
{
    if ( (currrelation = currnode -> GetRelation()) == NULL ) {
        relatenode = NULL;
        return FALSE;
    } else {
        relatenode = (NodeElem *) ( currrelation -> GetOtherNode() );
        return TRUE;
    }
}

/*----- NextRelation -----*/
/* Let currrelation the next after currrelation */
/* IN : - */
/* OUT : Was it the last relation of the list? */
/*-----*/
BOOL Graph :: NextRelation ( )
{
    if ( (currrelation = currrelation -> GetNext()) == NULL ) {
        relatenode = NULL;
        return FALSE;
    } else {
        relatenode = (NodeElem *) ( currrelation -> GetOtherNode() );
        return TRUE;
    }
}

/*----- RandomArrange -----*/
/* Random arrangement of nodes */
/*-----*/
void Graph :: RandomArrange( )
{
    /*
    * SKIP FIXED NODES
    */
    if ( !FirstMoveNode() ) return;

    /*
    * MAIN CYCLE OF PLACING MOVEABLE NODES RANDOMLY
    */
    do {
        currnode -> Position( ) = vector((OVERWINDOW_X - WALL_MARGIN * 2.0) /
                                           (double)RAND_MAX * (double)rand() + WALL_MARGIN,
                                           (OVERWINDOW_Y - WALL_MARGIN * 2.0) /
                                           (double)RAND_MAX * (double)rand() + WALL_MARGIN
        );
    } while ( NextNode() );
}

/*----- ObjectSpace Constructor -----*/
/* OBJECT SPACE */
/*-----*/
```

```
/* Initializes object space window */
/* IN : - */
/* OUT : - */
/*-----*/
ObjectSpace :: ObjectSpace( )
    :vwindow( 0, 0, (CoOrd)OVERWINDOW_Y, (CoOrd)OVERWINDOW_X ),
    viewport( 0, 0, WINDOW_WIDTH, WINDOW_HEIGHT )
{
}

/*----- SetScale -----*/
/* Initializes window -> viewport transform */
/* IN : - */
/* OUT : - */
/*-----*/
void ObjectSpace :: SetScale()
{
    scale_x = (double)viewport.Width() / (double)vwindow.Width();
    scale_y = (double)viewport.Height() / (double)vwindow.Height();
}

/*----- SetViewPort -----*/
/* Sets viewport (Canvas RectAngle) */
/* IN : new viewport */
/* OUT : - */
/* SIDE EFFECT: Recalculates window->viewport transform */
/*-----*/
void ObjectSpace :: SetViewPort( RectAngle v )
{
    viewport = v;
    SetScale();
}

/*----- ScreenPos -----*/
/* Transform a point from object space to screen space */
/* IN : object space position */
/* OUT : screen space coordinates of point */
/*-----*/
Point ObjectSpace :: ScreenPos( vector p )
{
    CoOrd x = (CoOrd)((p.X()-(double)vwindow.HorPos()) * scale_x);
    CoOrd y = (CoOrd)((p.Y()-(double)vwindow.VerPos()) * scale_y);
    return Point(x,y);
}

/*----- ScreenPos -----*/
/* Gets the position of a NODE in screen coordinate system */
/* IN : pointer to the NODE */
/* OUT : screen space coordinates of NODE position */
/*-----*/
Point ObjectSpace :: ScreenPos( NodeElem * pnode )
{
    return ScreenPos( pnode -> Position() );
}

/*****
/* GRAPH WINDOW */
*****/
/*----- GraphWindow constructor -----*/
```

```
/* Reads the input file defined in argv[1] */
/*-----*/
GraphWindow :: GraphWindow( int argc, char * argv[] )
    : AppWindow( argc, argv )
{
    if ( argc > 1 ) {
        graph.RestoreNodes( argv[1] );
    } else app.Error( "Input file missing" );
}

/*----- ExposeAll -----*/
/* Redraw the graph on the screen */
/*-----*/
void GraphWindow :: ExposeAll( ExposeEvent * event )
{
    Text( "<L> = Layout Algorithm", Point(20, 20) );
    Text( "<R> = Random Arrange", Point(20, 40) );
    Text( "<Q> = Quit & Save", Point(20, 60) );

/*
 *   SET WINDOW - VIEWPORT TRANSFORM
 */
    if ( event ) graph.SetViewPort( Canvas() );

/*
 *   DISPLAY RELATIONS
 */
    graph.FirstNode();
    do {
        if ( graph.FirstRelation() ) {
            do ShowRelation(); while ( graph.NextRelation() );
        }
    } while ( graph.NextNode() );

/*
 *   DISPLAY NODES
 */
    if ( !graph.FirstNode() ) return;
    do ShowNode( ); while ( graph.NextNode() );
}

/*----- MouseButtonDn -----*/
/* React to Mouse button down event! */
/*-----*/
void GraphWindow :: KeyPressed( KeyEvent * event )
{
    switch ( event -> GetASCII() ) {
    case 'L':
    case 'l': switch ( graph.Placement() ) {
        case STOPPED: RePaint();
            break;
        case INSTABLE: app.Warning("Instable system");
            break;
        case TOO_LONG: app.Warning("Solution takes too long");
            break;
        }
        break;
    case 'R':
    case 'r': graph.RandomArrange();
        RePaint();
        break;
    case 'Q':
    case 'q': graph.SaveNodes( "ggg.dat" );
    }
```

```
        app.Quit();
    }

/*----- ShowNode -----*/
/* Shows current node as a rectangle and a text */
/*-----*/
void GraphWindow :: ShowNode( )
{
    DrawRectangle( RectAngle( graph.ScreenPos().X() - NODESIZE_X / 2,
                             graph.ScreenPos().Y() - NODESIZE_Y / 2,
                             NODESIZE_X, NODESIZE_Y) );

    Text( graph.GetNode() -> GetName(), graph.ScreenPos() );
}

/*----- ShowRelation -----*/
/* Shows the current relation as a line and a text */
/*-----*/
void GraphWindow :: ShowRelation( )
{
    MoveTo( graph.ScreenPos() );
    LineTo( graph.RelScreenPos() );
    Text( graph.GetRelationName(),
          Point( (graph.ScreenPos().X() + graph.RelScreenPos().X()) / 2,
                 (graph.ScreenPos().Y() + graph.RelScreenPos().Y()) / 2 ) );
}

/*****
/* WINDOW MANAGER INDEPENDENT ENTRY POINT */
*****/
void App :: Start( int argc, char * argv[] )
{
    GraphWindow graphwindow( argc, argv );
    graphwindow.MessageLoop();
}
```

```

/*****
**      TEST FILE FOR graph (Dynamic Layout Alg)
**
**      HEADER - GRAPH DATA STRUCTURE MANIPULATION
**
** Author: dr. Szirmay-Kalos Laszlo (szirmay@fsz.bme.hu)
**      Technical University of Budapest, Hungary
*****/
#ifdef MSWINDOWS
#include "mswindow.hxx"
#include "vector.hxx"
#include "defines.h"
#else
#include "window.hxx"
#include "vector.hxx"
#include "defines.h"
#endif

typedef char TYPE;

/*****/
class Node {
/*****/
    char    name[MAXNAME + 1];    // node name
    TYPE    type;                 // fixed or movable
    vector  pos;                  // actual position
    vector  speed;                // speed
    vector  force;                // driving force to this node
public:
    Node( pchar, TYPE );          // constructor

    vector& Position( void )      { return pos;          }
    vector& Speed( void )         { return speed;         }
    vector& Force( void )         { return force;         }

    void    AddForce( vector& f ) { force += f;           }

    pchar   GetName( void )       { return name;          }
    TYPE    GetType( void )       { return type;          }
};

/*****/
class Relation {
/*****/
    char    name[MAXNAME + 1];    // relation name
    double  intensity;            // relation intensity
    Node    * relation_to;        // related node
public:
    Relation( pchar , Node *, double );

    void    SetRelation( pchar, double );
    pchar   GetName( void )       { return name;          }
    double  GetRelation( void )   { return intensity;      }
    Node    * GetOtherNode( void ) { return relation_to;    }
};

/*****/
class RelationElem : public Relation {
/*****/
    RelationElem    *    next_relation;    // next on the list
public:

```

```
RelationElem( pchar name, Node * p, double r );
void          SetNext( RelationElem * rn ) { next_relation = rn; }
RelationElem * GetNext( void )             { return next_relation; }
};

/*****
class NodeElem : public Node {
/*****/
    int          ser_num;          // serial number in list
    NodeElem     * next_node;       // pointer to next node
    RelationElem * relation;       // first relation of this node
public:
    NodeElem(pchar, TYPE);
    void      SetNext( NodeElem *p )      { next_node = p; }
    void      SetRelation( RelationElem *p ) { relation = p; }
    void      SetSerNum( int sernum )     { ser_num = sernum; }
    NodeElem * GetNext( void )            { return next_node; }
    RelationElem * GetRelation( void )    { return relation; }
    int        GetSerNum( void )          { return ser_num; }
};

/*****/
class Graph {
/*****/
    int          nfixnode;          // number of fix nodes
    int          nmovnode;          // number of movable nodes
    NodeElem *   currnode;          // current node
    NodeElem *   relatenode;        // actual relation of curr
    NodeElem *   start_node;        // start of list
    NodeElem *   last_node;         // end of list
    RelationElem * currelation;     // relation of nodes list
    RelationElem * prevrelation;    // previous to currelation

    void          SwapRelation( void ); // swap currnode and relatenode
                                         // if currnode is further in the
                                         // list

public:
    Graph( void );

    void          SetNodePos( vector ); // sets position of currnode
    void          SetRelation( double ); // sets intensity of currelation
    NodeElem *   GetNode( void )        { return currnode; }
    NodeElem *   GetRelateNode( void )  { return relatenode; }
    double       GetRelation( void );   // get intensity of currelation
    pchar        GetRelationName( void ); // get name of currelation
    BOOL         AddNode( pchar, TYPE ); // add new node to the list
    void         AddRelation( pchar, double ); // add new relation
    BOOL         SearchNode( pchar );    // search node by name
    BOOL         RelSearchNode( pchar );
    int          SearchRelation( void ); // search relation of currnode and relatenode

    BOOL         SaveNodes( pchar );     // save to a file
    void         RestoreNodes( pchar );  // restore from file

    int          Placement( void );       // place nodes step-by-step
    void         RandomArrange( void );  // arrange nodes randomly
    int          DynamicLayout( int );    // dynamic layout algorithm

    BOOL         FirstNode( void );       // select first node on the list
    BOOL         FirstMoveNode( void );   // select first moveable node
};
```



```
    BOOL        NextNode( int max = ALL_NODES ); // select next to currnode
    BOOL        FirstRelation( void );           // select first relation of currnode
    BOOL        NextRelation( void );            // select next relation
};

/*****
class ObjectSpace : public Graph {
/*****/
    double      scale_x;           // scale of window->viewport transform
    double      scale_y;
    RectAngle   vwindow;           // object space window
    RectAngle   viewport;          // viewport
    void        SetScale( void );  // calculate scale from vwindow and viewport
public:
    ObjectSpace( void );

    void        SetViewPort( RectAngle );
    void        SetWindow( RectAngle );

    Point       ScreenPos( NodeElem * ); // get screen coordinates of node
    Point       ScreenPos( vector );      // window -> viewport transform
    Point       ScreenPos( void )         { return ScreenPos( GetNode( ) ); }
    Point       RelScreenPos( void )      { return ScreenPos( GetRelateNode() ); }
};

/*****
class GraphWindow : public AppWindow {
/*****/
    ObjectSpace graph;

    void        ExposeAll( ExposeEvent * );
    void        KeyPressed( KeyEvent * );
    void        ShowNode( void );
    void        ShowRelation( void );

public:
    GraphWindow(int argc, char * argv[] );
};
```

.AUTODEPEND

.PATH.obj = .

```
#                *Translator Definitions*
CC = bcc -v -W -vi- -wpro -weas -wpre -n. -I$(INCLUDEPATH) -L$(LIBPATH)
TASM = TASM
TLIB = tlib
TLINK = tlink
LIBPATH = C:\BORLANDC\LIB
INCLUDEPATH = C:\BORLANDC\INCLUDE
```

```
#                *Implicit Rules*
.cpp.obj:
    $(CC) -c -DMSWINDOWS {$< }
```

```
#                *List Macros*
```

OBJS = fileio.obj layout.obj graph.obj mswindow.obj vector.obj

```
#                *Explicit Rules*
graph: $(OBJS)
    $(TLINK) /v/x/c/P-/Twe/L$(LIBPATH) @&&|
c0ws.obj+
fileio.obj+
layout.obj+
graph.obj+
mswindow.obj+
vector.obj
graph
    # no map file
mathws.lib+
import.lib+
cws.lib
```

```
|
    RC    .\graph.exe
```

```
#                *Individual File Dependencies*
mswindow.obj: mswindow.cpp
```

fileio.obj: fileio.cpp mswindow.hxx vector.hxx defines.h graph.hxx fileio.hxx

graph.obj: graph.cpp mswindow.hxx vector.hxx defines.h graph.hxx

layout.obj: layout.cpp mswindow.hxx r naor.hxx defines.h graph.hxx

vector.obj: vector.cpp vector.hxx

```

/*****
** DYNAMIC LAYOUT ALGORITHM OF GENERAL GRAPHS
**
** Author: dr. Szirmay-Kalos Laszlo (szirmay@fsz.bme.hu)
**          Technical University of Budapest, Hungary
***/

#ifdef MSWINDOWS
#include "graph.hxx"
#else
#include "graph.hxx"
#endif

/*
*   CONSTANTS
*/
const double TIME_STEP = 0.1;           // time step of diff equ

const double MAX_FORCE = 500.0;         // force shows instability
const double MIN_FORCE = 2.0;          // force cosidered as 0
const double MAX_TIME_SCALE = 10.0;    // scale of max time of solution

const double MINFRICTION = 0.6;        // friction boundaries
const double MAXFRICTION = 0.9;
const double MINIINERTIA = 0.1;        // inverse inertia boundaries
const double MAXIINERTIA = 0.4;

const double ZERO_DIST = 10.0;         // distance considered as 0
const double WALL_OUT_DRIVE = 80.0;    // forces of the wall
const double WALL_MARGIN_DRIVE = 1.0;

const double SCALECONSTRAINT = OVERWINDOW_X / 3.5 / MAXRELATION;
const double MINCONSTRAINT = OVERWINDOW_X / 7.0;    // minimal constraint

/*****
/* DYNAMIC LAYOUT base on MECHANICAL SYSTEM ANALOGY
/* IN  : The serial number of the maximal moveable node to be considered
/* OUT : STOPPED = All objects stopped
/*       INSTABLE = Instable, force goes to infinity
/*       TOO_LONG = Too much time elapsed
*****/
int Graph :: DynamicLayout( int maxsernum )
/*-----*/
{
/*
*   LOCALS
*/
double constraint, friction, iinertia, dist;
vector drive;           // drive forces
vector direction;       // direction of drives
double MAX_TIME = MAX_TIME_SCALE * (nmovnode + nfixnode + 1);

/*
*   INIT SPEED OF MOVEABLE NODES TO 0
*/
if ( !FirstMoveNode() ) return STOPPED;
do currnode -> Speed( ) = vector(0.0, 0.0); while ( NextNode( maxsernum ) );

/*
*   MAIN CYCLE OF TIME IN THE SOLUTION OF DIFF EQUATION
*/
for ( double t = 0.0 ; t < MAX_TIME ; t += TIME_STEP ) {

```

```
/*
 *   INITIALIZE FORCE IN NODES TO 0
 */
    FirstNode();
    do currnode -> Force( ) = vector( 0.0, 0.0 ); while ( NextNode( maxsernum ) );
/*
 *   CALCULATE FRICTION AND RESPONSE VALUES FROM t
 */
    friction = MINFRICTION + (MAXFRICTION - MINFRICTION) * t / MAX_TIME;
    iinertia = MAXIINERTIA - (MAXIINERTIA - MINIINERTIA) * t / MAX_TIME;
/*
 *   CALCULATE DRIVE FORCE BETWEEN EACH PAIR OF NODES
 */
    FirstNode();
    do {
        relatenode = currnode -> GetNext();
        while ( relatenode != NULL && relatenode -> GetSerNum() <= maxsernum ) {

            direction = currnode -> Position( ) - relatenode -> Position( );
            dist = direction.Size();
            if ( dist < ZERO_DIST ) dist = ZERO_DIST;

/*
 *   CALCULATE FORCE FROM THEIR RELATION
 */
            switch( SearchRelation( ) ) {
            case EMPTY_LIST:
            case NOT_FOUND:
                constraint = MINCONSTRAINT + MAXRELATION * SCALECONSTRAINT;
                break;
            case FOUND:
            case FIRST_FOUND:
                constraint = MINCONSTRAINT + (MAXRELATION - currelation->GetRelation()) *
SCALECONSTRAINT;
                break;
            }
            // SET FORCE
            drive = (constraint - dist) / dist * direction;
            drive /= (double)(maxsernum + nfixnode);
            currnode -> AddForce(drive);
            relatenode -> AddForce(-drive);

            relatenode = relatenode -> GetNext();
        }
    } while ( NextNode( maxsernum ) );
/*
 *   ADD ADDITIONAL FORCES AND DETERMINE MAXIMAL FORCE
 */
    double max_force = 0.0;

    FirstMoveNode();
    do {
/*
 *   CALCULATE DRIVE FORCE OF BOUNDARIES AND ADD TO RELATION FORCES
 */
        dist = currnode -> Position().X();
        /*
 *   FORCE OF LEFT WALL
 */
        if (dist < 0) {
            // OUT LEFT
            drive = vector( -dist * WALL_OUT_DRIVE + WALL_MARGIN * WALL_MARGIN_DRIVE,
0.0 );
```

```

    currnode -> AddForce(drive);
} else if (dist < WALL_MARGIN) {      // IN LEFT MARGIN
    drive = vector((WALL_MARGIN - dist) * WALL_MARGIN_DRIVE, 0.0);
    currnode -> AddForce(drive);
}
/*
*    FORCE OF THE RIGHT WALL
*/
dist = currnode -> Position().X() - OVERWINDOW_X;

if (dist > 0) {                        // OUT RIGHT
    drive = vector( -dist * WALL_OUT_DRIVE + WALL_MARGIN * WALL_MARGIN_DRIVE,
0.0);

    currnode -> AddForce(drive);
} else if (-dist < WALL_MARGIN) {      // IN RIGHT MARGIN
    drive = vector((-WALL_MARGIN - dist) * WALL_MARGIN_DRIVE, 0.0);
    currnode -> AddForce(drive);
}

dist = currnode -> Position().Y();
/*
*    FORCE OF BOTTOM WALL
*/
if (dist < 0) {                        // OUT BOTTOM
    drive = vector(0.0, -dist * WALL_OUT_DRIVE + WALL_MARGIN * WALL_MARGIN_DRIVE
);

    currnode -> AddForce(drive);
} else if (dist < WALL_MARGIN) {      // IN BOTTOM MARGIN
    drive = vector(0.0, (WALL_MARGIN - dist) * WALL_MARGIN_DRIVE);
    currnode -> AddForce(drive);
}
/*
*    FORCE OF THE TOP WALL
*/
dist = currnode -> Position().Y() - OVERWINDOW_Y;

if (dist > 0) {                        // OUT TOP
    drive = vector( 0.0, -dist * WALL_OUT_DRIVE + WALL_MARGIN *
WALL_MARGIN_DRIVE );
    currnode -> AddForce(drive);
} else if (-dist < WALL_MARGIN) {      // IN TOP MARGIN
    drive = vector(0.0, (-WALL_MARGIN - dist) * WALL_MARGIN_DRIVE);
    currnode -> AddForce(drive);
}

/*
*    MOVE NODE BY FORCE
*/

    vector old_speed = currnode -> Speed( );
    currnode -> Speed( ) = (1.0 - friction) * old_speed + iinertia * currnode ->
Force( );
    currnode -> Position( ) += 0.5 * (old_speed + currnode -> Speed( ) );

/*
*    CALCULATE MAXIMUM FORCE
*/

    double abs_force = currnode -> Force().Size( );
    if ( abs_force > max_force) max_force = abs_force;

} while ( NextNode( maxsernum ) );

/*
*    STOP CALCULATION IF

```

```
*/
    if ( max_force < MIN_FORCE ) return STOPPED; // All objects stopped
    if ( max_force > MAX_FORCE ) return INSTABLE; // Instable, force goes to infinity
}
return TOO_LONG; // Too much time elapsed
}

/*****
/* INITIAL PLACEMENT ALGORITHM */
/* OUT : STOPPED = All objects stopped */
/* INSTABLE = Instable, force goes to infinity */
/* TOO_LONG = Too much time elapsed */
*****/
int Graph :: Placement( )
/*-----*/
{
    vector    candidate;           // candidate position
    vector    relate_cent;         // center related objects
    vector    notrel_cent;         // center of related object
    vector    center( OVERWINDOW_X / 2, OVERWINDOW_Y / 2 );
    int       nrel;                // number of related objects
    int       nnotrel;             // displayed nodes
    double    perturb_x = OVERWINDOW_X / (double)RAND_MAX ;
    double    perturb_y = OVERWINDOW_Y / (double)RAND_MAX ;

/*
*   SKIP FIXED NODES
*/
    if ( !FirstMoveNode() ) return STOPPED;

/*
*   MAIN CYCLE OF INTRODUCING MOVABLE NODES STEP-BY-STEP
*/
    for( int inode = 1; ; inode++ ) {

/*
*   CALCULATE THE CENTER OF GRAVITY OF ALREADY INTRODUCED NODES
*   relate_cent IS FOR RELATED NODES
*   notrel_cent IS FOR NON_RELATED NODES
*/

        relate_cent = vector(0.0, 0.0);
        notrel_cent = vector(0.0, 0.0);
        nrel = 0;
        nnotrel = 0;           // displayed nodes
        relatenode = currnode;

        for( FirstNode(); currnode != relatenode; NextNode() ) {
            switch ( SearchRelation() ) {
                case EMPTY_LIST:
                case NOT_FOUND:
                    notrel_cent += currnode -> Position();
                    nnotrel++;
                    break;
                case FIRST_FOUND:
                case FOUND:
                    relate_cent += currnode -> Position();
                    nrel++;
                    break;
            }
        }
        if ( nrel != 0 )        relate_cent /= (double)nrel;
        if ( nnotrel != 0 )    notrel_cent /= (double)nnotrel;
    }
}
/*
```

```
*      IF THIS IS THE FIRST POINT -> PUT TO THE MIDDLE
*/
      if ( nrel == 0 && nnotrel == 0 ) candidate = center;
      else

/*
*      IF NO NOT_RELATED NODE -> PUT TO THE CENTRE OF GRAVITY OF RELATED NODES
*/
      if ( nnotrel == 0 ) candidate = relate_cent;
      else

/*
*      IF NO RELATED NODE -> PUT TO THE MIRROR OF THE nrel_cent ON THE CENTRE
*/
      if ( nrel == 0 )  candidate = 2.0 * center - notrel_cent;
      else

/*
*      BOTH TYPE OF NODES EXIST ->
*      CALCULATE THE CANDIDATE POINT AS THE HALF MIRROR OF notrel_cent TO relate_cent
*/
      candidate = 2.0 * relate_cent - 1.0 * notrel_cent;

/*
*      PERTURBATE RANDOMLY
*/
      candidate += vector( perturb_x / (double)(nfixnode + inode + 5) *
                          (double)( rand() - RAND_MAX / 2),
                          perturb_y / (double)(nfixnode + inode + 5) *
                          (double)( rand() - RAND_MAX / 2 ) );

/*
*      DECIDE IF IT IS OUTSIDE -> FIND THE NEAREST INSIDE POINT
*/
      if ( candidate.X() < WALL_MARGIN )
          candidate = vector( 2.0 * WALL_MARGIN, candidate.Y() );
      if ( candidate.X() > OVERWINDOW_X - WALL_MARGIN )
          candidate = vector( OVERWINDOW_X - 2.0 * WALL_MARGIN, candidate.Y() );

      if ( candidate.Y() < WALL_MARGIN )
          candidate = vector( candidate.X(), 2.0 * WALL_MARGIN );
      if ( candidate.Y() > OVERWINDOW_Y - WALL_MARGIN )
          candidate = vector( candidate.X(), OVERWINDOW_Y - 2.0 * WALL_MARGIN );

/*
*      SET POSITION OF THE NEW NODE
*/
      relatenode -> Position( ) = candidate;

/*
*      ARRANGE ALREADY DEFINED NODES BY DYNAMIC LAYOUT -> IGNORE EDGE CONSTRAINTS
*/
      NodeElem * oldcurrnode = currnode;
      char ret = DynamicLayout( inode );
      currnode = oldcurrnode;

      if ( ret != STOPPED || !NextNode() ) return ret;
  }
}
```

```
mkdir $1
cp defines.h      $1/defines.h
cp layout.C       $1/layout.cpp
cp fileio.C       $1/fileio.cpp
cp graph.C        $1/graph.cpp
cp mswindow.C     $1/mswindow.cpp
cp fileio.hxx     $1/fileio.hxx
cp graph.hxx      $1/graph.hxx
cp vector.C       $1/vector.cpp
cp mswindow.hxx   $1/mswindow.hxx
cp vector.hxx     $1/vector.hxx
cp g.dat          $1/g.dat
cp g20.dat        $1/g20.dat
cp README         $1/README
cp graph.mak      $1/graph.mak
echo "files are generated in directory:" $1
```



```

/*****
**      TEST FILE FOR graph (Dynamic Layout Alg)
**
**      MODUL: class library to interface MS-WINDOWS
**
** Author: dr. Szirmay-Kalos Laszlo (szirmay@fsz.bme.hu)
**          Technical University of Budapest, Hungary
*****/
#include "mswindow.hxx"

static AppWindow * pwindow;

long    FAR PASCAL _export WndProc( unsigned int, unsigned int, unsigned int, long );

//-----
void App :: Error( char * message, int line )
//-----
{
    fprintf( stderr, "ERROR: %s", message );
    if ( line >= 0 ) fprintf( stderr, " in line %d", line );
    fprintf( stderr, "\n" );
    Quit( );
}

//-----
void App :: Warning( char * message )
//-----
{
    fprintf( stderr, "ERROR: %s\n", message );
}

//-----
void App :: Quit( )
//-----
{
    fprintf( stderr, "Bye ( Graph )\n" );
    exit( -1 );
}

static HANDLE hInstance;
static HANDLE hPrevInstance;
static int    nCmdShow;

//-----
AppWindow :: AppWindow( int argc, char * argv[] )
//-----
: canvas( 0, 0, WINDOW_WIDTH, WINDOW_HEIGHT )
{
    WNDCLASS wndclass;

    pwindow = this;

    strcpy( szClassName, "GRAPH" );

    if ( ! hPrevInstance ) {
        wndclass.style          = CS_HREDRAW | CS_VREDRAW;
        wndclass.lpfnWndProc    = ::WndProc;
        wndclass.cbClsExtra     = 0;
        wndclass.cbWndExtra     = 0;
        wndclass.hInstance      = hInstance;
    }
}

```

```
        wndclass.hIcon          = LoadIcon( hInstance, IDI_APPLICATION );
        wndclass.hCursor        = LoadCursor( NULL, IDC_ARROW );
        wndclass.hbrBackground  = GetStockObject( WHITE_BRUSH );
        wndclass.lpszMenuName    = NULL;
        wndclass.lpszClassName  = szClassName;

        if ( ! RegisterClass( &wndclass ) )    exit( -1 );
    }

    hWnd = CreateWindow( szClassName,                // window class name
                        "Graph layout",              // window caption
                        WS_OVERLAPPEDWINDOW,         // window style
                        CW_USEDEFAULT,               // initial x pos
                        CW_USEDEFAULT,               // initial y pos
                        CW_USEDEFAULT,               // initial x size
                        CW_USEDEFAULT,               // initial y size
                        NULL,                          // parent window handle
                        NULL,                          // window menu handle
                        hInstance,                    // program instance handle
                        NULL );                       // creation params

    if ( ! hWnd ) exit( -1 );
    ShowWindow( hWnd, nCmdShow );
}
```

```
//-----
RectAngle AppWindow :: Canvas()
//-----
{
    return canvas;
}

//-----
void AppWindow :: RePaint()
//-----
{
    InvalidateRect( hWnd, NULL, TRUE );    // WM_PAINT message
}

//-----
void AppWindow :: Text( char * text, Point p)
//-----
{
    TEXTMETRIC tm;
    GetTextMetrics( hdc, &tm );

    TextOut(hdc,
            p.X() - tm.tmMaxCharWidth / 2,
            p.Y() - tm.tmHeight / 2,
            text, strlen( text ));
}

//-----
void AppWindow :: MoveTo( Point p )
//-----
{
    ::MoveTo(hdc, p.X(), p.Y() );
}

//-----
```

```
void AppWindow :: LineTo( Point p )
//-----
{
    ::LineTo(hdc, p.X(), p.Y());
}

//-----
void AppWindow :: DrawRectangle( RectAngle& rect )
//-----
{
    RECT r;
    r.left = rect.HorPos();
    r.top = rect.VerPos();
    r.right = r.left + rect.Width();
    r.bottom = r.top + rect.Height();
    FillRect( hdc, &r, GetStockObject( BLACK_BRUSH ) );
}

//-----
void AppWindow :: MessageLoop()
//-----
{
    MSG msg;

    while( GetMessage( &msg, NULL, 0, 0 ) ) {
        TranslateMessage( &msg );
        DispatchMessage( &msg );
    }
}

//-----
// FORWARD EVENT TO EVENT HANDLERS
//
long AppWindow::WindowProc( HWND hwnd, WORD wmsg, WORD wParam, LONG lParam )
//-----
{
    hwnd = hwnd;

    switch ( wmsg ) {
//~~~~~
        case WM_PAINT: {                                     // REPAINT WINDOW
//~~~~~
            RECT rect;
            GetClientRect( hwnd, &rect );
            canvas = RectAngle( rect.left, rect.top,
                               rect.right - rect.left,
                               rect.bottom - rect.top );
            ExposeEvent evt( &canvas );
            hdc = BeginPaint(hwnd, &ps);
            ExposeAll( &evt );
            EndPaint( hwnd, &ps );
        }
        break;

//~~~~~
        case WM_CHAR: {                                     // KEYBOARD EVENT
//~~~~~
            KeyEvent evt(wParam, lParam);
            hdc = GetDC( hwnd );
            KeyPressed( &evt );
            ReleaseDC( hwnd, hdc );
        }
    }
}
```

```
    }
    break;

//~~~~~
    case WM_DESTROY:                                // CLOSE WINDOW
//~~~~~
        PostQuitMessage( 0 );
        break;

//~~~~~
    default:                                         // ALL OTHER EVENTS
//~~~~~
        return DefWindowProc( hwnd, wmsg, wParam, lParam );
    }

    return 0;
}

/*****
*
*   MS-WINDOWS INTERFACE
*
*****/
//-----
// EVENT HANDLER, CALLED BY WINDOWS
//
long FAR PASCAL _export WndProc( unsigned int hwnd,
                                unsigned int message,
                                unsigned int wParam,
                                long lParam )
//-----
{
    return pwindow->WindowProc( hwnd, message, wParam, lParam );
}

//-----
// APPLICATION OBJECT
//-----
App app;

//-----
// PROGRAM ENTRY POINT, CALLED BY WINDOWS
//
int PASCAL WinMain( HANDLE hInstance,
                   HANDLE hPrevInstance,
                   LPSTR lpszCmdLine,
                   int nCmdShow )
//-----
{
    ::hInstance = hInstance;
    ::hPrevInstance = hPrevInstance;
    ::nCmdShow = nCmdShow;

/*
*   MAKE argc, argv FROM lpszCmdLine
*/
    char * argv[20];
    argv[0] = new char[ strlen( "graph" ) + 1 ];
    strcpy( argv[0], "graph" );

    int argc = 1;
    char far * start = lpszCmdLine;
    for( char far * ps = lpszCmdLine; *ps != '\0'; ps++ ) {
```

```
        if ( *ps != ' ' && *ps != '\t' ) continue;
        else if ( ps != start ) {
            argv[ argc ] = ( char * ) malloc( ps - start + 1 );
            for(int i = 0; i < ps - start; i++ )
                argv[argc][i] = start[i];
            argv[argc++][i] = '\0';
        }
    }
    if ( ps != start ) {
        argv[ argc ] = ( char * ) malloc( ps - start + 1 );
        for(int i = 0; i < ps - start; i++ )
            argv[argc][i] = start[i];
        argv[argc++][i] = '\0';
    }
}

/*
 * CALL APPLICATION DEPENDENT ENTRY
 */

app.Start( argc, argv );
return 0;
}
```

```

/*****
**      TEST FILE FOR graph (Dynamic Layout Alg)
**
**      HEADER - C++ X-WINDOW class library based on Xlib
**
** Author: dr. Szirmay-Kalos Laszlo (szirmay@fsz.bme.hu)
**      Technical University of Budapest, Hungary
*****/

#include <windows.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

/*****
//      CONSTANTS
//*****/
#define WINDOW_WIDTH      400
#define WINDOW_HEIGHT     400
#define FALSE              0
#define TRUE               1

/*****
//      TYPES
//*****/
typedef short  CoOrd;
typedef char * pchar;

//=====
class App {          // APPLICATION
//=====
public:
    void      Start( int argc, char * argv[] );
    void      Error( char * mess, int line = -1 );
    void      Warning( char * mess );
    void      Quit();
};

extern App app;

//=====
class Point {
//=====
private:
    CoOrd x;
    CoOrd y;
public:
    Point( )          { x = y = 0; }
    Point(CoOrd x0, CoOrd y0) { x = x0; y = y0; }
    CoOrd& X()        { return x; }
    CoOrd& Y()        { return y; }
};

//=====
class RectAngle {
//=====
private:
    CoOrd hor_pos;
    CoOrd ver_pos;
    CoOrd width;
    CoOrd height;
};
```

```
public:
    RectAngle( CoOrd hp, CoOrd vp, CoOrd w, CoOrd h)
        { hor_pos = hp; ver_pos = vp; width = w; height = h; }
    CoOrd& HorPos() { return hor_pos; }
    CoOrd& VerPos() { return ver_pos; }
    CoOrd& Width() { return width; }
    CoOrd& Height() { return height; }
};

//=====
class KeyEvent {
//=====
    WORD    wParam;    // word param
    LONG    lParam;    // long param
public:
    KeyEvent( WORD w, LONG l) { wParam = w; lParam = l; }
    int      GetASCII( void ) { return wParam; }
};

//=====
class ExposeEvent {
//=====
    RectAngle area;
public:
    ExposeEvent( RectAngle * pr )
        : area( *pr ) { }
    RectAngle& GetExposedArea( ) { return area; }
};

//=====
class AppWindow {
//=====
    char      szClassName[14];    // app window name
    HWND      hWnd;               // window handle
    HDC       hdc;                // device context
    PAINTSTRUCT ps;
    RectAngle canvas;
protected:
    virtual void KeyPressed( KeyEvent * ) {}
    virtual void ExposeAll( ExposeEvent * ) {}

    RectAngle Canvas( void );
    void RePaint( void );
    void Text( char * text, Point position );
    void MoveTo( Point to );
    void LineTo( Point to );
    void DrawRectangle( RectAngle& rect );
public:
    AppWindow( int argc, char * argv[] );
    long WindowProc( HWND hwnd, WORD wmsg, WORD wParam, LONG lParam );
    void MessageLoop( );
};
```

```

/*****
**      TEST FILE FOR graph (Dynamic Layout Alg)
**
**      MODUL      - 2D VECTOR OPERATIONS
**
** Author: dr. Szirmay-Kalos Laszlo (szirmay@fsz.bme.hu)
**      Technical University of Budapest, Hungary
*****/
#include <math.h>

#ifdef MSWINDOWS
#include "vector.hxx"
#else
#include "vector.hxx"
#endif

/*----- Overloaded + operator -----*/
/* Adds two vectors */
/* IN  : reference of the two operands */
/* OUT : result vector */
/*-----*/
vector operator+(vector& a, vector& b)
{
    vector sum;
    sum.x = a.x + b.x;
    sum.y = a.y + b.y;
    return sum;
}

/*----- Overloaded binary - operator -----*/
/* Subtract two vectors */
/* IN  : reference of the two operands */
/* OUT : result vector */
/*-----*/
vector operator-(vector& a, vector& b)
{
    vector dif;
    dif.x = a.x - b.x;
    dif.y = a.y - b.y;
    return dif;
}

/*----- Overloaded unary - operator -----*/
/* Negates a vector */
/* IN  : operand */
/* OUT : negated vector */
/*-----*/
vector operator-(vector& a)
{
    vector neg;
    neg.x = -a.x;
    neg.y = -a.y;
    return neg;
}

/*----- Overloaded * operator -----*/
/* Multiplies a vector with a scalar */
/* IN  : vector and scalar operand */
/* OUT : result vector */
/*-----*/
vector operator*(vector& a, double s)
```



```
{
    vector scaled;
    scaled.x = s * a.x;
    scaled.y = s * a.y;
    return scaled;
}

vector operator*(double s, vector& a)
{
    vector scaled;
    scaled.x = s * a.x;
    scaled.y = s * a.y;
    return scaled;
}

/*----- Size -----*/
/* Calculates the absolute value of the vector */
/* IN : - */
/* OUT : length */
/*-----*/
double vector::Size()
{
    return sqrt( x * x + y * y );
}
```

```

/*****
**      TEST FILE FOR graph (Dynamic Layout Alg)
**
**      HEADER      - 2D VECTOR OPERATIONS
**
** Author: dr. Szirmay-Kalos Laszlo (szirmay@fsz.bme.hu)
**      Technical University of Budapest, Hungary
*****/
/*
*      VECTOR - 2D GEOMETRIC VECTOR TYPE
*/
class      vector {
    double x;                      // coordinates
    double y;
public:
    vector()                        { x = 0.0; y = 0.0; }
    vector(double x0, double y0)   { x = x0; y = y0; }
    void      operator=(vector& a) { x = a.x; y = a.y; }
    void      operator+=(vector& a) { x += a.x; y += a.y; }
    void      operator/=(double d) { if (d != 0.0) {x /= d; y /= d;} }
    void      operator*=(double d) { x *= d; y *= d; }
    double      X()                  { return x; }
    double      Y()                  { return y; }
    double      Size();
// FRIENDS
    friend vector operator+(vector&, vector&);
    friend vector operator-(vector&, vector&);
    friend vector operator-(vector&);
    friend vector operator*(vector&, double);
    friend vector operator*(double, vector&);
};
```

```

/*****
**      TEST FILE FOR graph (Dynamic Layout Alg)
**
**      MODUL: C++ X-WINDOW class library based on Xlib
**
**      Author: dr. Szirmay-Kalos Laszlo (szirmay@fsz.bme.hu)
**              Technical University of Budapest, Hungary
*****/
#include "window.hxx"

//-----
void App :: Error( char * message, int line )
//-----
{
    fprintf( stderr, "ERROR: %s", message );
    if ( line >= 0 ) fprintf( stderr, " in line %d", line );
    fprintf( stderr, "\n" );
    Quit( );
}

//-----
void App :: Warning( char * message )
//-----
{
    fprintf( stderr, "WARNING: %s\n", message );
}

//-----
void App :: Quit( )
//-----
{
    fprintf( stderr, "Bye ( Graph )\n" );
    exit( -1 );
}

//-----
AppWindow :: AppWindow( int argc, char * argv[] )
//-----
    : canvas( 0, 0, WINDOW_WIDTH, WINDOW_HEIGHT ),
      beam( 0, 0 )
{
    if ( !(dpy = XOpenDisplay ( "" )) ) {
        fprintf( stderr, "Error: Can't open display\n" );
        exit (1);
    }

    screen = DefaultScreen (dpy);
    win = XCreateSimpleWindow(dpy,
                             DefaultRootWindow (dpy),
                             0, 0,
                             WINDOW_WIDTH, WINDOW_HEIGHT,
                             1,
                             WhitePixel (dpy, screen),
                             BlackPixel (dpy, screen));

    XSelectInput (dpy, win, StructureNotifyMask |
                    ExposureMask |
                    KeyPressMask );

    XStoreName (dpy, win, argv[0] );
    XMapWindow (dpy, win);
}

```

```
/*
 * White/Black graphics context
 */
    gc = XCreateGC (dpy, win, 0L, (XGCValues *) 0);
    XSetBackground (dpy, gc, BlackPixel (dpy, screen));
    XSetForeground (dpy, gc, WhitePixel (dpy, screen));

/*
 * Black/White graphics context
 */
    gc_inv = XCreateGC (dpy, win, 0L, (XGCValues *) 0);
    XSetForeground (dpy, gc_inv, BlackPixel (dpy, screen));
    XSetBackground (dpy, gc_inv, WhitePixel (dpy, screen));
}

//-----
void AppWindow :: MessageLoop()
//-----
{
    for ( ; ; ) {
        XEvent event;
        XNextEvent (dpy, &event);

        switch (event.type) {
            case ConfigureNotify:
                canvas = RectAngle( 0, 0, event.xconfigure.width,
event.xconfigure.height );
                break;
            case Expose:
                {
                    XClearWindow(dpy, win);
                    ExposeEvent evt( &event.xexpose );
                    ExposeAll( &evt );
                }
                break;
            case KeyPress:
                {
                    KeyEvent evt( &event.xkey );
                    KeyPressed( &evt );
                }
                break;
        }
    }
}

//-----
RectAngle AppWindow :: Canvas()
//-----
{
    return canvas;
}

//-----
void AppWindow :: RePaint()
//-----
{
    XClearWindow(dpy, win);
    ExposeAll( NULL );
}

//-----
void AppWindow :: Text( char * text, Point p)
```

```
//-----
{
    XDrawString( dpy, win, gc, p.X(), p.Y(), text, strlen(text) );
}

//-----
void AppWindow :: MoveTo( Point p )
//-----
{
    beam = p;
}

//-----
void AppWindow :: LineTo( Point p )
//-----
{
    XDrawLine( dpy, win, gc, beam.X(), beam.Y(), p.X(), p.Y() );
    beam = p;
}

//-----
void AppWindow :: DrawRectangle( RectAngle& rect )
//-----
{
    XFillRectangle( dpy, win, gc,
                    rect.HorPos(), rect.VerPos(),
                    rect.Width(), rect.Height() );
    XFillRectangle( dpy, win, gc_inv,
                    rect.HorPos() + 2, rect.VerPos() + 2,
                    rect.Width() - 4, rect.Height() - 4 );
}

/*****
/*  PROGRAM ENTRY POINT                                     */
*****/
App    app;    // Application program object

int main( int argc, char * argv[] )
{
    /*
    *  CALL APPLICATION DEPENDENT ENTRY
    */
    app.Start( argc, argv );
    app.Quit( );
}
```

```

/*****
**      TEST FILE FOR graph (Dynamic Layout Alg)
**
**      HEADER - C++ X-WINDOW class library based on Xlib
**
** Author: dr. Szirmay-Kalos Laszlo (szirmay@fsz.bme.hu)
**          Technical University of Budapest, Hungary
*****/

#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Xos.h>
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>

/*****
//      CONSTANTS
//*****/
#define WINDOW_WIDTH      400
#define WINDOW_HEIGHT     400
#define FALSE             0
#define TRUE              1

/*****
//      TYPES
//*****/
typedef short   CoOrd;
typedef char *  pchar;
typedef char    BOOL;

//=====
class App {
//=====
public:
    void      Start( int argc, char * argv[] );
    void      Error( char * mess, int line = -1 );
    void      Warning( char * mess );
    void      Quit( void );
};

extern App app;

//=====
class Point {
//=====
private:
    CoOrd x;
    CoOrd y;
public:
    Point( )                { x = y = 0; }
    Point(CoOrd x0, CoOrd y0) { x = x0; y = y0; }
    CoOrd&  X()              { return x; }
    CoOrd&  Y()              { return y; }
};

//=====
class Rectangle {
//=====
private:

```

```
CoOrd hor_pos;
CoOrd ver_pos;
CoOrd width;
CoOrd height;
```

```
public:
```

```
    RectAngle(CoOrd hp, CoOrd vp, CoOrd w, CoOrd h)
        { hor_pos = hp; ver_pos = vp; width = w; height = h; }
    CoOrd& HorPos()    { return hor_pos; }
    CoOrd& VerPos()    { return ver_pos; }
    CoOrd& Width()     { return width; }
    CoOrd& Height()    { return height; }
```

```
};
```

```
//=====
```

```
class KeyEvent {
```

```
//=====
```

```
    KeySym          key_sym;
    XComposeStatus   status;
    char             ascii;
```

```
public:
```

```
    KeyEvent( XKeyEvent * xe )
        { XLookupString( xe, &ascii, 1, &key_sym, &status ); }
    int      GetASCII( void ) { return ascii; }
```

```
};
```

```
//=====
```

```
class ExposeEvent {
```

```
//=====
```

```
    RectAngle area;
```

```
public:
```

```
    ExposeEvent( XExposeEvent * xe )
        :area(xe->x, xe->y, xe->width, xe->height) { }
```

```
    RectAngle& GetExposedArea( )    { return area; }
```

```
};
```

```
//=====
```

```
class AppWindow {
```

```
//=====
```

```
    GC          gc, gc_inv;
    Window       win;
    Display *    dpy;
    int          screen;
    RectAngle    canvas;
    Point        beam;
```

```
protected:
```

```
    virtual void KeyPressed( KeyEvent * ) {}
    virtual void ExposeAll( ExposeEvent * ) {}
```

```
    RectAngle    Canvas( void );
    void          RePaint( void );
    void          Text( char * text, Point position );
    void          MoveTo( Point to );
    void          LineTo( Point to );
    void          DrawRectangle( RectAngle& rect );
```

```
public:
```

```
    AppWindow( int argc, char * argv[] );
    void       MessageLoop( );
```

```
};
```

```
/*
 * Author: Filippo Tampieri
 */

#define FALSE 0
#define TRUE 1
#define DOT(A,B) (A[0] * B[0] + A[1] * B[1] + A[2] * B[2])

/*
 * vertexIsBehindPlane returns TRUE if point P is behind the
 * plane of normal N and coefficient d, FALSE otherwise.
 */
vertexIsBehindPlane(P, N, d)
float P[3], N[3], d;
{
    return(DOT(N, P) + d <= 0. ? TRUE : FALSE);
}

/*
 * boxIsBehindPlane returns TRUE if the axis-aligned box of
 * minimum corner Cmin and maximum corner Cmax is behind the
 * plane of normal N and coefficient d, FALSE otherwise.
 */
boxIsBehindPlane(Cmin, Cmax, N, d)
float Cmin[3], Cmax[3], N[3], d;
{
    register int i;
    float P[3];

    /*
     * assign to P the corner further away
     * along the direction of normal N
     */
    for(i = 0; i < 3; i++)
        P[i] = N[i] >= 0. ? Cmax[i] : Cmin[i];

    /* test P against the input plane */
    return(vertexIsBehindPlane(P, N, d));
}
```



```
/*  
NEWELL'S METHOD FOR COMPUTING THE PLANE EQUATION OF A POLYGON
```

```
Filippo Tampieri  
Cornell University  
*/
```

```
#include <math.h>  
  
/* definition for the components of vectors and plane equations */  
#define X    0  
#define Y    1  
#define Z    2  
#define D    3  
  
/* a few useful vector operations */  
#define VZERO(v)    (v[X] = v[Y] = v[Z] = 0.0)  
#define VNORM(v)    (sqrt(v[X] * v[X] + v[Y] * v[Y] + v[Z] * v[Z]))  
#define VDOT(u, v)  (u[0] * v[0] + u[1] * v[1] + u[2] * v[2])  
#define VINCR(u, v) (u[X] += v[X], u[Y] += v[Y], u[Z] += v[Z])  
  
/* type definitions for vectors and plane equations */  
typedef float Vector[3];  
typedef Vector Point;  
typedef Vector Normal;  
typedef float Plane[4];  
  
/*  
**  PlaneEquation--computes the plane equation of an arbitrary  
**  3D polygon using Newell's method.  
**  
**  Entry:  
**      verts - list of the vertices of the polygon  
**      nverts - number of vertices of the polygon  
**  Exit:  
**      plane - normalized (unit normal) plane equation  
**/  
  
PlaneEquation(verts, nverts, plane)  
Point *verts;  
int nverts;  
Plane plane;  
{  
    int i;  
    Point refpt;  
    Normal normal;  
    float *u, *v, len;
```

```
/* compute the polygon normal and a reference point on
   the plane. Note that the actual reference point is
   refpt / nverts
*/
VZERO(normal);
VZERO(refpt);
for(i = 0; i < nverts; i++) {
    u = verts[i];
    v = verts[(i + 1) % nverts];
    normal[X] += (u[Y] - v[Y]) * (u[Z] + v[Z]);
    normal[Y] += (u[Z] - v[Z]) * (u[X] + v[X]);
    normal[Z] += (u[X] - v[X]) * (u[Y] + v[Y]);
    VINCR(refpt, u);
}
/* normalize the polygon normal to obtain the first
   three coefficients of the plane equation
*/
len = VNORM(normal);
plane[X] = normal[X] / len;
plane[Y] = normal[Y] / len;
plane[Z] = normal[Z] / len;
/* compute the last coefficient of the plane equation */
len *= nverts;
plane[D] = -VDOT(refpt, normal) / len;
}
```

```
/*
ACCURATE FORM-FACTOR COMPUTATION

Filippo Tampieri
Cornell University
*/
/*
    The routines in this module provide a way of computing the
    radiosity contribution of a source patch to a point on a
    receiving surface.  The source must be a convex quadrilateral
    and have a uniform radiosity distribution (or approximately
    uniform).  For simplicity and clarity, the radiosity is
    computed for a single wavelength.
*/

#include <stdio.h>
#include <math.h>

static float computeFormFactor() ;
static float computeUnoccludedFormFactor() ;
static void splitQuad() ;
static float quadArea() ;

#define X    0
#define Y    1
#define Z    2

/* set vector v to zero */
#define VZERO(v)      (v[X] = v[Y] = v[Z] = 0.0)
/* increment vector v by vector dv */
#define VINCR(v, dv)  (v[X] += dv[X], v[Y] += dv[Y], v[Z] += dv[Z])
/* scale vector v by a constant c */
#define VSCALE(v, c)  (v[X] *= c, v[Y] *= c, v[Z] *= c)
/* copy vector u to vector v */
#define VCOPY(v, u)   (v[X] = u[X], v[Y] = u[Y], v[Z] = u[Z])
/* compute the dot product of vectors u and v */
#define VDOT(u, v)    (u[X] * v[X] + u[Y] * v[Y] + u[Z] * v[Z])
/* compute the cross product of vectors s and t */
#define VCROSS(v, s, t) (v[X] = s[Y] * t[Z] - s[Z] * t[Y], \
                        v[Y] = s[Z] * t[X] - s[X] * t[Z], \
                        v[Z] = s[X] * t[Y] - s[Y] * t[X])

/* compute the length of vector v */
#define VNORM(v)      (sqrt(VDOT(v, v)))
/* subtract vector t from vector s and assign the result to v */
#define VSUB(v, s, t) (v[X] = s[X] - t[X], \
                        v[Y] = s[Y] - t[Y], \
                        v[Z] = s[Z] - t[Z])

/* compute the average of vectors s and t */
#define VAVG2(v, s, t) (v[X] = 0.5 * (s[X] + t[X]), \
                        v[Y] = 0.5 * (s[Y] + t[Y]), \
                        v[Z] = 0.5 * (s[Z] + t[Z]))

typedef float Vector[3];
typedef Vector Point;
typedef Vector Normal;

/* represent a quadrilateral as an array of four points */
typedef Point Quad[4];

/*
    computeVisibility(Pr, S)

```

```
Point Pr;  
Quad S;
```

```
computeVisibility--takes as input a receiving point 'Pr' and a  
source 'S' and estimates the visibility of 'S' as seen from 'Pr'.  
It returns 0 for total occlusion, 1 for total visibility, and a  
number representing the fraction of 'S' that can be seen from  
'Pr' in the case of partial occlusion.
```

```
*/  
extern float computeVisibility();
```

```
static float eps_F, eps_A;
```

```
/*  
computeContribution--takes as input the descriptions of a source  
patch and a receiving point and returns the radiosity  
contribution of the source to the receiver. The source is  
described as a quadrilateral 'S', its UNIT normal 'Ns', and its  
average unshot radiosity 'Bs'. The receiving point is described  
by its location 'Pr', its UNIT normal 'Nr', and its reflectivity  
'rho'. The parameter 'eps_B' controls the accuracy of the  
computation in the presence of partial occlusion; smaller values  
of 'eps_B' will result in more accurate estimates at the cost of  
computation speed. The parameter 'minA' provides the means to  
control the recursive subdivision of the source: no region of  
area less than or equal to 'minA' will be subdivided.
```

```
*/  
float computeContribution(Pr, Nr, rho, S, Ns, Bs, eps_B, minA)  
Point Pr;
```

```
Normal Nr, Ns;
```

```
float rho, Bs, eps_B, minA;
```

```
Quad S;
```

```
{  
    Vector v;  
    float computeFormFactor();
```

```
    if(VDOT(Nr, Ns) >= 0.0)  
        /* the receiving point is oriented away from the source */  
        return 0.0;
```

```
    VSUB(v, Pr, S[0]);  
    if(VDOT(Ns, v) <= 0.0)  
        /* the receiving point is behind the source */  
        return 0.0;
```

```
    eps_F = eps_B / (rho * Bs);
```

```
    eps_A = minA;
```

```
    return(rho * Bs * computeFormFactor(Pr, Nr, S, Ns));
```

```
}
```

```
/*  
computeFormFactor--takes as input the descriptions of a source  
patch and a receiving point and returns the form-factor from a  
differential area centered around the receiving point to the  
finite area source. The source is described as a quadrilateral  
'S' and its UNIT normal 'Ns'. The receiving point is described  
by its location 'Pr' and its UNIT normal 'Nr'. This routine  
relies on the external function 'computeVisibility' to determine  
a visibility factor.
```

```
*/
static float computeFormFactor(Pr, Nr, S, Ns)
Point Pr;
Normal Nr, Ns;
Quad S;
{
    int j;
    Quad dS[4];
    void splitQuad();
    float Frs, vis, computeUnoccludedFormFactor(), quadArea();

    vis = computeVisibility(Pr, S);
    if(vis <= 0.0)
        Frs = 0.0;
    else if(vis >= 1.0)
        Frs = computeUnoccludedFormFactor(Pr, Nr, S);
    else {
        Frs = computeUnoccludedFormFactor(Pr, Nr, S);
        if(Frs <= eps_F || quadArea(S) <= eps_A)
            return(Frs * vis);
        else {
            splitQuad(S, dS);
            Frs = 0.0;
            for(j = 0; j < 4; j++)
                Frs += computeFormFactor(Pr, Nr, dS[j], Ns);
        }
    }

    return(Frs);
}

/*
computeUnoccludedFormFactor--takes as input the descriptions of
a source patch and a receiving point and returns the unoccluded
form-factor from a differential area centered around the
receiving point to the finite area source. The source is
described as a quadrilateral 'S' and its UNIT normal 'Ns'.
The receiving point is described by its location 'Pr' and its
UNIT normal 'Nr'. The form-factor is computed analytically.
*/
static float computeUnoccludedFormFactor(Pr, Nr, S)
Point Pr;
Normal Nr;
Quad S;
{
    int i;
    float f, c;
    Vector s, t, sxt, s0;

    f = 0.0;
    VSUB(s, S[3], Pr);
    c = 1.0 / VNORM(s);
    VSCALE(s, c);
    VCOPY(s0, s);
    for(i = 0; i < 4; i++) {
        if(i < 3) {
            VSUB(t, S[i], Pr);
            c = 1.0 / VNORM(t);
            VSCALE(t, c);
        } else
            VCOPY(t, s0);
    }
}
```

```
        VCROSS(sxt, s, t);
        c = 1.0 / VNORM(sxt);
        VSCALE(sxt, c);
        f -= acos(VDOT(s, t)) * VDOT(sxt, Nr);
        VCOPY(s, t);
    }

    return(f / (2.0 * M_PI));
}

/*
    splitQuad--takes as input a quadrilateral 'Q', splits it into
        four equal quadrilaterals, and returns the result in array 'dQ'.
*/
static void splitQuad(Q, dQ)
Quad Q, dQ[4];
{
    int i, j;
    Point center, midpt[4];

    /* compute the center of 'Q' and the midpoint of its edges */
    VZERO(center);
    for(i = 0; i < 4; i++) {
        j = (i + 1) % 4;
        VINCR(center, Q[i]);
        VAVG2(midpt[i], Q[i], Q[j]);
    }
    VSCALE(center, 0.25);

    /* initialize the four new quadrilaterals */
    VCOPY(dQ[0][0], Q[0]);
    VCOPY(dQ[0][1], midpt[0]);
    VCOPY(dQ[0][2], center);
    VCOPY(dQ[0][3], midpt[3]);

    VCOPY(dQ[1][1], Q[1]);
    VCOPY(dQ[1][2], midpt[1]);
    VCOPY(dQ[1][3], center);
    VCOPY(dQ[1][0], midpt[0]);

    VCOPY(dQ[2][2], Q[2]);
    VCOPY(dQ[2][3], midpt[2]);
    VCOPY(dQ[2][0], center);
    VCOPY(dQ[2][1], midpt[1]);

    VCOPY(dQ[3][3], Q[3]);
    VCOPY(dQ[3][0], midpt[3]);
    VCOPY(dQ[3][1], center);
    VCOPY(dQ[3][2], midpt[2]);
}

/*
    quadArea--takes as input a quadrilateral 'Q' and returns its
        area. The area of the quadrilateral is computed as the sum of
        the areas of the two triangles obtained by splitting the
        quadrilateral along one of its diagonals.
*/
static float quadArea(Q)
Quad Q;
{
    float area;
```





```
    Vector u, v, uxv;

    VSUB(u, Q[2], Q[1]);
    VSUB(v, Q[0], Q[1]);
    VCROSS(uxv, u, v);
    area = VNORM(uxv);

    VSUB(u, Q[0], Q[3]);
    VSUB(v, Q[2], Q[3]);
    VCROSS(uxv, u, v);
    area += VNORM(uxv);

    return area * 0.5;
}
```

# Index of /pubs/tog/GraphicsGems/gemsii/inv\_cmap/

Name	Last modified	Size	Description
 <a href="#">Parent Directory</a>			
 <a href="#">Makefile</a>	29-Jun-00 08:14	1K	
 <a href="#">inv_cmap.3</a>	29-Jun-00 08:14	1K	
 <a href="#">inv_cmap.c</a>	29-Jun-00 08:14	13K	



CFLAGS = -g

inv\_cmap.o: inv\_cmap.c  
cc \$(CFLAGS) -c inv\_cmap.c -o inv\_cmap.o

clean:  
/bin/rm -f inv\_cmap.o

```
.\ " *- Text *-
.\ " Copyright (c) 1990, University of Michigan
.\ " Template man page.
.TH INV_CMAP 3 "Month DD, YYYY" 1
.UC 4
.SH NAME
inv_cmap \- efficiently compute an inverse colormap
.SH SYNOPSIS
.HP
.B
void inv_cmap( colors, colormap, bits, dist_buf, rgbmap )
.LP
.B
int colors, bits;
.br
.B
unsigned char *colormap[3], *rgbmap;
.br
.B
unsigned long *dist_buf;
.SH DESCRIPTION
.I Inv_cmap
computes an inverse colormap to translate an RGB color to the nearest
color in the given \ficolormap\fP. The arguments are
.TP
.I colors
The number of colors in the input colormap. Must be  $\leq 256$ .
.TP
.I colormap
The input colormap. The  $i$ th color is  $(\text{colormap}[0][i], \text{colormap}[1][i], \text{colormap}[2][i])$ .
.TP
.I bits
Controls the size and precision of the inverse colormap. The
resulting colormap will be a cube  $2^{\text{bits}}$  on a side, and will
therefore contain  $2^{(3*\text{bits})}$  entries. RGB colors must be
quantized to  $\text{bits}$  bits before using the inverse colormap.
.TP
.I dist_buf
Temporary storage used by inv_cmap. It should contain at least
 $2^{(3*\text{bits})}$  elements.
.TP
.I rgbmap
The inverse colormap. Should be allocated with at least
 $2^{(3*\text{bits})}$  elements. After calling inv_cmap, an RGB color
(r,g,b) can be mapped to its closest representative in icolormap
by evaluating
.br
#define quantize(p) ((p)>>(8-bits))
.br
rgbmap[ (((quantize(r) << bits) | quantize(g)) << bits) | quantize(b) ]
.PP
Predicted performance is  $O(2^{(3*\text{bits})} * \log(\text{colors}))$ . The
measured performance is sublinear (but not as good as  $\log$ ) in
the number of input colors and also in the size of the output inverse
colormap. (I.e., it goes up more slowly than  $2^{(3*\text{bits})}$ .)
.SH SEE ALSO
.IR colorquant (3).
.SH AUTHOR
Spencer W. Thomas
```

```
/*
 * This software is copyrighted as noted below.  It may be freely copied,
 * modified, and redistributed, provided that the copyright notice is
 * preserved on all copies.
 *
 * There is no warranty or other guarantee of fitness for this software,
 * it is provided solely "as is".  Bug reports or fixes may be sent
 * to the author, who may or may not act on them as he desires.
 *
 * You may not include this software in a program or other software product
 * without supplying the source, or without informing the end-user that the
 * source is available for no extra charge.
 *
 * If you modify this software, you should include a notice giving the
 * name of the person performing the modification, the date of modification,
 * and the reason for such modification.
 */
/*
 * inv_cmap.c - Compute an inverse colormap.
 *
 * Author:      Spencer W. Thomas
 *              EECS Dept.
 *              University of Michigan
 * Date:        Thu Sep 20 1990
 * Copyright (c) 1990, University of Michigan
 *
 * $Id: inv_cmap.c,v 3.0.1.3 1992/04/30 14:07:28 spencer Exp $
 */
```

```
#include <math.h>
#include <stdio.h>
```

```
static int bcenter, gcenter, rcenter;
static long gdist, rdist, cdist;
static long cbinc, cginc, crinc;
static unsigned long *gdp, *rdp, *cdp;
static unsigned char *grgbp, *rrgbp, *crgbp;
static gstride, rstride;
static long x, xsqr, colormax;
static int cindex;
```

```
#ifdef USE_PROTOTYPES
static void maxfill( unsigned long *, long );
static int redloop( void );
static int greenloop( int );
static int blueloop( int );
#else
static void maxfill();
static int redloop();
static int greenloop();
static int blueloop();
#endif
```

```
/******
 * TAG( inv_cmap )
 *
 * Compute an inverse colormap efficiently.
 * Inputs:
 *     colors:      Number of colors in the forward colormap.
```

```

* colormap:      The forward colormap.
* bits:          Number of quantization bits.  The inverse
*                colormap will have  $(2^{\text{bits}})^3$  entries.
* dist_buf:      An array of  $(2^{\text{bits}})^3$  long integers to be
*                used as scratch space.
*
* Outputs:
*   rgbmap:      The output inverse colormap.  The entry
*                rgbmap[(r<<(2*bits)) + (g<<bits) + b]
*                is the colormap entry that is closest to the
*                (quantized) color (r,g,b).
*
* Assumptions:
*   Quantization is performed by right shift (low order bits are
*   truncated).  Thus, the distance to a quantized color is
*   actually measured to the color at the center of the cell
*   (i.e., to  $r+0.5$ ,  $g+0.5$ ,  $b+0.5$ , if (r,g,b) is a quantized color).
*
* Algorithm:
*   Uses a "distance buffer" algorithm:
*   The distance from each representative in the forward color map
*   to each point in the rgb space is computed.  If it is less
*   than the distance currently stored in dist_buf, then the
*   corresponding entry in rgbmap is replaced with the current
*   representative (and the dist_buf entry is replaced with the
*   new distance).
*
*   The distance computation uses an efficient incremental formulation.
*
*   Distances are computed "outward" from each color.  If the
*   colors are evenly distributed in color space, the expected
*   number of cells visited for color I is  $N^3/I$ .
*   Thus, the complexity of the algorithm is  $O(\log(K) N^3)$ ,
*   where  $K$  = colors, and  $N = 2^{\text{bits}}$ .
*/

```

```
*
* Here's the idea:  scan from the "center" of each cell "out"
* until we hit the "edge" of the cell -- that is, the point
* at which some other color is closer -- and stop.  In 1-D,
* this is simple:
*     for i := here to max do
*         if closer then buffer[i] = this color
*         else break
*     repeat above loop with i := here-1 to min by -1
```

```
* In 2-D, it's trickier, because along a "scan-line", the
* region might start "after" the "center" point.  A picture
* might clarify:
```

```

      *          |          . . .
      *          |          . . .
      *          |          . . .
      *          |          .
      *          |          .
      *          |          .
      *          +          .
      *          .          .
      *          .          .
      *          .          .
      *          . . . . .

```

\* The + marks the "center" of the above region. On the top 2  
\* lines, the region "begins" to the right of the "center".

```
* Thus, we need a loop like this:
*   detect := false
*   for i := here to max do
```

```
*           if closer then
*               buffer[..., i] := this color
*               if !detect then
*                   here = i
*                   detect = true
*           else
*               if detect then
*                   break
*
* Repeat the above loop with i := here-1 to min by -1. Note that
* the "detect" value should not be reinitialized. If it was
* "true", and center is not inside the cell, then none of the
* cell lies to the left and this loop should exit
* immediately.
*
* The outer loops are similar, except that the "closer" test
* is replaced by a call to the "next in" loop; its "detect"
* value serves as the test. (No assignment to the buffer is
* done, either.)
*
* Each time an outer loop starts, the "here", "min", and
* "max" values of the next inner loop should be
* re-initialized to the center of the cell, 0, and cube size,
* respectively. Otherwise, these values will carry over from
* one "call" to the inner loop to the next. This tracks the
* edges of the cell and minimizes the number of
* "unproductive" comparisons that must be made.
*
* Finally, the inner-most loop can have the "if !detect"
* optimized out of it by splitting it into two loops: one
* that finds the first color value on the scan line that is
* in this cell, and a second that fills the cell until
* another one is closer:
*     if !detect then      {needed for "down" loop}
*         for i := here to max do
*             if closer then
*                 buffer[..., i] := this color
*                 detect := true
*                 break
*         for i := i+1 to max do
*             if closer then
*                 buffer[..., i] := this color
*             else
*                 break
*
* In this implementation, each level will require the
* following variables. Variables labelled (l) are local to each
* procedure. The ? should be replaced with r, g, or b:
*     cdist:           The distance at the starting point.
*     ?center:         The value of this component of the color
*     c?inc:           The initial increment at the ?center position.
*     ?stride:         The amount to add to the buffer
*                     pointers (dp and rgbp) to get to the
*                     "next row".
*     min(l):          The "low edge" of the cell, init to 0
*     max(l):          The "high edge" of the cell, init to
*                     colormax-1
*     detect(l):       True if this row has changed some
*                     buffer entries.
*     i(l):            The index for this row.
*     ?xx:             The accumulated increment value.
```

```
*
*      here(1):      The starting index for this color.  The
*                    following variables are associated with here,
*                    in the sense that they must be updated if here
*                    is changed.
*      ?dist:        The current distance for this level.  The
*                    value of dist from the previous level (g or r,
*                    for level b or g) initializes dist on this
*                    level.  Thus gdist is associated with here(b)).
*      ?inc:          The initial increment for the row.
*      ?dp:           Pointer into the distance buffer.  The value
*                    from the previous level initializes this level.
*      ?rgbp:         Pointer into the rgb buffer.  The value
*                    from the previous level initializes this level.
*
* The blue and green levels modify 'here-associated' variables (dp,
* rgbp, dist) on the green and red levels, respectively, when here is
* changed.
*/
```

```
void
inv_cmap( colors, colormap, bits, dist_buf, rgbmap )
int colors, bits;
unsigned char *colormap[3], *rgbmap;
unsigned long *dist_buf;
{
    int nbits = 8 - bits;

    colormapax = 1 << bits;
    x = 1 << nbits;
    xsqr = 1 << (2 * nbits);

    /* Compute "strides" for accessing the arrays. */
    gstride = colormapax;
    rstride = colormapax * colormapax;

    maxfill( dist_buf, colormapax );

    for ( cindex = 0; cindex < colors; cindex++ )
    {
        /*
        * Distance formula is
        * (red - map[0])^2 + (green - map[1])^2 + (blue - map[2])^2
        *
        * Because of quantization, we will measure from the center of
        * each quantized "cube", so blue distance is
        * (blue + x/2 - map[2])^2,
        * where x = 2^(8 - bits).
        * The step size is x, so the blue increment is
        * 2*x*blue - 2*x*map[2] + 2*x^2
        *
        * Now, b in the code below is actually blue/x, so our
        * increment will be 2*(b*x^2 + x^2 - x*map[2]).  For
        * efficiency, we will maintain this quantity in a separate variable
        * that will be updated incrementally by adding 2*x^2 each time.
        */
        /* The initial position is the cell containing the colormap
        * entry.  We get this by quantizing the colormap values.
        */
        rcenter = colormap[0][cindex] >> nbits;
        gcenter = colormap[1][cindex] >> nbits;
```

```
bcenter = colormap[2][cindex] >> nbits;

rdist = colormap[0][cindex] - (rcenter * x + x/2);
gdist = colormap[1][cindex] - (gcenter * x + x/2);
cdist = colormap[2][cindex] - (bcenter * x + x/2);
cdist = rdist*rdist + gdist*gdist + cdist*cdist;

crinc = 2 * ((rcenter + 1) * xsqr - (colormap[0][cindex] * x));
cginc = 2 * ((gcenter + 1) * xsqr - (colormap[1][cindex] * x));
cbinc = 2 * ((bcenter + 1) * xsqr - (colormap[2][cindex] * x));

/* Array starting points. */
cdp = dist_buf + rcenter * rstride + gcenter * gstride + bcenter;
crgbp = rgbmap + rcenter * rstride + gcenter * gstride + bcenter;

(void)redloop();
```

```
    }
}

/* redloop -- loop up and down from red center. */
static int
redloop()
{
```

```
    int detect;
    int r;
    int first;
    long txsqr = xsqr + xsqr;
    static long rxx;

    detect = 0;

    /* Basic loop up. */
    for ( r = rcenter, rdist = cdist, rxx = crinc,
          rdp = cdp, rrgbp = crgbp, first = 1;
          r < colormap;
          r++, rdp += rstride, rrgbp += rstride,
          rdist += rxx, rxx += txsqr, first = 0 )
    {
        if ( greenloop( first ) )
            detect = 1;
        else if ( detect )
            break;
    }
```

```
    /* Basic loop down. */
    for ( r = rcenter - 1, rxx = crinc - txsqr, rdist = cdist - rxx,
          rdp = cdp - rstride, rrgbp = crgbp - rstride, first = 1;
          r >= 0;
          r--, rdp -= rstride, rrgbp -= rstride,
          rxx -= txsqr, rdist -= rxx, first = 0 )
    {
        if ( greenloop( first ) )
            detect = 1;
        else if ( detect )
            break;
    }
```

```
    return detect;
}
```

```
/* greenloop -- loop up and down from green center. */
```

```
static int
greenloop( restart )
int restart;
{
    int detect;
    int g;
    int first;
    long txsqr = xsqr + xsqr;
    static int here, min, max;
    static long ginc, gxx, gcdist;          /* "gc" variables maintain correct */
    static unsigned long *gcdp;             /* values for bcenter position, */
    static unsigned char *gcrGBP;          /* despite modifications by blueloop */
                                           /* to gdist, gdp, grGBP. */

    if ( restart )
    {
        here = gcenter;
        min = 0;
        max = colormax - 1;
        ginc = cginc;
    }

    detect = 0;

    /* Basic loop up. */
    for ( g = here, gcdist = gdist = rdist, gxx = ginc,
          gcdp = gdp = rdp, gcrGBP = grGBP = rrGBP, first = 1;
          g <= max;
          g++, gdp += gstride, gcdp += gstride, grGBP += gstride, gcrGBP += gstride,
          gdist += gxx, gcdist += gxx, gxx += txsqr, first = 0 )
    {
        if ( blueloop( first ) )
        {
            if ( !detect )
            {
                /* Remember here and associated data! */
                if ( g > here )
                {
                    here = g;
                    rdp = gcdp;
                    rrGBP = gcrGBP;
                    rdist = gcdist;
                    ginc = gxx;
                }
                detect = 1;
            }
        }
        else if ( detect )
        {
            break;
        }
    }

    /* Basic loop down. */
    for ( g = here - 1, gxx = ginc - txsqr, gcdist = gdist = rdist - gxx,
          gcdp = gdp = rdp - gstride, gcrGBP = grGBP = rrGBP - gstride,
          first = 1;
          g >= min;
          g--, gdp -= gstride, gcdp -= gstride, grGBP -= gstride, gcrGBP -= gstride,
          gxx -= txsqr, gdist -= gxx, gcdist -= gxx, first = 0 )
    {
```



```
    if ( blueloop( first ) )
    {
        if ( !detect )
        {
            /* Remember here! */
            here = g;
            rdp = gcdp;
            rrgbp = gcrgbp;
            rdist = gcdist;
            ginc = gxx;
            detect = 1;
        }
    }
    else if ( detect )
    {
        break;
    }
}

return detect;
}

/* blueloop -- loop up and down from blue center. */
static int
blueloop( restart )
int restart;
{
    int detect;
    register unsigned long *dp;
    register unsigned char *rgbp;
    register long bdist, bxx;
    register int b, i = cindex;
    register long txsqr = xsqr + xsqr;
    register int lim;
    static int here, min, max;
    static long binc;

    if ( restart )
    {
        here = bcenter;
        min = 0;
        max = colormax - 1;
        binc = cbinc;
    }

    detect = 0;

    /* Basic loop up. */
    /* First loop just finds first applicable cell. */
    for ( b = here, bdist = gdist, bxx = binc, dp = gdp, rgbp = grgbp, lim = max;
          b <= lim;
          b++, dp++, rgbp++,
          bdist += bxx, bxx += txsqr )
    {
        if ( *dp > bdist )
        {
            /* Remember new 'here' and associated data! */
            if ( b > here )
            {
                here = b;
            }
        }
    }
}
```

```
        gdp = dp;
        grgbp = rgbp;
        gdist = bdist;
        binc = bxx;
    }
    detect = 1;
    break;
}
}
/* Second loop fills in a run of closer cells. */
for ( ;
    b <= lim;
    b++, dp++, rgbp++,
    bdist += bxx, bxx += txsqr )
{
    if ( *dp > bdist )
    {
        *dp = bdist;
        *rgbp = i;
    }
    else
    {
        break;
    }
}

/* Basic loop down. */
/* Do initializations here, since the 'find' loop might not get
 * executed.
 */
lim = min;
b = here - 1;
bxx = binc - txsqr;
bdist = gdist - bxx;
dp = gdp - 1;
rgbp = grgbp - 1;
/* The 'find' loop is executed only if we didn't already find
 * something.
 */
if ( !detect )
    for ( ;
        b >= lim;
        b--, dp--, rgbp--,
        bxx -= txsqr, bdist -= bxx )
    {
        if ( *dp > bdist )
        {
            /* Remember here! */
            /* No test for b against here necessary because b <
             * here by definition.
             */
            here = b;
            gdp = dp;
            grgbp = rgbp;
            gdist = bdist;
            binc = bxx;
            detect = 1;
            break;
        }
    }
}
/* The 'update' loop. */
```

```
for ( ;
      b >= lim;
      b--, dp--, rgbp--,
      bxx -= txsqr, bdist -= bxx )
{
    if ( *dp > bdist )
    {
        *dp = bdist;
        *rgbp = i;
    }
    else
    {
        break;
    }
}

/* If we saw something, update the edge trackers. */

return detect;
}
```

```
static void
maxfill( buffer, side )
unsigned long *buffer;
long side;
{
    register unsigned long maxv = ~0L;
    register long i;
    register unsigned long *bp;

    for ( i = side * side * side, bp = buffer;
          i > 0;
          i--, bp++ )
        *bp = maxv;
}
```

```
/* unmatrix.c - given a 4x4 matrix, decompose it into standard operations.
 *
 * Author:      Spencer W. Thomas
 *              University of Michigan
 */
#include <math.h>
#include "GraphicsGems.h"
#include "unmatrix.h"

/* unmatrix - Decompose a non-degenerate 4x4 transformation matrix into
 * the sequence of transformations that produced it.
 * [Sx][Sy][Sz][Shearx/y][Sx/z][Sz/y][Rx][Ry][Rz][Tx][Ty][Tz][P(x,y,z,w)]
 *
 * The coefficient of each transformation is returned in the corresponding
 * element of the vector tran.
 *
 * Returns 1 upon success, 0 if the matrix is singular.
 */
int
unmatrix( mat, tran )
Matrix4 *mat;
double tran[16];
{
    register int i, j;
    Matrix4 locmat;
    Matrix4 pmat, invpmat, tinvpmat;
    /* Vector4 type and functions need to be added to the common set. */
    Vector4 prhs, psol;
    Point3 row[3], pdum3;

    locmat = *mat;
    /* Normalize the matrix. */
    if ( locmat.element[3][3] == 0 )
        return 0;
    for ( i=0; i<4; i++ )
        for ( j=0; j<4; j++ )
            locmat.element[i][j] /= locmat.element[3][3];
    /* pmat is used to solve for perspective, but it also provides
     * an easy way to test for singularity of the upper 3x3 component.
     */
    pmat = locmat;
    for ( i=0; i<3; i++ )
        pmat.element[i][3] = 0;
    pmat.element[3][3] = 1;

    if ( det4x4(pmat) == 0.0 )
        return 0;

    /* First, isolate perspective.  This is the messiest. */
    if ( locmat.element[0][3] != 0 || locmat.element[1][3] != 0 ||
        locmat.element[2][3] != 0 ) {
        /* prhs is the right hand side of the equation. */
        prhs.x = locmat.element[0][3];
        prhs.y = locmat.element[1][3];
        prhs.z = locmat.element[2][3];
        prhs.w = locmat.element[3][3];

        /* Solve the equation by inverting pmat and multiplying
         * prhs by the inverse.  (This is the easiest way, not
         * necessarily the best.)
         * inverse function (and det4x4, above) from the Matrix

```

```
        * Inversion gem in the first volume.
        */
        inverse( &pmat, &invpmat );
        TransposeMatrix4( &invpmat, &tinvpmat );
        V4MulPointByMatrix(&prhs, &tinvpmat, &psol);

        /* Stuff the answer away. */
        tran[U_PERSPX] = psol.x;
        tran[U_PERSPY] = psol.y;
        tran[U_PERSPZ] = psol.z;
        tran[U_PERSPW] = psol.w;
        /* Clear the perspective partition. */
        locmat.element[0][3] = locmat.element[1][3] =
            locmat.element[2][3] = 0;
        locmat.element[3][3] = 1;
    } else /* No perspective. */
        tran[U_PERSPX] = tran[U_PERSPY] = tran[U_PERSPZ] =
            tran[U_PERSPW] = 0;

    /* Next take care of translation (easy). */
    for ( i=0; i<3; i++ ) {
        tran[U_TRANSX + i] = locmat.element[3][i];
        locmat.element[3][i] = 0;
    }

    /* Now get scale and shear. */
    for ( i=0; i<3; i++ ) {
        row[i].x = locmat.element[i][0];
        row[i].y = locmat.element[i][1];
        row[i].z = locmat.element[i][2];
    }

    /* Compute X scale factor and normalize first row. */
    tran[U_SCALEX] = V3Length(&row[0]);
    row[0] = *V3Scale(&row[0], 1.0);

    /* Compute XY shear factor and make 2nd row orthogonal to 1st. */
    tran[U_SHEARXY] = V3Dot(&row[0], &row[1]);
    (void)V3Combine(&row[1], &row[0], &row[1], 1.0, -tran[U_SHEARXY]);

    /* Now, compute Y scale and normalize 2nd row. */
    tran[U_SCALEY] = V3Length(&row[1]);
    V3Scale(&row[1], 1.0);
    tran[U_SHEARXY] /= tran[U_SCALEY];

    /* Compute XZ and YZ shears, orthogonalize 3rd row. */
    tran[U_SHEARXZ] = V3Dot(&row[0], &row[2]);
    (void)V3Combine(&row[2], &row[0], &row[2], 1.0, -tran[U_SHEARXZ]);
    tran[U_SHEARYZ] = V3Dot(&row[1], &row[2]);
    (void)V3Combine(&row[2], &row[1], &row[2], 1.0, -tran[U_SHEARYZ]);

    /* Next, get Z scale and normalize 3rd row. */
    tran[U_SCALEZ] = V3Length(&row[2]);
    V3Scale(&row[2], 1.0);
    tran[U_SHEARXZ] /= tran[U_SCALEZ];
    tran[U_SHEARYZ] /= tran[U_SCALEZ];

    /* At this point, the matrix (in rows[]) is orthonormal.
     * Check for a coordinate system flip. If the determinant
     * is -1, then negate the matrix and the scaling factors.
     */
```

```
if ( V3Dot( &row[0], V3Cross( &row[1], &row[2], &pdum3) ) < 0 )
    for ( i = 0; i < 3; i++ ) {
        tran[U_SCALEX+i] *= -1;
        row[i].x *= -1;
        row[i].y *= -1;
        row[i].z *= -1;
    }












/* Now, get the rotations out, as described in the gem. */
tran[U_ROTATEY] = asin(-row[0].z);
if ( cos(tran[U_ROTATEY]) != 0 ) {
    tran[U_ROTATEX] = atan2(row[1].z, row[2].z);
    tran[U_ROTATEZ] = atan2(row[0].y, row[0].x);
} else {
    tran[U_ROTATEX] = atan2(row[1].x, row[1].y);
    tran[U_ROTATEZ] = 0;
}
/* All done! */
return 1;
}
```

```
/* transpose rotation portion of matrix a, return b */
Matrix4 *TransposeMatrix4(a, b)
Matrix4 *a, *b;
{
    int i, j;
    for (i=0; i<4; i++)
        for (j=0; j<4; j++)
            b->element[i][j] = a->element[j][i];

    return(b);
}
```

```
/* multiply a hom. point by a matrix and return the transformed point */
Vector4 *V4MulPointByMatrix(pin, m, pout)
Vector4 *pin, *pout;
Matrix4 *m;
{
    pout->x = (pin->x * m->element[0][0]) + (pin->y * m->element[1][0]) +
              (pin->z * m->element[2][0]) + (pin->w * m->element[3][0]);
    pout->y = (pin->x * m->element[0][1]) + (pin->y * m->element[1][1]) +
              (pin->z * m->element[2][1]) + (pin->w * m->element[3][1]);
    pout->z = (pin->x * m->element[0][2]) + (pin->y * m->element[1][2]) +
              (pin->z * m->element[2][2]) + (pin->w * m->element[3][2]);
    pout->w = (pin->x * m->element[0][3]) + (pin->y * m->element[1][3]) +
              (pin->z * m->element[2][3]) + (pin->w * m->element[3][3]);
    return(pout);
}
```

# Index of /pubs/tog/GraphicsGems/gems/AALines/

Name	Last modified	Size	Description
 <a href="#">Parent Directory</a>			
 <a href="#">_00README</a>	29-Jun-00 08:12	1K	
 <a href="#">AALines.c</a>	29-Jun-00 08:12	5K	
 <a href="#">AALines.h</a>	29-Jun-00 08:12	1K	
 <a href="#">AAMain.c</a>	29-Jun-00 08:12	1K	
 <a href="#">AATables.c</a>	29-Jun-00 08:12	5K	
 <a href="#">FastMatMul.c</a>	29-Jun-00 08:12	12K	
 <a href="#">LongConst.h</a>	29-Jun-00 08:12	1K	
 <a href="#">Makefile</a>	29-Jun-00 08:12	1K	
 <a href="#">utah.c</a>	29-Jun-00 08:12	4K	
 <a href="#">utah.h</a>	29-Jun-00 08:12	1K	

This group of files is a simple demonstration of an anti-aliased line renderer from `_Graphics_Gems_`. Files in the release are:

`00README` -- This information file.

`Makefile` -- Makefile for creating the demo executable.

`AALines.h` -- Include file for demo source files.

`AALines.c` -- Rendering code from `_Graphics_Gems_` pages 690-693.

`AATables.c` -- Initialization code for frame buffer and lookup tables.

`AAMain.c` -- Calling routine for the renderer.

`utah.h` -- Include file for friendly Utah RLE front end.

`utah.c` -- Source for friendly Utah RLE front end.

As it is written, the program dumps its frame buffer to a Utah RLE file. You need to obtain the Utah RLE library from another source; try the following FTP sites:

`cs.utah.edu` (128.110.4.21)  
`weedeater.math.yale.edu` (130.132.23.17)  
`freebie.engin.umich.edu` (35.2.68.23)

It should be fairly easy to dump the frame buffer to another type of file, or straight to a display device. See `AAMain.c`.

Have fun.

-- Kelvin Thompson, 18 August 1990  
`kelvin@cs.utexas.edu`



```
/*  FILENAME: AALines.c  [revised 17 AUG 90]

AUTHOR:  Kelvin Thompson

DESCRIPTION:  Code to render an anti-aliased line, from
             "Rendering Anti-Aliased Lines" in _Graphics_Gems_.

             This is almost exactly the code printed on pages 690-693
             of _Graphics_Gems_.  An overview of the code is on pages
             105-106.

LINK WITH:
    AALines.h -- Shared tables, symbols, etc.
    AAMain.c -- Calling code for this subroutine.
    AATables.c -- Initialize lookup tables.
*/

#include "AALines.h"

#define SWAPVARS(v1,v2) ( temp=v1, v1=v2, v2=temp )

#define FIXMUL(f1,f2) \
( \
    ((f1&0x0000ffff) * (f2&0x0000ffff)) >> 16) + \
    ((f1&0xffff0000)>>16) * (f2&0x0000ffff) + \
    ((f2&0xffff0000)>>16) * (f1&0x0000ffff) + \
    ((f1&0xffff0000)>>16) * (f2&0xffff0000) \
)

/* HARDWARE ASSUMPTIONS:
/*      * 32-bit, signed ints
/*      * 8-bit pixels, with initialized color table
/*      * pixels are memory mapped in a rectangular fashion */

/* FIXED-POINT DATA TYPE */
#ifndef FX_FRACBITS
    typedef int FX;
# define FX_FRACBITS 16 /* bits of fraction in FX format */
# define FX_0        0 /* zero in fixed-point format */
#endif

/* ASSUMED MACROS:
/*  SWAPVARS(v1,v2) -- swaps the contents of two variables
/*  PIXADDR(x,y) -- returns address of pixel at (x,y)
/*  COVERAGE(FXdist) -- lookup macro for pixel coverage
/*                      given perpendicular distance; takes a fixed-point
/*                      integer and returns an integer in the range [0,255]
/*  SQRTFUNC(FXval) -- lookup macro for sqrt(1/(1+FXval^2))
/*                   accepts and returns fixed-point numbers
/*  FIXMUL(FX1,FX2) -- multiplies two fixed-point numbers
/*                   and returns the product as a fixed-point number */

/* BLENDING FUNCTION:
/*  'cover' is coverage -- in the range [0,255]
/*  'back' is background color -- in the range [0,255] */
#define BLEND(cover,back) (((255-(cover))*(back))>>8)+(cover))

/* LINE DIRECTION bits and tables */
#define DIR_STEEP  1 /* set when abs(dy) > abs(dx) */
#define DIR_NEGY   2 /* set when dy < 0 */
```

```
/* pixel increment values
/*  -- assume PIXINC(dx,dy) is a macro such that:
/*  PIXADDR(x0,y0) + PIXINC(dx,dy) = PIXADDR(x0+dx,y0+dy)  */
static int adj_pixinc[4] =
    { PIXINC(1,0), PIXINC(0,1), PIXINC(1,0), PIXINC(0,-1) };
static int diag_pixinc[4] =
    { PIXINC(1,1), PIXINC(1,1), PIXINC(1,-1), PIXINC(1,-1) };
static int orth_pixinc[4] =
    { PIXINC(0,1), PIXINC(1,0), PIXINC(0,-1), PIXINC(1,0) };

/* Global 'Pmax' is initialized elsewhere.  It is the
   "maximum perpendicular distance" -- the sum of half the
   line width and the effective pixel radius -- in fixed format */
FX Pmax;

/***** FUNCTION ANTI_LINE *****/

void Anti_Line ( X1, Y1, X2, Y2 )
int X1, Y1, X2, Y2;
{
int      Bvar,      /* decision variable for Bresenham's */
    Bainc,      /* adjacent-increment for 'Bvar' */
    Bdinc;      /* diagonal-increment for 'Bvar' */
FX      Pmid,      /* perp distance at Bresenham's pixel */
    Pnow,      /* perp distance at current pixel (ortho loop) */
    Painc,      /* adjacent-increment for 'Pmid' */
    Pdinc,      /* diagonal-increment for 'Pmid' */
    Poinc;      /* orthogonal-increment for 'Pnow'--also equals 'k' */
char      *mid_addr,      /* pixel address for Bresenham's pixel */
    *now_addr;      /* pixel address for current pixel */
int      addr_ainc,      /* adjacent pixel address offset */
    addr_dinc,      /* diagonal pixel address offset */
    addr_oinc;      /* orthogonal pixel address offset */
int dx,dy,dir;      /* direction and deltas */
FX slope;      /* slope of line */
int temp;

/* rearrange ordering to force left-to-right */
if      ( X1 > X2 )
    { SWAPVARS(X1,X2);  SWAPVARS(Y1,Y2); }

/* init deltas */
dx = X2 - X1;  /* guaranteed non-negative */
dy = Y2 - Y1;

/* calculate direction (slope category) */
dir = 0;
if ( dy < 0 )    { dir |= DIR_NEGY;  dy = -dy; }
if ( dy > dx )  { dir |= DIR_STEEP; SWAPVARS(dx,dy); }

/* init address stuff */
mid_addr = PIXADDR(X1,Y1);
addr_ainc = adj_pixinc[dir];
addr_dinc = diag_pixinc[dir];
addr_oinc = orth_pixinc[dir];

/* perpendicular measures */
```

```
slope = (dy << FX_FRACBITS) / dx;
Poinc = SQRTFUNC( slope );
Painc = FIXMUL( slope, Poinc );
Pdinc = Painc - Poinc;
Pmid = FX_0;

/* init Bresenham's */
Bainc = dy << 1;
Bdinc = (dy-dx) << 1;
Bvar = Bainc - dx;

do
{
    /* do middle pixel */
    *mid_addr = BLEND( COVERAGE(abs(Pmid)), *mid_addr );

    /* go up orthogonally */
    for (
        Pnow = Poinc-Pmid, now_addr = mid_addr+addr_oinc;
        Pnow < Pmax;
        Pnow += Poinc, now_addr += addr_oinc
    )
        *now_addr = BLEND( COVERAGE(Pnow), *now_addr );

    /* go down orthogonally */
    for (
        Pnow = Poinc+Pmid, now_addr = mid_addr-addr_oinc;
        Pnow < Pmax;
        Pnow += Poinc, now_addr -= addr_oinc
    )
        *now_addr = BLEND( COVERAGE(Pnow), *now_addr );

    /* update Bresenham's */
    if ( Bvar < 0 )
    {
        Bvar += Bainc;
        mid_addr = (char *) ((int)mid_addr + addr_ainc);
        Pmid += Painc;
    }
    else
    {
        Bvar += Bdinc;
        mid_addr = (char *) ((int)mid_addr + addr_dinc);
        Pmid += Pdinc;
    }

    --dx;
} while ( dx >= 0 );
}
```

```
/*  FILENAME:  AALines.h  [revised 17 AUG 90]

AUTHOR:  Kelvin Thompson

DESCRIPTION:  Symbols and globals for the anti-aliased line
              renderer.

#INCLUDED IN:
    AAMain.c -- Calling routine for renderer.
    AATables.c -- Initialization routines for lookup tables.
    AALines.c -- Rendering code.
*/

/* frame buffer to hold the anti-aliased line */
#define xpix 60
#define ypix 60
extern char *fbuff;

/* macros to access the frame buffer */
#define PIXADDR(xx,yy) (fbuff+(yy)*xpix+(xx))
#define PIXINC(dx,dy) ((dy)*xpix+(dx))

/* fixed-point data types and macros */
typedef int FX;
typedef unsigned int UFX;
#define FX_FRACBITS 16 /* bits of fraction in FX format */
#define FX_0 0 /* zero in fixed-point format */
#define FLOAT_TO_FX(flt) ((FX)((flt)*(1<FX_FRACBITS)+0.5))

/* some important constants */
#define PI 3.1415926535897932384626433832795028841971693993751
#define SQRT_2 1.4142135623730950488016887242096980785696718753769

/* square-root function globals */
extern UFX *sqrtfunc;
extern int sqrtcells;
extern int sqrtshift;
#define SQRTFUNC(fxval) (sqrtfunc[ (fxval) >> sqrtshift ])

/* AA globals */
extern float line_r; /* line radius */
extern float pix_r; /* pixel radius */
extern FX *coverage;
extern int covercells;
extern int covershift;
#define COVERAGE(fxval) (coverage[ (fxval) >> covershift ])
```

```
/*  FILENAME:  AAMain.c  [revised 17 AUG 90]
```

```
    AUTHOR:  Kelvin Thompson
```

```
DESCRIPTION:  Calling routine for anti-aliased line renderer.
    This routine calls the line renderer to draw a single
    anti-aliased line into a small frame buffer.  The
    routine then dumps the frame buffer to a Utah RLE file
    'anti.rle'.
```

```
LINK WITH:
```

```
    utah.h -- Definitions for friendly Utah RLE front end.
    AALines.h -- Shared tables, symbols, etc. for renderer.
    AALines.c -- Rendering code.
    AATables.c -- Table initialization.
```

```
*/
```

```
#include <stdio.h>
#include <math.h>
#include "AALines.h"
#include "utah.h"
```

```
main ( argc, argv )
int argc;
char *argv[];
{
    int i;
    char *scanptr;
    int x1,y1,x2,y2;
```

```
/* initialize frame buffer and look-up tables */
Anti_Init();
```

```
/* set line endpoints */
x1 =  2;  y1 =  2;
x2 = 25;  y2 = 55;
```

```
/* render anti-aliased line to a frame buffer */
Anti_Line( x1,y1, x2,y2 );
```

```
/* The code below dumps the frame buffer to a Utah RLE file.
** It should be pretty easy to rewrite so that it dumps to
** any other kind of output file...or straight to a display
** device.  The frame buffer is an array of characters
** starting at 'fbuff' with size 'xpix' by 'ypix'.  */
```

```
{
    /* thanks to A.T. Campbell for the friendly front end */
    UTAH_FILE *picout;
    picout = utah_write_init( "anti.rle", xpix, ypix );
    if ( picout == NULL )
        { perror("anti.rle"); exit(1); }
    for ( i=0; i<ypix; i++ )
        {
            scanptr = &fbuff[i*xpix];
            utah_write_rgb( picout, scanptr, scanptr, scanptr );
        }
    utah_write_close(picout);
```

```
}  
}
```

```
/*  FILENAME: AATables.c  [revised 18 AUG 90]

DESCRIPTION:  Initialization of lookup tables and frame buffer
              for anti-aliased line rendering demo.

LINK WITH:
    AALines.h -- Shared variables and symbols for renderer.
    AAMain.c -- Calling routine for renderer.
    AALines.c -- Anti-aliased line rendering code.
*/

#include <math.h>
#include "AALines.h"

/* programs in this file */
extern void Anti_Init();
static void Sqrt_Init();

/* globals defined here */
char *fbuff;
UFX *sqrtfunc=0;
int sqrtcells=1024;
int sqrtshift;

/* AA sizes */
float line_r=1.0;      /* line radius */
float pix_r=SQRT_2;    /* pixel radius */
FX *coverage=0;
int covercells=128;
int covershift;

/* *****
 *
 *      Anti_Init
 *
 * *****
 */

DESCRIPTION:  Initialize everything for the anti-aliased line
              renderer in 'AALines.c':  allocate the frame buffer,
              set up lookup tables, etc.

              For hints about initializing the coverage table, see
              "Area of Intersection: Circle and a Thick Line" on pages
              40-42 of _Graphics_Gems_.

*/

#define FLOAT_TO_CELL(flt)  ((int) ((flt) * 255.0 + 0.5))
#define MAXVAL_CELL        255

void Anti_Init ( )
{
    int *thiscell;
    double maxdist,nowdist,incdist;
    int tablebits,radbits;
    int tablecells;
    static int tablesz=0;
    double fnear,ffar,fcover;
    double half,invR,invpiRsq,invpi,Rsq;
```

```
double sum_r;
double inv_log_2;
extern FX Pmax;

/* alloc & init frame buffer */
fbuff = (char *) malloc( xpix*ypix );
{
    register int i;
    for ( i=xpix*ypix-1; i>=0; --i )    fbuff[i] = 0;
}

/* init */
inv_log_2 = 1.0 / log( 2.0 );
sum_r = line_r + pix_r;
tablebits = (int) ( log((double)covercells) * inv_log_2 + 0.99 );
radbits = (int) ( log((double)sum_r) * inv_log_2 ) + 1;
covershift = FX_FRACBITS - (tablebits-radbits);

/* constants */
half = 0.5;
invR = 1.0 / pix_r;
invpi = 1.0 / PI;
invpiRsq = invpi * invR * invR;
Rsq = pix_r * pix_r;
#define FRACCOVER(d) (half - d*sqrt(Rsq-d*d)*invpiRsq - invpi*asin(d*invR))

/* allocate table */
Pmax = FLOAT_TO_FX(sum_r);
Pmax >>= covershift;
tablecells = Pmax + 2;
Pmax <<= covershift;
if ( coverage && tablecells > tablesizesize )
    { free( coverage ); coverage = 0;    tablesizesize = 0; }
if ( coverage == 0 )
    {
        coverage = (FX *) malloc( tablecells * sizeof(int) );
        tablesizesize = tablecells;
    }

/* init for fill loops */
nowdist = 0.0;
thiscell = coverage;
incdist = sum_r / (double)(tablecells-2);

/* fill fat portion */
if ( pix_r <= line_r )
    {
        maxdist = line_r - pix_r;
        for ( ;
            nowdist <= maxdist;
            nowdist += incdist, ++thiscell
        )
        {
            *thiscell = MAXVAL_CELL;
        }
    }

/* fill skinny portion */
else
    {
        /* loop till edge of line, or end of skinny, whichever comes first */
    }
```



```
maxdist = pix_r - line_r;
if ( maxdist > line_r )
    maxdist = line_r;
for ( ;
    nowdist < maxdist;
    nowdist += incdist, ++thiscell
    )
{
    fnear = line_r - nowdist;
    ffar = line_r + nowdist;
    fcover = 1.0 - FRACCOVER(fnear) - FRACCOVER(ffar);
    *thiscell = FLOAT_TO_CELL(fcover);
}

/* loop till end of skinny -- only run on super-skinny */
maxdist = pix_r - line_r;
for ( ;
    nowdist < maxdist;
    nowdist += incdist, ++thiscell
    )
{
    fnear = nowdist - line_r;
    ffar = nowdist + line_r;
    fcover = FRACCOVER(fnear) - FRACCOVER(ffar);
    *thiscell = FLOAT_TO_CELL(fcover);
}
}

/* loop till edge of line */
maxdist = line_r;
for ( ;
    nowdist < maxdist;
    nowdist += incdist, ++thiscell
    )
{
    fnear = line_r - nowdist;
    fcover = 1.0 - FRACCOVER(fnear);
    *thiscell = FLOAT_TO_CELL(fcover);
}

/* loop till max separation */
maxdist = line_r + pix_r;
for ( ;
    nowdist < maxdist;
    nowdist += incdist, ++thiscell
    )
{
    fnear = nowdist - line_r;
    fcover = FRACCOVER(fnear);
    *thiscell = FLOAT_TO_CELL(fcover);
}

/* finish off table */
*thiscell = FLOAT_TO_CELL(0.0);
coverage[tablecells-1] = FLOAT_TO_CELL(0.0);

Sqrt_Init();
}
```

```
/* *****  
 *  
 *      Sqrt_Init      *  
 *  
 *****  
  
DESCRIPTION:  Initialize the lookup table for the function  
              sqrt(1/(1+x^2)).  The table takes a shifted fixed-point  
              value as an index and returns a fixed-point value.  Input  
              values are in the range [0,1] inclusive.  The number of  
              cells in the table is a power of two plus one (the extra  
              cell provides an entry for an input value of exactly 1).  
  
GLOBALS:  
  sqrtcells -- Number of cells to use in the table (must  
                be set before calling this routine).  This number is  
                rounded up to the nearest power of two (the global  
                variable itself is unchanged).  
  sqrtshift -- Bits to shift a fixed-point (FX) number  
                to generate a table index.  
  sqrtfunc -- Lookup table for the function.  
  
*/  
  
static void Sqrt_Init ( )  
{  
  UFX *thiscell;  
  double nowval,incval;  
  int tablebits;  
  int tablecells;  
  double one;  
  
  /* init */  
  tablebits = (int) ( log((double)sqrtcells) / log(2.0) + 0.999 );  
  tablecells = (1 << tablebits) + 1; /* one more than requested */  
  sqrtshift = FX_FRACBITS - tablebits;  
  one = 1.0;  
  
  /* allocate table */  
  if ( sqrtfunc )  
    free( sqrtfunc );  
  sqrtfunc = (UFX *) malloc( tablecells * sizeof(int) );  
  
  /* init for fill loop */  
  incval = one / (double)(tablecells-1); /* a negative power of two */  
  
  for (   
    nowval = 0.0,      thiscell = sqrtfunc;  
    nowval < 1.0;  
    nowval += incval, ++thiscell  
  )  
  {  
    *thiscell = FLOAT_TO_FX( sqrt(one/(one+nowval*nowval)) );  
  }  
  
  sqrtfunc[tablecells-1] = FLOAT_TO_FX( sqrt(0.5) );  
}
```

```
/*  FILENAME: FastMatMul.c  [revised 18 AUG 90]
```

AUTHOR: Kelvin Thompson

DESCRIPTION: Routines to multiply different kinds of 4x4 matrices as fast as possible. Based on ideas on pages 454, 460-461, and 646 of `_Graphics_Gems_`.

These routines offer two advantages over the standard `V3MatMul` in the `_Graphics_Gems_` vector library `GGVecLib.c`. First, the routines are faster. Second, they can handle input matrices that are the same as the output matrix. The routines have the disadvantage of taking more code space (from unwound loops).

The routines are about as fast as you can get for general-purpose multiplication. If you have special knowledge about your system, you may be able to improve them a little more:

[1] If you know that your input and output matrices will never overlap: remove the tests near the beginning and end of each routine, and just `#define 'mptr' to 'result'`. (The standard library's `V3MatMul` makes this assumption.)

[2] If you know that your compiler supports more than three pointer-to-double registers in a subroutine: make `'result'` in each routine a register variable. You might also make the `'usetemp'` boolean a register.

[3] If you have limited stack space, or your system can access global memory faster than stack: make each routine's `'temp'` a static, or let all routines share a global `'temp'`. (This is useless if assumption [1] holds.)

```
*/
```

```
/* definitions from "GraphicsGems.h" */
```

```
typedef struct Matrix3Struct { /* 3-by-3 matrix */
    double element[3][3];
} Matrix3;
typedef struct Matrix4Struct { /* 4-by-4 matrix */
    double element[4][4];
} Matrix4;
```

```
/* routines in this file */
```

```
Matrix3 *V2MatMul(); /* general 3x3 matrix multiplier */
Matrix4 *V3MatMul(); /* general 4x4 matrix multiplier */
Matrix4 *V3AffMatMul(); /* affine 4x4 matrix multiplier */
Matrix4 *V3LinMatMul(); /* linear 4x4 matrix multiplier */
```

```
/* macro to access matrix element */
```

```
#define MVAL(mptr,row,col) ((mptr)->element[row][col])
```

```
/* *****
 *
 *          V2MatMul
 *
 *          *
```

\*\*\*\*\*

DESCRIPTION: Multiply two general 3x3 matrices. If one of the input matrices is the same as the output, write the result to a temporary matrix during multiplication, then copy to the output matrix.

ENTRY:

a -- pointer to left matrix  
b -- pointer to right matrix  
result -- result matrix

EXIT: returns 'result'

\*/

```
Matrix3 *V2MatMul ( a, b, result )
register Matrix3 *a,*b;
Matrix3 *result;
{
register Matrix3 *mptr;
int usetemp; /* boolean */
Matrix3 tempx;

/* decide where intermediate result goes */
usetemp = ( a == result || b == result );
if ( usetemp )
    mptr = & tempx;
else
    mptr = result;

MVAL(mptr,0,0) =
    MVAL(a,0,0) * MVAL(b,0,0)
+ MVAL(a,0,1) * MVAL(b,1,0)
+ MVAL(a,0,2) * MVAL(b,2,0);

MVAL(mptr,0,1) =
    MVAL(a,0,0) * MVAL(b,0,1)
+ MVAL(a,0,1) * MVAL(b,1,1)
+ MVAL(a,0,2) * MVAL(b,2,1);

MVAL(mptr,0,2) =
    MVAL(a,0,0) * MVAL(b,0,2)
+ MVAL(a,0,1) * MVAL(b,1,2)
+ MVAL(a,0,2) * MVAL(b,2,2);

MVAL(mptr,1,0) =
    MVAL(a,1,0) * MVAL(b,0,0)
+ MVAL(a,1,1) * MVAL(b,1,0)
+ MVAL(a,1,2) * MVAL(b,2,0);

MVAL(mptr,1,1) =
    MVAL(a,1,0) * MVAL(b,0,1)
+ MVAL(a,1,1) * MVAL(b,1,1)
+ MVAL(a,1,2) * MVAL(b,2,1);

MVAL(mptr,1,2) =
    MVAL(a,1,0) * MVAL(b,0,2)
+ MVAL(a,1,1) * MVAL(b,1,2)
+ MVAL(a,1,2) * MVAL(b,2,2);

MVAL(mptr,2,0) =
```

```
        MVAL(a,2,0) * MVAL(b,0,0)
+   MVAL(a,2,1) * MVAL(b,1,0)
+   MVAL(a,2,2) * MVAL(b,2,0);

MVAL(mptr,2,1) =
    MVAL(a,2,0) * MVAL(b,0,1)
+   MVAL(a,2,1) * MVAL(b,1,1)
+   MVAL(a,2,2) * MVAL(b,2,1);

MVAL(mptr,2,2) =
    MVAL(a,2,0) * MVAL(b,0,2)
+   MVAL(a,2,1) * MVAL(b,1,2)
+   MVAL(a,2,2) * MVAL(b,2,2);

/* copy temp matrix to result if needed */
if ( usetemp )
    *result = *mptr;

return result;
}

/* *****
*
*           V3MatMul
*
* *****
*
DESCRIPTION:  Multiply two general 4x4 matrices.  If one of
the input matrices is the same as the output, write the
result to a temporary matrix during multiplication, then
copy to the output matrix.

ENTRY:
    a -- pointer to left matrix
    b -- pointer to right matrix
    result -- result matrix

EXIT:  returns 'result'
*/

Matrix4 *V3MatMul ( a, b, result )
register Matrix4 *a,*b;
Matrix4 *result;
{
    register Matrix4 *mptr;
    int usetemp; /* boolean */
    Matrix4 tempx;

    /* decide where intermediate result goes */
    usetemp = ( a == result || b == result );
    if ( usetemp )
        mptr = & tempx;
    else
        mptr = result;

    MVAL(mptr,0,0) =
        MVAL(a,0,0) * MVAL(b,0,0)
    +   MVAL(a,0,1) * MVAL(b,1,0)
```

```
+ MVAL(a,0,2) * MVAL(b,2,0)
+ MVAL(a,0,3) * MVAL(b,3,0);
```

```
MVAL(mptr,0,1) =
    MVAL(a,0,0) * MVAL(b,0,1)
+   MVAL(a,0,1) * MVAL(b,1,1)
+   MVAL(a,0,2) * MVAL(b,2,1)
+   MVAL(a,0,3) * MVAL(b,3,1);
```

```
MVAL(mptr,0,2) =
    MVAL(a,0,0) * MVAL(b,0,2)
+   MVAL(a,0,1) * MVAL(b,1,2)
+   MVAL(a,0,2) * MVAL(b,2,2)
+   MVAL(a,0,3) * MVAL(b,3,2);
```

```
MVAL(mptr,0,3) =
    MVAL(a,0,0) * MVAL(b,0,3)
+   MVAL(a,0,1) * MVAL(b,1,3)
+   MVAL(a,0,2) * MVAL(b,2,3)
+   MVAL(a,0,3) * MVAL(b,3,3);
```

```
MVAL(mptr,1,0) =
    MVAL(a,1,0) * MVAL(b,0,0)
+   MVAL(a,1,1) * MVAL(b,1,0)
+   MVAL(a,1,2) * MVAL(b,2,0)
+   MVAL(a,1,3) * MVAL(b,3,0);
```

```
MVAL(mptr,1,1) =
    MVAL(a,1,0) * MVAL(b,0,1)
+   MVAL(a,1,1) * MVAL(b,1,1)
+   MVAL(a,1,2) * MVAL(b,2,1)
+   MVAL(a,1,3) * MVAL(b,3,1);
```

```
MVAL(mptr,1,2) =
    MVAL(a,1,0) * MVAL(b,0,2)
+   MVAL(a,1,1) * MVAL(b,1,2)
+   MVAL(a,1,2) * MVAL(b,2,2)
+   MVAL(a,1,3) * MVAL(b,3,2);
```

```
MVAL(mptr,1,3) =
    MVAL(a,1,0) * MVAL(b,0,3)
+   MVAL(a,1,1) * MVAL(b,1,3)
+   MVAL(a,1,2) * MVAL(b,2,3)
+   MVAL(a,1,3) * MVAL(b,3,3);
```

```
MVAL(mptr,2,0) =
    MVAL(a,2,0) * MVAL(b,0,0)
+   MVAL(a,2,1) * MVAL(b,1,0)
+   MVAL(a,2,2) * MVAL(b,2,0)
+   MVAL(a,2,3) * MVAL(b,3,0);
```

```
MVAL(mptr,2,1) =
    MVAL(a,2,0) * MVAL(b,0,1)
+   MVAL(a,2,1) * MVAL(b,1,1)
+   MVAL(a,2,2) * MVAL(b,2,1)
+   MVAL(a,2,3) * MVAL(b,3,1);
```

```
MVAL(mptr,2,2) =
    MVAL(a,2,0) * MVAL(b,0,2)
+   MVAL(a,2,1) * MVAL(b,1,2)
+   MVAL(a,2,2) * MVAL(b,2,2)
```

```
+   MVAL(a,2,3) * MVAL(b,3,2);

MVAL(mptr,2,3) =
    MVAL(a,2,0) * MVAL(b,0,3)
+   MVAL(a,2,1) * MVAL(b,1,3)
+   MVAL(a,2,2) * MVAL(b,2,3)
+   MVAL(a,2,3) * MVAL(b,3,3);

MVAL(mptr,3,0) =
    MVAL(a,3,0) * MVAL(b,0,0)
+   MVAL(a,3,1) * MVAL(b,1,0)
+   MVAL(a,3,2) * MVAL(b,2,0)
+   MVAL(a,3,3) * MVAL(b,3,0);

MVAL(mptr,3,1) =
    MVAL(a,3,0) * MVAL(b,0,1)
+   MVAL(a,3,1) * MVAL(b,1,1)
+   MVAL(a,3,2) * MVAL(b,2,1)
+   MVAL(a,3,3) * MVAL(b,3,1);

MVAL(mptr,3,2) =
    MVAL(a,3,0) * MVAL(b,0,2)
+   MVAL(a,3,1) * MVAL(b,1,2)
+   MVAL(a,3,2) * MVAL(b,2,2)
+   MVAL(a,3,3) * MVAL(b,3,2);

MVAL(mptr,3,3) =
    MVAL(a,3,0) * MVAL(b,0,3)
+   MVAL(a,3,1) * MVAL(b,1,3)
+   MVAL(a,3,2) * MVAL(b,2,3)
+   MVAL(a,3,3) * MVAL(b,3,3);

/* copy temp matrix to result if needed */
if ( usetemp )
    *result = *mptr;

return result;
}

/* *****
 *
 *          V3AffMatMul
 *
 * *****
 *
 * DESCRIPTION:  Multiply two affine 4x4 matrices.  The
 *               routine assumes the rightmost column of each input
 *               matrix is [0 0 0 1].  The output matrix will have the
 *               same property.
 *
 *               If one of the input matrices is the same as the output,
 *               write the result to a temporary matrix during multiplication,
 *               then copy to the output matrix.
 *
 * ENTRY:
 *   a -- pointer to left matrix
 *   b -- pointer to right matrix
```

```
result -- result matrix
```

```
EXIT:  returns 'result'
```

```
*/
```

```
Matrix4 *V3AffMatMul ( a, b, result )
```

```
register Matrix4 *a,*b;
```

```
Matrix4 *result;
```

```
{
```

```
register Matrix4 *mptr;
```

```
int usetemp;  /* boolean */
```

```
Matrix4 tempx;
```

```
/* decide where intermediate result goes */
```

```
usetemp = ( a == result  ||  b == result );
```

```
if ( usetemp )
```

```
    mptr = & tempx;
```

```
else
```

```
    mptr = result;
```

```
MVAL(mptr,0,0) =
```

```
    MVAL(a,0,0) * MVAL(b,0,0)
```

```
  + MVAL(a,0,1) * MVAL(b,1,0)
```

```
  + MVAL(a,0,2) * MVAL(b,2,0);
```

```
MVAL(mptr,0,1) =
```

```
    MVAL(a,0,0) * MVAL(b,0,1)
```

```
  + MVAL(a,0,1) * MVAL(b,1,1)
```

```
  + MVAL(a,0,2) * MVAL(b,2,1);
```

```
MVAL(mptr,0,2) =
```

```
    MVAL(a,0,0) * MVAL(b,0,2)
```

```
  + MVAL(a,0,1) * MVAL(b,1,2)
```

```
  + MVAL(a,0,2) * MVAL(b,2,2);
```

```
MVAL(mptr,0,3) = 0.0;
```

```
MVAL(mptr,1,0) =
```

```
    MVAL(a,1,0) * MVAL(b,0,0)
```

```
  + MVAL(a,1,1) * MVAL(b,1,0)
```

```
  + MVAL(a,1,2) * MVAL(b,2,0);
```

```
MVAL(mptr,1,1) =
```

```
    MVAL(a,1,0) * MVAL(b,0,1)
```

```
  + MVAL(a,1,1) * MVAL(b,1,1)
```

```
  + MVAL(a,1,2) * MVAL(b,2,1);
```

```
MVAL(mptr,1,2) =
```

```
    MVAL(a,1,0) * MVAL(b,0,2)
```

```
  + MVAL(a,1,1) * MVAL(b,1,2)
```

```
  + MVAL(a,1,2) * MVAL(b,2,2);
```

```
MVAL(mptr,1,3) = 0.0;
```

```
MVAL(mptr,2,0) =
```

```
    MVAL(a,2,0) * MVAL(b,0,0)
```

```
  + MVAL(a,2,1) * MVAL(b,1,0)
```

```
  + MVAL(a,2,2) * MVAL(b,2,0);
```

```
MVAL(mptr,2,1) =
```

```
    MVAL(a,2,0) * MVAL(b,0,1)
```



```
+ MVAL(a,2,1) * MVAL(b,1,1)
+ MVAL(a,2,2) * MVAL(b,2,1);

MVAL(mptr,2,2) =
    MVAL(a,2,0) * MVAL(b,0,2)
+ MVAL(a,2,1) * MVAL(b,1,2)
+ MVAL(a,2,2) * MVAL(b,2,2);

MVAL(mptr,2,3) = 0.0;

MVAL(mptr,3,0) =
    MVAL(a,3,0) * MVAL(b,0,0)
+ MVAL(a,3,1) * MVAL(b,1,0)
+ MVAL(a,3,2) * MVAL(b,2,0)
+ MVAL(b,3,0);

MVAL(mptr,3,1) =
    MVAL(a,3,0) * MVAL(b,0,1)
+ MVAL(a,3,1) * MVAL(b,1,1)
+ MVAL(a,3,2) * MVAL(b,2,1)
+ MVAL(b,3,1);

MVAL(mptr,3,2) =
    MVAL(a,3,0) * MVAL(b,0,2)
+ MVAL(a,3,1) * MVAL(b,1,2)
+ MVAL(a,3,2) * MVAL(b,2,2)
+ MVAL(b,3,2);

MVAL(mptr,3,3) = 1.0;

/* copy temp matrix to result if needed */
if ( usetemp )
    *result = *mptr;

return result;
}

/* *****
*
*          V3LinMatMul
*
* *****
DESCRIPTION:  Multiply two affine 4x4 matrices.  The
routine assumes the right column and bottom line
of each input matrix is [0 0 0 1].  The output matrix
will have the same property.  This is pretty much the
same thing as multiplying two 3x3 matrices.

If one of the input matrices is the same as the output,
write the result to a temporary matrix during multiplication,
then copy to the output matrix.

ENTRY:
a -- pointer to left matrix
b -- pointer to right matrix
result -- result matrix
```

```
EXIT:  returns 'result'
*/

Matrix4 *V3LinMatMul ( a, b, result )
register Matrix4 *a,*b;
Matrix4 *result;
{
register Matrix4 *mptr;
int usetemp;  /* boolean */
Matrix4 tempx;

/* decide where intermediate result goes */
usetemp = ( a == result  ||  b == result );
if ( usetemp )
    mptr = & tempx;
else
    mptr = result;

MVAL(mptr,0,0) =
    MVAL(a,0,0) * MVAL(b,0,0)
+   MVAL(a,0,1) * MVAL(b,1,0)
+   MVAL(a,0,2) * MVAL(b,2,0);

MVAL(mptr,0,1) =
    MVAL(a,0,0) * MVAL(b,0,1)
+   MVAL(a,0,1) * MVAL(b,1,1)
+   MVAL(a,0,2) * MVAL(b,2,1);

MVAL(mptr,0,2) =
    MVAL(a,0,0) * MVAL(b,0,2)
+   MVAL(a,0,1) * MVAL(b,1,2)
+   MVAL(a,0,2) * MVAL(b,2,2);

MVAL(mptr,0,3) = 0.0;

MVAL(mptr,1,0) =
    MVAL(a,1,0) * MVAL(b,0,0)
+   MVAL(a,1,1) * MVAL(b,1,0)
+   MVAL(a,1,2) * MVAL(b,2,0);

MVAL(mptr,1,1) =
    MVAL(a,1,0) * MVAL(b,0,1)
+   MVAL(a,1,1) * MVAL(b,1,1)
+   MVAL(a,1,2) * MVAL(b,2,1);

MVAL(mptr,1,2) =
    MVAL(a,1,0) * MVAL(b,0,2)
+   MVAL(a,1,1) * MVAL(b,1,2)
+   MVAL(a,1,2) * MVAL(b,2,2);

MVAL(mptr,1,3) = 0.0;

MVAL(mptr,2,0) =
    MVAL(a,2,0) * MVAL(b,0,0)
+   MVAL(a,2,1) * MVAL(b,1,0)
+   MVAL(a,2,2) * MVAL(b,2,0);

MVAL(mptr,2,1) =
    MVAL(a,2,0) * MVAL(b,0,1)
+   MVAL(a,2,1) * MVAL(b,1,1)
```

```
+   MVAL(a,2,2) * MVAL(b,2,1);

MVAL(mptr,2,2) =
    MVAL(a,2,0) * MVAL(b,0,2)
+   MVAL(a,2,1) * MVAL(b,1,2)
+   MVAL(a,2,2) * MVAL(b,2,2);

MVAL(mptr,2,3) = 0.0;

MVAL(mptr,3,0) = 0.0;
MVAL(mptr,3,1) = 0.0;
MVAL(mptr,3,2) = 0.0;
MVAL(mptr,3,3) = 1.0;

/* copy temp matrix to result if needed */
if ( usetemp )
    *result = *mptr;

return result;
}
```

```
/*  FILENAME:    LongConst.h  [revised 18 AUG 90]
```

```
    AUTHOR:    Kelvin Thompson
```

```
DESCRIPTION:  High-precision constants.  If this file is included
              in the same file as GraphicsGems.h, this file must come *after*
              GraphicsGems.h.  (It's okay to use this file without GraphicsGems.h.)
```

```
    The standard _Graphics_Gems_ include file has some constants
    that do not have full double-precision accuracy.  This file
    has the constants to a ridiculously high precision.  See pages
    434-435 of _Graphics_Gems_.  I got the constants from Mathematica.
```

```
    Also, this file has a constant and macro for finding the base-two
    logarithm of a number.
```

```
*/
```

```
/* prevent multiple inclusion */
```

```
#ifndef __LONGCONST_H__
```

```
#define __LONGCONST_H__
```

```
/* first get rid of stuff from GraphicsGems.h */
```

```
#undef PI
```

```
#undef PITIMES2
```

```
#undef PIOVER2
```

```
#undef E
```

```
#undef SQRT2
```

```
#undef SQRT3
```

```
#undef GOLDEN
```

```
#undef DTOR
```

```
#undef RTOD
```

```
/* re-define basic constants with high precision */
```

```
#define PI      3.141592653589793238462643383279502884197169399375105820975
```

```
#define E      2.718281828459045235360287471352662497757247093699959574967
```

```
#define SQRT2   1.414213562373095048801688724209698078569671875376948073177
```

```
#define SQRT3   1.732050807568877293527446341505872366942805253810380628056
```

```
#define GOLDEN  1.618033988749894848204586834365638117720309179805762862135
```

```
/* re-define derived constants */
```

```
#define PITIMES2 (2.0*PI)
```

```
#define PIOVER2  (0.5*PI)
```

```
#define DTOR     (PI/180.0)
```

```
#define RTOD     (180.0/PI)
```

```
/* macro and constant for base 2 logarithm */
```

```
#define LN2      0.693147180559945309417232121458176568075500134360255254121
```

```
#define LOG2(val) (log(val)*(1.0/LN_2))
```

```
#endif  /* __LONGCONST_H__ */
```

```
# FILENAME:  Makefile  [revised 18 AUG 90]
#
# AUTHOR:    Kelvin Thompson
#
# DESCRIPTION:  Makefile for anti-aliased line rendering demo.

# locations of Utah RLE information
UTAH_RLE_INCLUDE_DIR = /usr/contrib/include
UTAH_RLE_LIB_FILE = /usr/contrib/lib/librle.a

CFLAGS = -I$(UTAH_RLE_INCLUDE_DIR)

OBJS = AAMain.o AALines.o AATables.o utah.o

.o :      .c
          $(CC) -c $(CFLAGS) $(CPPFLAGS) $<

AALine : $(OBJS)
          cc $(CFLAGS) -o $@ $(OBJS) $(UTAH_RLE_LIB_FILE) -lm

clean:
          /bin/rm -f $(OBJS) AALine
```

```
/*
    file:          utah.c
    description:    interface to Utah RLE toolkit
    author:        A. T. Campbell
    date:          October 27, 1989
*/

#ifndef lint
static char      sccsid[] = "%W% %G%";          /* SCCS info */
#endif lint

#include <math.h>
#include <stdio.h>
#ifdef sequent
#include <strings.h>
#else
#include <string.h>
#endif
#include "utah.h"

/*****

/* return values */
extern void      free();
extern char      *malloc();

*****/

utah_read_close(ufp)
UTAH_FILE      *ufp;
{
    return(0);
}

/*****

UTAH_FILE *
utah_read_init(fname, ht, wd)

char      *fname;
int       *ht, *wd;
{
    FILE          *fp;
    UTAH_FILE      *ufp;

    /* open output stream */
    if (!strcmp(fname, ""))
        fp = stdin;
    else {
        if ((fp = fopen(fname, "r")) == NULL)
            return(NULL);
    }

    /* change the default rle_dflt_hdr struct to match what we need */
    ufp = (UTAH_FILE *) malloc(sizeof(UTAH_FILE));
    *ufp = rle_dflt_hdr;
    ufp->rle_file = fp;

    /* read the header in the input file */
    if (rle_get_setup(ufp) != 0)
        return(NULL);
}
*****/
```

```
/* get image size */
*wd = ufp->xmax - ufp->xmin + 1;
*ht = ufp->ymax - ufp->ymin + 1;

/* normal termination */
return(ufp);
}
```

/\*\*\*/

utah\_read\_pixels(ufp, pixels)

```
UTAH_FILE      *ufp;
unsigned char   pixels[][3];
{
    static unsigned n = 0;
    static unsigned char *r = NULL, *g = NULL, *b = NULL;
    int i, width;

    /* allocate storage */
    width = ufp->xmax + 1;
    if (width > n) {
        if (n > 0) {
            free((char *)r);
            free((char *)g);
            free((char *)b);
        }
        n = width;
        r = (unsigned char *) malloc(n * sizeof(unsigned char));
        g = (unsigned char *) malloc(n * sizeof(unsigned char));
        b = (unsigned char *) malloc(n * sizeof(unsigned char));
    }

    /* read this row */
    utah_read_rgb(ufp, r, g, b);

    /* convert to pixels */
    for (i = 0; i < width; i++) {
        pixels[i][0] = r[i];
        pixels[i][1] = g[i];
        pixels[i][2] = b[i];
    }

    return(0);
}
```

/\*\*\*/

utah\_read\_rgb(ufp, r, g, b)

```
UTAH_FILE      *ufp;
unsigned char   r[], g[], b[];
{
    rle_pixel    *rows[3];

    /* set color channels */
    rows[0] = r;
    rows[1] = g;
    rows[2] = b;
}
```

```
    /* read this row */
    rle_getrow(ufp, rows);
    return(0);
}

/*****

utah_write_close(ufp)

UTAH_FILE      *ufp;
{
    if (!ufp) return(-1);
    rle_puteof(ufp);
    return(0);
}

/*****

UTAH_FILE *
utah_write_init(fname, ht, wd)

char      *fname;
int       ht, wd;
{
    FILE          *fp;
    UTAH_FILE     *ufp;

    /* open output stream */
    if (!strcmp(fname, ""))
        fp = stdout;
    else {
        if ((fp = fopen(fname, "w")) == NULL)
            return(NULL);
    }

    /* change the default rle_dflt_hdr struct to match what we need */
    ufp = (UTAH_FILE *) malloc(sizeof(UTAH_FILE));
    *ufp = rle_dflt_hdr;
    ufp->rle_file = fp;
    ufp->xmax = wd - 1;
    ufp->ymax = ht - 1;
    ufp->alpha = 0; /* No coverage (alpha) */

    /* create the header in the output file */
    rle_put_setup(ufp);

    /* normal termination */
    return(ufp);
}

/*****

utah_write_pixels(ufp, pixels)

UTAH_FILE      *ufp;
unsigned char   pixels[][3];
{
    static unsigned n = 0;
    static unsigned char *r = NULL, *g = NULL, *b = NULL;
    int                i, width;
```



```
/* allocate storage */
width = ufp->xmax + 1;
if (width > n) {
    if (n > 0) {
        free((char *)r);
        free((char *)g);
        free((char *)b);
    }
    n = width;
    r = (unsigned char *) malloc(n * sizeof(unsigned char));
    g = (unsigned char *) malloc(n * sizeof(unsigned char));
    b = (unsigned char *) malloc(n * sizeof(unsigned char));
}

/* convert to color channels */
for (i = 0; i < width; i++) {
    r[i] = pixels[i][0];
    g[i] = pixels[i][1];
    b[i] = pixels[i][2];
}

/* write this row */
utah_write_rgb(ufp, r, g, b);
return(0);
}
```

/\*\*\*\*\*\*

utah\_write\_rgb(ufp, r, g, b)

```
UTAH_FILE      *ufp;
unsigned char   r[], g[], b[];
{
    rle_pixel    *rows[3];
    int          width;

    /* set color channels */
    rows[0] = r;
    rows[1] = g;
    rows[2] = b;

    /* write this row */
    width = ufp->xmax - ufp->xmin + 1;
    rle_putrow(rows, width, ufp);
    return(0);
}
```

/\*\*\*\*\*\*

```
/*
    file:          utah.h
    description:    interface to Utah RLE toolkit
    author:        A. T. Campbell
    date:          October 30, 1989
*/

#ifndef UTAH_H
#define UTAH_H

/*****

/* include files */
#include "rle.h"

/*****

/* type definitions */
typedef rle_hdr UTAH_FILE;

/*****

/* return values */
extern int          utah_read_close();
extern UTAH_FILE    *utah_read_init();
extern int          utah_read_pixels();
extern int          utah_read_rgb();
extern int          utah_write_close();
extern UTAH_FILE    *utah_write_init();
extern int          utah_write_pixels();
extern int          utah_write_rgb();

/*****

#endif UTAH_H
```

```
#include <math.h>
```

```
/*
*****
These definitions of vectors and matrices are used throughout this code.
The constant M_PI_2 is equal to PI/2, or 1.57079632679489661923.
*****
*/
```

```
typedef float    vec3[3];
typedef float    mat4[4][4];
```

```
/*
*****
Transforms an object space point into world coordinates using the provided
cumulative transformation matrix.
*****
*/
```

```
transform_point(world, local, ctm)
vec3 world;                /* returned world space point */
vec3 local;                /* provided local space point */
mat4 ctm;                  /* cumulative transformation matrix */
{
    world[0] = local[0] * ctm[0][0] + local[1] * ctm[1][0] +
                local[2] * ctm[2][0] + ctm[3][0];
    world[1] = local[0] * ctm[0][1] + local[1] * ctm[1][1] +
                local[2] * ctm[2][1] + ctm[3][1];
    world[2] = local[0] * ctm[0][2] + local[1] * ctm[1][2] +
                local[2] * ctm[2][2] + ctm[3][2];
}
```

```
/*
*****
The provided point is compared against the current minimum and maximum
values in X, Y, and Z.  If a new maximum or minimum is found, the
min and max variables are updated.
*****
*/
```

```
update_min_and_max(min, max, point)
vec3 min, max;            /* provided extent that is to be modified */
vec3 point;               /* world space point to be tested */
{
    int i;

    for (i=0; i<3; i++) {
        if (point[0] < min[0]) min[0] = point[0];
        if (point[0] > max[0]) max[0] = point[0];
        if (point[1] < min[1]) min[1] = point[1];
        if (point[1] > max[1]) max[1] = point[1];
        if (point[2] < min[2]) min[2] = point[2];
        if (point[2] > max[2]) max[2] = point[2];
    }
}
```

```
/*
*****
The bounding volume of a cube is found.  Each of the cube's eight vertices
is transformed into world space, and then compared to the current minimum

```

and maximum values, which are updated if a new extremum is found. The canonical cube is centered about the origin, with faces at X=-1, X=1, Y=-1, Y=1, Z=-1, and Z=1.

\*\*\*\*\*/

```
cube_volume(min, max, ctm)
vec3 min, max;          /* returned minimum and maximum of extent */
mat4 ctm;               /* cumulative transformation */
{
    int      i;
    vec3     point;
    static vec3 corners[8] = {
        -1, -1, -1,      1, -1, -1,      -1,  1, -1,      1,  1, -1,
        -1, -1,  1,      1, -1,  1,      -1,  1,  1,      1,  1,  1
    };

    min[0] = min[1] = min[2] = MAXFLOAT;
    max[0] = max[1] = max[2] = -MAXFLOAT;

    for (i=0; i<8; i++) {
        transform_point(point, corners[i], ctm);
        update_min_and_max(min, max, point);
    }
}
```

\*\*\*\*\*  
The bounding volume of a collection of polygons is found. The verts variable is a list of the vertices of the polygon collection, and the vertices need not be unique. The num variable specifies how many vertices are in the list. Each vertex is transformed into world space, and then compared to the current minimum and maximum values, which are updated if a new extremum is found.  
\*\*\*\*\*/

```
polygons_volume(min, max, ctm, verts, num)
vec3 min, max;          /* returned minimum and maximum of extent */
mat4 ctm;              /* cumulative transformation */
vec3 verts[];          /* list of vertices from polygons */
int  num;              /* number of vertices in list */
{
    int  i;
    vec3 point;

    min[0] = min[1] = min[2] = MAXFLOAT;
    max[0] = max[1] = max[2] = -MAXFLOAT;

    for (i=0; i<num; i++) {
        transform_point(point, verts[i], ctm);
        update_min_and_max(min, max, point);
    }
}
```

\*\*\*\*\*  
Performs a safe arctangent calculation for the two provided numbers. If the denominator is 0.0, atan2 is not called (it results in a floating exception on some machines). Instead, + or - PI/2 is returned.  
\*\*\*\*\*/

```
float
arctangent(a, b)
float a, b;          /* operands for tan'(a/b) */
{
    if (b != 0.0)
        return atan2(a, b);
    else if (a > 0.0)
        return M_PI_2;
    else
        return -M_PI_2;
}

/*****
The bounding volume of a cylinder is found.  The algorithm operates in
three passes, one for each dimension.  In each pass the extrema of the
bottom circle of the cylinder are found as values of the parameter t.
These values are then used to calculate the position of the two points
in the current dimension.  Finally, these points from the bottom circle
and corresponding points from the top circle are considered while computing
the minimum and maximum values of the current dimension.  The canonical
cylinder has a radius of 1.0 from the Y axis, and ranges from Z=-1 to Z=1.
*****/

cylinder_volume(min, max, ctm)
vec3 min, max;          /* returned minimum and maximum of extent */
mat4 ctm;               /* cumulative transformation matrix */
{
    int    i;
    float t1, t2, p1, p2, tmp;

    for (i=0; i<3; i++) {

        /* calculate first extremum.  second is +/- PI on other side of circle */
        t1 = arctangent(ctm[2][i], ctm[0][i]);
        if (t1 <= 0)
            t2 = t1 + M_PI;
        else
            t2 = t1 - M_PI;

        /* find and sort extrema locations in this dimension */
        p1 = ctm[0][i]*cos(t1) - ctm[1][i] + ctm[2][i]*sin(t1) + ctm[3][i];
        p2 = ctm[0][i]*cos(t2) - ctm[1][i] + ctm[2][i]*sin(t2) + ctm[3][i];
        if (p1 > p2) {
            tmp = p1;
            p1 = p2;
            p2 = tmp;
        }

        /* add the difference between bottom and top circles to an extremum */
        if (ctm[1][i] < 0) {
            min[i] = p1 + 2 * ctm[1][i];
            max[i] = p2;
        }
        else {
            min[i] = p1;
            max[i] = p2 + 2 * ctm[1][i];
        }
    }
}
```

```
}
```

```

/*****

```

The bounding volume of a cone is found. The algorithm checks the cone's bottom in three passes, one for each dimension. In each pass the extrema of the circle are found as values of the parameter  $t$ . These values are then used to calculate the position of the two points in the current dimension. Finally, these points are considered along with the transformed apex of the cone to compute the minimum and maximum extents. The canonical cone has a base of radius 1.0 at  $Z=-1$  and an apex at the point  $0,1,0$ .

```

*****/

```

```
cone_volume(min, max, ctm)
vec3 min, max;          /* returned minimum and maximum of extent */
mat4 ctm;               /* cumulative transformation matrix */
{
    int          i;
    float        t1, t2, tmp;
    vec3         point;
    static vec3  apex = { 0, 1, 0 };

    for (i=0; i<3; i++) {

        /* calculate first extremum.  second is +/- PI on other side of circle */
        t1 = arctangent(ctm[2][i], ctm[0][i]);
        if (t1 <= 0)
            t2 = t1 + M_PI;
        else
            t2 = t1 - M_PI;

        /* find and sort extrema locations in this dimension */
        min[i] = ctm[0][i]*cos(t1) - ctm[1][i] + ctm[2][i]*sin(t1) + ctm[3][i];
        max[i] = ctm[0][i]*cos(t2) - ctm[1][i] + ctm[2][i]*sin(t2) + ctm[3][i];
        if (min[i] > max[i]) {
            tmp = max[i];
            max[i] = min[i];
            min[i] = tmp;
        }
    }

    /* transform and check apex of cone */
    transform_point(point, apex, ctm);
    update_min_and_max(min, max, point);
}
```

```

/*****

```

The bounding volume of a conic is found. The algorithm checks the conic's bottom and top in three passes, one for each dimension. In each pass the extrema of the circles are found as values of the parameter  $t$ . These values are then used to calculate the position of the four points in the current dimension. The conic has a base of radius 1.0 at  $Z=-1$  and a top in the  $Y=1$  plane with a radius  $r$  around the  $Y$  axis.

```

*****/

```

```
conic_volume(min, max, ctm, r)
vec3 min, max;          /* returned minimum and maximum of extent */
mat4 ctm;               /* cumulative transformation matrix */
```

```

float r;                                /* radius of top of conic */
{
    int    i;
    float t1, t2, p1, p2;

    for (i=0; i<3; i++) {

        /* calculate first extremum.  second is +/- PI on other side of circle */
        t1 = arctangent(ctm[2][i], ctm[0][i]);
        if (t1 <= 0)
            t2 = t1 + M_PI;
        else
            t2 = t1 - M_PI;

        /* find and sort bottom extrema locations in this dimension */
        p1 = ctm[0][i]*cos(t1) - ctm[1][i] + ctm[2][i]*sin(t1) + ctm[3][i];
        p2 = ctm[0][i]*cos(t2) - ctm[1][i] + ctm[2][i]*sin(t2) + ctm[3][i];
        if (p1 < p2) {
            min[i] = p1;
            max[i] = p2;
        }
        else {
            min[i] = p2;
            max[i] = p1;
        }

        /* find, sort and compare top extrema locations in this dimension */
        p1 = r*ctm[0][i]*cos(t1) + ctm[1][i] + r*ctm[2][i]*sin(t1) + ctm[3][i];
        p2 = r*ctm[0][i]*cos(t2) + ctm[1][i] + r*ctm[2][i]*sin(t2) + ctm[3][i];
        if (p1 < p2) {
            if (p1 < min[i])
                min[i] = p1;
            if (p2 > max[i])
                max[i] = p2;
        }
        else {
            if (p2 < min[i])
                min[i] = p2;
            if (p1 > max[i])
                max[i] = p1;
        }
    }
}

```

\*\*\*\*\*  
 The bounding volume of a sphere is found. The algorithm performs three passes, one for each dimension. In each pass the extrema of the surface are found as values of the parameters u and v. These values are then used to calculate the position of the two points in the current dimension. Finally, these points are sorted to find the minimum and maximum extents. The canonical sphere has a radius of 1.0 and is centered at the origin.  
 \*\*\*\*\*/

```

sphere_volume(min, max, ctm)
vec3 min, max;                /* returned minimum and maximum of extent */
mat4 ctm;                     /* cumulative transformation matrix */
{
    int    i;
    float u1, u2, v1, v2, tmp, denominator;

```

```
for (i=0; i<3; i++) {

    /* calculate first extremum. */
    u1 = arctangent(ctm[2][i], ctm[0][i]);
    denominator = ctm[0][i]*cos(u1) + ctm[2][i]*sin(u1);
    v1 = arctangent(ctm[1][i], denominator);

    /* second extremum is +/- PI from u1, negative of v1 */
    if (u1 <= 0)
        u2 = u1 + M_PI;
    else
        u2 = u1 - M_PI;
    v2 = -v1;

    /* find and sort extrema locations in this dimension */
    min[i] =
        ctm[0][i] * cos(u1) * cos(v1) +
        ctm[1][i] * sin(v1) +
        ctm[2][i] * sin(u1) * cos(v1) +
        ctm[3][i];
    max[i] =
        ctm[0][i] * cos(u2) * cos(v2) +
        ctm[1][i] * sin(v2) +
        ctm[2][i] * sin(u2) * cos(v2) +
        ctm[3][i];
    if (min[i] > max[i]) {
        tmp = max[i];
        max[i] = min[i];
        min[i] = tmp;
    }
}
}
```

\*\*\*\*\*

The bounding volume of a torus is found. The algorithm performs three passes, one for each dimension. In each pass the extrema of the surface are found as values of the parameters  $u$  and  $v$ . These values are then used to calculate the position of the two points in the current dimension. Finally, these points are sorted to find the minimum and maximum extents. The torus is defined as the rotation of a circle that is perpendicular to the XZ plane. This circle has radius  $q$ , and its center lies in the XZ plane and is rotated about the Y axis in a circle of radius  $r$ . The value of  $q$  must be less than that of  $r$ .

\*\*\*\*\*/

```
torus_volume(min, max, ctm, r, q)
vec3 min, max;          /* returned minimum and maximum of extent */
mat4 ctm;               /* cumulative transformation matrix */
float r;                /* major radius of torus */
float q;                /* minor radius of torus */
{
    int i;
    float u1, u2, v1, v2, tmp, denominator;

    for (i=0; i<3; i++) {

        /* calculate first extremum. assure that  $-\pi/2 \leq v1 \leq \pi/2$  */
        u1 = arctangent(ctm[2][i], ctm[0][i]);
```



```
denominator = ctm[0][i]*cos(u1) + ctm[2][i]*sin(u1);
v1 = arctangent(ctm[1][i], denominator);

/* second extremum is +/- PI from u1, negative of v1 */
if (u1 <= 0)
    u2 = u1 + M_PI;
else
    u2 = u1 - M_PI;
v2 = -v1;

/* find and sort extrema locations in this dimension */
min[i] =
    ctm[0][i] * cos(u1) * (r + q * cos(v1)) +
    ctm[1][i] * sin(v1) * q +
    ctm[2][i] * sin(u1) * (r + q * cos(v1)) +
    ctm[3][i];
max[i] =
    ctm[0][i] * cos(u2) * (r + q * cos(v2)) +
    ctm[1][i] * sin(v2) * q +
    ctm[2][i] * sin(u2) * (r + q * cos(v2)) +
    ctm[3][i];
if (min[i] > max[i]) {
    tmp = max[i];
    max[i] = min[i];
    min[i] = tmp;
}
}
```

```
/*
Generating Random Points in Triangles
by Greg Turk
from "Graphics Gems", Academic Press, 1990
*/

#include <math.h>
#include "GraphicsGems.h"

/*****
Compute relative areas of sub-triangles that form a convex polygon.
There are vcount-2 sub-triangles, each defined by the first point
in the polygon and two other adjacent points.

This procedure should be called once before using
square_to_polygon().

Entry:
    vertices - list of the vertices of a convex polygon
    vcount   - number of vertices of polygon
Exit:
    areas - relative areas of sub-triangles of polygon
*****/

triangle_areas (vertices, vcount, areas)
    Point3 vertices[];
    int vcount;
    float areas[];
{
    int i;
    float area_sum = 0;
    Vector3 v1,v2,v3;

    /* compute relative areas of the sub-triangles of polygon */

    for (i = 0; i < vcount - 2; i++) {
        V3Sub(&vertices[i+1], &vertices[0], &v1);
        V3Sub(&vertices[i+2], &vertices[0], &v2);
        V3Cross(&v1, &v2, &v3);
        areas[i] = V3Length(&v3);
        area_sum += areas[i];
    }

    /* normalize areas so that the sum of all sub-triangles is one */

    for (i = 0; i < vcount - 2; i++)
        areas[i] /= area_sum;
}

/*****
Map a point from the square [0,1] x [0,1] into a convex polygon.
Uniform random points in the square will generate uniform random
points in the polygon.

The procedure triangle_areas() must be called once to compute
'areas', and then this procedure can be called repeatedly.

Entry:
    vertices - list of the vertices of a convex polygon
    vcount   - number of vertices of polygon
*****/
```

areas - relative areas of sub-triangles of polygon

s,t - position in the square [0,1] x [0,1]

Exit:

p - position in polygon

\*\*\*\*\*/

square\_to\_polygon (vertices, vcount, areas, s, t, p)

Point3 vertices[];

int vcount;

float areas[];

float s,t;

Point3 \*p;

{

int i;

float area\_sum = 0;

float a,b,c;

/\* use 's' to pick one sub-triangle, weighted by relative \*/

/\* area of triangles \*/

for (i = 0; i < vcount - 2; i++) {

area\_sum += areas[i];

if (area\_sum >= s)

break;

}

/\* map 's' into the interval [0,1] \*/

s = (s - area\_sum + areas[i]) / areas[i];

/\* map (s,t) to a point in that sub-triangle \*/

t = sqrt(t);

a = 1 - t;

b = (1 - s) \* t;

c = s \* t;

p->x = a \* vertices[0].x + b \* vertices[i+1].x + c \* vertices[i+2].x;

p->y = a \* vertices[0].y + b \* vertices[i+1].y + c \* vertices[i+2].y;

p->z = a \* vertices[0].z + b \* vertices[i+1].z + c \* vertices[i+2].z;

}

```
/*
Fixed-Point Trigonometry with CORDIC Iterations
by Ken Turkowski
from "Graphics Gems", Academic Press, 1990
```

```
provided by user:
    frmul(a,b)=(a*b)>>31, high part of 64-bit product
*/
```

```
#define COSCALE 0x22c2dd1c /* 0.271572 */
#define QUARTER ((int)(3.141592654 / 2.0 * (1 << 28)))
static long arctantab[32] = { /* MS 4 integral bits for radians */
    297197971, 210828714, 124459457, 65760959, 33381290, 16755422,
    8385879, 4193963, 2097109, 1048571, 524287, 262144, 131072,
    65536, 32768, 16384, 8192, 4096, 2048, 1024, 512, 256, 128, 64,
    32, 16, 8, 4, 2, 1, 0, 0,
};
```

```
CordicRotate(px, py, theta)
long *px, *py;
register long theta; /* Assume that abs(theta) <= pi */
{
    register int i;
    register long x = *px, y = *py, xtemp;
    register long *arctanptr = arctantab;

    /* The -1 may need to be pulled out and done as a left shift */
    for (i = -1; i <= 28; i++) {
        if (theta < 0) {
            xtemp = x + (y >> i);
            y      = y - (x >> i);
            x = xtemp;
            theta += *arctanptr++;
        } else {
            xtemp = x - (y >> i);
            y      = y + (x >> i);
            x = xtemp;
            theta -= *arctanptr++;
        }
    }

    *px = frmul(x, COSCALE); /* Compensate for CORDIC enlargement */
    *py = frmul(y, COSCALE); /* frmul(a,b)=(a*b)>>31, high part */
                               /* of 64-bit product */
}
```



```
CordicPolarize(argx, argy)
long *argx, *argy; /* We assume these are already in the */
                   /* right half plane */
{
    register long theta = 0, yi, i;
    register long x = *argx, y = *argy;
    register long *arctanptr = arctantab;
    for (i = -1; i <= 28; i++) {
        if (y < 0) { /* Rotate positive */
            yi = y + (x >> i);
            x  = x - (y >> i);
            y  = yi;
        }
    }
}
```

```
        theta -= *arctanptr++;
    } else {
        /* Rotate negative */
        yi = y - (x >> i);
        x  = x + (y >> i);
        y  = yi;
        theta += *arctanptr++;
    }
}

*argx = frmul(x, COSCALE);
*argy = theta;
}
```

# Index of

## /pubs/tog/GraphicsGems/gemsv/ch1-2/

Name	Last modified	Size	Description
 <a href="#">_Parent Directory</a>			
 <a href="#">_invsgt.c</a>	29-Jun-00 08:22	2K	

```
/* Compute the Inverse Square Root
 * of an IEEE Single Precision Floating-Point number.
 *
 * Written by Ken Turkowski.
 */

/* Specified parameters */
#define LOOKUP_BITS      6      /* Number of mantissa bits for lookup */
#define EXP_POS          23     /* Position of the exponent */
#define EXP_BIAS         127    /* Bias of exponent */
/* The mantissa is assumed to be just down from the exponent */

/* Type of result */
#ifndef DOUBLE_PRECISION
    typedef float FLOAT;
#else /* DOUBLE_PRECISION */
    typedef double FLOAT;
#endif /* DOUBLE_PRECISION */

/* Derived parameters */
#define LOOKUP_POS      (EXP_POS-LOOKUP_BITS) /* Position of mantissa lookup */
#define SEED_POS        (EXP_POS-8)          /* Position of mantissa seed */
#define TABLE_SIZE     (2 << LOOKUP_BITS)   /* Number of entries in table */
#define LOOKUP_MASK     (TABLE_SIZE - 1)      /* Mask for table input */
#define GET_EXP(a)      (((a) >> EXP_POS) & 0xFF) /* Extract exponent */
#define SET_EXP(a)      ((a) << EXP_POS)      /* Set exponent */
#define GET_EMANT(a)    (((a) >> LOOKUP_POS) & LOOKUP_MASK) /* Extended mantissa
                                                             * MSB's */
#define SET_MANTSEED(a) (((unsigned long)(a)) << SEED_POS) /* Set mantissa
                                                             * 8 MSB's */

#include <stdlib.h>
#include <math.h>

static unsigned char *iSqrt = NULL;

union _flint {
    unsigned long    i;
    float           f;
} _fi, _fo;

static void
MakeInverseSqrtLookupTable(void)
{
    register long f;
    register unsigned char *h;
    union _flint fi, fo;

    iSqrt = malloc(TABLE_SIZE);
    h = iSqrt;
    for (f = 0, h = iSqrt; f < TABLE_SIZE; f++) {
        fi.i = ((EXP_BIAS-1) << EXP_POS) | (f << LOOKUP_POS);
        fo.f = 1.0 / sqrt(fi.f);
        *h++ = ((fo.i + (1<<(SEED_POS-2))) >> SEED_POS) & 0xFF; /* rounding */
    }
    iSqrt[TABLE_SIZE / 2] = 0xFF; /* Special case for 1.0 */
}

/* The following returns the inverse square root */
FLOAT
InvSqrt(float x)
```

```
{
    register unsigned long a = ((union _flint*)(&x))->i;
    register float arg = x;
    union _flint seed;
    register FLOAT r;

    if (iSqrt == NULL) MakeInverseSqrtLookupTable();

    seed.i = SET_EXP(((3*EXP_BIAS-1) - GET_EXP(a)) >> 1)
        | SET_MANTSEED(iSqrt[GET_EMANT(a)]);

    /* Seed: accurate to LOOKUP_BITS */
    r = seed.f;

    /* First iteration: accurate to 2*LOOKUP_BITS */
    r = (3.0 - r * r * arg) * r * 0.5;



    /* Second iteration: accurate to 4*LOOKUP_BITS */
    r = (3.0 - r * r * arg) * r * 0.5;

#ifdef DOUBLE_PRECISION
    /* Third iteration: accurate to 8*LOOKUP_BITS */
    r = (3.0 - r * r * arg) * r * 0.5;
#endif /* DOUBLE_PRECISION */
    return(r);
}
```



# Index of

## /pubs/tog/GraphicsGems/gemsv/ch1-3/

Name	Last modified	Size	Description
 <a href="#">_Parent Directory</a>			
 <a href="#">_fixsqrt.c</a>	29-Jun-00 08:22	1K	

```
/* The definitions below yield 2 integer bits, 30 fractional bits */
#define FRACBITS 30      /* Must be even! */
#define ITERS      (15 + (FRACBITS >> 1))
typedef long TFract;

TFract
FFracSqrt(TFract x)
{
    register unsigned long root, remHi, remLo, testDiv, count;



    root = 0;           /* Clear root */
    remHi = 0;          /* Clear high part of partial remainder */
    remLo = x;          /* Get argument into low part of partial remainder */
    count = ITERS;      /* Load loop counter */

    do {
        remHi = (remHi << 2) | (remLo >> 30); remLo <= 2; /* get 2 bits of arg */
        root <= 1;    /* Get ready for the next bit in the root */
        testDiv = (root << 1) + 1; /* Test radical */
        if (remHi >= testDiv) {
            remHi -= testDiv;
            root += 1;
        }
    } while (count-- != 0);

    return(root);
}
```

# Index of

## /pubs/tog/GraphicsGems/gemsv/ch4-3/

Name	Last modified	Size	Description
 <a href="#">_Parent Directory</a>			
 <a href="#">_arcdivid.c</a>	29-Jun-00 08:23	1K	

```
/* arcdivide.c - recursive circular arc subdivision (FP version) */

#define DMAX 0.5    /* max chordal deviation = 1/2 pixel */

#include <math.h>
#include "../ch7-7/GG4D/GGems.h"

/* Function prototype for externally defined functions */
void DrawLine(Point2 p0, Point2 p1);

void
DrawArc(Point2 p0, Point2 p1, double d)
{
    if (fabs(d) <= DMAX) DrawLine(p0, p1);
    else {
        Vector2 v;
        Point2  pm, pb;
        double  dSub;

        v.x = p1.x - p0.x;      /* vector from p0 to p1 */
        v.y = p1.y - p0.y;

        pm.x = p0.x + 0.5 * v.x; /* midpoint */
        pm.y = p0.y + 0.5 * v.y;


        dSub = d / 4;
        V2Scale(&v, dSub);      /* subdivided vector */

        pb.x = pm.x - v.y;      /* bisection point */
        pb.y = pm.y + v.x;

        DrawArc(p0, pb, dSub);   /* first half arc */
        DrawArc(pb, p1, dSub);   /* second half arc */
    }
}
```

# Index of

## /pubs/tog/GraphicsGems/gemsv/ch7-4/

Name	Last modified	Size	Description
 <a href="#">_Parent Directory</a>			
 <a href="#">_Makefile</a>	29-Jun-00 08:24	1K	
 <a href="#">_basic.h</a>	29-Jun-00 08:24	1K	
 <a href="#">_dedge.cc</a>	29-Jun-00 08:24	1K	
 <a href="#">_dedge.h</a>	29-Jun-00 08:24	1K	
 <a href="#">_list.cc</a>	29-Jun-00 08:24	1K	
 <a href="#">_list.h</a>	29-Jun-00 08:24	1K	
 <a href="#">_main.cc</a>	29-Jun-00 08:24	1K	
 <a href="#">_plane.cc</a>	29-Jun-00 08:24	1K	
 <a href="#">_plane.h</a>	29-Jun-00 08:24	1K	
 <a href="#">_point.h</a>	29-Jun-00 08:24	1K	
 <a href="#">_polygon.cc</a>	29-Jun-00 08:24	6K	
 <a href="#">_polygon.h</a>	29-Jun-00 08:24	1K	
 <a href="#">_vector.h</a>	29-Jun-00 08:24	1K	

```
# -*- Makefile -*-
# based on gnu's make

.SUFFIXES: .cc .h

GPP      = g++
# Compiler options, such as -g or -O :
COPTS    = -O2
# Linker/loader options, such as -s :
LOPTS    = -s
LIBS     = -lm

MOBJS = main.o polygon.o dedge.o plane.o list.o
SRCS  = ${MOBJS:.o=.cc}

.cc.o:; $(GPP) -c $(COPTS)  $*.cc

ptest: $(MOBJS)
      $(GPP) $(COPTS) $(LOPTS) -o $@ $(MOBJS) $(LIBS)

# -----
clean:; $(RM) Makefile.bak *.o *~ a.out core errs TAGS

depend:
      /usr/local/X11/makedepend $(SRCS)
# DO NOT DELETE THIS LINE -- make depend depends on it.

main.o: polygon.h list.h plane.h vector.h point.h dedge.h
polygon.o: /usr/include/assert.h polygon.h list.h plane.h vector.h point.h
polygon.o: dedge.h
dedge.o: /usr/include/stdlib.h /usr/include/sys/feature_tests.h
dedge.o: /usr/include/assert.h dedge.h
plane.o: plane.h vector.h point.h /usr/include/assert.h
list.o: list.h
```

```
// -*- C++ -*-
// basic.h by George Vanecsek, Jr. June 1994

#ifndef _BASIC_H_
#define _BASIC_H_

#ifndef _IOSTREAM_H
#include <ostream.h>
#endif

#ifndef __MATH_H__
#include <math.h>
#endif

#ifndef _ASSERT_H_
#include <assert.h>
#endif

typedef int      Counter;      // 0,1,2,...
typedef int      Index;       // Array Index: 0,1,2,...
enum Boolean { FALSE, TRUE };
enum Where {
    NOWHERE,
    ABOVE    = 0x01,
    ON        = 0x02,
    ONABOVE   = 0x03,          // ON      | ABOVE
    BELOW     = 0x04,
    ABOVEBELOW = 0x05,        // ABOVE  | BELOW
    ONBELOW   = 0x06,        // ON     | BELOW
    CROSS     = 0x07          // ABOVE  | ON      | BELOW
};

template<class Type>
inline void swap( Type& a, Type& b )
{
    const Type c = a;
    a = b;
    b = c;
}

#ifndef NULL
#define NULL 0
#endif

#endif
```

```
// -*- C++ -*-
// dedge.cc by George Vanecsek Jr. June 1994
//

#include <stdlib.h>
#include <assert.h>
#include "dedge.h"

DEdge::DEdge( const Point& srcP, DEdge* const last )
: sP(srcP), nxt(NULL), sPW(NOWHERE)
{
    last->nxt = this;
    prv      = last;
}

void DEdge::closeCycle( DEdge* const first, DEdge* const last )
{
    first->prv = last;
    last->nxt  = first;
}

void DEdge::split( const Point& p )
{
    DEdge* const n = next();
    DEdge* const d = new DEdge( p, this );
    closeCycle( n, d );
}
```



```
// -*- C++ -*-
// dedge.h by George Vaneczek Jr, June 1994

#ifndef _DEdge_H_
#define _DEdge_H_

#ifndef _POINT_H_
#include "point.h"
#endif

class DEdge {                                // Directed Edge
public:
    DEdge ( const Point& srcP )
    : sP(srcP), nxt(this), prv(this), sPW(NOWHERE) { }
    DEdge*      next( ) const { return nxt; }
    DEdge*      prev( ) const { return prv; }
    const Point& srcPoint( ) const { return sP; }
    const Point& dstPoint( ) const { return nxt->sP; }
    Where&      srcWhere( ) { return sPW; }
    Where&      dstWhere( ) { return nxt->sPW; }
    Where        where( ) const { return Where( sPW | nxt->sPW ); }
    double&     distFromRefP( ) { return t; }

private:
    DEdge ( const Point& srcP, DEdge* const last );
    static void closeCycle( DEdge* const first, DEdge* const last );
    void        split      ( const Point& );

    DEdge*      nxt;                // Next DEdge on cycle
    DEdge*      prv;                // Previous DEdge on cycle
    const Point sP;                 // Source Point
    Where        sPW;               // Where is Source Point?
    double        t;                // Related to sP. Used in complexCut(...)

friend class Polygon;
};

typedef DEdge* DEdgePtr;

#endif
```

```
// -*- C++ -*-
//
// list.cc
// by George Vanecek, 1994
// This List template is used frequently, and consequently, we
// want to improve its performance.  Since memory allocation and
// deallocation is very time consuming, we keep a stack of free ListNodes,
// and reuse them.

#include "list.h"

ListNode* ListNode::freeList = NULL;

// Allocate ListNode first from our free list, then from system heap.
void* ListNode::operator new( size_t s )
{
    if( freeList ) {
        ListNode* const n = freeList;
        freeList = freeList->nxt;
        return n;
    }
    return ::operator new(s);
}

// Delete node by placing it on our free list.
void ListNode::operator delete( void* p )
{
    ListNode* const n = (ListNode*)p;
    n->nxt = freeList;
    freeList = n;
}
```

```
// -*- C++ -*-
// list.h by George Vaneczek Jr. June 1994

#ifndef _LIST_H_
#define _LIST_H_

#ifndef __STDLIB_H__
#include <stdlib.h>
#endif

#ifndef _BASIC_H_
#include "basic.h"
#endif

class ListNode {
public:
    ListNode( void* const t, ListNode* const n ) : val(t), nxt(n) {}
    void*      value ( ) const { return val; }
    ListNode*  next  ( ) { return nxt; }
    void* operator new    ( size_t );
    void operator delete( void* );

private:
    void* const val;
    ListNode*  nxt;
    static ListNode* freeList;
};

template <class T>
class List {
public:
    List( ) : anchor(NULL), nNodes(0) { }
    Boolean empty      ( ) const { return Boolean( anchor == NULL ); }
    T*      first      ( ) const { return (T*)(anchor->value()); }
    ListNode* head      ( ) const { return anchor; }
    Counter size        ( ) const { return nNodes; }
    void operator <<( T* const ); // insert (i.e. push)
    Boolean operator >>( T*& );   // remove (i.e. pop)

private:
    ListNode* anchor;           // Start of List
    Counter  nNodes;           // Number of Nodes on list
};

template <class T>
void List<T>::operator <<( T* const t )
{
    anchor = new ListNode( t, anchor );
    ++nNodes;
}

template <class T>
Boolean List<T>::operator >>( T*& t )
{
    if( empty() ) {
        t = NULL;
        return FALSE;
    }
    t = first();
    ListNode* f = anchor;
    anchor = f->next();
}
```

```
    delete f;
    --nNodes;
    return TRUE;
}

#define foreachItemOnList(p) \
    for( ListNode* pp = p.head(); pp != NULL; pp = pp->next() )

#define getItem(Type) \
    (Type*)(pp->value())

#endif
```

```
// -*- C++ -*-
// Gems V: Spatial Partitioning of a Polygon by a Plane
// by George Vaneczek Jr, Sept. 1994

#include "polygon.h"

static void printPolys( const char* const label, const List<Polygon>& pL )
{
    if( pL.size() ) {
        cout << "-----" << endl
             << pL.size() << " polygon(s) " << label << endl;
        foreachItemOnList( pL ) {
            cout << " Polygon:" << endl;
            const Polygon* const g = getItem(Polygon);
            foreachDEdgeOfPoly(d1,g) {
                const Point& p = d1->srcPoint();
                cout << " " << p.x() << ' ' << p.y() << ' ' << p.z() << endl;
            }
        }
    }
}

int main( )
{
    // Sample polygon shown in Figure 1(a) of the Gems V article.
    const Point pts[] = {
        Point( 0,0,0), Point( 2,0,0), Point( 3,3,0),
        Point( 4,0,0), Point( 9,0,0), Point( 9,3,0),
        Point(10,3,0), Point(10,0,0), Point(13,0,0),
        Point(13,3,0), Point(14,3,0), Point(14,6,0),
        Point( 6,6,0), Point( 6,2,0), Point( 7,2,0),
        Point( 7,5,0), Point(12,5,0), Point(12,1,0),
        Point(11,1,0), Point(11,4,0), Point( 8,4,0),
        Point( 8,1,0), Point( 5,1,0), Point( 5,3,0),
        Point( 4,3,0), Point( 4,6,0), Point( 3,6,0),
        Point( 2,3,0), Point( 1,6,0), Point( 0,6,0)
    };
    Polygon* g = new Polygon( 30, pts );
    cout << "Before:" << endl;
    foreachDEdgeOfPoly(d1,g)
        cout << d1->srcPoint() << endl;
    List<Polygon> above;
    List<Polygon> on;
    List<Polygon> below;
    split( g, Plane(Vector(0.0,1.0,0.0),-3.0), above, on, below);
    printPolys( "Above", above);
    printPolys( "On",    on);
    printPolys( "Below", below);
}
```

```
// -*- C++ -*-
// plane.cc by George Vaneczek Jr., June 1994
//

#include "plane.h"

// Computes the plane equation using Newell's averaging algorithm.
Plane::Plane( const Counter nPoints, const Point points[] )
: n(0.0,0.0,0.0), d(0.0), eps(0.0)
{
    assert( nPoints > 2 );
    Vector avrPnt = Point(0,0,0);
    for( Index i = 0; i < nPoints; ++i ) {
        avrPnt += points[i-1];
        n      += Vector(points[i-1]) ^ Vector(points[i]);
        updateEpsilon( points[i] );
    }
    avrPnt += points[nPoints-1];
    n      += Vector(points[nPoints-1]) ^ Vector(points[0]);
    n.normalize();
    d = normal() * ((-1.0 / nPoints) * avrPnt );
}

// Compute the intersection point with the transversal line (p,q).
Point Plane::onPoint( const Point &p, const Point &q ) const
{
    const Vector v(q - p);
    const double c = normal() * v;
    assert( c != 0.0 );
    const double t = -sDistance(p) / c;
    return p + t * v;
}

void Plane::updateEpsilon ( const Point& p )
{
    double d = sDistance(p);
    if( d < 0.0 )
        d = -d;
    if( d > eps )
        eps = d;
}
```

```
// -*- C++ -*-
//
// plane.h by George Vaneczek Jr. June 1994

#ifndef _PLANE_H_
#define _PLANE_H_

#ifndef _VECTOR_H_
#include "vector.h"
#endif

// Provide a minimal 3D plane support sufficine for the Gem.
//
// Any point p that is known topologically to lie on a plane pN+d~0
// is included in the plane by enlarging the epsilon, so the
// equation |pN+d| <= eps holds. The point/plane relationship must
// be established by the application code.

class Plane
{
public:
    Plane ( const Counter nPoints, const Point[] );
    Plane ( const Vector& v, double x ) : n(v), d(x), eps(0.0) { }
    Plane ( const Plane& pl ) : n(pl.n), d(pl.d), eps(pl.eps) { }

    const Vector& normal( ) const { return n; }

    // The point is topologically known to lie in the plane, so check
    // and update epsilon accordingly:
    void updateEpsilon( const Point& );

    // Signed distance from the point to the plane.
    double sDistance( const Point& p ) const { return Vector(p) * n + d; }

    // Intersection point between this Plane and transversal line.
    Point onPoint( const Point&, const Point& ) const;

    // Which side of the plane is this point on?
    Where whichSide( const Point& ) const;

private:
    Vector n; // unit normal vector
    double d; // shortest distance from origin
    double eps; // point/plane distance epsilon
};

inline Where Plane::whichSide( const Point& p ) const
{ const double d = sDistance( p );
  return d < -eps ? BELOW : (d > eps ? ABOVE : ON);
}

#endif
```

```
// -*- C++ -*-
// point.h by George Vaneczek Jr. June 1994

#ifndef _POINT_H_
#define _POINT_H_

#ifndef _BASIC_H_
#include "basic.h"
#endif

class Point
{
public:
    Point ( const double x, const double y, const double z)
        : _x(x), _y(y), _z(z) { }

    double x() const { return _x; }
    double y() const { return _y; }
    double z() const { return _z; }
    Point& operator +=( const Point& p);

protected:
    double _x, _y, _z;           // Point Coordinates
    double& x() { return _x; }
    double& y() { return _y; }
    double& z() { return _z; }
};

inline Point& Point::operator +=( const Point& p )
{ x() += p.x(), y() += p.y(), z() += p.z(); return *this; }

inline ostream& operator << ( ostream& outs, const Point& p )
{
    outs << '(' << p.x() << ' ' << p.y() << ' ' << p.z() << ')';
    return outs;
}

#endif
```



```
// -*- C++ -*-
// polygon.cc by George Vanecsek Jr. June 1994
//

#include <assert.h>
#include "polygon.h"

Polygon::Polygon( const Counter nPoints, const Point pts[] )
: supportPlane( nPoints, pts )
{
    DEdge* last = ( anchor = new DEdge( pts[0] ) );
    for( Index i = 1; i < nPoints;++i )
        last = new DEdge( pts[i], last );
    DEdge::closeCycle( anchor, last );
    nDEdges= nPoints;
}

// Split Directed-Edge d of this Polygon by cut Plane:
void Polygon::split( const Plane& cut, DEdge* const d )
{
    assert( cut.sDistance(d->srcPoint()) *
            cut.sDistance(d->dstPoint()) < 0.0 ); // same as sgn(a)!=sgn(b)
    const Point onP = cut.onPoint( d->srcPoint(), d->dstPoint() );
    d->split( onP );
    ++nDEdges;
}

// Assign each srcPoint of every DEdge ABOVE, ON, or BELOW depending
// where they are in relation to the cut plane, and split any DEdges
// that cross the cut plane.
Where Polygon::classifyPoints( const Plane& cut,
                               Counter&      nOnDEdges,
                               DEdge*        onDEdges[] )
{
    first()->srcWhere() = cut.whichSide( first()->srcPoint() );
    Where polyW = first()->srcWhere();
    forEachDEdge( d ) {
        d->dstWhere() = cut.whichSide( d->dstPoint() );
        polyW = Where( polyW | d->dstWhere() );
        if( d->where() == ABOVEBELOW ) {
            split( cut, d );
            onDEdges[nOnDEdges++] = ( d = d->next() );
            d->srcWhere() = ON;
        } else if( d->srcWhere() == ON )
            onDEdges[nOnDEdges++] = d;
    }
    return polyW;
}

Polygon::Polygon( DEdge* const start, const Plane& sPl )
: supportPlane( sPl )
{
    anchor = start;
    nDEdges = 0;
    forEachDEdge( d ) {
        d->srcWhere() = NOWHERE;
        ++nDEdges;
    }
}
```

```
void Polygon::maximize( DEdge* const d )
{
    DEdge* dN = d->next();
    if( d->srcWhere() == ON && dN->srcWhere() == ON && dN->dstWhere() == ON ) {
        // Merge two adjacent and colinear DEdges:
        DEdge::closeCycle( dN->next(), d );
        anchor = d;
        delete dN;
        --nDEdges;
    }
}

// Insert two new Directed Edges, between srcD->srcPoint() and
// dstD->srcPoint(); one for the above loop and one for the below loop.
void Polygon::addBridge( DEdge* const leftBelow,
                        DEdge* const rghtAbove )
{
    assert( leftBelow->srcWhere() == ON );
    assert( rghtAbove->srcWhere() == ON );
    assert( leftBelow != rghtAbove );
    DEdge* const leftAbove = leftBelow->prev();
    DEdge* const rghtBelow = rghtAbove->prev();
    DEdge* const onAbove = new DEdge( leftBelow->srcPoint(), leftAbove );
    DEdge* const onBelow = new DEdge( rghtAbove->srcPoint(), rghtBelow );
    DEdge::closeCycle( rghtAbove, onAbove );
    DEdge::closeCycle( leftBelow, onBelow );
    onAbove->srcWhere() = onBelow->srcWhere() = ON;
    maximize( onAbove->prev() );
    maximize( onBelow );
}

// Sort directed edges that have srcPoints ON the cut plane
// left to right (in direction of cutDir) by their source points.
void Polygon::sortDEdges( const Counter nOnDs, DEdge* const onDs[],
                        const Vector& cutDir )
{
    assert( nOnDs >= 2 );
    const Point& refP = onDs[0]->srcPoint();
    for( Index i = 0; i < nOnDs; ++i )
        onDs[i]->distFromRefP() = cutDir * (onDs[i]->srcPoint() - refP );
    for( i = nOnDs-1; i > 0; --i )
        for( Index j = 0, k = 1; k <= i; j = k++ )
            if( onDs[j]->distFromRefP() > onDs[k]->distFromRefP() ||
                onDs[j]->distFromRefP() == onDs[k]->distFromRefP() &&
                onDs[j]->dstWhere() == ABOVE )
                swap( onDs[j], onDs[k] );
}

static DEdge* useSrc = NULL;

// Get the next directed edge that starts a cut.
// This assumes all vertices on the cut Plane have manifold sectors.
static DEdge* getSrcD( DEdge* const onDs[],
                    Counter& start, const Counter nOnDs )
{
    {
        if( useSrc ) {
            DEdge* const gotIt = useSrc;
            useSrc = NULL;
            return gotIt;
        }
    }
}
```

```
while( start < nOnDs ) {
    const Where prevW = onDs[start]->prev()->srcWhere();
    const Where nextW = onDs[start]->dstWhere();
    if( prevW == ABOVE && nextW == BELOW ||
        prevW == ABOVE && nextW == ON &&
            onDs[start]->next()->distFromRefP() < onDs[start]->distFromRefP() ||
        prevW == ON && nextW == BELOW &&
            onDs[start]->prev()->distFromRefP() < onDs[start]->distFromRefP() )
        return onDs[start++];
    ++start;
}
return NULL;
}

// Get the next directed edge that ends a cut.
static DEdge* getDstD( DEdge* const onDs[],
                      Counter& start, const Counter nOnDs )
{
    while( start < nOnDs ) {
        const Where prevW = onDs[start]->prev()->srcWhere();
        const Where nextW = onDs[start]->dstWhere();
        if( prevW == BELOW && nextW == ABOVE ||
            prevW == BELOW && nextW == BELOW ||
            prevW == ABOVE && nextW == ABOVE ||
            prevW == BELOW && nextW == ON &&
                onDs[start]->distFromRefP() < onDs[start]->next()->distFromRefP() ||
            prevW == ON && nextW == ABOVE &&
                onDs[start]->distFromRefP() < onDs[start]->prev()->distFromRefP() )
            return onDs[start++];
        ++start;
    }
    return NULL;
}

void Polygon::complexCut( const Plane& cut,
                        const Counter nOnDs, DEdge* const onDs[],
                        List<Polygon>& above, List<Polygon>& below)
{
    sortDEdges( nOnDs, onDs, cut.normal() ^ plane().normal() );
    Index startOnD = 0;
    DEdge* srcD = NULL;
    while( srcD = getSrcD( onDs, startOnD, nOnDs ) ) {
        DEdge* const dstD = getDstD( onDs, startOnD, nOnDs );
        assert( dstD != NULL );
        addBridge( srcD, dstD );
        if( srcD->prev()->prev()->srcWhere() == ABOVE )
            useSrc = srcD->prev();
        else if( dstD->dstWhere() == BELOW )
            useSrc = dstD;
    }
    // Collect new Polygons:
    for( Index i = 0; i < nOnDs; ++i )
        if( onDs[i]->srcWhere() == ON )
            if( onDs[i]->dstWhere() == ABOVE )
                above << new Polygon( onDs[i], plane() );
            else if( onDs[i]->dstWhere() == BELOW )
                below << new Polygon( onDs[i], plane() );
    }

void split( Polygon*& g, const Plane& cut,
          List<Polygon>& above,
```

```
        List<Polygon>& on,
        List<Polygon>& below )
{
    DEdge*  onDEdges[g.nPoints()];
    Counter nOnDEdges = 0;
    switch( g->classifyPoints( cut, nOnDEdges, onDEdges ) ) {
    case ONABOVE:
    case ABOVE:
        above << g;
        break;
    case ON:
        on << g;
        break;
    case ONBELOW:
    case BELOW:
        on << g;
        break;
    default: /* case CROSS */
        assert( nOnDEdges >= 2 );
        g->complexCut( cut, nOnDEdges, onDEdges, above, below );
        g->anchor    = NULL;
        g->nDEdges    = 0;
        delete g;
    }
    g = NULL;
}
```

```
// -*- C++ -*-
// polygon.h by George Vaneczek Jr, June 1994

#ifndef _POLYGON_H_
#define _POLYGON_H_

#ifndef _LIST_H_
#include "list.h"
#endif

#ifndef _PLANE_H_
#include "plane.h"
#endif

#ifndef _DEDGE_H_
#include "dedge.h"
#endif

class Polygon {
public:
    Polygon( const Counter nPoints, const Point [] );
    Counter    nPoints( ) const { return nDEdges; }
    const Plane& plane( ) const { return supportPlane; }
    DEdge*      first( ) const { return anchor; }

private:
    Polygon( DEdge* const, const Plane& );
    Where    classifyPoints( const Plane&, Counter&, DEdge*[] );
    void      addBridge      ( DEdge* const, DEdge* const );
    void      complexCut     ( const Plane&, const Counter, DEdge* const [],
                               List<Polygon>&, List<Polygon>& );
    static void sortDEdges( const Counter, DEdge* const [], const Vector& );
    void      maximize      ( DEdge* const );
    void      split         ( const Plane&, DEdge* const );

    const Plane supportPlane;
    Counter    nDEdges;           // Number of DEdges in loop...
    DEdge*      anchor;           // Edge Loop

friend void split( Polygon*&, const Plane& cut,
                  List<Polygon>& above,
                  List<Polygon>& on,
                  List<Polygon>& below );
};

// Juxtapose two strings to form a single identifier:
#define name2(a,b) a##b

// Iterate over all Directed Edges within a Polygon member function:
#define foreachDEdge(d) \
    for( DEdgePtr d = first(), name2(last,d) = NULL;\
        d != first() || name2(last,d) == NULL;\
        name2(last,d) = d, d = d->next() )

// Iterate over all Directed Edges of a Polygon *g:
#define foreachDEdgeOfPoly(d,g) \
    for( DEdgePtr d = g->first(), name2(last,d) = NULL;\
        d != g->first() || name2(last,d) == NULL;\
        name2(last,d) = d, d = d->next() )

#endif
```

```
// -*- C++ -*-
// vector.h by George Vanecsek Jr. June 1994

#ifndef _VECTOR_H_
#define _VECTOR_H_

#ifndef _POINT_H_
#include "point.h"
#endif

class Vector : public Point {
public:
    Vector( const Point& p ) : Point(p) {}
    Vector( double a, double b, double c ) : Point(a, b, c) {}
    inline double operator * ( const Vector& v ) const;
    inline Vector operator ^ ( const Vector& v ) const;
    void          normalize ( );
};

inline Vector operator + ( const Point& p, const Point& q )
{ return Point(p.x()+q.x(), p.y()+q.y(), p.z()+q.z()); }

inline Vector operator - ( const Point& p, const Point& q )
{ return Point(p.x()-q.x(), p.y()-q.y(), p.z()-q.z()); }

inline Vector operator * ( const double s, const Vector& v )
{ return Vector( s*v.x(), s*v.y(), s*v.z() ); }

// Vector Dot Product.
inline double Vector::operator * ( const Vector& v ) const
{ return x() * v.x() + y() * v.y() + z() * v.z(); }

// Vector Cross product.
inline Vector Vector::operator ^ ( const Vector& v ) const
{ return Vector(y() * v.z() - z() * v.y(),
               z() * v.x() - x() * v.z(),
               x() * v.y() - y() * v.x());
}

inline void Vector::normalize()
{
    const double n = *this * *this;
    assert( n != 0.0 );
    x() /= n;
    y() /= n;
    z() /= n;
}

#endif
```

```
#include <gl/gl.h>      /* SGI Graphics Library assumed */

#define STEP_SIZE 4      /* # of pixels in each step */

long coord[2];           /* X,Y for graphics calls */

void step(long angle)
{
    while (angle > 270) angle -= 360;    /* Fold ANGLE to be 0, 90, 180, 270 */
    while (angle < 0) angle += 360;
    if (angle == 0) coord[0] += STEP_SIZE; /* +X */
    else if (angle == 90) coord[1] += STEP_SIZE; /* +Y */
    else if (angle == 180) coord[0] -= STEP_SIZE; /* -X */
    else if (angle == 270) coord[1] -= STEP_SIZE; /* -Y */
    v2i(coord);                /* Draw (poly)line to new X,Y = coord */
}

/* Recursive Hilbert-curve generation algorithm */
/* ORIENT is either +1 or -1...it swaps left turns and right turns */
/* ANGLE is some multiple of 90 degrees...positive or negative */
/* LEVEL is the recursion level */
/* 2^LEVEL by 2^LEVEL points will be visited in total */

void hilbert (orient,angle,level)
long orient,*angle,level;
{
    if (level-- <= 0) return;
    *angle += orient * 90;
    hilbert(-orient,angle,level);
    step(*angle);
    *angle -= orient * 90;
    hilbert(orient,angle,level);
    step(*angle);
    hilbert(orient,angle,level);
    *angle -= orient * 90;
    step(*angle);
    hilbert(-orient,angle,level);
    *angle += orient * 90;
}

/* Recursive Peano-curve generation */
/* Same parameters as Hilbert above */
/* 3^LEVEL by 3^LEVEL points visited */

void peano (orient,angle,level)
long orient,*angle,level;
{
    if (level-- <= 0) return;
    peano(orient,angle,level);
    step(*angle);
    peano(-orient,angle,level);
    step(*angle);
    peano(orient,angle,level);
    *angle -= orient * 90;
    step(*angle);
    *angle -= orient * 90;
    peano(-orient,angle,level);
    step(*angle);
    peano(orient,angle,level);
    step(*angle);
}
```

```
    peano(-orient,angle,level);
    *angle += orient * 90;
    step(*angle);
    *angle += orient * 90;
    peano(orient,angle,level);
    step(*angle);
    peano(-orient,angle,level);
    step(*angle);
    peano(orient,angle,level);
}

void main()
{
    long initial_angle;

    /* Set up window on screen for 24-bit drawing */
    /* This presumes SGI graphics library          */
    preposition(192,1088,236,788);
    foreground();
    winopen("Hilbert and Peano curves");
    RGBmode();
    gconfig();
    cpack(0x00701030); /* Background = indigo */
    clear();
    cpack(0x00FFFFFF); /* Curve = white */

    /* Start polyline near bottom left corner */
    bgnline();
    coord[0] = 20;
    coord[1] = 20;
    v2i(coord);

    /* Visit 128x128 points along Hilbert curve using STEP_SIZE steps, */
    /* so pattern will fill 512x512 area on screen since STEP_SIZE = 4 */
    initial_angle = 0;
    hilbert(1,&initial_angle,7);

    /* Start polyline to right of other curve */
    bgnline();
    coord[0] = 552;
    coord[1] = 20;
    v2i(coord);

    /* Visit 81x81 points along Peano curve using STEP_SIZE steps,      */
    /* so pattern will fill 324x324 area on screen since STEP_SIZE = 4 */
    initial_angle = 0;
    peano(-1,&initial_angle,4);

    /* All done...admire it for 10 seconds */
    endlne();
    sleep(10);
}
```



```
#include <math.h>

/* this version of SIGN3 shows some numerical instability, and is improved
 * by using the uncommented macro that follows, and a different test with it */
#ifdef OLD_TEST
    #define SIGN3( A ) (((A).x<0)?4:0 | ((A).y<0)?2:0 | ((A).z<0)?1:0)
#else
    #define EPS 10e-5
    #define SIGN3( A ) \
        (((A).x < EPS) ? 4 : 0 | ((A).x > -EPS) ? 32 : 0 | \
         ((A).y < EPS) ? 2 : 0 | ((A).y > -EPS) ? 16 : 0 | \
         ((A).z < EPS) ? 1 : 0 | ((A).z > -EPS) ? 8 : 0)
#endif

#define CROSS( A, B, C ) { \
    (C).x = (A).y * (B).z - (A).z * (B).y; \
    (C).y = -(A).x * (B).z + (A).z * (B).x; \
    (C).z = (A).x * (B).y - (A).y * (B).x; \
}

#define SUB( A, B, C ) { \
    (C).x = (A).x - (B).x; \
    (C).y = (A).y - (B).y; \
    (C).z = (A).z - (B).z; \
}

#define LERP( A, B, C ) ((B)+(A)*((C)-(B)))
#define MIN3(a,b,c) (((a)<(b))&&((a)<(c))) ? (a) : (((b)<(c)) ? (b) : (c))
#define MAX3(a,b,c) (((a)>(b))&&((a)>(c))) ? (a) : (((b)>(c)) ? (b) : (c))
#define INSIDE 0
#define OUTSIDE 1

typedef struct {
    float      x;
    float      y;
    float      z;
} Point3;

typedef struct{
    Point3 v1;          /* Vertex1 */
    Point3 v2;          /* Vertex2 */
    Point3 v3;          /* Vertex3 */
} Triangle3;

/* _____ */

/* Which of the six face-plane(s) is point P outside of? */

long face_plane(Point3 p)
{
    long outcode;

    outcode = 0;
    if (p.x > .5) outcode |= 0x01;
    if (p.x < -.5) outcode |= 0x02;
    if (p.y > .5) outcode |= 0x04;
    if (p.y < -.5) outcode |= 0x08;
    if (p.z > .5) outcode |= 0x10;
    if (p.z < -.5) outcode |= 0x20;
    return(outcode);
}

/* . . . . . */
```

```
/* Which of the twelve edge plane(s) is point P outside of? */

long bevel_2d(Point3 p)
{
    long outcode;

    outcode = 0;
    if ( p.x + p.y > 1.0) outcode |= 0x001;
    if ( p.x - p.y > 1.0) outcode |= 0x002;
    if (-p.x + p.y > 1.0) outcode |= 0x004;
    if (-p.x - p.y > 1.0) outcode |= 0x008;
    if ( p.x + p.z > 1.0) outcode |= 0x010;
    if ( p.x - p.z > 1.0) outcode |= 0x020;
    if (-p.x + p.z > 1.0) outcode |= 0x040;
    if (-p.x - p.z > 1.0) outcode |= 0x080;
    if ( p.y + p.z > 1.0) outcode |= 0x100;
    if ( p.y - p.z > 1.0) outcode |= 0x200;
    if (-p.y + p.z > 1.0) outcode |= 0x400;
    if (-p.y - p.z > 1.0) outcode |= 0x800;
    return(outcode);
}

/* . . . . . */

/* Which of the eight corner plane(s) is point P outside of? */

long bevel_3d(Point3 p)
{
    long outcode;

    outcode = 0;
    if (( p.x + p.y + p.z) > 1.5) outcode |= 0x01;
    if (( p.x + p.y - p.z) > 1.5) outcode |= 0x02;
    if (( p.x - p.y + p.z) > 1.5) outcode |= 0x04;
    if (( p.x - p.y - p.z) > 1.5) outcode |= 0x08;
    if ((-p.x + p.y + p.z) > 1.5) outcode |= 0x10;
    if ((-p.x + p.y - p.z) > 1.5) outcode |= 0x20;
    if ((-p.x - p.y + p.z) > 1.5) outcode |= 0x40;
    if ((-p.x - p.y - p.z) > 1.5) outcode |= 0x80;
    return(outcode);
}

/* . . . . . */

/* Test the point "alpha" of the way from P1 to P2 */
/* See if it is on a face of the cube */
/* Consider only faces in "mask" */

long check_point(Point3 p1, Point3 p2, float alpha, long mask)
{
    Point3 plane_point;

    plane_point.x = LERP(alpha, p1.x, p2.x);
    plane_point.y = LERP(alpha, p1.y, p2.y);
    plane_point.z = LERP(alpha, p1.z, p2.z);
    return(face_plane(plane_point) & mask);
}

/* . . . . . */
```

```
/* Compute intersection of P1 --> P2 line segment with face planes */
/* Then test intersection point to see if it is on cube face */
/* Consider only face planes in "outcode_diff" */
/* Note: Zero bits in "outcode_diff" means face line is outside of */

long check_line(Point3 p1, Point3 p2, long outcode_diff)
{
    if ((0x01 & outcode_diff) != 0)
        if (check_point(p1,p2,(.5-p1.x)/(p2.x-p1.x),0x3e) == INSIDE) return(INSIDE);
    if ((0x02 & outcode_diff) != 0)
        if (check_point(p1,p2,(-.5-p1.x)/(p2.x-p1.x),0x3d) == INSIDE) return(INSIDE);
    if ((0x04 & outcode_diff) != 0)
        if (check_point(p1,p2,(.5-p1.y)/(p2.y-p1.y),0x3b) == INSIDE) return(INSIDE);
    if ((0x08 & outcode_diff) != 0)
        if (check_point(p1,p2,(-.5-p1.y)/(p2.y-p1.y),0x37) == INSIDE) return(INSIDE);
    if ((0x10 & outcode_diff) != 0)
        if (check_point(p1,p2,(.5-p1.z)/(p2.z-p1.z),0x2f) == INSIDE) return(INSIDE);
    if ((0x20 & outcode_diff) != 0)
        if (check_point(p1,p2,(-.5-p1.z)/(p2.z-p1.z),0x1f) == INSIDE) return(INSIDE);
    return(OUTSIDE);
}

/* . . . . . */

/* Test if 3D point is inside 3D triangle */

long point_triangle_intersection(Point3 p, Triangle3 t)
{
    long sign12,sign23,sign31;
    Point3 vect12,vect23,vect31,vect1h,vect2h,vect3h;
    Point3 cross12_1p,cross23_2p,cross31_3p;

    /* First, a quick bounding-box test: */
    /* If P is outside triangle bbox, there cannot be an intersection. */

    if (p.x > MAX3(t.v1.x, t.v2.x, t.v3.x)) return(OUTSIDE);
    if (p.y > MAX3(t.v1.y, t.v2.y, t.v3.y)) return(OUTSIDE);
    if (p.z > MAX3(t.v1.z, t.v2.z, t.v3.z)) return(OUTSIDE);
    if (p.x < MIN3(t.v1.x, t.v2.x, t.v3.x)) return(OUTSIDE);
    if (p.y < MIN3(t.v1.y, t.v2.y, t.v3.y)) return(OUTSIDE);
    if (p.z < MIN3(t.v1.z, t.v2.z, t.v3.z)) return(OUTSIDE);

    /* For each triangle side, make a vector out of it by subtracting vertexes; */
    /* make another vector from one vertex to point P. */
    /* The crossproduct of these two vectors is orthogonal to both and the */
    /* signs of its X,Y,Z components indicate whether P was to the inside or */
    /* to the outside of this triangle side. */

    SUB(t.v1, t.v2, vect12)
    SUB(t.v1, p, vect1h);
    CROSS(vect12, vect1h, cross12_1p)
    sign12 = SIGN3(cross12_1p); /* Extract X,Y,Z signs as 0..7 or 0...63 integer */

    SUB(t.v2, t.v3, vect23)
    SUB(t.v2, p, vect2h);
    CROSS(vect23, vect2h, cross23_2p)
    sign23 = SIGN3(cross23_2p);

    SUB(t.v3, t.v1, vect31)
    SUB(t.v3, p, vect3h);
```

```
CROSS(vect31, vect3h, cross31_3p)
sign31 = SIGN3(cross31_3p);

/* If all three crossproduct vectors agree in their component signs, */
/* then the point must be inside all three. */
/* P cannot be OUTSIDE all three sides simultaneously. */

/* this is the old test; with the revised SIGN3() macro, the test
 * needs to be revised. */
#ifdef OLD_TEST
    if ((sign12 == sign23) && (sign23 == sign31))
        return(INSIDE);
    else
        return(OUTSIDE);
#else
    return (sign12 & sign23 & sign31 == 0) ? OUTSIDE : INSIDE;
#endif
}

/* . . . . . */

/*****/
/* This is the main algorithm procedure. */
/* Triangle t is compared with a unit cube, */
/* centered on the origin. */
/* It returns INSIDE (0) or OUTSIDE(1) if t */
/* intersects or does not intersect the cube. */
/*****/

long t_c_intersection(Triangle3 t)
{
    long v1_test,v2_test,v3_test;
    float d;
    Point3 vect12,vect13,norm;
    Point3 hitpp,hitpn,hitnp,hitnn;

    /* First compare all three vertexes with all six face-planes */
    /* If any vertex is inside the cube, return immediately! */

    if ((v1_test = face_plane(t.v1)) == INSIDE) return(INSIDE);
    if ((v2_test = face_plane(t.v2)) == INSIDE) return(INSIDE);
    if ((v3_test = face_plane(t.v3)) == INSIDE) return(INSIDE);

    /* If all three vertexes were outside of one or more face-planes, */
    /* return immediately with a trivial rejection! */

    if ((v1_test & v2_test & v3_test) != 0) return(OUTSIDE);

    /* Now do the same trivial rejection test for the 12 edge planes */

    v1_test |= bevel_2d(t.v1) << 8;
    v2_test |= bevel_2d(t.v2) << 8;
    v3_test |= bevel_2d(t.v3) << 8;
    if ((v1_test & v2_test & v3_test) != 0) return(OUTSIDE);

    /* Now do the same trivial rejection test for the 8 corner planes */

    v1_test |= bevel_3d(t.v1) << 24;
    v2_test |= bevel_3d(t.v2) << 24;
    v3_test |= bevel_3d(t.v3) << 24;
    if ((v1_test & v2_test & v3_test) != 0) return(OUTSIDE);
```

```
/* If vertex 1 and 2, as a pair, cannot be trivially rejected */
/* by the above tests, then see if the v1-->v2 triangle edge */
/* intersects the cube. Do the same for v1-->v3 and v2-->v3. */
/* Pass to the intersection algorithm the "OR" of the outcode */
/* bits, so that only those cube faces which are spanned by */
/* each triangle edge need be tested. */

    if ((v1_test & v2_test) == 0)
        if (check_line(t.v1,t.v2,v1_test|v2_test) == INSIDE) return(INSIDE);
    if ((v1_test & v3_test) == 0)
        if (check_line(t.v1,t.v3,v1_test|v3_test) == INSIDE) return(INSIDE);
    if ((v2_test & v3_test) == 0)
        if (check_line(t.v2,t.v3,v2_test|v3_test) == INSIDE) return(INSIDE);

/* By now, we know that the triangle is not off to any side, */
/* and that its sides do not penetrate the cube. We must now */
/* test for the cube intersecting the interior of the triangle. */
/* We do this by looking for intersections between the cube */
/* diagonals and the triangle...first finding the intersection */
/* of the four diagonals with the plane of the triangle, and */
/* then if that intersection is inside the cube, pursuing */
/* whether the intersection point is inside the triangle itself. */

/* To find plane of the triangle, first perform crossproduct on */
/* two triangle side vectors to compute the normal vector. */

    SUB(t.v1,t.v2,vect12);
    SUB(t.v1,t.v3,vect13);
    CROSS(vect12,vect13,norm)

/* The normal vector "norm" X,Y,Z components are the coefficients */
/* of the triangles  $AX + BY + CZ + D = 0$  plane equation. If we */
/* solve the plane equation for  $X=Y=Z$  (a diagonal), we get */
/*  $-D/(A+B+C)$  as a metric of the distance from cube center to the */
/* diagonal/plane intersection. If this is between -0.5 and 0.5, */
/* the intersection is inside the cube. If so, we continue by */
/* doing a point/triangle intersection. */
/* Do this for all four diagonals. */

    d = norm.x * t.v1.x + norm.y * t.v1.y + norm.z * t.v1.z;
    hitpp.x = hitpp.y = hitpp.z = d / (norm.x + norm.y + norm.z);
    if (fabs(hitpp.x) <= 0.5)
        if (point_triangle_intersection(hitpp,t) == INSIDE) return(INSIDE);
    hitpn.z = -(hitpn.x = hitpn.y = d / (norm.x + norm.y - norm.z));
    if (fabs(hitpn.x) <= 0.5)
        if (point_triangle_intersection(hitpn,t) == INSIDE) return(INSIDE);
    hitnp.y = -(hitnp.x = hitnp.z = d / (norm.x - norm.y + norm.z));
    if (fabs(hitnp.x) <= 0.5)
        if (point_triangle_intersection(hitnp,t) == INSIDE) return(INSIDE);
    hitnn.y = hitnn.z = -(hitnn.x = d / (norm.x - norm.y - norm.z));
    if (fabs(hitnn.x) <= 0.5)
        if (point_triangle_intersection(hitnn,t) == INSIDE) return(INSIDE);

/* No edge touched the cube; no cube diagonal touched the triangle. */
/* We're done...there was no intersection. */

    return(OUTSIDE);

}
```

```
/*
Forms, Vectors, and Transforms
by Bob Wallis
from "Graphics Gems", Academic Press, 1990
*/

/*-----
The main program below is set up to solve the Bezier subdivision problem
in "Forms, Vectors, and Transforms". The subroutines are useful in
solving general problems which require manipulating matrices via exact
integer arithmetic. The intended application is validating or avoiding
tedious algebraic calculations. As such, no thought was given to
efficiency.
-----*/
#define ABS(x) ((x)>(0)? (x):(-x))
#define N 4 /* size of matrices to deal with */
int M[N][N] = /* Bezier weights */
{
    1, 0, 0, 0,
    -3, 3, 0, 0,
    3, -6, 3, 0,
    -1, 3, -3, 1,
};
int T[N][N] = /* re-parameterization xform for top half */
{
    1, -1, 1, -1,
    0, 2, -4, 6,
    0, 0, 4, -12,
    0, 0, 0, 8
};
main ()
{
    int i,
        j,
        scale,
        gcd,
        C[N][N],
        S[N][N],
        Madj[N][N],
        Tadj[N][N],
        Mdet,
        Tdet;

    Tdet = adjoint (T, Tadj); /* inverse without division by */
    Mdet = adjoint (M, Madj); /* determinant of T and M */
    matmult (Madj, Tadj, C);
    matmult (C, M, S); /* Madj*Tadj*M -> S */
    scale = gcd = Mdet * Tdet; /* scale factors of both determinants */
    for (i = 0; i < N; i++) /* find the greatest common */
    { /* denominator of S and determinants */
        for (j = 0; j < N; j++)
            gcd = Gcd (gcd, S[i][j]);
    }
    scale /= gcd; /* divide everything by gcd to get */
    for (i = 0; i < N; i++) /* matrix and scale factor in lowest */
    { /* integer terms possible */
        for (j = 0; j < N; j++)
            S[i][j] /= gcd;
    }
    printf ("scale factor = 1/%d ", scale);
}
```

```
print_mat ("M=", M, N);      /* display the results */
print_mat ("T=", T, N);
print_mat ("S=", S, N);      /* subdivision matrix */
exit (0);
}
Gcd (a, b)                    /*returns greatest common denominator */
int  a,                        /* of (a,b) */
    b;
{
    int    i,
           r;

    a = ABS (a);               /* force positive */
    b = ABS (b);
    if (a < b)                  /* exchange so that a >= b */
    {
        i = b;
        b = a;
        a = i;
    }
    if (b == 0)
        return (a);           /* finished */
    r = a % b;                  /* remainder */
    if (r == 0)
        return (b);           /* finished */
    else
        return (Gcd (b, r));  /* recursive call */
}

adjoint (A, Aadj)             /* returns determinant of A */
int  A[N][N],                 /* input matrix */
    Aadj[N][N];               /* output = adjoint of A */
{
    int    i,
           j,
           I[N],               /* arrays of row and column indices */
           J[N],
           Isub[N],            /* sub-arrays of the above */
           Jsub[N],
           cofactor,
           det;

    if (N < 3)
    {
        printf ("must have N >= 3\n");
        exit (1);
    }
    for (i = 0; i < N; i++)
    {
        I[i] = i;              /* lookup tables to select a */
        J[i] = i;              /* particular subset of */
                                /* rows and columns */
    }
    det = 0;
    for (i = 0; i < N; i++)
    {
        subarray (I, Isub, N, i); /* delete ith row */
        for (j = 0; j < N; j++)
        {
            subarray (J, Jsub, N, j); /* delete jth column */
            cofactor = determinant (A, Isub, Jsub, N - 1, (i + j) & 1);
            if (j == 0)           /* use 0th column for det */
                det += cofactor * A[i][0];
            Aadj[j][i] = cofactor;
        }
    }
}
```

```

    }
    }
    return (det);
}
determinant (A, I, J, n, parity)/* actually gets a sub-determinant */
int      A[N][N],                /* input = entire matrix */
      I[N],                      /* row sub-array we want */
      J[N],                      /* col sub-array we want */
      parity,                    /* 1-> flip polarity */
      n;                         /* # elements in subarrays */

{
    int      i,
            j,
            det,
            j_,
            Jsub[N];
    if (n <= 2)                  /* call ourselves till we get down to */
    {                             /* a 2x2 matrix */
        det =
            (A[I[0]][J[0]] * A[I[1]][J[1]]) -
            (A[I[1]][J[0]] * A[I[0]][J[1]]);
        if (parity)
            det = -det;
        return (det);
    }
    det = 0;                     /* if (n <= 2) */
    i = I[0];                    /* n > 2; call recursively */
    for (j_ = 0; j_ < n; j_++)  /* strike out 0th row */
    {                             /* strike out jth column */
        subarray (J, Jsub, n, j_);
        j = J[j_];               /* I + 1 => struck out 0th row */
        det += A[i][j] * determinant (A, I + 1, Jsub, n - 1, j_ & 1);
    }
    if (parity)
        det = -det;
    return (det);
}

subarray (src, dest, n, k)      /* strike out kth row/column */
int      *src,                 /* source array of n indices */
      *dest,                   /* dest array formed by deleting k */
      n,
      k;

{
    int      i;
    for (i = 0; i < n; i++, src++)
        if (i != k)             /* skip over k */
            *dest++ = *src;
}

matmult (A, B, C)              /* C = A*B */
int      A[N][N],
      B[N][N],
      C[N][N];

{
    int      i,
            j,
            k,
            sum;
    for (i = 0; i < N; i++)
    {

```













```
        for (k = 0; k < N; k++)
        {
            sum = 0;
            for (j = 0; j < N; j++)
                sum += A[i][j] * B[j][k];
            C[i][k] = sum;
        }
    }
}

print_mat (string, mat, n)
char      *string;
int       mat[N][N],
          n;
{
    int     i,
            j;
    printf ("%s\n", string);
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
            printf (" %8ld", mat[i][j]);
        printf ("\n");
    }
}
```

# Index of

## /pubs/tog/GraphicsGems/gemsiii/luminaire/

Name	Last modified	Size	Description
 <a href="#">_</a> <a href="#">Parent Directory</a>			
 <a href="#">_</a> <a href="#">Makefile</a>	29-Jun-00 08:16	1K	
 <a href="#">_</a> <a href="#">geometry_object.C</a>	29-Jun-00 08:16	1K	
 <a href="#">_</a> <a href="#">geometry_object.h</a>	29-Jun-00 08:16	1K	
 <a href="#">_</a> <a href="#">special_instruction</a>	29-Jun-00 08:16	1K	
 <a href="#">_</a> <a href="#">sphere_luminaire.C</a>	29-Jun-00 08:16	1K	
 <a href="#">_</a> <a href="#">sphere_luminaire.h</a>	29-Jun-00 08:16	1K	
 <a href="#">_</a> <a href="#">triangle_luminaire.C</a>	29-Jun-00 08:16	1K	
 <a href="#">_</a> <a href="#">triangle_luminaire.h</a>	29-Jun-00 08:16	1K	
 <a href="#">_</a> <a href="#">utility.h</a>	29-Jun-00 08:16	4K	

```
# code is C++
CFLAGS =

all:    geometry_object.o sphere_luminaire.o triangle_luminaire.o

geometry_object.o:    geometry_object.C geometry_object.h utility.h
    CC $(CFLAGS) -c geometry_object.C -o geometry_object.o -lm

sphere_luminaire.o:    sphere_luminaire.C sphere_luminaire.h utility.h
    CC $(CFLAGS) -c sphere_luminaire.C -o sphere_luminaire.o -lm

triangle_luminaire.o:    triangle_luminaire.C triangle_luminaire.h utility.h
    CC $(CFLAGS) -c triangle_luminaire.C -o triangle_luminaire.o -lm

clean:
    rm -rf geometry_object.o sphere_luminaire.o triangle_luminaire.o
```

```
// *****  
//  
// Physically Correct Direct Lighting For Distribution Ray Tracing  
//      by Changyaw Wang  
//  
// geometry_object.c  
//  
// *****
```

```
#include "utility.h"
```

```
void geom_obj::select_visible_point(  
    const point&,    // not used  
    const double,    // not used  
    const double,    // not used  
    point&,          // not used  
    double&)         // not used  
{ }
```

```
// *****
//
// Physically Correct Direct Lighting For Distribution Ray Tracing
//      by Changyaw Wang
//
// geometry_object.h
//
// *****

// Abstract class geom_obj may be subclassed to a particular type of
// object such as a sphere, or a triangle.

class point;

class geom_obj {
public:

    // Selects a point visible from x given (r1,r2).
    // Here, visible means not SELF-shadowed.

    virtual void select_visible_point(
        const point& x,    // viewpoint
        const double r1,  // random number
        const double r2,  // random number
        point& on_light,   // point corresponding to (r1,r2)
        double& prob);     // probability of selecting on_light
};
```

```
// *****  
//  
// The following C++ code computes a sample point and its probability  
// on the spherical or triangular luminaire according to the solid  
// angle from the view point. These are the center part of the direct  
// lighting computation via Monte Carlo integration.  
//  
// The function hit() is assumed to exist. It returns the intersection  
// point of a ray and a luminaire.  
//  
// Use the command, CC -c *.C -lm, to compile (not link). The code  
// needs to be combined with a ray tracer to complete.  
// *****
```

```
// *****
//
// Physically Correct Direct Lighting For Distribution Ray Tracing
//      by Changyaw Wang
//
// sphere_luminaire.c
//
// *****

#include "utility.h"

#ifndef PI
#define PI          3.14159265358979323846
#endif

// Selects a point visible from x given (r1,r2).
// Here, visible means not SELF-shadowed.

void sphere::select_visible_point(
    const point& x,    // viewpoint
    const double r1,  // random number
    const double r2,  // random number
    point& on_light,  // point corresponding to (r1,r2)
    double& prob)     // probability of selecting on_light
{
    rotation_matrix m;
    double theta, phi, theta_max;
    vector v1, v2, u, v, w, t, psi;

    theta_max = asin(radius/distance(x,center));
    theta = acos(1.0 - r1 + r1 * cos(theta_max));
    phi    = 2.0 * PI * r2;
    psi    = spherical_to_vector(theta,phi);
    w = center - x;           // (u,v,w) is (X',Y',Z') in the text
    w.normalize();
    if(fabs(w.data[0]) >= fabs(w.data[1]))
        { t.data[0] = 0.0; t.data[1] = 1.0; t.data[2] = 0.0; }
    else
        { t.data[0] = 1.0; t.data[1] = 0.0; t.data[2] = 0.0; }
    u = cross(w,t);
    u.normalize();
    v = cross(w,u);
    v.normalize();
    m.set_identity();
    m.set_xyz_to_uvw(u,v,w);
    psi = m * psi;
    psi.normalize();
    hit(x, psi, on_light);    // on_light is x" in the text
    v1 = on_light - center;
    v1.normalize();
    v2 = x - on_light;
    v2.normalize();
    prob = dot(v1,v2) * 0.5 /
        (distance_squared(x,on_light) * PI * (1.0 - cos(theta_max)));
}
```

```
// *****  
//  
// Physically Correct Direct Lighting For Distribution Ray Tracing  
// by Changyaw Wang  
//  
// sphere_luminaire.h  
//  
// *****
```

```
class sphere : public geom_obj {  
public:  
    point center;  
    double radius;  
  
    void hit(const point& x,    // viewpoint  
             const vector& v,  // viewing direction  
             const point on_light); // hit point  
  
    // Selects a point visible from x given (r1,r2).  
    // Here, visible means not SELF-shadowed.  
  
    virtual void select_visible_point(  
        const point& x,    // viewpoint  
        const double r1,   // random number  
        const double r2,   // random number  
        point& on_light,    // point corresponding to (r1,r2)  
        double& prob);      // probability of selecting on_light  
};
```



```
// *****  
//  
// Physically Correct Direct Lighting For Distribution Ray Tracing  
//      by Changyaw Wang  
//  
// triangle_luminaire.c  
//  
// *****
```

```
#include "utility.h"
```

```
// Selects a point visible from x given (r1,r2).  
// Here, visible means not SELF-shadowed.
```

```
void triangle::select_visible_point(  
    const point& x,    // viewpoint  
    const double r1,   // random number  
    const double r2,   // random number  
    point& on_light,   // point corresponding to (r1,r2)  
    double& probab)    // probability of selecting on_light  
{  
    point pt, pt1, pt2, pt3;  
    vector v1, v2, v3, psi, temp1, temp2;  
    double u, v, area;  
  
    v1 = p1 - x;  
    v1.normalize();  
    pt1 = x + v1;  
    v2 = p2 - x;  
    v2.normalize();  
    pt2 = x + v2;  
    v3 = p3 - x;  
    v3.normalize();  
    pt3 = x + v3;  
    u = 1.0 - sqrt(1.0 - r1);  
    v = r2 * sqrt(1.0 - r1);  
  
    pt = pt1 + u*(pt2 - pt1) + v*(pt3 - pt1);  
    psi = pt - x;  
    psi.normalize();  
    hit(x, psi, on_light);  
    temp1 = pt2 - pt1;  
    temp2 = pt3 - pt1;  
    temp1 = cross(temp1,temp2);  
    // area is the area of pt1,pt2,pt3  
    area = 0.5 * sqrt(dot(temp1, temp1));  
    temp1.normalize();  
    probab = distance_squared(x,pt)*dot(-1.0*psi,normal) /  
            (distance_squared(x,on_light)*dot(-1.0*psi,temp1)*area);  
}
```

```
// *****
//
// Physically Correct Direct Lighting For Distribution Ray Tracing
//      by Changyaw Wang
//
// triangle_luminaire.h
//
// *****

class triangle : public geom_obj {
public:
    point p1;           // vertex
    point p2;           // vertex
    point p3;           // vertex
    vector normal;      // normal vector

    void hit(const point& x,    // viewpoint
            const vector& v,    // viewing direction
            const point on_light); // hit point

    // Selects a point visible from x given (r1,r2).
    // Here, visible means not SELF-shadowed.

    virtual void select_visible_point(
        const point& x,    // viewpoint
        const double r1,   // random number
        const double r2,   // random number
        point& on_light,    // point corresponding to (r1,r2)
        double& prob);      // probability of selecting on_light
};
```

```
// *****
//
// Physically Correct Direct Lighting For Distribution Ray Tracing
//          by Changyaw Wang
//
// utility.h
//
// *****

#include <math.h>
#include <stream.h>

class point;
class vector;
class rotation_matrix;

class point {
public:
    double data[3];
    point();
    point(double, double, double);
    point& operator=(const point&);
};

class vector {
public:
    double data[3];
    vector();
    vector(double,double,double);
    vector& operator=(const vector&);
    void normalize();
    friend vector operator*(double, const vector&);
};

class rotation_matrix {
protected:
    double data[4][4];
public:
    void set_identity();
    void set_xyz_to_uvw(const vector&, const vector&, const vector&);
    friend vector operator*(const rotation_matrix&, const vector&);
};

inline point::point() {}

inline vector::vector() {}

inline point& point::operator=(const point& v)
{
    data[0] = v.data[0];
    data[1] = v.data[1];
    data[2] = v.data[2];
    return *this;
}

inline point::point(double a, double b, double c)
{
    data[0] = a; data[1] = b; data[2] = c;
}

inline vector::vector(double a, double b, double c)
```

```
{
    data[0] = a; data[1] = b; data[2] = c;
}

inline vector& vector::operator=(const vector& v)
{
    data[0] = v.data[0];
    data[1] = v.data[1];
    data[2] = v.data[2];
    return *this;
}

inline vector operator*(double d, const vector& v)
{
    return vector(d * v.data[0],
                  d * v.data[1],
                  d * v.data[2] );
}

inline void vector::normalize()
{
    double length;

    length = sqrt(data[0]*data[0] + data[1]*data[1] + data[2]*data[2]);
    if (length <= 1.0e-6)
    {
        data[0] = 1.0; data[1] = 0.0; data[2] = 0.0;
    }
    else
    {
        length = 1.0 / length;
        data[0] *= length; data[1] *= length; data[2] *= length;
    }
}

inline void rotation_matrix::set_identity()
{
    data[0][0] = data[1][1] = data[2][2] = data[3][3] = 1.0;
    data[0][1] = data[0][2] = data[0][3] = 0.0;
    data[1][0] = data[1][2] = data[1][3] = 0.0;
    data[2][0] = data[2][1] = data[2][3] = 0.0;
    data[3][0] = data[3][1] = data[3][2] = 0.0;
}

void rotation_matrix::set_xyz_to_uvw(const vector& u, const vector& v,
                                     const vector& w)
{
    this->set_identity();
    data[0][0] = u.data[0];
    data[1][0] = u.data[1];
    data[2][0] = u.data[2];

    data[0][1] = v.data[0];
    data[1][1] = v.data[1];
    data[2][1] = v.data[2];

    data[0][2] = w.data[0];
    data[1][2] = w.data[1];
    data[2][2] = w.data[2];
}
```

```
inline point operator+(const point& p, const vector& v)
{
    return point(p.data[0] + v.data[0], p.data[1] + v.data[1],
                p.data[2] + v.data[2]);
}

inline vector operator*(const rotation_matrix& r, const vector& v)
{
    return vector(r.data[0][0]*v.data[0] + r.data[0][1]*v.data[1] +
                r.data[0][2]*v.data[2] + r.data[0][3],
                r.data[1][0]*v.data[0] + r.data[1][1]*v.data[1] +
                r.data[1][2]*v.data[2] + r.data[1][3],
                r.data[2][0]*v.data[0] + r.data[2][1]*v.data[1] +
                r.data[2][2]*v.data[2] + r.data[2][3]);
}

inline vector operator-(const point& u, const point& v)
{
    return vector(u.data[0]-v.data[0], u.data[1]-v.data[1],
                u.data[2]-v.data[2]);
}

inline double dot(const vector& a, const vector& b)
{
    return ( a.data[0]*b.data[0] + a.data[1]*b.data[1] +
            a.data[2] * b.data[2] );
}

inline double distance_squared(const point& a, const point& b)
{
    double x,y,z;
    x = a.data[0] - b.data[0];
    y = a.data[1] - b.data[1];
    z = a.data[2] - b.data[2];
    return x*x + y*y + z*z;
}

inline double distance(const point& a, const point& b)
{
    double t;
    t = distance_squared(a,b);
    return sqrt(t);
}

inline vector spherical_to_vector(double theta, double phi)
{
    return vector(sin(theta)*cos(phi),
                sin(theta)*sin(phi),
                cos(theta));
}










inline vector cross(const vector& a, const vector &b)
{
    return vector(a.data[1]*b.data[2] - a.data[2]*b.data[1],
                a.data[2]*b.data[0] - a.data[0]*b.data[2],
                a.data[0]*b.data[1] - a.data[1]*b.data[0]);
}

#include "geometry_object.h"
```

```
#include "sphere_luminaire.h"  
#include "triangle_luminaire.h"
```

# Index of

## /pubs/tog/GraphicsGems/gemsii/RealPixels/

Name	Last modified	Size	Description
 <a href="#">Parent Directory</a>			
 <a href="#">Makefile</a>	29-Jun-00 08:15	1K	
 <a href="#">color.c</a>	29-Jun-00 08:15	5K	
 <a href="#">color.h</a>	29-Jun-00 08:15	2K	
 <a href="#">colrops.c</a>	29-Jun-00 08:15	3K	
 <a href="#">header.c</a>	29-Jun-00 08:15	3K	
 <a href="#">ra_pr24.c</a>	29-Jun-00 08:15	4K	
 <a href="#">rasterfile.h</a>	29-Jun-00 08:15	1K	
 <a href="#">resolu.c</a>	29-Jun-00 08:15	1K	

```
CFLAGS = -g
LIBS = -lm
```

```
ra_pr24:      color.o colrops.o header.o resolu.o rasterfile.h color.h
              cc $(CFLAGS) ra_pr24.c -o ra_pr24 \
                  color.o colrops.o header.o resolu.o $(LIBS)
```

```
color.o:      color.c color.h
              cc $(CFLAGS) -c color.c -o color.o
```

```
colrops.o:    colrops.c color.h
              cc $(CFLAGS) -c colrops.c -o colrops.o
```

```
header.o:     header.c
              cc $(CFLAGS) -c header.c -o header.o
```

```
resolu.o:     resolu.c color.h
              cc $(CFLAGS) -c resolu.c -o resolu.o
```

```
clean:

              /bin/rm -f color.o colrops.o header.o ra_pr24 resolu.o
```



```
/* Copyright (c) 1991 Regents of the University of California */
```

```
#ifndef lint
```

```
static char SCCSid[] = "@(#)color.c 1.15 8/28/91 LBL";
```

```
#endif
```

```
/*
```

```
 * color.c - routines for color calculations.
```

```
 *
```

```
 * 10/10/85
```

```
*/
```

```
#include <stdio.h>
```

```
#include "color.h"
```

```
#define MINELEN 8 /* minimum scanline length for encoding */
```

```
#define MINRUN 4 /* minimum run length */
```

```
char *
```

```
tempbuffer(len) /* get a temporary buffer */
```

```
unsigned len;
```

```
{
```

```
    extern char *malloc(), *realloc();
```

```
    static char *tempbuf = NULL;
```

```
    static int tempbuflen = 0;
```

```
    if (len > tempbuflen) {
```

```
        if (tempbuflen > 0)
```

```
            tempbuf = realloc(tempbuf, len);
```

```
        else
```

```
            tempbuf = malloc(len);
```

```
        tempbuflen = tempbuf==NULL ? 0 : len;
```

```
    }
```

```
    return(tempbuf);
```

```
}
```

```
fwritecolrs(scanline, len, fp) /* write out a colr scanline */
```

```
register COLR *scanline;
```

```
int len;
```

```
register FILE *fp;
```

```
{
```

```
    register int i, j, beg, cnt;
```

```
    int c2;
```

```
    if (len < MINELEN) /* too small to encode */
```

```
        return(fwrite((char *)scanline, sizeof(COLR), len, fp) - len);
```

```
    if (len > 32767) /* too big! */
```

```
        return(-1);
```

```
    putc(2, fp); /* put magic header */
```

```
    putc(2, fp);
```

```
    putc(len>>8, fp);
```

```
    putc(len&255, fp);
```

```
    /* put components separately */
```

```
    for (i = 0; i < 4; i++) {
```

```
        for (j = 0; j < len; j += cnt) { /* find next run */
```

```
            for (beg = j; beg < len; beg += cnt) {
```

```
                for (cnt = 1; cnt < 127 && beg+cnt < len &&
```

```
                    scanline[beg+cnt][i] == scanline[beg][i]; cnt++)
```

```

        ;
        if (cnt >= MINRUN)
            break; /* long enough */
    }
    if (beg-j > 1 && beg-j < MINRUN) {
        c2 = j+1;
        while (scanline[c2++][i] == scanline[j][i])
            if (c2 == beg) { /* short run */
                putc(128+beg-j, fp);
                putc(scanline[j][i], fp);
                j = beg;
                break;
            }
    }
    while (j < beg) { /* write out non-run */
        if ((c2 = beg-j) > 128) c2 = 128;
        putc(c2, fp);
        while (c2--)
            putc(scanline[j++][i], fp);
    }
    if (cnt >= MINRUN) { /* write out run */
        putc(128+cnt, fp);
        putc(scanline[beg][i], fp);
    } else
        cnt = 0;
}
}
return(ferror(fp) ? -1 : 0);
}

```

```

freadcolrs(scanline, len, fp) /* read in an encoded colr scanline */
register COLR *scanline;
int len;
register FILE *fp;
{
    register int i, j;
    int code;

    /* determine scanline type */
    if (len < MINELEN)
        return(oldreadcolrs(scanline, len, fp));
    if ((i = getc(fp)) == EOF)
        return(-1);
    if (i != 2) {
        ungetc(i, fp);
        return(oldreadcolrs(scanline, len, fp));
    }
    scanline[0][GRN] = getc(fp);
    scanline[0][BLU] = getc(fp);
    if ((i = getc(fp)) == EOF)
        return(-1);
    if (scanline[0][GRN] != 2 || scanline[0][BLU] & 128) {
        scanline[0][RED] = 2;
        scanline[0][EXP] = i;
        return(oldreadcolrs(scanline+1, len-1, fp));
    }
    if ((scanline[0][BLU]<8 | i) != len)
        return(-1); /* length mismatch! */
    /* read each component */
    for (i = 0; i < 4; i++)
        for (j = 0; j < len; ) {

```

```
        if ((code = getc(fp)) == EOF)
            return(-1);
        if (code > 128) {           /* run */
            scanline[j++][i] = getc(fp);
            for (code &= 127; --code; j++)
                scanline[j][i] = scanline[j-1][i];
        } else                     /* non-run */
            while (code--)
                scanline[j++][i] = getc(fp);
    }
    return(feof(fp) ? -1 : 0);
}

oldreadcolrs(scanline, len, fp)           /* read in an old colr scanline */
register COLR *scanline;
int len;
register FILE *fp;
{
    int rshift;
    register int i;

    rshift = 0;

    while (len > 0) {
        scanline[0][RED] = getc(fp);
        scanline[0][GRN] = getc(fp);
        scanline[0][BLU] = getc(fp);
        scanline[0][EXP] = getc(fp);
        if (feof(fp) || ferror(fp))
            return(-1);
        if (scanline[0][RED] == 1 &&
            scanline[0][GRN] == 1 &&
            scanline[0][BLU] == 1) {
            for (i = scanline[0][EXP] << rshift; i > 0; i--) {
                copycolr(scanline[0], scanline[-1]);
                scanline++;
                len--;
            }
            rshift += 8;
        } else {
            scanline++;
            len--;
            rshift = 0;
        }
    }
    return(0);
}
```

```
fwritescan(scanline, len, fp)           /* write out a scanline */
register COLOR *scanline;
int len;
FILE *fp;
{
    COLOR *clrscan;
    int n;
    register COLOR *sp;

    /* get scanline buffer */
    if ((sp = (COLOR *)tempbuffer(len*sizeof(COLOR))) == NULL)
        return(-1);
```

```
    clrscan = sp;
                                     /* convert scanline */
    n = len;
    while (n-- > 0) {
        setcolr(sp[0], scanline[0][RED],
                scanline[0][GRN],
                scanline[0][BLU]);
        scanline++;
        sp++;
    }
    return(fwritecolrs(clrscan, len, fp));
}

freadscan(scanline, len, fp)          /* read in a scanline */
register COLOR *scanline;
int len;
FILE *fp;
{
    register COLR *clrscan;

    if ((clrscan = (COLR *)tempbuffer(len*sizeof(COLR))) == NULL)
        return(-1);
    if (freadcolrs(clrscan, len, fp) < 0)
        return(-1);
                                     /* convert scanline */
    colr_color(scanline[0], clrscan[0]);
    while (--len > 0) {
        scanline++; clrscan++;
        if (clrscan[0][RED] == clrscan[-1][RED] &&
            clrscan[0][GRN] == clrscan[-1][GRN] &&
            clrscan[0][BLU] == clrscan[-1][BLU] &&
            clrscan[0][EXP] == clrscan[-1][EXP])
            copycolor(scanline[0], scanline[-1]);
        else
            colr_color(scanline[0], clrscan[0]);
    }
    return(0);
}

setcolr(clr, r, g, b)                  /* assign a short color value */
register COLOR clr;
double r, g, b;
{
    double frexp();
    double d;
    int e;

    d = r > g ? r : g;
    if (b > d) d = b;

    if (d <= 1e-32) {
        clr[RED] = clr[GRN] = clr[BLU] = 0;
        clr[EXP] = 0;
        return;
    }

    d = frexp(d, &e) * 256.0 / d;

    clr[RED] = r * d;
```

```
    clr[GRN] = g * d;
    clr[BLU] = b * d;
    clr[EXP] = e + COLXS;
}
```

```
colr_color(col, clr)                /* convert short to float color */
register COLOR  col;
register COLR  clr;
{
    double  f;

    if (clr[EXP] == 0)
        col[RED] = col[GRN] = col[BLU] = 0.0;
    else {
        f = ldexp(1.0, (int)clr[EXP]-(COLXS+8));
        col[RED] = (clr[RED] + 0.5)*f;
        col[GRN] = (clr[GRN] + 0.5)*f;
        col[BLU] = (clr[BLU] + 0.5)*f;
    }
}
```

```
bigdiff(c1, c2, md)                /* c1 delta c2 > md? */
register COLOR  c1, c2;
double  md;
{
    register int  i;

    for (i = 0; i < 3; i++)
        if (colval(c1,i)-colval(c2,i) > md*colval(c2,i) ||
            colval(c2,i)-colval(c1,i) > md*colval(c1,i))
            return(1);

    return(0);
}
```

```
/* Copyright (c) 1986 Regents of the University of California */
```

```
/* SCCSid "@(#)color.h 1.12 7/17/91 LBL" */
```

```
/*
 * color.h - header for routines using pixel color values.
 *
 *      12/31/85
 *
 * Two color representations are used, one for calculation and
 * another for storage. Calculation is done with three floats
 * for speed. Stored color values use 4 bytes which contain
 * three single byte mantissas and a common exponent.
 */
```

```
#define RED          0
#define GRN          1
#define BLU          2
#define EXP          3
#define COLXS        128      /* excess used for exponent */

typedef unsigned char BYTE;    /* 8-bit unsigned integer */

typedef BYTE  COLR[4];        /* red, green, blue, exponent */

#define copycolr(c1,c2)      (c1[0]=c2[0],c1[1]=c2[1], \
                             c1[2]=c2[2],c1[3]=c2[3])

typedef float  COLOR[3];      /* red, green, blue */

#define colval(col,pri)      ((col)[pri])

#define setcolor(col,r,g,b)  ((col)[RED]=(r),(col)[GRN]=(g),(col)[BLU]=(b))

#define copycolor(c1,c2)     ((c1)[0]=(c2)[0],(c1)[1]=(c2)[1],(c1)[2]=(c2)[2])

#define scalecolor(col,sf)   ((col)[0]*=(sf),(col)[1]*=(sf),(col)[2]*=(sf))

#define addcolor(c1,c2)      ((c1)[0]+=(c2)[0],(c1)[1]+=(c2)[1],(c1)[2]+=(c2)[2])

#define multcolor(c1,c2)     ((c1)[0]*=(c2)[0],(c1)[1]*=(c2)[1],(c1)[2]*=(c2)[2])

#ifdef NTSC
#define bright(col)          (.295*(col)[RED]+.636*(col)[GRN]+.070*(col)[BLU])
#define normbright(c)        (int)((74L*(c)[RED]+164L*(c)[GRN]+18L*(c)[BLU])/256)
#else
#define bright(col)          (.263*(col)[RED]+.655*(col)[GRN]+.082*(col)[BLU])
#define normbright(c)        (int)((67L*(c)[RED]+168L*(c)[GRN]+21L*(c)[BLU])/256)
#endif

#define luminance(col)       (470. * bright(col))

#define intens(col)          ( (col)[0] > (col)[1] \
                               ? (col)[0] > (col)[2] ? (col)[0] : (col)[2] \
                               : (col)[1] > (col)[2] ? (col)[1] : (col)[2] )

#define colrval(c,p)         ( (c)[EXP] ? \
                               ldexp((c)[p]+.5,(int)(c)[EXP]-(COLXS+8)) : \
                               0. )

#define WHTCOLOR              {1.0,1.0,1.0}
```

```
#define BLKCOLOR      {0.0,0.0,0.0}
#define WHTCOLR      {128,128,128,COLXS+1}
#define BLKCOLR      {0,0,0,0}

/* definitions for resolution header */
#define XDECR        1
#define YDECR        2
#define YMAJOR        4

/* picture format identifier */
#define COLRFMT      "32-bit_rle_rgbe"

/* macros for exposures */
#define EXPOSSTR      "EXPOSURE="
#define LEXPOSSTR      9
#define isexpos(hl)   (!strcmp(hl,EXPOSSTR,LEXPOSSTR))
#define exposval(hl)   atof((hl)+LEXPOSSTR)
#define fputexpos(ex,fp) fprintf(fp,"%s%e\n",EXPOSSTR,ex)

/* macros for pixel aspect ratios */
#define ASPECTSTR      "PIXASPECT="
#define LASPECTSTR      10
#define isaspect(hl)   (!strcmp(hl,ASPECTSTR,LASPECTSTR))
#define aspectval(hl)   atof((hl)+LASPECTSTR)
#define fputaspect(pa,fp) fprintf(fp,"%s%f\n",ASPECTSTR,pa)

/* macros for color correction */
#define COLCORSTR      "COLORCORR="
#define LCOLCORSTR      10
#define iscolcor(hl)   (!strcmp(hl,COLCORSTR,LCOLCORSTR))
#define colcorval(cc,hl) sscanf(hl+LCOLCORSTR,"%f %f %f", \
                                &(cc)[RED],&(cc)[GRN],&(cc)[BLU])
#define fputcolcor(cc,fp) fprintf(fp,"%s %f %f %f\n",COLCORSTR, \
                                (cc)[RED],(cc)[GRN],(cc)[BLU])

extern double ldexp(), atof();
```

```
/* Copyright (c) 1990 Regents of the University of California */
```

```
#ifndef lint
static char SCCSid[] = "@(#)colrops.c 1.3 11/9/90 LBL";
#endif
```

```
/*
 * Integer operations on COLR scanlines
 */
```

```
#include "color.h"
```

```
#define MAXGSHIFT      15                /* maximum shift for gamma table */
```

```
static BYTE      g_mant[256], g_nexp[256];
```

```
static BYTE      g_bval[MAXGSHIFT+1][256];
```

```
setcolrgam(g)                /* set gamma conversion */
```

```
double  g;
{
    extern double  pow();
    double  mult;
    register int   i, j;

    /* compute colr -> gamb mapping */
    for (i = 0; i <= MAXGSHIFT; i++) {
        mult = pow(0.5, (double)(i+8));
        for (j = 0; j < 256; j++)
            g_bval[i][j] = 256.0 * pow((j+.5)*mult, 1.0/g);
    }

    /* compute gamb -> colr mapping */
    i = 0;
    mult = 256.0;
    for (j = 255; j > 0; j--) {
        while ((g_mant[j] = mult * pow(j/256.0, g)) < 128) {
            i++;
            mult *= 2.0;
        }
        g_nexp[j] = i;
    }
    g_mant[0] = 0;
    g_nexp[0] = COLXS;
}
```

```
colrs_gambs(scan, len)        /* convert scanline of colrs to gamma bytes */
```

```
register COLR      *scan;
int               len;
{
    register int     i, expo;

    while (len-- > 0) {
        expo = scan[0][EXP] - COLXS;
        if (expo < -MAXGSHIFT) {
            if (expo < -MAXGSHIFT-8) {
                scan[0][RED] =
                scan[0][GRN] =
                scan[0][BLU] = 0;
            } else {
                i = (-MAXGSHIFT-1) - expo;

```



```

        scan[0][RED] =
        g_bval[MAXGSHIFT][((scan[0][RED]>>i)+1)>>1];
        scan[0][GRN] =
        g_bval[MAXGSHIFT][((scan[0][GRN]>>i)+1)>>1];
        scan[0][BLU] =
        g_bval[MAXGSHIFT][((scan[0][BLU]>>i)+1)>>1];
    }
} else if (expo > 0) {
    if (expo > 8) {
        scan[0][RED] =
        scan[0][GRN] =
        scan[0][BLU] = 255;
    } else {
        i = (scan[0][RED]<<1 | 1) << (expo-1);
        scan[0][RED] = i > 255 ? 255 : g_bval[0][i];
        i = (scan[0][GRN]<<1 | 1) << (expo-1);
        scan[0][GRN] = i > 255 ? 255 : g_bval[0][i];
        i = (scan[0][BLU]<<1 | 1) << (expo-1);
        scan[0][BLU] = i > 255 ? 255 : g_bval[0][i];
    }
} else {
    scan[0][RED] = g_bval[-expo][scan[0][RED]];
    scan[0][GRN] = g_bval[-expo][scan[0][GRN]];
    scan[0][BLU] = g_bval[-expo][scan[0][BLU]];
}
scan[0][EXP] = COLXS;
scan++;
}
}

```

```

gambs_colrs(scan, len)          /* convert gamma bytes to colr scanline */
register COLR    *scan;
int    len;
{
    register int    nexpo;

    while (len-- > 0) {
        nexpo = g_nexp[scan[0][RED]];
        if (g_nexp[scan[0][GRN]] < nexpo)
            nexpo = g_nexp[scan[0][GRN]];
        if (g_nexp[scan[0][BLU]] < nexpo)
            nexpo = g_nexp[scan[0][BLU]];
        if (nexpo < g_nexp[scan[0][RED]])
            scan[0][RED] = g_mant[scan[0][RED]]
                >> (g_nexp[scan[0][RED]]-nexpo);
        else
            scan[0][RED] = g_mant[scan[0][RED]];
        if (nexpo < g_nexp[scan[0][GRN]])
            scan[0][GRN] = g_mant[scan[0][GRN]]
                >> (g_nexp[scan[0][GRN]]-nexpo);
        else
            scan[0][GRN] = g_mant[scan[0][GRN]];
        if (nexpo < g_nexp[scan[0][BLU]])
            scan[0][BLU] = g_mant[scan[0][BLU]]
                >> (g_nexp[scan[0][BLU]]-nexpo);
        else
            scan[0][BLU] = g_mant[scan[0][BLU]];
        scan[0][EXP] = COLXS - nexpo;
        scan++;
    }
}

```

```
}
```

```
shiftcolrs(scan, len, adjust)    /* shift a scanline of colors by 2^adjust */
register COLR    *scan;
register int     len;
register int     adjust;
{
    while (len-- > 0) {
        scan[0][EXP] += adjust;
        scan++;
    }
}
```

```
normcolrs(scan, len, adjust)    /* normalize a scanline of colrs */
register COLR    *scan;
int len;
int adjust;
{
    register int c;
    register int shift;

    while (len-- > 0) {
        shift = scan[0][EXP] + adjust - COLXS;
        if (shift > 0) {
            if (shift > 8) {
                scan[0][RED] =
                scan[0][GRN] =
                scan[0][BLU] = 255;
            } else {
                shift--;
                c = (scan[0][RED]<<1 | 1) << shift;
                scan[0][RED] = c > 255 ? 255 : c;
                c = (scan[0][GRN]<<1 | 1) << shift;
                scan[0][GRN] = c > 255 ? 255 : c;
                c = (scan[0][BLU]<<1 | 1) << shift;
                scan[0][BLU] = c > 255 ? 255 : c;
            }
        } else if (shift < 0) {
            if (shift < -8) {
                scan[0][RED] =
                scan[0][GRN] =
                scan[0][BLU] = 0;
            } else {
                shift = -1-shift;
                scan[0][RED] = ((scan[0][RED]>>shift)+1)>>1;
                scan[0][GRN] = ((scan[0][GRN]>>shift)+1)>>1;
                scan[0][BLU] = ((scan[0][BLU]>>shift)+1)>>1;
            }
        }
        scan[0][EXP] = COLXS - adjust;
        scan++;
    }
}
```

```
/* Copyright (c) 1991 Regents of the University of California */
```

```
#ifndef lint
```

```
static char SCCSid[] = "@(#)header.c 1.4 4/22/91 LBL";
```

```
#endif
```

```
/*
```

```
 * header.c - routines for reading and writing information headers.
```

```
 *
```

```
 *      8/19/88
```

```
 *
```

```
 * printargs(ac,av,fp) print an argument list to fp, followed by '\n'
```

```
 * isformat(s) returns true if s is of the form "FORMAT=*"
```

```
 * formatval(r,s) copy the format value in s to r
```

```
 * fputformat(s,fp) write "FORMAT=%s" to fp
```

```
 * getheader(fp,f,p) read header from fp, calling f(s,p) on each line
```

```
 * checkheader(i,p,o) check header format from i against p and copy to o
```

```
 *
```

```
 * To copy header from input to output, use getheader(fin, fputs, fout)
```

```
 */
```

```
#include <stdio.h>
```

```
#include <ctype.h>
```

```
#define MAXLINE 512
```

```
char FMTSTR[] = "FORMAT=";
```

```
int FMTSTRL = 7;
```

```
printargs(ac, av, fp) /* print arguments to a file */
```

```
int ac;
```

```
char **av;
```

```
FILE *fp;
```

```
{
    while (ac-- > 0) {
        fputs(*av++, fp);
        putc(' ', fp);
    }
    putc('\n', fp);
}
```

```
isformat(s) /* is line a format line? */
```

```
char *s;
```

```
{
    return(!strcmp(s,FMTSTR,FMTSTRL));
}
```

```
formatval(r, s) /* return format value */
```

```
register char *r;
```

```
register char *s;
```

```
{
    s += FMTSTRL;
    while (isspace(*s)) s++;
    if (!*s) { *r = '\0'; return; }
    while(*s) *r++ = *s++;
    while (isspace(r[-1])) r--;
    *r = '\0';
}
```

```
fputformat(s, fp)                /* put out a format value */
char  *s;
FILE  *fp;
{
    fputs(FMTSTR, fp);
    fputs(s, fp);
    putc('\n', fp);
}

getheader(fp, f, p)              /* get header from file */
FILE  *fp;
int   (*f)();
char  *p;
{
    char  buf[MAXLINE];

    for ( ; ; ) {
        buf[MAXLINE-2] = '\n';
        if (fgets(buf, sizeof(buf), fp) == NULL)
            return(-1);
        if (buf[0] == '\n')
            return(0);
        if (buf[MAXLINE-2] != '\n') {
            ungetc(buf[MAXLINE-2], fp);    /* prevent false end */
            buf[MAXLINE-2] = '\0';
        }
        if (f != NULL)
            (*f)(buf, p);
    }
}

struct check {
    FILE  *fp;
    char  fs[64];
};

static
mycheck(s, cp)                  /* check a header line for format info. */
char  *s;
register struct check  *cp;
{
    if (!strncmp(s, FMTSTR, FMTSTRL))
        formatval(cp->fs, s);
    else if (cp->fp != NULL)      /* don't copy format info. */
        fputs(s, cp->fp);
}

/*
 * Copymatch(pat, str) checks pat for wildcards, and
 * copies str into pat if there is a match (returning true).
 */

#ifdef COPYMATCH
copymatch(pat, str)
char  *pat, *str;
```

```
{
    int      docopy = 0;
    register char  *p = pat, *s = str;

    do {
        switch (*p) {
            case '?': /* match any character */
                if (!*s++)
                    return(0);
                docopy++;
                break;
            case '*': /* match any string */
                while (p[1] == '*') p++;
                do
                    if ( (p[1]=='?' || p[1]==*s)
                        && copymatch(p+1,s) ) {
                        strcpy(pat, str);
                        return(1);
                    }
                while (*s++);
                return(0);
            case '\\': /* literal next */
                p++;
            /* fall through */
            default: /* normal character */
                if (*p != *s)
                    return(0);
                s++;
                break;
        }
    } while (*p++);
    if (docopy)
        strcpy(pat, str);
    return(1);
}

#else
#define copymatch(pat, s)      (!strcmp(pat, s))
#endif
```

```
/*
 * Checkheader(fin,fmt,fout) returns a value of 1 if the input format
 * matches the specification in fmt, 0 if no input format was found,
 * and -1 if the input format does not match or there is an
 * error reading the header.  If fmt is empty, then -1 is returned
 * if any input format is found (or there is an error), and 0 otherwise.
 * If fmt contains any '*' or '?' characters, then checkheader
 * does wildcard expansion and copies a matching result into fmt.
 * Be sure that fmt is big enough to hold the match in such cases!
 * The input header (minus any format lines) is copied to fout
 * if fout is not NULL.
 */
```

```
checkheader(fin, fmt, fout)
FILE  *fin;
char  *fmt;
FILE  *fout;
{
    struct check    cdat;

    cdat.fp = fout;
```

```
    cdat.fs[0] = '\\0';  
    if (getheader(fin, mycheck, &cdat) < 0)  
        return(-1);  
    if (cdat.fs[0] != '\\0')  
        return(copymatch(fmt, cdat.fs) ? 1 : -1);  
    return(0);  
}
```

```
/* Copyright (c) 1991 Regents of the University of California */
```

```
#ifndef lint
```

```
static char SCCSid[] = "@(#)ra_pr24.c 1.8 8/15/91 LBL";
```

```
#endif
```

```
/*
```

```
 * program to convert between RADIANCE and 24-bit rasterfiles.
```

```
*/
```

```
#include <stdio.h>
```

```
#include "rasterfile.h"
```

```
#include "color.h"
```

```
extern double  atof(), pow();
```

```
double  gamma = 2.0;                /* gamma correction */
```

```
int  bradj = 0;                      /* brightness adjustment */
```

```
char  *programe;
```

```
int  xmax, ymax;
```

```
main(argc, argv)
```

```
int  argc;
```

```
char  *argv[];
```

```
{
```

```
    struct rasterfile  head;
```

```
    int  reverse = 0;
```

```
    int  i;
```

```
    programe = argv[0];
```

```
    for (i = 1; i < argc; i++)
```

```
        if (argv[i][0] == '-')
```

```
            switch (argv[i][1]) {
```

```
                case 'g':
```

```
                    gamma = atof(argv[++i]);
```

```
                    break;
```

```
                case 'e':
```

```
                    if (argv[i+1][0] != '+' && argv[i+1][0] != '-')
```

```
                        goto userr;
```

```
                    bradj = atoi(argv[++i]);
```

```
                    break;
```

```
                case 'r':
```

```
                    reverse = !reverse;
```

```
                    break;
```

```
                default:
```

```
                    goto userr;
```

```
            }
```

```
        else
```

```
            break;
```

```
    if (i < argc-2)
```

```
        goto userr;
```

```
    if (i <= argc-1 && freopen(argv[i], "r", stdin) == NULL) {
```

```
        fprintf(stderr, "%s: can't open input \"%s\"\n",
```

```

        progname, argv[i]);
    exit(1);
}
if (i == argc-2 && freopen(argv[i+1], "w", stdout) == NULL) {
    fprintf(stderr, "can't open output \"%s\"\n",
        progname, argv[i+1]);
    exit(1);
}
setcolrgam(gamma);
if (reverse) {
    /* get header */
    if (fread((char *)&head, sizeof(head), 1, stdin) != 1)
        quiterr("missing header");
    if (head.ras_magic != RAS_MAGIC)
        quiterr("bad raster format");
    xmax = head.ras_width;
    ymax = head.ras_height;
    if ((head.ras_type != RT_STANDARD
        && head.ras_type != RT_FORMAT_RGB)
        || head.ras_maptype != RMT_NONE
        || head.ras_depth != 24)
        quiterr("incompatible format");
    /* put header */
    printargs(i, argv, stdout);
    fputformat(COLRFMT, stdout);
    putchar('\n');
    fputresolu(YMAJOR|YDECR, xmax, ymax, stdout);
    /* convert file */
    pr2ra(head.ras_type);
} else {
    /* get header info. */
    if (checkheader(stdin, COLRFMT, NULL) < 0 ||
        fgetresolu(&xmax, &ymax, stdin) != (YMAJOR|YDECR))
        quiterr("bad picture format");
    /* write rasterfile header */
    head.ras_magic = RAS_MAGIC;
    head.ras_width = xmax;
    head.ras_height = ymax;
    head.ras_depth = 24;
    head.ras_length = xmax*ymax*3;
    head.ras_type = RT_STANDARD;
    head.ras_maptype = RMT_NONE;
    head.ras_maplength = 0;
    fwrite((char *)&head, sizeof(head), 1, stdout);
    /* convert file */
    ra2pr();
}
exit(0);
userr:
    fprintf(stderr, "Usage: %s [-r][-g gamma][-e +/-stops] [input [output]]\n",
        progname);
    exit(1);
}

quiterr(err)          /* print message and exit */
char *err;
{
    if (err != NULL) {
        fprintf(stderr, "%s: %s\n", progname, err);
        exit(1);
    }
}

```



```
    }
    exit(0);
}

pr2ra(rf)                /* convert 24-bit scanlines to Radiance picture */
int    rf;
{
    COLR    *scanout;
    register int    x;
    int    y;

                                /* allocate scanline */
    scanout = (COLR *)malloc(xmax*sizeof(COLR));
    if (scanout == NULL)
        quiterr("out of memory in pr2ra");

                                /* convert image */
    for (y = ymax-1; y >= 0; y--) {
        for (x = 0; x < xmax; x++)
            if (rf == RT_FORMAT_RGB) {
                scanout[x][RED] = getc(stdin);
                scanout[x][GRN] = getc(stdin);
                scanout[x][BLU] = getc(stdin);
            } else {
                scanout[x][BLU] = getc(stdin);
                scanout[x][GRN] = getc(stdin);
                scanout[x][RED] = getc(stdin);
            }
        if (feof(stdin) || ferror(stdin))
            quiterr("error reading rasterfile");
        gambs_colrs(scanout, xmax);
        if (bradj)
            shiftcolrs(scanout, xmax, bradj);
        if (fwritecolrs(scanout, xmax, stdout) < 0)
            quiterr("error writing Radiance picture");
    }

                                /* free scanline */
    free((char *)scanout);
}

ra2pr()                  /* convert Radiance scanlines to 24-bit rasterfile */
{
    COLR    *scanin;
    register int    x;
    int    y;

                                /* allocate scanline */
    scanin = (COLR *)malloc(xmax*sizeof(COLR));
    if (scanin == NULL)
        quiterr("out of memory in ra2pr");

                                /* convert image */
    for (y = ymax-1; y >= 0; y--) {
        if (freadcolrs(scanin, xmax, stdin) < 0)
            quiterr("error reading Radiance picture");
        if (bradj)
            shiftcolrs(scanin, xmax, bradj);
        colrs_gambs(scanin, xmax);
        for (x = 0; x < xmax; x++) {
            putc(scanin[x][BLU], stdout);
            putc(scanin[x][GRN], stdout);
            putc(scanin[x][RED], stdout);
        }
    }
}
```

```
        if (ferror(stdout))
            quiterr("error writing rasterfile");
    }
    free((char *)scanin);
    /* free scanline */
}
```

```
/*      @(#)rasterfile.h 1.11 89/08/21 SMI      */

/*
 * Description of header for files containing raster images
 */

#ifndef _rasterfile_h
#define _rasterfile_h

struct rasterfile {
    int      ras_magic;          /* magic number */
    int      ras_width;         /* width (pixels) of image */
    int      ras_height;        /* height (pixels) of image */
    int      ras_depth;         /* depth (1, 8, or 24 bits) of pixel */
    int      ras_length;        /* length (bytes) of image */
    int      ras_type;          /* type of file; see RT_* below */
    int      ras_maptype;       /* type of colormap; see RMT_* below */
    int      ras_maplength;     /* length (bytes) of following map */
    /* color map follows for ras_maplength bytes, followed by image */
};

#define RAS_MAGIC      0x59a66a95

    /* Sun supported ras_type's */
#define RT_OLD          0      /* Raw pixrect image in 68000 byte order */
#define RT_STANDARD     1      /* Raw pixrect image in 68000 byte order */
#define RT_BYTE_ENCODED 2      /* Run-length compression of bytes */
#define RT_FORMAT_RGB   3      /* XRGB or RGB instead of XBGR or BGR */
#define RT_FORMAT_TIFF  4      /* tiff <-> standard rasterfile */
#define RT_FORMAT_IFF   5      /* iff (TAAC format) <-> standard rasterfile */
#define RT_EXPERIMENTAL 0xffff /* Reserved for testing */

    /* Sun registered ras_maptype's */
#define RMT_RAW          2

    /* Sun supported ras_maptype's */
#define RMT_NONE          0      /* ras_maplength is expected to be 0 */
#define RMT_EQUAL_RGB     1      /* red[ras_maplength/3],green[],blue[] */

/*
 * NOTES:
 *      Each line of the image is rounded out to a multiple of 16 bits.
 *      This corresponds to the rounding convention used by the memory pixrect
 *      package (/usr/include/pixrect/memvar.h) of the SunWindows system.
 *      The ras_encoding field (always set to 0 by Sun's supported software)
 *      was renamed to ras_length in release 2.0. As a result, rasterfiles
 *      of type 0 generated by the old software claim to have 0 length; for
 *      compatibility, code reading rasterfiles must be prepared to compute the
 *      true length from the width, height, and depth fields.
 */

#endif /* !_rasterfile_h */
```

```
#ifndef lint
static char SCCSid[] = "@(#)resolu.c 1.1 9/22/90 LBL";
#endif

/*
 * Read and write image resolutions.
 */

#include <stdio.h>

#include "color.h"

fputresolu(ord, xres, yres, fp)          /* put x and y resolution */
register int  ord;
int  xres, yres;
FILE  *fp;
{
    if (ord & YMAJOR)
        fprintf(fp, "%cY %d %cX %d\n",
                ord & YDECR ? '-' : '+', yres,
                ord & XDECR ? '-' : '+', xres);
    else
        fprintf(fp, "%cX %d %cY %d\n",
                ord & XDECR ? '-' : '+', xres,
                ord & YDECR ? '-' : '+', yres);
}

fgetresolu(xrp, yrp, fp)                /* get x and y resolution */
int  *xrp, *yrp;
FILE  *fp;
{
    char  buf[64], *xndx, *yndx;
    register char  *cp;
    register int  ord;

    if (fgets(buf, sizeof(buf), fp) == NULL)
        return(-1);
    xndx = yndx = NULL;
    for (cp = buf+1; *cp; cp++)
        if (*cp == 'X')
            xndx = cp;
        else if (*cp == 'Y')
            yndx = cp;
    if (xndx == NULL || yndx == NULL)
        return(-1);
    ord = 0;
    if (xndx > yndx) ord |= YMAJOR;
    if (xndx[-1] == '-') ord |= XDECR;
    if (yndx[-1] == '-') ord |= YDECR;
    if ((*xrp = atoi(xndx+1)) <= 0)
        return(-1);
    if ((*yrp = atoi(yndx+1)) <= 0)
        return(-1);
    return(ord);
}
```

```
/* Copyright (c) 1988 Regents of the University of California */
```

```
#ifndef lint
```

```
static char SCCSid[] = "@(#)noise3.c 2.1 11/12/91 LBL";
```

```
#endif
```

```
/*
```

```
 * noise3.c - noise functions for random textures.
```

```
 *
```

```
 * Credit for the smooth algorithm goes to Ken Perlin.
```

```
 * (ref. SIGGRAPH Vol 19, No 3, pp 287-96)
```

```
 *
```

```
 * 4/15/86
```

```
 * 5/19/88 Added fractal noise function
```

```
 */
```

```
#define A 0
```

```
#define B 1
```

```
#define C 2
```

```
#define D 3
```

```
#define rand3a(x,y,z) frand(67*(x)+59*(y)+71*(z))
```

```
#define rand3b(x,y,z) frand(73*(x)+79*(y)+83*(z))
```

```
#define rand3c(x,y,z) frand(89*(x)+97*(y)+101*(z))
```

```
#define rand3d(x,y,z) frand(103*(x)+107*(y)+109*(z))
```

```
/* hermite */
```

```
#define hpoly1(t) ((2.0*t-3.0)*t*t+1.0)
```

```
#define hpoly2(t) (-2.0*t+3.0)*t*t
```

```
#define hpoly3(t) ((t-2.0)*t+1.0)*t
```

```
#define hpoly4(t) (t-1.0)*t*t
```

```
double *noise3(), fnoise3(), frand();
```

```
static interpolate();
```

```
static long xlim[3][2];
```

```
static double xarg[3];
```

```
#define EPSILON .0001 /* error allowed in fractal */
```

```
#define frand3(x,y,z) frand(17*(x)+23*(y)+29*(z))
```

```
double *
```

```
noise3(xnew) /* compute the noise function */
```

```
register double xnew[3];
```

```
{
```

```
    extern double floor();
```

```
    static double x[3] = {-100000.0, -100000.0, -100000.0};
```

```
    static double f[4];
```

```
    if (x[0]==xnew[0] && x[1]==xnew[1] && x[2]==xnew[2])
```

```
        return(f);
```

```
    x[0] = xnew[0]; x[1] = xnew[1]; x[2] = xnew[2];
```

```
    xlim[0][0] = floor(x[0]); xlim[0][1] = xlim[0][0] + 1;
```

```
    xlim[1][0] = floor(x[1]); xlim[1][1] = xlim[1][0] + 1;
```

```
    xlim[2][0] = floor(x[2]); xlim[2][1] = xlim[2][0] + 1;
```

```
    xarg[0] = x[0] - xlim[0][0];
```

```
    xarg[1] = x[1] - xlim[1][0];
```

```
xarg[2] = x[2] - xlim[2][0];
interpolate(f, 0, 3);
return(f);
```

```
}
```

```
static
interpolate(f, i, n)
double f[4];
register int i, n;
{
    double f0[4], f1[4], hp1, hp2;

    if (n == 0) {
        f[A] = rand3a(xlim[0][i&1],xlim[1][i>>1&1],xlim[2][i>>2]);
        f[B] = rand3b(xlim[0][i&1],xlim[1][i>>1&1],xlim[2][i>>2]);
        f[C] = rand3c(xlim[0][i&1],xlim[1][i>>1&1],xlim[2][i>>2]);
        f[D] = rand3d(xlim[0][i&1],xlim[1][i>>1&1],xlim[2][i>>2]);
    } else {
        n--;
        interpolate(f0, i, n);
        interpolate(f1, i | 1<<n, n);
        hp1 = hpoly1(xarg[n]); hp2 = hpoly2(xarg[n]);
        f[A] = f0[A]*hp1 + f1[A]*hp2;
        f[B] = f0[B]*hp1 + f1[B]*hp2;
        f[C] = f0[C]*hp1 + f1[C]*hp2;
        f[D] = f0[D]*hp1 + f1[D]*hp2 +
                f0[n]*hpoly3(xarg[n]) + f1[n]*hpoly4(xarg[n]);
    }
}
```

```
double
frand(s)                                /* get random number from seed */
register long s;
{
    s = s<<13 ^ s;
    return(1.0-((s*(s*s*15731+789221)+1376312589)&0x7fffffff)/1073741824.0);
}
```




```
double
fnoise3(p)                              /* compute fractal noise function */
double p[3];
{
    double floor();
    long t[3], v[3], beg[3];
    double fval[8], fc;
    int branch;
    register long s;
    register int i, j;

    /* get starting cube */
    s = (long)(1.0/EPSILON);
    for (i = 0; i < 3; i++) {
        t[i] = s*p[i];
        beg[i] = s*floor(p[i]);
    }
    for (j = 0; j < 8; j++) {
        for (i = 0; i < 3; i++) {
            v[i] = beg[i];
            if (j & 1<<i)
```

```
        v[i] += s;
    }
    fval[j] = frand3(v[0],v[1],v[2]);
}

/* compute fractal */
for ( ; ; ) {
    fc = 0.0;
    for (j = 0; j < 8; j++)
        fc += fval[j];
    fc *= 0.125;
    if ((s >= 1) == 0)
        return(fc); /* close enough */
    branch = 0;
    for (i = 0; i < 3; i++) { /* do center */
        v[i] = beg[i] + s;
        if (t[i] > v[i]) {
            branch |= 1<<i;
        }
    }
    fc += s*EPSILON*frand3(v[0],v[1],v[2]);
    fval[~branch & 7] = fc;
    for (i = 0; i < 3; i++) { /* do faces */
        if (branch & 1<<i)
            v[i] += s;
        else
            v[i] -= s;
        fc = 0.0;
        for (j = 0; j < 8; j++)
            if (~(j^branch) & 1<<i)
                fc += fval[j];
        fc = 0.25*fc + s*EPSILON*frand3(v[0],v[1],v[2]);
        fval[~(branch^1<<i) & 7] = fc;
        v[i] = beg[i] + s;
    }
    for (i = 0; i < 3; i++) { /* do edges */
        j = (i+1)%3;
        if (branch & 1<<j)
            v[j] += s;
        else
            v[j] -= s;
        j = (i+2)%3;
        if (branch & 1<<j)
            v[j] += s;
        else
            v[j] -= s;
        fc = fval[branch & ~(1<<i)];
        fc += fval[branch | 1<<i];
        fc = 0.5*fc + s*EPSILON*frand3(v[0],v[1],v[2]);
        fval[branch^1<<i] = fc;
        j = (i+1)%3;
        v[j] = beg[j] + s;
        j = (i+2)%3;
        v[j] = beg[j] + s;
    }
    for (i = 0; i < 3; i++) /* new cube */
        if (branch & 1<<i)
            beg[i] += s;
}
}
```

# Index of /pubs/tog/GraphicsGems/gemsiv/ptpoly\_weiler/

Name	Last modified	Size	Description
 <a href="#">_Parent Directory</a>			
 <a href="#">_polygon.h</a>	29-Jun-00 08:20	1K	
 <a href="#">_pt_poly.c</a>	29-Jun-00 08:20	2K	



```
#include <stdio.h>

/* polygon vertex definition */
typedef struct vertex_struct {
    double x,y; /* coordinate values */
    struct vertex_struct *next; /* circular singly linked list from poly */
} vtx, *vtx_ptr;

/* polygon definition */
typedef struct polygon_struct {
    vtx_ptr last; /* pointer to end of circular vertex list */
} polygon, *polygon_ptr;

/* return next vertex in polygon list of vertices */
#define polygon_get_vertex(poly, vertex) \
    ((vertex == NULL) ? poly->last->next : vertex->next)

/* create and insert new polygon vertex */
#define polygon_new_vertex(vertex, poly, xx, yy, new_vertex) { \
    new_vertex = (vtx_ptr) malloc (sizeof(vtx)); \
    new_vertex->x = xx; \
    new_vertex->y = yy; \
    if (poly->last != NULL) { \
        new_vertex->next = poly->last->next; \
        poly->last->next = new_vertex; \
    } \
    else { \
        new_vertex->next = new_vertex; \
    } \
    poly->last = new_vertex; \
}

/* create new polygon */
#define polygon_create(poly) \
    poly = (polygon_ptr) malloc(sizeof (polygon)); \
    poly->last = NULL;
```

```
/*
 * C code from the article
 * "An Incremental Angle Point in Polygon Test"
 * by Kevin Weiler, kjw@autodesk.com
 * in "Graphics Gems IV", Academic Press, 1994
 */

#include "polygon.h"

/* quadrant id's, incremental angles, accumulated angle values */
typedef short quadrant_type;

/* result value from point in polygon test */
typedef enum pt_poly_relation {INSIDE, OUTSIDE} pt_poly_relation;

/* determine the quadrant of a polygon point
   relative to the test point */
#define quadrant(vertex, x, y) \
    ( (vertex->x > x) ? ((vertex->y > y) ? 0 : 3) : ( (vertex->y > y) ? 1 : 2) )

/* determine x intercept of a polygon edge
   with a horizontal line at the y value of the test point */
#define x_intercept(pt1, pt2, yy) \
    (pt2->x - ( (pt2->y - yy) * ((pt1->x - pt2->x) / (pt1->y - pt2->y)) ) )

/* adjust delta */
#define adjust_delta(delta, vertex, next_vertex, xx, yy) \
    switch (delta) { \
        /* make quadrant deltas wrap around */ \
        case 3: delta = -1; break; \
        case -3: delta = 1; break; \
        /* check if went around point cw or ccw */ \
        case 2: case -2: if (x_intercept(vertex, next_vertex, yy) > xx) \
            delta = - (delta); \
        break; \
    }

/* determine if a test point is inside of or outside of a polygon */
/* polygon is "poly", test point is at "x","y" */
pt_poly_relation
point_in_poly(polygon_ptr poly, double x, double y)
{
    vtx_ptr vertex, first_vertex, next_vertex;
    quadrant_type quad, next_quad, delta, angle;

    /* initialize */
    vertex = NULL; /* because polygon_get_vertex is a macro */
    vertex = first_vertex = polygon_get_vertex(poly, vertex);
    quad = quadrant(vertex, x, y);
    angle = 0;

    /* loop on all vertices of polygon */
    do {
        next_vertex = polygon_get_vertex(poly, vertex);
        /* calculate quadrant and delta from last quadrant */
        next_quad = quadrant(next_vertex, x, y);
        delta = next_quad - quad;
        adjust_delta(delta, vertex, next_vertex, x, y);
        /* add delta to total angle sum */
        angle = angle + delta;
        /* increment for next step */
        quad = next_quad;
    }
```

```
vertex = next_vertex;
} while (vertex != first_vertex);

/* complete 360 degrees (angle of + 4 or -4 ) means inside */
if ((angle == +4) || (angle == -4)) return INSIDE; else return OUTSIDE;

/* odd number of windings rule */
/* if (angle & 4) return INSIDE; else return OUTSIDE; */
/* non-zero winding rule */
/* if (angle != 0) return INSIDE; else return OUTSIDE; */
}
```

```
/*
Fast Ray-Box Intersection
by Andrew Woo
from "Graphics Gems", Academic Press, 1990
*/

#include "GraphicsGems.h"

#define NUMDIM 3
#define RIGHT 0
#define LEFT 1
#define MIDDLE 2

char HitBoundingBox(minB,maxB, origin, dir,coord)
double minB[NUMDIM], maxB[NUMDIM];          /*box */
double origin[NUMDIM], dir[NUMDIM];          /*ray */
double coord[NUMDIM];                        /* hit point */
{
    char inside = TRUE;
    char quadrant[NUMDIM];
    register int i;
    int whichPlane;
    double maxT[NUMDIM];
    double candidatePlane[NUMDIM];

    /* Find candidate planes; this loop can be avoided if
    rays cast all from the eye(assume perpsective view) */
    for (i=0; i<NUMDIM; i++)
        if(origin[i] < minB[i]) {
            quadrant[i] = LEFT;
            candidatePlane[i] = minB[i];
            inside = FALSE;
        }else if (origin[i] > maxB[i]) {
            quadrant[i] = RIGHT;
            candidatePlane[i] = maxB[i];
            inside = FALSE;
        }else {
            quadrant[i] = MIDDLE;
        }

    /* Ray origin inside bounding box */
    if(inside) {
        coord = origin;
        return (TRUE);
    }

    /* Calculate T distances to candidate planes */
    for (i = 0; i < NUMDIM; i++)
        if (quadrant[i] != MIDDLE && dir[i] !=0.)
            maxT[i] = (candidatePlane[i]-origin[i]) / dir[i];
        else
            maxT[i] = -1.;

    /* Get largest of the maxT's for final choice of intersection */
    whichPlane = 0;
    for (i = 1; i < NUMDIM; i++)
        if (maxT[whichPlane] < maxT[i])
            whichPlane = i;

    /* Check final candidate actually inside box */
}
```

```
    if (maxT[whichPlane] < 0.) return (FALSE);
    for (i = 0; i < NUMDIM; i++)
        if (whichPlane != i) {
            coord[i] = origin[i] + maxT[whichPlane] *dir[i];
            if (coord[i] < minB[i] || coord[i] > maxB[i])
                return (FALSE);
        } else {
            coord[i] = candidatePlane[i];
        }
    return (TRUE);
}                                     /* ray hits box */
```

```
/*
 *      Name (person to blame): Andrew Woo
 *      Gem: Article 7.1, "The Shadow Depth Map Revisited"
 *      Affiliation: Alias Research, Style! Division
 */
#include      <math.h>

/*****
Reeves' SampleShadow code using BIAS
*****/

#define REEVES_APPROACH {
\
    inshadow = 0;
\
    for (i = 0, s = smin; i < ns; i++, s += ds) {
\
        for (j = 0, t = tmin; j < nt; j++, t += dt) {
\
            iu = s + Rand()*js;                                /* jitter s, t */
\
            iv = t + Rand()*jt;
\
\
            bias = Rand() * (Bias1 - Bias0) + Bias0; /* pick bias */
\
            if (iu >= bbox->r_umin && iu <= bbox->r_umax &&
\
                iv >= bbox->r_vmin && iv <= bbox->r_vmax)
\
                if (z > depthMap[iu][iv] + bias) inshadow++;
\
        }
\
    }
\
}

/*****
New SampleShadow code with no BIAS and doing
integer comparisons for depth values first.
*****/

#define NEW_APPROACH {
\
    register int integerZ = (int) z, intDepth;
\
    register float depthValue;
\
    inshadow = 0;
\
\
    for (i = 0, s = smin; i < ns; i++, s += ds) {
\
        for (j = 0, t = tmin; j < nt; j++, t += dt) {
\
```

```

        iu = s + Rand()*js;
\
        iv = t + Rand()*jt;
\
        if (iu >= bbox->r_umin && iu <= bbox->r_umax &&
\
            iv >= bbox->r_vmin && iv <= bbox->r_vmax) {
\
            /* do integer comparison first */
\
            depthValue = depthMap[iu][iv];
\
            intDepth = (int) depthValue;
\
            if ((integerZ > intDepth) ||
\
                (integerZ == intDepth && z > depthValue))
\
                inshadow++;
\
        } else {
\
            /* boundary case error */
\
            inshadow++;
\
        }
\
    }
\
}

/* Start of Reeves' code in Siggraph 87 paper */

float ResFactor = 3;
float MinSize = 0;
float Bias0 = 0.3;
float Bias1 = 0.4;
int NumSamples = 16;
int MinSamples = 1;

#define MAPRES                256                /* I tested the code at 256x256
*/
float depthMap[MAPRES][MAPRES];

#define CLAMP(a,min,max)      (a<min?min:(a>max?max:a))
float Rand();

typedef struct {
    int r_umin, r_umax;
    int r_vmin, r_vmax;
} TextureRect;

float SampleShadow (s, t, z, sres, tres, bbox)
float s, t, z, sres, tres;
TextureRect *bbox;
```

```
{
    int i, j, inshadow, iu, iv, ns, nt, lu, hu, lv, hv;
    float smin, tmin, ds, dt, js, jt;

    /* convert to coordinates of depth map */
    sres = MAPRES * sres * ResFactor;
    tres = MAPRES * tres * ResFactor;
    if (sres < MinSize) sres = MinSize;
    if (tres < MinSize) tres = MinSize;
    s *= MAPRES; t *= MAPRES;

    /* cull if outside bounding box */
    lu = floor (s-sres); hu = ceil (s+sres);
    lv = floor (t-tres); hv = ceil (t+tres);
    if (lu > bbox->r_umax || hu < bbox->r_umin ||
        lv > bbox->r_vmax || hv < bbox->r_vmin)
        return (1.0); /* error in Reeves' code at boundary cases */

    /* calculate number of samples */
    if (sres*tres*4 < NumSamples) {
        ns = sres + sres + 0.5;
        ns = CLAMP(ns, MinSamples, NumSamples);
        nt = tres + tres + 0.5;
        nt = CLAMP(nt, MinSamples, NumSamples);
    }
    else {
        nt = sqrt(tres*NumSamples/sres) + 0.5;
        nt = CLAMP(nt, MinSamples, NumSamples);
        ns = ((float)NumSamples)/nt + 0.5;
        ns = CLAMP(ns, MinSamples, NumSamples);
    }

    /* setup jitter variables */
    ds = 2*sres/ns; dt = 2*tres/nt;
    js = ds*.5; jt = dt*.5;
    smin = s - sres + js; tmin = t - tres + jt;

    /* decide which version you want... */
#ifdef OLD_WAY
    REEVES_APPROACH;
#else
    NEW_APPROACH;
#endif

    return (((float) inshadow) / (ns*nt));
}
```



```
#include "GraphicsGems.h"
#include <stdio.h>

/****
 *
 * affine_matrix4_inverse
 *
 * Computes the inverse of a 3D affine matrix; i.e. a matrix with a dimen-
 * sionality of 4 where the right column has the entries (0, 0, 0, 1).
 *
 * This procedure treats the 4 by 4 matrix as a block matrix and
 * calculates the inverse of one submatrix for a significant perform-
 * ance improvement over a general procedure that can invert any non-
 * singular matrix:
 *
 *      --      --      --      --
 *      |      |      |      |
 *      |  A    |  0  |  -1   |  -1   |
 *      |      |      |      |  A    |  0   |
 *      |  C    |  1  |      |  -C  A   |  1   |
 *      |      |      |      |      |      |
 *      --      --      --      --
 *
 *      M-1 =
 *
 * where      M is a 4 by 4 matrix,
 *            A is the 3 by 3 upper left submatrix of M,
 *            C is the 1 by 3 lower left submatrix of M.
 *
 * Input:
 *   in   - 3D affine matrix
 *
 * Output:
 *   out  - inverse of 3D affine matrix
 *
 * Returned value:
 *   TRUE  if input matrix is nonsingular
 *   FALSE otherwise
 *
 ****/
```

```
boolean
affine_matrix4_inverse (in, out)
    register Matrix4 *in;
    register Matrix4 *out;
{
    register double det_1;
    register double pos, neg, temp;

#define ACCUMULATE \
    if (temp >= 0.0) \
        pos += temp; \
    else \
        neg += temp;

#define PRECISION_LIMIT (1.0e-15)

    /*
     * Calculate the determinant of submatrix A and determine if the
     * the matrix is singular as limited by the double precision
     * floating-point data representation.
     */
    pos = neg = 0.0;
```

```
temp = in->element[0][0] * in->element[1][1] * in->element[2][2];
ACCUMULATE
temp = in->element[0][1] * in->element[1][2] * in->element[2][0];
ACCUMULATE
temp = in->element[0][2] * in->element[1][0] * in->element[2][1];
ACCUMULATE
temp = -in->element[0][2] * in->element[1][1] * in->element[2][0];
ACCUMULATE
temp = -in->element[0][1] * in->element[1][0] * in->element[2][2];
ACCUMULATE
temp = -in->element[0][0] * in->element[1][2] * in->element[2][1];
ACCUMULATE
det_1 = pos + neg;

/* Is the submatrix A singular? */
if ((det_1 == 0.0) || (ABS(det_1 / (pos - neg)) < PRECISION_LIMIT)) {

    /* Matrix M has no inverse */
    fprintf (stderr, "affine_matrix4_inverse: singular matrix\n");
    return FALSE;
}

else {

    /* Calculate inverse(A) = adj(A) / det(A) */
    det_1 = 1.0 / det_1;
    out->element[0][0] = ( in->element[1][1] * in->element[2][2] -
                          in->element[1][2] * in->element[2][1] )
                        * det_1;
    out->element[1][0] = - ( in->element[1][0] * in->element[2][2] -
                           in->element[1][2] * in->element[2][0] )
                        * det_1;
    out->element[2][0] = ( in->element[1][0] * in->element[2][1] -
                          in->element[1][1] * in->element[2][0] )
                        * det_1;
    out->element[0][1] = - ( in->element[0][1] * in->element[2][2] -
                           in->element[0][2] * in->element[2][1] )
                        * det_1;
    out->element[1][1] = ( in->element[0][0] * in->element[2][2] -
                          in->element[0][2] * in->element[2][0] )
                        * det_1;
    out->element[2][1] = - ( in->element[0][0] * in->element[2][1] -
                           in->element[0][1] * in->element[2][0] )
                        * det_1;
    out->element[0][2] = ( in->element[0][1] * in->element[1][2] -
                          in->element[0][2] * in->element[1][1] )
                        * det_1;
    out->element[1][2] = - ( in->element[0][0] * in->element[1][2] -
                           in->element[0][2] * in->element[1][0] )
                        * det_1;
    out->element[2][2] = ( in->element[0][0] * in->element[1][1] -
                          in->element[0][1] * in->element[1][0] )
                        * det_1;

    /* Calculate -C * inverse(A) */
    out->element[3][0] = - ( in->element[3][0] * out->element[0][0] +
                           in->element[3][1] * out->element[1][0] +
                           in->element[3][2] * out->element[2][0] );
    out->element[3][1] = - ( in->element[3][0] * out->element[0][1] +
                           in->element[3][1] * out->element[1][1] +
                           in->element[3][2] * out->element[2][1] );
```

```
out->element[3][2] = - ( in->element[3][0] * out->element[0][2] +
                        in->element[3][1] * out->element[1][2] +
                        in->element[3][2] * out->element[2][2] );

/* Fill in last column */
out->element[0][3] = out->element[1][3] = out->element[2][3] = 0.0;
out->element[3][3] = 1.0;

return TRUE;
```

```
    }
}
```

```

/*
 * ANSI C code from the article
 * "Fast Inversion of Length- and Angle-Preserving Matrices"
 * by Kevin Wu, Kevin.Wu@eng.sun.com
 * in "Graphics Gems IV", Academic Press, 1994
 *
 * compile with "cc -DMAIN ..." to create a test program
 */

#include "GraphicsGems.h"
#include <stdio.h>
/****
 *
 * angle_preserving_matrix4_inverse
 *
 * Computes the inverse of a 3-D angle-preserving matrix.
 *
 * This procedure treats the 4 by 4 angle-preserving matrix as a block
 * matrix and calculates the inverse of one submatrix for a significant
 * performance improvement over a general procedure that can invert any
 * nonsingular matrix:
 *
 *      -1      -1      -1      -1
 *      M      =  [ A      C ]  =  [ -2 T      -2 T      ]
 *                  [ 0      1 ]    [ s  A      - s  A  C ]
 *
 * where      M is a 4 by 4 angle-preserving matrix,
 *             A is the 3 by 3 upper-left submatrix of M,
 *             C is the 3 by 1 upper-right submatrix of M.
 *
 * Input:
 *   in   - 3-D angle-preserving matrix
 *
 * Output:
 *   out  - inverse of 3-D angle-preserving matrix
 *
 * Returned value:
 *   TRUE   if input matrix is nonsingular
 *   FALSE  otherwise
 *
 ****/

boolean
angle_preserving_matrix4_inverse (Matrix4 *in, Matrix4 *out)
{
    double  scale;

    /* Calculate the square of the isotropic scale factor */
    scale = in->element[0][0] * in->element[0][0] +
            in->element[0][1] * in->element[0][1] +
            in->element[0][2] * in->element[0][2];

    /* Is the submatrix A singular? */
    if (scale == 0.0) {

        /* Matrix M has no inverse */
        fprintf (stderr, "angle_preserving_matrix4_inverse: singular matrix\n");
        return FALSE;
    }
}

```

```
}

/* Calculate the inverse of the square of the isotropic scale factor */
scale = 1.0 / scale;

/* Transpose and scale the 3 by 3 upper-left submatrix */
out->element[0][0] = scale * in->element[0][0];
out->element[1][0] = scale * in->element[0][1];
out->element[2][0] = scale * in->element[0][2];
out->element[0][1] = scale * in->element[1][0];
out->element[1][1] = scale * in->element[1][1];
out->element[2][1] = scale * in->element[1][2];
out->element[0][2] = scale * in->element[2][0];
out->element[1][2] = scale * in->element[2][1];
out->element[2][2] = scale * in->element[2][2];

/* Calculate -(transpose(A) / s*s) C */
out->element[0][3] = - ( out->element[0][0] * in->element[0][3] +
                        out->element[0][1] * in->element[1][3] +
                        out->element[0][2] * in->element[2][3] );
out->element[1][3] = - ( out->element[1][0] * in->element[0][3] +
                        out->element[1][1] * in->element[1][3] +
                        out->element[1][2] * in->element[2][3] );
out->element[2][3] = - ( out->element[2][0] * in->element[0][3] +
                        out->element[2][1] * in->element[1][3] +
                        out->element[2][2] * in->element[2][3] );

/* Fill in last row */
out->element[3][0] = out->element[3][1] = out->element[3][2] = 0.0;
out->element[3][3] = 1.0;

return TRUE;
}

#ifdef MAIN      /* test program for inverter */

/*
 * Angle preserving matrix:
 * M = S(-3.67, 1.85, 9.52) T(6.93, 6.93, 6.93) Ry(0.19) Rz(-1.32) Rx(0.87)
 * where the angles are in radians.
 */
static double  m[4][4] =
    {{ 1.6889057579031668e+00,  5.2512935661266260e+00,
      -4.1948078887213214e+00, -3.6699999999999999e+00 },
     {-6.7131956438195779e+00,  1.1090087288191814e+00,
      -1.3145356165599698e+00,  1.8500000000000001e+00 },
     {-3.2481008100637232e-01,  4.3839383574315880e+00,
       5.3572831630889803e+00,  9.5199999999999996e+00 },
     { 0.0000000000000000e+00,  0.0000000000000000e+00,
       0.0000000000000000e+00,  1.0000000000000000e+00 }};

main()
{
    Matrix4      in, out, prod;
    int          i, j, k;

    for (i = 0; i < 4; i++) {
        for (j = 0; j < 4; j++) {
            in.element[i][j] = m[i][j];
        }
    }
}
```

```
printf ("Original matrix:\n");
printf ("%13.6e %13.6e %13.6e %13.6e\n", in.element[0][0],
        in.element[0][1], in.element[0][2], in.element[0][3]);
printf ("%13.6e %13.6e %13.6e %13.6e\n", in.element[1][0],
        in.element[1][1], in.element[1][2], in.element[1][3]);
printf ("%13.6e %13.6e %13.6e %13.6e\n", in.element[2][0],
        in.element[2][1], in.element[2][2], in.element[2][3]);
printf ("%13.6e %13.6e %13.6e %13.6e\n", in.element[3][0],
        in.element[3][1], in.element[3][2], in.element[3][3]);

/* Calculate inverse with utility */
angle_preserving_matrix4_inverse(&in, &out);

printf ("\nCalculated inverse matrix:\n");
printf ("%13.6e %13.6e %13.6e %13.6e\n", out.element[0][0],
        out.element[0][1], out.element[0][2], out.element[0][3]);
printf ("%13.6e %13.6e %13.6e %13.6e\n", out.element[1][0],
        out.element[1][1], out.element[1][2], out.element[1][3]);
printf ("%13.6e %13.6e %13.6e %13.6e\n", out.element[2][0],
        out.element[2][1], out.element[2][2], out.element[2][3]);
printf ("%13.6e %13.6e %13.6e %13.6e\n", out.element[3][0],
        out.element[3][1], out.element[3][2], out.element[3][3]);

/*
 * Calculate the product of the original matrix and calculated inverse.
 * The product should be the identity if the utility is correct.
 */

for (i = 0; i < 4; i++) {
    for (j = 0; j < 4; j++) {
        prod.element[i][j] = 0.0;
    }
}

for (i = 0; i < 4; i++) {
    for (j = 0; j < 4; j++) {
        for (k = 0; k < 4; k++) {
            prod.element[i][j] += in.element[i][k] * out.element[k][j];
        }
    }
}

printf ("\nProduct of original matrix and calculated inverse:\n");
printf ("%13.6e %13.6e %13.6e %13.6e\n", prod.element[0][0],
        prod.element[0][1], prod.element[0][2], prod.element[0][3]);
printf ("%13.6e %13.6e %13.6e %13.6e\n", prod.element[1][0],
        prod.element[1][1], prod.element[1][2], prod.element[1][3]);
printf ("%13.6e %13.6e %13.6e %13.6e\n", prod.element[2][0],
        prod.element[2][1], prod.element[2][2], prod.element[2][3]);
printf ("%13.6e %13.6e %13.6e %13.6e\n", prod.element[3][0],
        prod.element[3][1], prod.element[3][2], prod.element[3][3]);
}

#endif
```

```

/*****
    C Implementation of Wu's Color Quantizer (v. 2)
    (see Graphics Gems vol. II, pp. 126-133)
*****/
```

Author: Xiaolin Wu  
Dept. of Computer Science  
Univ. of Western Ontario  
London, Ontario N6A 5B7  
wu@csd.uwo.ca

Algorithm: Greedy orthogonal bipartition of RGB space for variance minimization aided by inclusion-exclusion tricks.  
For speed no nearest neighbor search is done. Slightly better performance can be expected by more sophisticated but more expensive versions.

The author thanks Tom Lane at Tom\_Lane@G.GP.CS.CMU.EDU for much of additional documentation and a cure to a previous bug.

Free to distribute, comments and suggestions are appreciated.

```
*****/
```

```
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <math.h>

#define MAXCOLOR      256
#define RED           2
#define GREEN         1
#define BLUE          0

struct box {
    int r0;                /* min value, exclusive */
    int r1;                /* max value, inclusive */
    int g0;
    int g1;
    int b0;
    int b1;
    int vol;
};

/* Histogram is in elements 1..HISTSZ along each axis,
 * element 0 is for base or marginal value
 * NB: these must start out 0!
 */

float      gm2[33][33][33];
long int   wt[33][33][33], mr[33][33][33], mg[33][33][33], mb[33][33][33];
unsigned char *Ir, *Ig, *Ib;
int        size; /*image size*/
int        K;    /*color look-up table size*/
unsigned short int *Qadd;

void
Hist3d(vwt, vmr, vmg, vmb, m2)
/* build 3-D color histogram of counts, r/g/b, c^2 */
long int *vwt, *vmr, *vmg, *vmb;
float *m2;
{
    register int ind, r, g, b;
```

```
int      inr, ing, inb, table[256];
register long int i;

    for(i=0; i<256; ++i) table[i]=i*i;
    Qadd = (unsigned short int *)malloc(sizeof(short int)*size);
    if (Qadd==NULL) {printf("Not enough space\n"); exit(1);}
    for(i=0; i<size; ++i){
        r = Ir[i]; g = Ig[i]; b = Ib[i];
        inr=(r>>3)+1;
        ing=(g>>3)+1;
        inb=(b>>3)+1;
        Qadd[i]=ind=(inr<<10)+(inr<<6)+inr+(ing<<5)+ing+inb;
        /*[inr][ing][inb]*/
        ++vwt[ind];
        vmr[ind] += r;
        vmg[ind] += g;
        vmb[ind] += b;
        m2[ind] += (float)(table[r]+table[g]+table[b]);
    }
}

/* At conclusion of the histogram step, we can interpret
 *   wt[r][g][b] = sum over voxel of P(c)
 *   mr[r][g][b] = sum over voxel of r*P(c) ,   similarly for mg, mb
 *   m2[r][g][b] = sum over voxel of c^2*P(c)
 * Actually each of these should be divided by 'size' to give the usual
 * interpretation of P() as ranging from 0 to 1, but we needn't do that here.
 */

/* We now convert histogram into moments so that we can rapidly calculate
 * the sums of the above quantities over any desired box.
 */

void
M3d(vwt, vmr, vmg, vmb, m2) /* compute cumulative moments. */
long int *vwt, *vmr, *vmg, *vmb;
float *m2;
{
    register unsigned short int ind1, ind2;
    register unsigned char i, r, g, b;
    long int line, line_r, line_g, line_b,
        area[33], area_r[33], area_g[33], area_b[33];
    float line2, area2[33];

    for(r=1; r<=32; ++r){
        for(i=0; i<=32; ++i)
            area2[i]=area[i]=area_r[i]=area_g[i]=area_b[i]=0.0f;
        for(g=1; g<=32; ++g){
            line2 = line = line_r = line_g = line_b = 0.0f;
            for(b=1; b<=32; ++b){
                ind1 = (r<<10) + (r<<6) + r + (g<<5) + g + b; /* [r][g][b] */
                line += vwt[ind1];
                line_r += vmr[ind1];
                line_g += vmg[ind1];
                line_b += vmb[ind1];
                line2 += m2[ind1];
                area[b] += line;
                area_r[b] += line_r;
                area_g[b] += line_g;
                area_b[b] += line_b;
            }
        }
    }
}
```



```
        area2[b] += line2;
        ind2 = ind1 - 1089; /* [r-1][g][b] */
        vwt[ind1] = vwt[ind2] + area[b];
        vmr[ind1] = vmr[ind2] + area_r[b];
        vmg[ind1] = vmg[ind2] + area_g[b];
        vmb[ind1] = vmb[ind2] + area_b[b];
        m2[ind1] = m2[ind2] + area2[b];
    }
}
}
```

```
long int Vol(cube, mmt)
/* Compute sum over a box of any given statistic */
struct box *cube;
long int mmt[33][33][33];
{
```

```
    return( mmt[cube->r1][cube->g1][cube->b1]
        -mmt[cube->r1][cube->g1][cube->b0]
        -mmt[cube->r1][cube->g0][cube->b1]
        +mmt[cube->r1][cube->g0][cube->b0]
        -mmt[cube->r0][cube->g1][cube->b1]
        +mmt[cube->r0][cube->g1][cube->b0]
        +mmt[cube->r0][cube->g0][cube->b1]
        -mmt[cube->r0][cube->g0][cube->b0] );
}
```

```
/* The next two routines allow a slightly more efficient calculation
 * of Vol() for a proposed subbox of a given box. The sum of Top()
 * and Bottom() is the Vol() of a subbox split in the given direction
 * and with the specified new upper bound.
 */
```

```
long int Bottom(cube, dir, mmt)
/* Compute part of Vol(cube, mmt) that doesn't depend on r1, g1, or b1 */
/* (depending on dir) */
struct box *cube;
unsigned char dir;
long int mmt[33][33][33];
{
    switch(dir){
        case RED:
            return( -mmt[cube->r0][cube->g1][cube->b1]
                +mmt[cube->r0][cube->g1][cube->b0]
                +mmt[cube->r0][cube->g0][cube->b1]
                -mmt[cube->r0][cube->g0][cube->b0] );
            break;
        case GREEN:
            return( -mmt[cube->r1][cube->g0][cube->b1]
                +mmt[cube->r1][cube->g0][cube->b0]
                +mmt[cube->r0][cube->g0][cube->b1]
                -mmt[cube->r0][cube->g0][cube->b0] );
            break;
        case BLUE:
            return( -mmt[cube->r1][cube->g1][cube->b0]
                +mmt[cube->r1][cube->g0][cube->b0]
                +mmt[cube->r0][cube->g1][cube->b0]
                -mmt[cube->r0][cube->g0][cube->b0] );
            break;
    }
}
```

```
}
```

```
long int Top(cube, dir, pos, mmt)
/* Compute remainder of Vol(cube, mmt), substituting pos for */
/* r1, g1, or b1 (depending on dir) */
struct box *cube;
unsigned char dir;
int pos;
long int mmt[33][33][33];
{
    switch(dir){
        case RED:
            return( mmt[pos][cube->g1][cube->b1]
                    -mmt[pos][cube->g1][cube->b0]
                    -mmt[pos][cube->g0][cube->b1]
                    +mmt[pos][cube->g0][cube->b0] );
            break;
        case GREEN:
            return( mmt[cube->r1][pos][cube->b1]
                    -mmt[cube->r1][pos][cube->b0]
                    -mmt[cube->r0][pos][cube->b1]
                    +mmt[cube->r0][pos][cube->b0] );
            break;
        case BLUE:
            return( mmt[cube->r1][cube->g1][pos]
                    -mmt[cube->r1][cube->g0][pos]
                    -mmt[cube->r0][cube->g1][pos]
                    +mmt[cube->r0][cube->g0][pos] );
            break;
    }
}
```

```
float Var(cube)
/* Compute the weighted variance of a box */
/* NB: as with the raw statistics, this is really the variance * size */
struct box *cube;
{
    float dr, dg, db, xx;
    float result;

    dr = (float)Vol(cube, mr);
    dg = (float)Vol(cube, mg);
    db = (float)Vol(cube, mb);
    xx = gm2[cube->r1][cube->g1][cube->b1]
        -gm2[cube->r1][cube->g1][cube->b0]
        -gm2[cube->r1][cube->g0][cube->b1]
        +gm2[cube->r1][cube->g0][cube->b0]
        -gm2[cube->r0][cube->g1][cube->b1]
        +gm2[cube->r0][cube->g1][cube->b0]
        +gm2[cube->r0][cube->g0][cube->b1]
        -gm2[cube->r0][cube->g0][cube->b0];

    result = xx - (dr*dr+dg*dg+db*db)/(float)Vol(cube,wt);
    return (float)fabs((float)result);
}
```

```
/* We want to minimize the sum of the variances of two subboxes.
 * The sum(c^2) terms can be ignored since their sum over both subboxes
 * is the same (the sum for the whole box) no matter where we split.
```

```
* The remaining terms have a minus sign in the variance formula,  
* so we drop the minus sign and MAXIMIZE the sum of the two terms.  
*/
```

```
float Maximize(cube, dir, first, last, cut,  
               whole_r, whole_g, whole_b, whole_w)  
struct box *cube;  
unsigned char dir;  
int first, last, *cut;  
long int whole_r, whole_g, whole_b, whole_w;  
{  
    register long int half_r, half_g, half_b, half_w;  
    long int base_r, base_g, base_b, base_w;  
    register int i;  
    register float temp, max;  
  
    base_r = Bottom(cube, dir, mr);  
    base_g = Bottom(cube, dir, mg);  
    base_b = Bottom(cube, dir, mb);  
    base_w = Bottom(cube, dir, wt);  
    max = 0.0;  
    *cut = -1;  
    for(i=first; i<last; ++i){  
        half_r = base_r + Top(cube, dir, i, mr);  
        half_g = base_g + Top(cube, dir, i, mg);  
        half_b = base_b + Top(cube, dir, i, mb);  
        half_w = base_w + Top(cube, dir, i, wt);  
        /* now half_x is sum over lower half of box, if split at i */  
        if (half_w == 0) {          /* subbox could be empty of pixels! */  
            continue;              /* never split into an empty box */  
        } else  
            temp = ((float)half_r*half_r + (float)half_g*half_g +  
                   (float)half_b*half_b)/half_w;  
  
        half_r = whole_r - half_r;  
        half_g = whole_g - half_g;  
        half_b = whole_b - half_b;  
        half_w = whole_w - half_w;  
        if (half_w == 0) {          /* subbox could be empty of pixels! */  
            continue;              /* never split into an empty box */  
        } else  
            temp += ((float)half_r*half_r + (float)half_g*half_g +  
                   (float)half_b*half_b)/half_w;  
  
        if (temp > max) {max=temp; *cut=i;}  
    }  
    return(max);  
}  
  
int  
Cut(set1, set2)  
struct box *set1, *set2;  
{  
    unsigned char dir;  
    int cutr, cutg, cutb;  
    float maxr, maxg, maxb;  
    long int whole_r, whole_g, whole_b, whole_w;  
  
    whole_r = Vol(set1, mr);  
    whole_g = Vol(set1, mg);
```

```
whole_b = Vol(set1, mb);
whole_w = Vol(set1, wt);

maxr = Maximize(set1, RED, set1->r0+1, set1->r1, &cutr,
                whole_r, whole_g, whole_b, whole_w);
maxg = Maximize(set1, GREEN, set1->g0+1, set1->g1, &cutg,
                whole_r, whole_g, whole_b, whole_w);
maxb = Maximize(set1, BLUE, set1->b0+1, set1->b1, &cutb,
                whole_r, whole_g, whole_b, whole_w);

if( (maxr>=maxg)&&(maxr>=maxb) ) {
    dir = RED;
    if (cutr < 0) return 0; /* can't split the box */
}
else
if( (maxg>=maxr)&&(maxg>=maxb) )
    dir = GREEN;
else
    dir = BLUE;

set2->r1 = set1->r1;
set2->g1 = set1->g1;
set2->b1 = set1->b1;

switch (dir){
    case RED:
        set2->r0 = set1->r1 = cutr;
        set2->g0 = set1->g0;
        set2->b0 = set1->b0;
        break;
    case GREEN:
        set2->g0 = set1->g1 = cutg;
        set2->r0 = set1->r0;
        set2->b0 = set1->b0;
        break;
    case BLUE:
        set2->b0 = set1->b1 = cutb;
        set2->r0 = set1->r0;
        set2->g0 = set1->g0;
        break;
}
set1->vol=(set1->r1-set1->r0)*(set1->g1-set1->g0)*(set1->b1-set1->b0);
set2->vol=(set2->r1-set2->r0)*(set2->g1-set2->g0)*(set2->b1-set2->b0);
return 1;
}
```

Mark(cube, label, tag)

```
struct box *cube;
int label;
unsigned char *tag;
{
    register int r, g, b;

    for(r=cube->r0+1; r<=cube->r1; ++r)
        for(g=cube->g0+1; g<=cube->g1; ++g)
            for(b=cube->b0+1; b<=cube->b1; ++b)
                tag[(r<<10) + (r<<6) + r + (g<<5) + g + b] = label;
}
```

main()

```
{
struct box      cube[MAXCOLOR];
unsigned char    *tag;
unsigned char    lut_r[MAXCOLOR], lut_g[MAXCOLOR], lut_b[MAXCOLOR];
int             next;
register long int i, weight;
register int     k;
float           vv[MAXCOLOR], temp;

/* input R,G,B components into Ir, Ig, Ib;
   set size to width*height */

printf("no. of colors:\n");
scanf("%d", &K);

Hist3d(wt, mr, mg, mb, gm2); printf("Histogram done\n");
free(Ig); free(Ib); free(Ir);

M3d(wt, mr, mg, mb, gm2);   printf("Moments done\n");

cube[0].r0 = cube[0].g0 = cube[0].b0 = 0;
cube[0].r1 = cube[0].g1 = cube[0].b1 = 32;
next = 0;
for(i=1; i<K; ++i){
    if (Cut(&cube[next], &cube[i])) {
        /* volume test ensures we won't try to cut one-cell box */
        vv[next] = (cube[next].vol>1) ? Var(&cube[next]) : 0.0f;
        vv[i] = (cube[i].vol>1) ? Var(&cube[i]) : 0.0f;
    } else {
        vv[next] = 0.0;    /* don't try to split this box again */
        i--;              /* didn't create box i */
    }
    next = 0; temp = vv[0];
    for(k=1; k<=i; ++k)
        if (vv[k] > temp) {
            temp = vv[k]; next = k;
        }
    if (temp <= 0.0) {
        K = i+1;
        fprintf(stderr, "Only got %d boxes\n", K);
        break;
    }
}
printf("Partition done\n");

/* the space for array gm2 can be freed now */

tag = (unsigned char *)malloc(33*33*33);
if (tag==NULL) {printf("Not enough space\n"); exit(1);}
for(k=0; k<K; ++k){
    Mark(&cube[k], k, tag);
    weight = Vol(&cube[k], wt);
    if (weight) {
        lut_r[k] = (unsigned char)(Vol(&cube[k], mr) / weight);
        lut_g[k] = (unsigned char)(Vol(&cube[k], mg) / weight);
        lut_b[k] = (unsigned char)(Vol(&cube[k], mb) / weight);
    }
    else{
        fprintf(stderr, "bogus box %d\n", k);
        lut_r[k] = lut_g[k] = lut_b[k] = 0;
    }
}
```

```
    }  
  
    for(i=0; i<size; ++i) Qadd[i] = tag[Qadd[i]];  
  
    /* output lut_r, lut_g, lut_b as color look-up table contents,  
       Qadd as the quantized image (array of table addresses). */  
}
```

```
/*
Symmetric Double Step Line Algorithm
by Brian Wyvill
from "Graphics Gems", Academic Press, 1990

user provides "setpixel()" function for output.
*/

#define swap(a,b)          {a^=b; b^=a; a^=b;}
#define absolute(i,j,k)    ( (i-j)*(k = ( (i-j)<0 ? -1 : 1)))
int
symwuline(a1, b1, a2, b2) int a1, b1, a2, b2;
{
    int          dx, dy, incr1, incr2, D, x, y, xend, c, pixels_left;
    int          x1, y1;
    int          sign_x, sign_y, step, reverse, i;

    dx = absolute(a2, a1, sign_x);
    dy = absolute(b2, b1, sign_y);
    /* decide increment sign by the slope sign */
    if (sign_x == sign_y)
        step = 1;
    else
        step = -1;

    if (dy > dx) {                /* chooses axis of greatest movement (make
                                   * dx) */
        swap(a1, b1);
        swap(a2, b2);
        swap(dx, dy);
        reverse = 1;
    } else
        reverse = 0;
    /* note error check for dx==0 should be included here */
    if (a1 > a2) {                /* start from the smaller coordinate */
        x = a2;
        y = b2;
        x1 = a1;
        y1 = b1;
    } else {
        x = a1;
        y = b1;
        x1 = a2;
        y1 = b2;
    }

    /* Note dx=n implies 0 - n or (dx+1) pixels to be set */
    /* Go round loop dx/4 times then plot last 0,1,2 or 3 pixels */
    /* In fact (dx-1)/4 as 2 pixels are already plotted */
    xend = (dx - 1) / 4;
    pixels_left = (dx - 1) % 4;    /* number of pixels left over at the
                                   * end */

    plot(x, y, reverse);
    if ( pixels_left < 0 ) return ; /* plot only one pixel for zero
                                   * length vectors */
    plot(x1, y1, reverse); /* plot first two points */
    incr2 = 4 * dy - 2 * dx;
    if (incr2 < 0) {              /* slope less than 1/2 */
        c = 2 * dy;
        incr1 = 2 * c;
    }
}
```

```
D = incr1 - dx;

for (i = 0; i < xend; i++) {    /* plotting loop */
    ++x;
    --x1;
    if (D < 0) {
        /* pattern 1 forwards */
        plot(x, y, reverse);
        plot(++x, y, reverse);
        /* pattern 1 backwards */
        plot(x1, y1, reverse);
        plot(--x1, y1, reverse);
        D += incr1;
    } else {
        if (D < c) {
            /* pattern 2 forwards */
            plot(x, y, reverse);
            plot(++x, y += step, reverse);
            /* pattern 2 backwards */
            plot(x1, y1, reverse);
            plot(--x1, y1 -= step, reverse);
        } else {
            /* pattern 3 forwards */
            plot(x, y += step, reverse);
            plot(++x, y, reverse);
            /* pattern 3 backwards */
            plot(x1, y1 -= step, reverse);
            plot(--x1, y1, reverse);
        }
        D += incr2;
    }
}

/* end for */

/* plot last pattern */
if (pixels_left) {
    if (D < 0) {
        plot(++x, y, reverse); /* pattern 1 */
        if (pixels_left > 1)
            plot(++x, y, reverse);
        if (pixels_left > 2)
            plot(--x1, y1, reverse);
    } else {
        if (D < c) {
            plot(++x, y, reverse); /* pattern 2 */
            if (pixels_left > 1)
                plot(++x, y += step, reverse);
            if (pixels_left > 2)
                plot(--x1, y1, reverse);
        } else {
            /* pattern 3 */
            plot(++x, y += step, reverse);
            if (pixels_left > 1)
                plot(++x, y, reverse);
            if (pixels_left > 2)
                plot(--x1, y1 -= step, reverse);
        }
    }
}

/* end if pixels_left */
}

/* end slope < 1/2 */
else {    /* slope greater than 1/2 */
```



```
c = 2 * (dy - dx);
incr1 = 2 * c;
D = incr1 + dx;
for (i = 0; i < xend; i++) {
    ++x;
    --x1;
    if (D > 0) {
        /* pattern 4 forwards */
        plot(x, y += step, reverse);
        plot(++x, y += step, reverse);
        /* pattern 4 backwards */
        plot(x1, y1 -= step, reverse);
        plot(--x1, y1 -= step, reverse);
        D += incr1;
    } else {
        if (D < c) {
            /* pattern 2 forwards */
            plot(x, y, reverse);
            plot(++x, y += step, reverse);

            /* pattern 2 backwards */
            plot(x1, y1, reverse);
            plot(--x1, y1 -= step, reverse);
        } else {
            /* pattern 3 forwards */
            plot(x, y += step, reverse);
            plot(++x, y, reverse);
            /* pattern 3 backwards */
            plot(x1, y1 -= step, reverse);
            plot(--x1, y1, reverse);
        }
        D += incr2;
    }
}
/* end for */
/* plot last pattern */
if (pixels_left) {
    if (D > 0) {
        plot(++x, y += step, reverse); /* pattern 4 */
        if (pixels_left > 1)
            plot(++x, y += step, reverse);
        if (pixels_left > 2)
            plot(--x1, y1 -= step, reverse);
    } else {
        if (D < c) {
            plot(++x, y, reverse); /* pattern 2 */
            if (pixels_left > 1)
                plot(++x, y += step, reverse);
            if (pixels_left > 2)
                plot(--x1, y1, reverse);
        } else {
            /* pattern 3 */
            plot(++x, y += step, reverse);
            if (pixels_left > 1)
                plot(++x, y, reverse);
            if (pixels_left > 2) {
                if (D > c) /* step 3 */
                    plot(--x1, y1 -= step, reverse);
                else /* step 2 */
                    plot(--x1, y1, reverse);
            }
        }
    }
}
```

```
        }
    }
}
/* non-zero flag indicates the pixels needing swap back. */
plot(x, y, flag) int x, y, flag;
{
    if (flag)
        setpixel(y, x);
    else
        setpixel(x, y);
}
```

```
/*
A 3D Grid Hashing Function
by Brian Wyvill
from "Graphics Gems", Academic Press, 1990
*/

/* Test Program for 3D hash function.
In C the hash function can be defined in a macro which
avoids a function call
and the bit operations are defined in the language.
*/

#include <stdio.h>
#include <math.h>
#include "GraphicsGems.h"

#define RANGE          256
#define NBITS          4
#define RBITS          4
#define MASK           0360
#define HASH(a,b,c) (((a&MASK)<<NBITS|b&MASK)<<NBITS|c&MASK)>>RBITS)
#define HSIZE          1<<NBITS*3
#define IABS(x)        (int)((x) < 0 ? -(x) : (x))

typedef struct {
    double x,y,z;
} Triple, *RefTriple;

typedef struct { /* linked list of objects to be stored */
    Triple origin;
    struct Object *link;
} Object, *RefObject;

typedef struct { /* linked list of voxels (object pointers) */
    RefObject objectList;
    struct Voxel *link;
} Voxel, *RefVoxel;

RefVoxel table[HSIZE]; /* Table of pointers to Voxels */

checkrange(z) double z;
{
    if (z < 0 || z >= RANGE)
        fprintf(stderr,"%f out of range\n",z),          exit(1);
}

double getcoord()
{
    char buf[80];
    double z;
    scanf("%s",buf);
    z = atof(buf);
    checkrange(z);
    return z;
}

main()
{
    Triple a;
    while (TRUE) {
```

```
printf("Enter object position x y z ==> ");
a.x = getcoord();
a.y = getcoord();
a.z = getcoord();
printf("\ncoord: %d %d %d Hashes to %d\n", IABS(a.x), IABS(a.y), IABS(a.z),
      HASH(IABS(a.x), IABS(a.y), IABS(a.z) ));
};
}
```

```
/*
** Rotate an 8x8 tile clockwise by table lookup
** and write to destination directly.
** Large bitmaps can be rotated an 8x8 tile at a time.
** The extraction is done a nybble at a time to reduce the
** size of the tables.
**
** Input parameters:
** src          starting address of source 8x8 tile
** srcstep      difference in byte address between
**              adjacent rows in source bitmap
** dst          starting address of destination 8x8 tile
** dststep      difference in byte address between
**              adjacent rows in destination bitmap
**
** Ken Yap (Centre for Spatial Information Systems, CSIRO DIT, Australia)
** after an idea suggested by Alan Paeth (U of Waterloo).
*/

typedef long    bit32;

#define table(name,n)\
static bit32 name[16] =\
{\
    0x00000000<<n, 0x00000001<<n, 0x00000100<<n, 0x00000101<<n,\
    0x00010000<<n, 0x00010001<<n, 0x00010100<<n, 0x00010101<<n,\
    0x01000000<<n, 0x01000001<<n, 0x01000100<<n, 0x01000101<<n,\
    0x01010000<<n, 0x01010001<<n, 0x01010100<<n, 0x01010101<<n,\
};

table(ltab0,7)
table(ltab1,6)
table(ltab2,5)
table(ltab3,4)
table(ltab4,3)
table(ltab5,2)
table(ltab6,1)
table(ltab7,0)

void rotate8x8(src, srcstep, dst, dststep)
    unsigned char  *src, *dst;
    int            srcstep, dststep;
{
    register unsigned char  *p;
    register int            pstep, lownyb, hinyb;
    register bit32          low, hi;

    low = hi = 0;

#define extract(d,t)\
    lownyb = *d & 0xf; hinyb = *d >> 4;\
    low |= t[lownyb]; hi |= t[hinyb]; d += pstep;

    p = src; pstep = srcstep;
    extract(p,ltab0) extract(p,ltab1) extract(p,ltab2) extract(p,ltab3)
    extract(p,ltab4) extract(p,ltab5) extract(p,ltab6) extract(p,ltab7)

#define unpack(d,w)\
    *d = w & 0xff;      d += pstep;\
    *d = (w >> 8) & 0xff; d += pstep;\
    *d = (w >> 16) & 0xff; d += pstep;\
```

```
*d = (w >> 24) & 0xff;
```

```
p = dst; pstep = dststep;
```

```
unpack(p,low) p += pstep; unpack(p,hi)
```

```
}
```

```
/*
 * ANSI C code from the article
 * "Contrast Limited Adaptive Histogram Equalization"
 * by Karel Zuiderveld, karel@cv.ruu.nl
 * in "Graphics Gems IV", Academic Press, 1994
 *
 *
 * These functions implement Contrast Limited Adaptive Histogram Equalization.
 * The main routine (CLAHE) expects an input image that is stored contiguously in
 * memory; the CLAHE output image overwrites the original input image and has the
 * same minimum and maximum values (which must be provided by the user).
 * This implementation assumes that the X- and Y image resolutions are an integer
 * multiple of the X- and Y sizes of the contextual regions. A check on various other
 * error conditions is performed.
 *
 * #define the symbol BYTE_IMAGE to make this implementation suitable for
 * 8-bit images. The maximum number of contextual regions can be redefined
 * by changing uiMAX_REG_X and/or uiMAX_REG_Y; the use of more than 256
 * contextual regions is not recommended.
 *
 * The code is ANSI-C and is also C++ compliant.
 *
 * Author: Karel Zuiderveld, Computer Vision Research Group,
 *         Utrecht, The Netherlands (karel@cv.ruu.nl)
 */

#ifdef BYTE_IMAGE
typedef unsigned char kz_pixel_t;          /* for 8 bit-per-pixel images */
#define uiNR_OF_GREY (256)
#else
typedef unsigned short kz_pixel_t;         /* for 12 bit-per-pixel images (default) */
#define uiNR_OF_GREY (4096)
#endif

/***** Prototype of CLAHE function. Put this in a separate include file. *****/
int CLAHE(kz_pixel_t* pImage, unsigned int uiXRes, unsigned int uiYRes, kz_pixel_t Min,
          kz_pixel_t Max, unsigned int uiNrX, unsigned int uiNrY,
          unsigned int uiNrBins, float fCliplimit);

/***** Local prototypes *****/
static void ClipHistogram (unsigned long*, unsigned int, unsigned long);
static void MakeHistogram (kz_pixel_t*, unsigned int, unsigned int, unsigned int,
                           unsigned long*, unsigned int, kz_pixel_t*);
static void MapHistogram (unsigned long*, kz_pixel_t, kz_pixel_t,
                          unsigned int, unsigned long);
static void MakeLut (kz_pixel_t*, kz_pixel_t, kz_pixel_t, unsigned int);
static void Interpolate (kz_pixel_t*, int, unsigned long*, unsigned long*,
                        unsigned long*, unsigned long*, unsigned int, unsigned int, kz_pixel_t*);

/***** Start of actual code *****/
#include <stdlib.h>                                /* To get prototypes of malloc() and free() */

const unsigned int uiMAX_REG_X = 16;             /* max. # contextual regions in x-direction */
const unsigned int uiMAX_REG_Y = 16;             /* max. # contextual regions in y-direction */

/***** main function CLAHE *****/
int CLAHE (kz_pixel_t* pImage, unsigned int uiXRes, unsigned int uiYRes,
           kz_pixel_t Min, kz_pixel_t Max, unsigned int uiNrX, unsigned int uiNrY,
           unsigned int uiNrBins, float fCliplimit)
```

```
/*  pImage - Pointer to the input/output image
 *   uiXRes - Image resolution in the X direction
 *   uiYRes - Image resolution in the Y direction
 *   Min - Minimum greyvalue of input image (also becomes minimum of output image)
 *   Max - Maximum greyvalue of input image (also becomes maximum of output image)
 *   uiNrX - Number of contextial regions in the X direction (min 2, max uiMAX_REG_X)
 *   uiNrY - Number of contextial regions in the Y direction (min 2, max uiMAX_REG_Y)
 *   uiNrBins - Number of greybins for histogram ("dynamic range")
 *   float fClimplimit - Normalized cliplimit (higher values give more contrast)
 * The number of "effective" greylevels in the output image is set by uiNrBins; selecting
 * a small value (eg. 128) speeds up processing and still produce an output image of
 * good quality. The output image will have the same minimum and maximum value as the
input
 * image. A clip limit smaller than 1 results in standard (non-contrast limited) AHE.
 */
{
    unsigned int uiX, uiY; /* counters */
    unsigned int uiXSize, uiYSize, uiSubX, uiSubY; /* size of context. reg. and subimages
*/
    unsigned int uiXL, uiXR, uiYU, uiYB; /* auxiliary variables interpolation routine */
    unsigned long ulClipLimit, ulNrPixels; /* clip limit and region pixel count */
    kz_pixel_t* pImPointer; /* pointer to image */
    kz_pixel_t aLUT[uiNR_OF_GREY]; /* lookup table used for scaling of input
image */
    unsigned long* pulHist, *pulMapArray; /* pointer to histogram and mappings*/
    unsigned long* pulLU, *pulLB, *pulRU, *pulRB; /* auxiliary pointers interpolation */

    if (uiNrX > uiMAX_REG_X) return -1; /* # of regions x-direction too large */
    if (uiNrY > uiMAX_REG_Y) return -2; /* # of regions y-direction too large */
    if (uiXRes % uiNrX) return -3; /* x-resolution no multiple of uiNrX */
    if (uiYRes % uiNrY) return -4; /* y-resolution no multiple of uiNrY */
    if (Max >= uiNR_OF_GREY) return -5; /* maximum too large */
    if (Min >= Max) return -6; /* minimum equal or larger than maximum */
    if (uiNrX < 2 || uiNrY < 2) return -7; /* at least 4 contextual regions required */
    if (fClimplimit == 1.0) return 0; /* is OK, immediately returns original image.
*/
    if (uiNrBins == 0) uiNrBins = 128; /* default value when not specified */

    pulMapArray=(unsigned long *)malloc(sizeof(unsigned long)*uiNrX*uiNrY*uiNrBins);
    if (pulMapArray == 0) return -8; /* Not enough memory! (try reducing uiNrBins)
*/

    uiXSize = uiXRes/uiNrX; uiYSize = uiYRes/uiNrY; /* Actual size of contextual regions
*/
    ulNrPixels = (unsigned long)uiXSize * (unsigned long)uiYSize;

    if(fClimplimit > 0.0) { /* Calculate actual cliplimit */
        ulClipLimit = (unsigned long) (fClimplimit * (uiXSize * uiYSize) / uiNrBins);
        ulClipLimit = (ulClipLimit < 1UL) ? 1UL : ulClipLimit;
    }
    else ulClipLimit = 1UL<<14; /* Large value, do not clip (AHE) */
    MakeLut(aLUT, Min, Max, uiNrBins); /* Make lookup table for mapping of greyvalues
*/

    /* Calculate greylevel mappings for each contextual region */
    for (uiY = 0, pImPointer = pImage; uiY < uiNrY; uiY++) {
        for (uiX = 0; uiX < uiNrX; uiX++, pImPointer += uiXSize) {
            pulHist = &pulMapArray[uiNrBins * (uiY * uiNrX + uiX)];
            MakeHistogram(pImPointer,uiXRes,uiXSize,uiYSize,pulHist,uiNrBins,aLUT);
            ClipHistogram(pulHist, uiNrBins, ulClipLimit);
            MapHistogram(pulHist, Min, Max, uiNrBins, ulNrPixels);
        }
    }
}
```



```

    pImPointer += (uiYSize - 1) * uiXRes;          /* skip lines, set pointer */
}

/* Interpolate greylevel mappings to get CLAHE image */
for (pImPointer = pImage, uiY = 0; uiY <= uiNrY; uiY++) {
    if (uiY == 0) {                                /* special case: top row */
        uiSubY = uiYSize >> 1;  uiYU = 0; uiYB = 0;
    }
    else {
        if (uiY == uiNrY) {                         /* special case: bottom row */
            uiSubY = uiYSize >> 1;  uiYU = uiNrY-1;  uiYB = uiYU;
        }
        else {                                       /* default values */
            uiSubY = uiYSize; uiYU = uiY - 1; uiYB = uiYU + 1;
        }
    }
    for (uiX = 0; uiX <= uiNrX; uiX++) {
        if (uiX == 0) {                             /* special case: left column */
            uiSubX = uiXSize >> 1; uiXL = 0; uiXR = 0;
        }
        else {
            if (uiX == uiNrX) {                     /* special case: right column
*/
                uiSubX = uiXSize >> 1;  uiXL = uiNrX - 1; uiXR = uiXL;
            }
            else {                                   /* default values */
                uiSubX = uiXSize; uiXL = uiX - 1; uiXR = uiXL + 1;
            }
        }

        pulLU = &pulMapArray[uiNrBins * (uiYU * uiNrX + uiXL)];
        pulRU = &pulMapArray[uiNrBins * (uiYU * uiNrX + uiXR)];
        pulLB = &pulMapArray[uiNrBins * (uiYB * uiNrX + uiXL)];
        pulRB = &pulMapArray[uiNrBins * (uiYB * uiNrX + uiXR)];
        Interpolate(pImPointer, uiXRes, pulLU, pulRU, pulLB, pulRB, uiSubX, uiSubY, aLUT);
        pImPointer += uiSubX;                        /* set pointer on next matrix
*/
    }
    pImPointer += (uiSubY - 1) * uiXRes;
}
free(pulMapArray);                                /* free space for histograms */
return 0;                                           /* return status OK */
}

void ClipHistogram (unsigned long* pulHistogram, unsigned int
                    uiNrGreylevels, unsigned long ulClipLimit)
/* This function performs clipping of the histogram and redistribution of bins.
 * The histogram is clipped and the number of excess pixels is counted. Afterwards
 * the excess pixels are equally redistributed across the whole histogram (providing
 * the bin count is smaller than the cliplimit).
 */
{
    unsigned long* pulBinPointer, *pulEndPoint, *pulHisto;
    unsigned long ulNrExcess, ulUpper, ulBinIncr, ulStepSize, i;
    long lBinExcess;

    ulNrExcess = 0;  pulBinPointer = pulHistogram;
    for (i = 0; i < uiNrGreylevels; i++) { /* calculate total number of excess pixels */
        lBinExcess = (long) pulBinPointer[i] - (long) ulClipLimit;
        if (lBinExcess > 0) ulNrExcess += lBinExcess;    /* excess in current bin */
    }
}

```

```
/* Second part: clip histogram and redistribute excess pixels in each bin */
ulBinIncr = ulNrExcess / uiNrGreylevels; /* average binincrement */
ulUpper = ulClipLimit - ulBinIncr; /* Bins larger than ulUpper set to cliplimit */

for (i = 0; i < uiNrGreylevels; i++) {
    if (pulHistogram[i] > ulClipLimit) pulHistogram[i] = ulClipLimit; /* clip bin */
    else {
        if (pulHistogram[i] > ulUpper) { /* high bin count */
            ulNrExcess -= pulHistogram[i] - ulUpper; pulHistogram[i]=ulClipLimit;
        }
        else { /* low bin count */
            ulNrExcess -= ulBinIncr; pulHistogram[i] += ulBinIncr;
        }
    }
}

while (ulNrExcess) { /* Redistribute remaining excess */
    pulEndPointer = &pulHistogram[uiNrGreylevels]; pulHisto = pulHistogram;

    while (ulNrExcess && pulHisto < pulEndPointer) {
        ulStepSize = uiNrGreylevels / ulNrExcess;
        if (ulStepSize < 1) ulStepSize = 1; /* stepsize at least 1 */
        for (pulBinPointer=pulHisto; pulBinPointer < pulEndPointer && ulNrExcess;
            pulBinPointer += ulStepSize) {
            if (*pulBinPointer < ulClipLimit) {
                (*pulBinPointer)++; ulNrExcess--; /* reduce excess */
            }
        }
        pulHisto++; /* restart redistributing on other bin location */
    }
}

void MakeHistogram (kz_pixel_t* pImage, unsigned int uiXRes,
    unsigned int uiSizeX, unsigned int uiSizeY,
    unsigned long* pulHistogram,
    unsigned int uiNrGreylevels, kz_pixel_t* pLookupTable)
/* This function classifies the greylevels present in the array image into
 * a greylevel histogram. The pLookupTable specifies the relationship
 * between the greyvalue of the pixel (typically between 0 and 4095) and
 * the corresponding bin in the histogram (usually containing only 128 bins).
 */
{
    kz_pixel_t* pImagePointer;
    unsigned int i;

    for (i = 0; i < uiNrGreylevels; i++) pulHistogram[i] = 0L; /* clear histogram */

    for (i = 0; i < uiSizeY; i++) {
        pImagePointer = &pImage[uiSizeX];
        while (pImage < pImagePointer) pulHistogram[pLookupTable[*pImage++]]++;
        pImagePointer += uiXRes;
        pImage = &pImagePointer[-uiSizeX];
    }
}

void MapHistogram (unsigned long* pulHistogram, kz_pixel_t Min, kz_pixel_t Max,
    unsigned int uiNrGreylevels, unsigned long ulNrOfPixels)
/* This function calculates the equalized lookup table (mapping) by
 * cumulating the input histogram. Note: lookup table is rescaled in range [Min..Max].
 */
{
```

```
    unsigned int i; unsigned long ulSum = 0;
    const float fScale = ((float)(Max - Min)) / ulNrOfPixels;
    const unsigned long ulMin = (unsigned long) Min;

    for (i = 0; i < uiNrGreylevels; i++) {
        ulSum += pulHistogram[i]; pulHistogram[i]=(unsigned long)(ulMin+ulSum*fScale);
        if (pulHistogram[i] > Max) pulHistogram[i] = Max;
    }
}

void MakeLut (kz_pixel_t * pLUT, kz_pixel_t Min, kz_pixel_t Max, unsigned int uiNrBins)
/* To speed up histogram clipping, the input image [Min,Max] is scaled down to
 * [0,uiNrBins-1]. This function calculates the LUT.
 */
{
    int i;
    const kz_pixel_t BinSize = (kz_pixel_t) (1 + (Max - Min) / uiNrBins);

    for (i = Min; i <= Max; i++) pLUT[i] = (i - Min) / BinSize;
}

void Interpolate (kz_pixel_t * pImage, int uiXRes, unsigned long * pulMapLU,
    unsigned long * pulMapRU, unsigned long * pulMapLB, unsigned long * pulMapRB,
    unsigned int uiXSize, unsigned int uiYSize, kz_pixel_t * pLUT)
/* pImage      - pointer to input/output image
 * uiXRes      - resolution of image in x-direction
 * pulMap*     - mappings of greylevels from histograms
 * uiXSize     - uiXSize of image submatrix
 * uiYSize     - uiYSize of image submatrix
 * pLUT        - lookup table containing mapping greyvalues to bins
 * This function calculates the new greylevel assignments of pixels within a submatrix
 * of the image with size uiXSize and uiYSize. This is done by a bilinear interpolation
 * between four different mappings in order to eliminate boundary artifacts.
 * It uses a division; since division is often an expensive operation, I added code to
 * perform a logical shift instead when feasible.
 */
{
    const unsigned int uiIncr = uiXRes-uiXSize; /* Pointer increment after processing row
 */
    kz_pixel_t GreyValue; unsigned int uiNum = uiXSize*uiYSize; /* Normalization factor
 */

    unsigned int uiXCoef, uiYCoef, uiXInvCoef, uiYInvCoef, uiShift = 0;

    if (uiNum & (uiNum - 1)) /* If uiNum is not a power of two, use division */
    for (uiYCoef = 0, uiYInvCoef = uiYSize; uiYCoef < uiYSize;
        uiYCoef++, uiYInvCoef--, pImage+=uiIncr) {
        for (uiXCoef = 0, uiXInvCoef = uiXSize; uiXCoef < uiXSize;
            uiXCoef++, uiXInvCoef--) {
            GreyValue = pLUT[*pImage]; /* get histogram bin value */
            *pImage++ = (kz_pixel_t) ((uiYInvCoef * (uiXInvCoef*pulMapLU[GreyValue]
                + uiXCoef * pulMapRU[GreyValue])
                + uiYCoef * (uiXInvCoef * pulMapLB[GreyValue]
                + uiXCoef * pulMapRB[GreyValue])) / uiNum);
        }
    }
    else {
        /* avoid the division and use a right shift instead */
        while (uiNum >= 1) uiShift++; /* Calculate 2log of uiNum */
        for (uiYCoef = 0, uiYInvCoef = uiYSize; uiYCoef < uiYSize;
            uiYCoef++, uiYInvCoef--, pImage+=uiIncr) {
            for (uiXCoef = 0, uiXInvCoef = uiXSize; uiXCoef < uiXSize;
```

```
        uiXCoef++, uiXInvCoef--) {
    GreyValue = pLUT[*pImage];          /* get histogram bin value */
    *pImage++ = (kz_pixel_t)((uiYInvCoef* (uiXInvCoef * pulMapLU[GreyValue]
        + uiXCoef * pulMapRU[GreyValue])
        + uiYCoef * (uiXInvCoef * pulMapLB[GreyValue]
        + uiXCoef * pulMapRB[GreyValue])) >> uiShift);
    }
}
}
```

# Eric Haines



Mail me at [erich@acm.org](mailto:erich@acm.org). I work for [Autodesk, Inc.](#) Sorry, your browser doesn't support Java.

This page can now be reached by the URL <http://www.erichaines.com/>, the short version being just typing "erichaines.com" as the address.

## Links

My [portal page](#) sums up what web computer graphics resources I use the most. The [ACM TOG resources](#) area is where I put computer graphics research and education related links. The [Ray Tracing News](#) contains links to all sorts of computer graphics related resources, as does our [Real-Time Rendering](#) page (obligatory [Amazon link](#) here). I also have a [personal page](#) with pictures of all my fly fishing lures.

## Current Interests

Here are some resources I work on:

- [Real-Time Rendering](#) - a book (and website) on real-time rendering algorithms, coauthored with Tomas Möller. A page from this site I like to use as a jump-off spot is [my portal page](#). A mirror for this site is <http://www.acm.org/tog/resources/RTR/>.
- [journal of graphics tools](#) - a journal dedicated to presenting practical tools and techniques. The web site has useful code for some of the articles.
- [Ray Tracing News](#) - an electronic magazine/journal concerned with ray tracing and other rendering algorithms, published when I find the time.
- [Graphics Gems Repository](#) - a repository of the code from the popular *Graphics Gems* series of books.
- [ACM Transactions on Graphics](#) - a computer graphics research journal; I webmaster for it.

- [3D Object Intersection Page](#) - a handy table of references to algorithms for object/object intersection.
- [Ray tracing bibliography](#) - a free bibliography of ray tracing references.
- [Standard Procedural Databases \(SPD\) software package](#) - creates 3D models, mostly for testing ray tracer efficiency schemes. The source is in the public domain, and includes some useful 3D graphics code. I wrote the original, Xander Enzmann and others wrote converters to other common file formats.
- [Fantasy Graphics League](#) - demented or silly, you decide...
- [The Realtime Raytracing Realm](#) - put real-time rendering and ray tracing together and what do you get? I maintain this page of PC demos for Piero Foscari.
- [pellucid](#) - a java applet which shows the VRML rendering equation in action.
- [Educational computer graphics applets](#) - I've barely begun a collection of sites for these. Well-designed Java applets which teach computer graphics seem like a great thing that would benefit us all.

## Publications

- [Ray Tracing News](#), ed. Eric Haines. I have compiled articles related to ray tracing and rendering in general since 1987 and put them here. The collection is [indexed by category](#). There is also a [text archive](#) of the issues, useful for doing grep or find-in-files on.
- ["Shaft Culling Tool,"](#) Eric Haines, *journal of graphics tools*, v. 5, no. 1, 2000, p. 23-26. Efficient, compact code for generating and testing 3D shafts. A shaft is the volume between two axis-aligned bounding boxes. This algorithm quickly forms this volume and efficiently tests boxes and spheres against it. It's handy for culling out things between a light source and a ground plane, for example. The [abstract and code](#) are also online.
- "Triangle Intersection Tests," Eric Haines and Tomas Möller, *Dr. Dobb's Journal*, August 2000, [code and listings online](#).
- [Real-Time Rendering](#), by Tomas Möller and Eric Haines, published by A.K. Peters, 512 pages, July 1999, ISBN 1-56881-101-2. A mirror for this site is <http://www.realtimerendering.com>
- "Fast, Low Memory Z-Buffering when Performing Medium-Quality Rendering," Eric Haines and Steven Worley, *journal of graphics tools*, v. 1, no. 3, 1996, p. 1-5. The [abstract](#) is available online.
- "Point in Polygon Strategies," Eric Haines, *Graphics Gems IV*, ed. Paul Heckbert, Academic Press, San Diego, 1994, p. 24-46. Various methods for testing whether a point is inside a polygon. Code available online in the [Graphics Gems Repository](#); note that code has a newer test (CrossingsMultiply) which is particularly fast on Intel (slow divide) architectures.
- "Shaft Culling for Efficient Ray-Traced Radiosity," Eric A. Haines and John R. Wallace, *Photorealistic Rendering in Computer Graphics (Proceedings of the Second Eurographics Workshop on Rendering)*, Springer-Verlag, New York, 1994, p.122-138. Also in *SIGGRAPH '91 Frontiers in Rendering course notes*. Computing the amount two objects see each other can be approximated with ray tracing. By forming a tight polyhedral volume around the two objects and

quickly comparing it to a bounding volume hierarchy, we generate a reusable candidate set of objects and bounding volumes to test with a set of rays. A [postscript version of this paper](#) is available online.

- "Efficiency Improvements for Hierarchy Traversal," Eric Haines, *Graphics Gems II*, ed. James Arvo, Academic Press, San Diego, 1991, p. 267-273. Expansion of parts of the "Tracing Tricks" article.
- "Radiosity Bibliography," Eric Haines, in *Global Illumination Algorithms*, Donald P. Greenberg and Francois Sillion, Eurographics Technical report EG 91 TN, Eurographics Association, Aire-la-Ville, Switzerland, 1991. This bibliography is now maintained (and vastly expanded) by Ian Ashdown and is [available online](#) (follow the resources link).
- "Ronchamp: A Case Study for Radiosity," Eric Haines, *SIGGRAPH '91 Frontiers in Rendering course notes*, July 1991. Discusses practical meshing, energy balance, sampling, and display problems when using meshed radiosity. Text is available online in [TROFF format](#).
- "Beams O' Light: Confessions of a Hacker," Eric Haines, *SIGGRAPH '91 Frontiers in Rendering course notes*, July 1991. Discusses using monte carlo techniques and ray tracing to create atmospheric volume effects. Technique used in *The Key is Light* film to create the dusty shafts of light in the church. Text is available online in [TROFF format](#).
- "Fast Ray-Convex Polyhedron Intersection", Eric Haines, *Graphics Gems II*, ed. James Arvo, Academic Press, San Diego, 1991, p. 247-250 and code. Essentially quickly clipping a ray against the set of planes defining the polyhedron. Code available online in the [Graphics Gems Repository](#).
- "A Ray Tracing Algorithm for Progressive Radiosity," John R. Wallace, Kells A. Elmquist, Eric A. Haines, *Computer Graphics (SIGGRAPH '89 Proceedings)*, v. 23, no. 3, July 1989, p. 315-24. Occlusion testing for meshed radiosity can be done with ray tracing. Further discussion on the topic is available in [The Ray Tracing News, v. 2, no. 6](#).
- "Tracing Tricks," Eric A. Haines, *SIGGRAPH '89 Introduction to Ray Tracing course notes*, July 1989. Various efficiency scheme, spline surface intersection, and ambient lighting tricks. Reprinted in [The Ray Tracing News, v. 2, no. 8](#).
- "A Proposal for Standard Graphics Environments," Eric Haines, *IEEE Computer Graphics and Applications*, v. 7, no. 11, Nov. 1987, p. 3-5. Presentation of a set of programs which generate sets of standard scenes for testing ray tracing efficiency schemes. The [SPD software package](#) discussed is available online and is still being expanded (e.g. the famous teapot has been added, and the SPD now exports to many different file formats).
- "Essential Ray Tracing Algorithms," Eric Haines, *An Introduction to Ray Tracing*, ed. Andrew Glassner, Academic Press, London, 1989, p. 33-77. Ray/object intersections and mappings, including ray/sphere, polygon, box, and quadrics. Some of the material in this and other chapters of this book is available at the [SIGGRAPH HyperGraph](#) web site.
- "Ray Tracing Bibliography," Paul S. Heckbert and Eric Haines, *An Introduction to Ray Tracing*, ed. Andrew Glassner, Academic Press, London, 1989, p. 295-303. I currently maintain this [free online bibliography](#).
- "The Light Buffer: A Ray Tracer Shadow Testing Accelerator," Eric A. Haines, Donald P. Greenberg, *IEEE Computer Graphics and Applications*, v. 6, no. 9, Sept. 1986, p. 6-16. The basic

ideas presented are classifying objects from the light's viewpoint, and caching shadowing objects. The classification scheme uses a modified z-buffer to create lists of objects in sorted order for each "pixel" the light sees and determining depths beyond which no light passes. The other technique presented is caching the object that was last intersected by a shadow ray and immediately testing this object for the next shadow ray for the same light at the same location in the ray tree. Shadow caching is simple and applicable to almost any ray tracer. Dieter Bayer implemented the [light buffer for POV-Ray](#).

- *The Light Buffer: A Ray Tracer Shadow Testing Accelerator*, Eric A. Haines, Masters Thesis, Program of Computer Graphics, Cornell University, Jan. 1986.

## Imagery

Here are a variety of images I've made or was involved with. Click on a thumbnail to see the full-sized version.



For the 25th anniversary of the [Program of Computer Graphics](#) at Cornell, I made a [photomosaic](#) of Don Greenberg out of students, teachers, and staff who had been at the lab over the years.



I worked on the renderers and ray tracer for [TriSpectives](#); see their pages for many more images. The ray tracer was nothing too special (it used a hierarchy of grids efficiency scheme), except that it was integrated into the hidden surface renderers to be used on demand - this makes rendering faster than pure ray tracing. It can also be more accurate; subpixel rendering using an A-buffer is usually better than adaptive subsampled ray tracing because small features are caught more often (for example, wheel spokes can be fully missed by ray tracers). Click on the image to see renderings of a set of 3D models from one of the [TriGallery](#) collections. Most of the jewelry models here were created by Nancy Heinz.



I made the interlinked rings image for [Zap Andersson](#) on the occasion of his wedding. The image is interesting technically in that it was ray traced, with the shadows computed using adaptive radiosity meshes. It used bump mapping for the engraving (though I should have made the maps higher resolution).



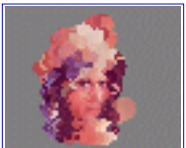


These are some still images of a model of Ronchamp, a chapel designed by Le Corbusier. To compare, see [photos of Ronchamp](#), which include a similar [side view](#) and [front view](#). The 3D model was created by Paul Boudreau and Keith Howie. I used the ArtCore radiosity and ray tracing system we developed for Hewlett-Packard, with some custom add-ins (see "[Confessions of a Hacker](#)"), to render the model. There are a number of bugs in the images, which are discussed in "[Ronchamp: A Case Study for Radiosity](#)". A minute long walkthrough of the church was shown at the SIGGRAPH '91 film show and is available through [SIGGRAPH](#) publications' Video Review. The February 1991 issue of Scientific American includes an introductory article about the rendering techniques used and includes more stills. I believe the SIGGRAPH '91 art slide set also contains stills.

After reading Takafumi Sato & Tokiichiro Takahashi's article on "Comprehensible Rendering of 3-D Shapes," in SIGGRAPH '90, I tried some experiments with illustrative rendering styles. Here are some old (1991?) renderings of a soda can model (the spline surface model is from the University of Utah):

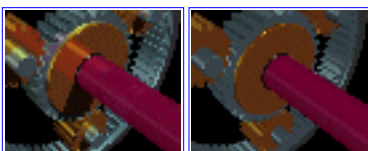
- [Can 1](#) (random noise bit shading)
- [Can 2](#) (vertical lines)
- [Can 3](#) (textured, thick horizontal lines)
- [Can 4](#) (horizontal lines with no noise)
- [Can 5](#) (horizontal lines with noise added for hatching)

Non-photorealistic rendering is interesting in that it widens the user's range of expressive styles enormously. Other places to look on the web at some interesting results are [Thinkfish](#), [Piranesi](#), [Inklination](#), and [Graphisoft](#).



Paul Haeberli's "Paint by Numbers" was another great article from SIGGRAPH 1990. It inspired me to write a system based on his ideas for my Hewlett Packard workstation (why should people using SGI's have all the fun?). This is a sample creation (made in about 5 minutes), using the famous Lenna image for its basis. Nowadays programs like Fractal Painter have taken these sorts of techniques miles beyond this point.

Here are some images comparing various rendering techniques; feel free to use them for educational purposes.

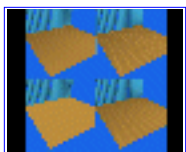


- This planetary gears assembly is a nice example of what ray tracing adds to a scene compared to standard hidden surface (z-buffer) rendering. The reflections are nice enough (though I personally find too many reflections to be more confusing than none at all). It's the shadows, for me, that give the 3D cues that make the image both more realistic and more

understandable. Image composition idea from Paul Booth.



- This set of three images shows a statue of liberty model rendered with hidden surface (z-buffer), ray tracing, and radiosity. Note how the ray trace shadows give the model a reality, but their sharpness can also detract from understanding (the shadow from the nose, for example: is that a crease or a shadow?). The meshed radiosity shadows are softer, but there are gouraud shading artifacts (e.g. on the tablet and at the base at the entryway) and a loss of shadow detail (the bottom of the robe's shadowed areas are simplistic). The radiosity solution could be improved some by finer meshing and by tessellation of complex polygons. Its advantage is, of course, that once the solution is computed, the set of polygons generated by the process can be displayed using accelerated Z-buffer hardware and display at near the same speed (and in some cases faster, since illumination does not have to be computed for it; balancing this is that there are more polygons to display) as the original hidden surface model.



- This set of four images, clockwise from the upper left, shows distributed ray tracing, traditional ray tracing, radiosity, and hidden surface renderings of the same scene. Distributed (a.k.a. stochastic) ray tracing gives the truest image (and is most expensive to compute), traditional ray tracing gives the usual sharp shadows, and meshed radiosity gives soft shadows everywhere (even when they should be sharp at the bases of the pipes) and mesh artifacts. The radiosity rendering could be improved by increasing the mesh resolution (which is generated with quadrees and is already fairly fine). Efficiently creating sharp shadows at the bases of objects while using meshed radiosity is still an active area of research (see the [recent paper by Telea et al.](#)).



- As above, this set of four images, clockwise from the upper left, shows distributed ray tracing, traditional ray tracing, radiosity, and hidden surface renderings. The shadows on the distributed ray trace are noticeably noisy, even though a large number of shadow rays were shot per pixel. Traditional ray tracing is sharp shadowed, as usual. Meshed radiosity without using blending techniques has a few problems besides those detailed above: loss of specular component (shine), meshed spheres result in visible gap where spheres are supposed to touch, and gouraud shading artifacts. There is also a bit of "shadow leak" visible at the base of the closest cone, where the shadowed samples under the cone affect the shading outside of the cone. All of these problems are curable; I wish I had rendered this same scene with our final product. We used the radiosity mesh imposed on the original geometry, combined with ray tracing the specular components, to solve the first two; we used surface modeling techniques to determine where the cones touched the floor and created mesh edges on the floor at these points to avoid shadow leaks.



- Here is an early radiosity/ray-tracing blend experiment we tried. Clockwise from left

is the hidden surface version, the traditional ray tracer version, the blend (radiosity shadowing and interreflection and ray traced reflections), and the pure radiosity version. There are edge artifacts where the pink sides met the base in all but the ray traced image due to hidden surface renderer inaccuracy (this was done in the days before sub-pixel addressing was available in hardware, so cracking like this was common). The blend technique was a simple proof of concept image I made by using the pure radiosity image and adding it to a ray traced image which had only the reflections and highlight rendered in it. There are some fairly bad registration problems where the ray traced and hidden surface images didn't align properly (look at the handle). We later integrated ray tracing, radiosity and A-buffer renderers into a system which cleanly and quickly produced such images. Pity it didn't sell...



This is a camshaft image I rendered with an early version of our ray tracer, created for Hewlett Packard. It was used in HP advertising literature, and has the claim to fame of being one of the physically largest computer graphics images ever displayed, as it was printed on the side of HP's trade show tractor-trailer. The model was created by HP's German CAD group (now CoCreate GmbH).



This is the version of Spheraflake which was a part of the SIGGRAPH '87 art show. There are 7381 spheres. The model is from the free [Standard Procedural Databases software package](#), available online. The floor plane texture was done with a procedural function.



Countertop ray traced image from my thesis used on the Sept. 1986 cover of IEEE CG&A. The shiny bowl and shadows from ray tracing add some realism, but the extensive use of texture mapping is what gives the image most of its visual interest. Cornell's system was great in that you could see the power of combining a good modeler and good material designer with powerful rendering algorithms. This image was produced around mid-1985.



A ray trace from my [Master's Thesis](#), the image was produced during December 1984 and used on the SIGGRAPH '86 advanced program. This board position is from Raymond Smullyan's wonderful *The Chess Mysteries of the Arabian Knights*, Knopf, 1981. White has not castled; is the black pawn which started on b7 still on the board, and if he is, is he still a pawn or promoted?

---

Eric Haines / [erich@acm.org](mailto:erich@acm.org)

Last change: *February 5, 2001*



# ACM Transactions on Graphics

This page provides information about the journal ACM Transactions on Graphics (TOG), a publication of the [ACM](#). In addition to information about the journal itself, we include links to resources and sites on the net which are related to computer graphics research. You may [search](#) the site.



## Overview

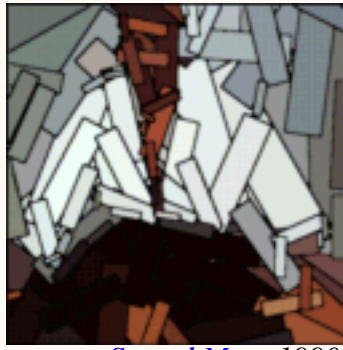
- Search [ACM TOG](#) or the SIGGRAPH [Bibliography Database](#)
- [Contents](#)
- [Background](#)
- [Resources](#)



*Printing Press, 1991, using radiosity,  
by [John Wallace](#), [3D/EYE Inc](#)  
and [Hewlett-Packard](#)*

## Contents

- [Titles in Current Issue](#)
- [Addenda and Errata](#)
- [Titles to Appear in Future Issues](#)
- [Tables of Contents by Volume](#) ([compact version](#))



*Seated Man*, 1990,  
using relaxation painting,  
by [Paul Haeberli](#), [SGI](#)

## Background

- [Editorial Charter](#)
- [Editorial Board](#)
- [Subscriptions/Access to Articles](#)
- [Information for Authors](#) (and [upon acceptance](#))
- [Reviewing Guidelines and Forms](#)



*The Mighty Maple*, 1985,  
by [Jules Bloomenthal](#)

## Resources

- [Literature On-Line](#)
- [Software Related Tools](#)
- [Research Sites](#) (linked to University of California - Santa Cruz)

---

For further information about TOG contact the Editor in Chief, [Jessica Hodgins](#), at [jkh+tog@cs.cmu.edu](mailto:jkh+tog@cs.cmu.edu).

---

[Eric Haines](#), On-Line Editor / [erich@acm.org](mailto:erich@acm.org) Last change: December 12, 2000



# Ray Tracing News Guide

*"Light Makes Right"*

Compiled by [Eric Haines](#), [erich@acm.org](mailto:erich@acm.org). Opinions expressed are mine, not Autodesk's [my current employer].

A wonderful [index](#) of the Ray Tracing News was begun by [Paul Heckbert](#).

## Search For:

**Match:**    Any word    All words    Exact phrase

**Show:**  
                 results                   summaries

For a wider view of the world, check out the [ray tracing FAQ](#).

## Archive locations:

- Text version at <http://www.acm.org/tog/resources/RTNews/text/>
- Zip file of all text files at [http://www.acm.org/tog/resources/RTNews/text/rtn\\_text.zip](http://www.acm.org/tog/resources/RTNews/text/rtn_text.zip)
- HTML at <http://www.acm.org/tog/resources/RTNews/html/>
- Zip file of all HTML pages at [http://www.acm.org/tog/resources/RTNews/html/rtn\\_html.zip](http://www.acm.org/tog/resources/RTNews/html/rtn_html.zip)

If you would like to be informed when a new issue of the Ray Tracing News has come out, let [erich@acm.org](mailto:erich@acm.org) know.

This cumulative table of contents was started by Joe Cychosz (*thanks, Joe!*).

---

## [RTNv13n1](#): July 16, 2000

- [Introduction](#)
- [Ray Tracing Roundup](#)
- [BART: A Benchmark for Animated Ray Tracing](#), by Jonas Lext, Ulf Assarsson, and Tomas Moller
- [Global Illumination Test Scenes](#), by Brian Smits and Henrik Wann Jensen
- [Realtime Techniques for Ray Tracing](#), by Ádám Nagy, Eric Haines, Russel Simmons, Paul Kahler, Factory, and Bruno Schödlbauer
- [Fastest Way to Generate Eye Rays in Ray Tracing](#), by Samuel Paik
- [On Transforming Planes](#), by Steve Hollasch
- [Intersecting a Ray and a Triangle with Plücker Coordinates](#), by Ray Jones
- [Volumetric Rendering](#), by Paul Kahler

- [Recent Papers](#), summarized by Eric Haines and Vlastimil Havran
- 

## **RTNv12n2: December 21, 1999**

- [Introduction](#)
  - [Ray Tracing Roundup](#)
  - [The Ups and Downs of Multithreaded Ray Tracing and Optimization](#), by John Stone
  - [A Summary of Octree Ray Traversal Algorithms](#), by Vlastimil Havran
  - [Additional Octree Traversal Notes](#), by John Spackman, Erik Jansen, and Joel Welling
  - [Z-buffer and Raytracing](#), by Sam Paik
  - [Ray Tracing Procedural & Parametric Surfaces](#), by Mark VandeWettering, Stephen Westin, and Matt Pharr
  - [Raytracing Gloss/Translucency](#), by Peter Eastman, Matt Pharr, and Stephen Westin
  - [Importance for Ray Tracing](#), by Per H. Christensen
  - [Good Computer Graphics Books](#), by Brent McPherson
- 

## **RTNv12n1: June 25, 1999**

- [Introduction](#)
  - [Ray Tracing Roundup](#)
  - [Comparison of Hierarchical Grids](#), by Vlastimil Havran and Filip Sixta
  - [Great Shadow Hack](#), summarized by Eric Haines
  - [Octree Traversal and the Best Efficiency Scheme](#), by Ben Hutchison, Eric Haines, Hanan Samet, and Erik Jansen
  - [Quicker Grid Generation via Memory Allocation](#), by Eric Haines
- 

## **RTNv11n1: July 11, 1998**

- [Introduction](#)
- [Ray Tracing Roundup](#)
- [Pluecker Coordinate Tutorial](#), by Ken Shoemake
- [A Short Note on Kalra and Barr's Algorithm](#), by Andrei Sherstyuk
- [Origins of Point In Polygon](#), Take 10..., by Neil Stewart
- [Info on REYES Algorithm](#), by Robert Speranza and Tom Duff
- [Polygon Shrinking](#), by Dave Rusin and Jeff Erickson

- [Correcting Normals on "Flipped" Polygons](#), by Kev, Duncan Colvin, Steve Baker, John Nagle, Dennis Jiang, Alejo Hausner, and Eric Haines
  - [What's Mesa?](#) by Brian Paul
  - [Multithreading Mesa](#), by John Stone
  - [Recent Ray Tracing European Conference Papers](#)
  - [Attenuation in Water](#), by Bretton Wade and Ian Ashdown
- 

## **[RTNv10n3](#): December 2, 1997**

- [Introduction](#)
  - [New People](#)
  - [Ray Tracing Roundup](#)
  - [Two New Graphics Books plus One](#), by Eric Haines
  - [Ray Classification for Animation](#), by Matt Quail
  - [Raytracker Tricks](#), by Hakan "Zap" Andersson
  - [Even Faster Crossings Test](#), by Philip Brown
  - [Additional Notes on Nested Grids](#), by Kris Klimaszewski, Andrew Woo, Frederic Cazals, and Eric Haines
  - [Recursive Grids and Ray Bounding Box Comments and Timings](#), by Andrew Woo
  - [CD-ROM: The Internet Raytracing Competition](#), Year One, review by Eric Haines
  - [Plücker Coordinates](#), by Jeff Erickson
  - [Errors of Commission and Omission](#), by Eric Haines
  - [Cyrus-Beck as a Ray/Polygon Tester](#), by David Rogers
  - [Mirror Reflectivity](#), by Greg W. Larson
  - [Testing a Matrix for Uniform Scaling](#), by Charles Iliya Krempeaux and Eduardo Marreiros
  - [Declassified CIA BRDFs now available](#), by Greg Ward Larson
  - [Lifting the Monkey's Paw Curse](#), by Jeff Goldsmith
  - [Eccentricity Effects in Blobs](#), by Alfonso Hermida
- 

## **[RTNv10n2](#): June 26, 1997**

- [SIGGRAPH Roundtable](#)
- [Introduction](#)
- [New People: Marcos Fajardo Orellana](#)
- [Ray Tracing Roundup](#)



- [Illuminated by Black Light](#), by Dan Wexler
  - [What's Wrong with Monte-Carlo Methods?](#) by D.C. Nguyen (JuHu), Pete Shirley, Stephen Westin, and Eric Veach
  - [More on Rendering Bugs](#), by Eric Haines
  - [The Curse of the Monkey's Paw](#), by Eric Haines
  - [Global Illumination "Bible"?](#) by Ian Ashdown
  - [Megahedron](#)
  - [Detecting Points on the Edge of a Polygon](#), by Eric Haines
  - [Rendering Unflat Polygons](#), by Eric Haines
  - [Free Polygon Tessellators](#), by Eric Haines
  - [Octree Neighbor Finding](#), by Andrew Glassner, Francois Sillion, and Paul Heckbert
  - [Author Sues Publisher](#), by Ralph Grabowski
  - [European Graphics Tour](#), by Nelson Max
  - [Fast Shadow Testing Bibliography](#), by Slawomir Kilanowski
- 

## **RTNv10n1: January 21, 1997**

- [Introduction](#)
  - [New People: Jim Roedder](#)
  - [Faster Refraction Formula, and Transmission Color Filtering](#), by Xavier Bec
  - [Results of Sphere in Box Ratio Contest](#), by Han-Wen Nienhuys, Jim Arvo, and Eric Haines
  - [Ray Tracing Roundup](#)
  - [Papers of Interest](#), by Eric Haines
  - [Synthetic Textures and Genetic Algorithms](#), by Steve Worley
  - [Ray Tracing, What is it Good For?](#) by Arijan Siska
  - [Fast Soft Shadows](#), by Steve Worley
  - [Progressive Ray Tracing and Fast Previews](#), by Eric Haines
  - [Free Radiosity Renderer \(inc. C++ source code\)](#), by Ian Ashdown
  - [Cells and Portals Resources](#), by Randy Stiles
  - [New Books](#), by Eric Haines
- 

## **RTNv9n2: January 24, 1996**

- Introduction
- Advice to Authors, by Anonymae

## **RTNv9n1: January 18, 1996**

- Introduction
  - Results of the Name that Area Ratio Contest, by Eric Haines and Arijan Siska
  - Another Contest: Sphere in Box ratio, by Eric Haines
  - Ray Tracing Roundup
  - A Freeware modeller supporting VRML, by Stephen Cheney
  - 3D Point Hashing, by Steve Worley
  - 5D vs. BSP and ABVH, by Nguyen Duc Cuong aka "JuHu"
  - Interacting with Ray Traces, by Bert Peers
  - Hierarchical Techniques for Glossy Global Illumination, by Per Christensen
  - Global Illumination SIGGRAPH meeting, by Greg Ward
  - Total Internal Reflection, by Eric Haines, Greg Ward, and Chris Larson
  - Pretemporal Imaging Systems, by Ken Musgrave and Eric Haines
  - PNG Status, by Tim Wegner and Tom Lane
  - Cinematography References, by Dan Wexler
  - Getting Rid of the Divide, by Frank Compagner and Bob Pendleton
- 

## **RTNv8n3: August 2, 1995**

- Introduction
  - Contest: Name that Area Ratio
  - Ray Tracing Roundup
  - On-Line Computer Science Bibliography Collection, by Alf-Christian Achilles
  - Comments on RTNv8n1, by Alexander Enzmann
  - Dore' Now Free, and Dore' Mailing List, by Len Wanger
  - Fooling Around, by Eric Haines
  - Beware of VIDEA! by W. Purgathofer, E. Groeller, M. Feda
  - Still More on Optical Ray Tracing, by Dan Reiley
  - Raytracing and 3D Studio, by Michael Adams and Brian Hoffman
  - Testing SIPP versus Raytraces under DOS, by Alexander Enzmann
- 

## **RTNv8n2: May 16, 1995**

- Introduction
- Ray Tracing Roundup

- Scanline vs. Ray Tracing, by Loren Carpenter and Sam Richards
  - Pyramid Clipping, by Erik Reinhard and Erik Jansen
  - Copyrighting Raytraced Images, by J Edward Bell, Benton Holzwarth, and Keith
  - Octrees and Whatnot, by Steve Worley and Eric Haines
  - Learning Radiance, by Kevin Matthews
  - Radiance vs. Pov, by "Cloister Bell"
  - Overlapping Refractors, by Steve Worley and Eric Haines
  - Raytracing Particle Systems, by Jos Stam
  - Answers/References on Cone Tracing, by Brent Baker
  - Dumb Radiosity Question, by Steve Worley and Eric Haines
  - Good Book on 3d Animation, by Scott McCaskill
  - Animation Books, by Dave DeBry
  - Fast Algorithms for 3D Graphics, by Glaeser: Any Good? by Brian Hook
  - Order of Rendering and Fast Texturing, by Bruno Levy, Dan Piponi, and Bernie Roehl
  - FREE E-Mag on VR, Computer Graphics, et al: Issue 2, by David Lewis
  - Color Quantization Bibliography, by Ian Ashdown
- 

## **RTNv8n1: January 23, 1995**

- Introduction
- People
- Fetishes and Perversions, by Doug Andersen
- Ray Tracing Roundup
- New Web Sites of Note, by Eric Haines
- MGF: Materials and Geometry Format, by Greg Ward
- Blue Moon Rendering Tools RenderMan Implementation, by Larry Gritz
- RenderMan Implementation Paper On-Line, by Philipp Slusallek
- TGA BBS CD ROM review, by Eric Haines
- Review of Syndesis Avalon CDROM, by Keith Rule
- SIRDS/SIS/stereograms/Holusions/whatever, by Eric Haines
- Two Books on Stereograms, by Eric Haines
- Siggraph Report: The Exhibits, by Techs Avery
- Scanline Rendering Previewer, by Eric Haines due to Chris Cason
- Raysmith, by Eduardo Bustillo Iceta
- Terminology Survey Results, by Peter Shirley
- Radiosity Online: A Bibliography, by Ian Ashdown
- Jevans' Temporal Coherence for Ray Tracing, by Eric Haines

- Reviews of \_Encyclopedia of Graphics File Formats\_, from Steve Lamont, John Grieggs, Tom Lane, Brian Erwin, and Andy Key
  - SDSC Image Tools library, by Matt Hughes
  - LUG Library v1.0, by Raul Rivero
- 

## [RTNv7n5](#): October 13, 1994

- Introduction
  - New People
  - Ray Tracing Roundup
  - Recent Ray Tracing Papers
  - Rumor Mill
  - AERO Animation/Simulation System version 1.5.1, by Thomas Braeunl
  - A Brief Review of [an old] AERO, by Dave Negro
  - Photon Tracing, by Chris Thornborrow and Greg Ward
  - Faster Than POV-RAY 2.1, by Dieter Bayer
  - Z Buffer Based Rendering Program, by Raghu Karinithi
  - Gossamer, a Free Macintosh VR/3D Renderer, by Jon Blossom
  - Antialiasing Issues, by Arijan Siska
  - Microcosm, by Abe Megahed of Cosmic Software
  - Fisheye Lens Distortion, by Greg Ward
  - Optical Ray Tracers
  - Correcting Normal Direction, by Gavin Bell
  - Graphics Gems IV Table of Contents, by Paul Heckbert
  - Beyond Graphics Gems, by Paul Heckbert
  - Radiosity vs. Ray Tracing, by Rico Tsang
  - ACM SIGGRAPH Online Bibliography Updated, by Frank Kappe
  - How to be Notified of New POV Releases
  - PoVSB Windows-based Modeler v0.85, by Jeff Hauswirth
  - Porting Rayshade, PBM, etc from Unix to DOS, by Mike Castle
  - REYES & Patents, William C. Archibald
  - Going from AutoCAD and 3DS into Ray Tracing, by Sean Ross
  - Computer Lego Modeling, by Paul Gyugyi
  - Rowe's Ray Tracing World BBS, by Harry Rowe
  - On Using BSP trees, by Benton Jackson
  - Books about Commercial Renderers, by Don Lewis, Jimbo and Yury German
-

## **RTNv7n4: July 14, 1994**

- Introduction and Announcement
  - Call For Participation: POV Ray Tracing CD ROM, by Christopher Cason
  - Tracers and Tracings #1, by Tom Hoover
  - Review of Tracers and Tracings CD ROM, by Eric Haines
  - Ray Tracer Comparison, by Amanda L. Osborne
  - Blob Sculptor, by Alfonso Hermida
  - Book Announcements, and Modeler vs. Scripting Language, by Mitch Waite
  - A Quick Writeup of Ray Tracing for the Macintosh CD, from notes by Eduard Schwan
  - Ray Tracing Roundup
  - POV-Help, by Chris Cason
  - Lenticular 3D, by Keith Rule
  - POV Official News, Issue #1, POV Team
  - A Quick Explanation of Radiosity Computation vs. Output, by Steve Hollasch
  - A Grand Unified Modeller (GUM), by Lex van der Sluijs
  - New (at least to me) Bulletin Boards, collected by Eric Haines
  - GemCAD Review, by Greg Prior
  - A Summary of the Reyes Algorithm, by Steve Demlow
  - Version 1.7 of Polyray Available, by Alexander Enzmann
  - Use of External Program Calling for Polyray, by Alexander Enzmann
  - POV-Ray Legal Stipulations Correction, by Dan Farmer
  - 3D Graphics Book List, by Brian Hook
- 

## **RTNv7n3: July 6, 1994**

- Introduction
- Ray Tracing Roundtable at SIGGRAPH '94, by Eric Haines
- Books at SIGGRAPH '94, by Eric Haines
- Radiosity: A Programmer's Perspective, by Ian Ashdown
- Computational Geometry in C, by Joseph O'Rourke
- Simple Databases Available For Rendering (Global Illumination), by Peter Shirley
- Radiance Version 2.4 Available, by Greg Ward
- Faster Ray-Torus Intersection, by Eric Haines
- RAT PACK: Free Ray Tracing Research Software, by Tom Wilson
- Light Beams Tricks, by Chris Thornborrow
- On-Line Ray Tracing Bibliography, by Ian Grimstead

- Polynomials, by Han-Wen Nienhuys
  - Shadow Caching Observation, by Han-Wen Nienhuys
  - Small Comment on GGems III bsp.c, by Han-Wen Nienhuys
  - Fogsources, by Han-Wen Nienhuys
  - The Rayce Ray Tracer, by Han-Wen Nienhuys
  - Ray Tracing Roundup
  - A Storage Trick for 3D Polygons, by Eric Haines
  - Illumination-Related Abstracts from the Proceedings of Graphics Interface '93
  - Design and Aims of the YART Graphics Kernel, by Ekkehard 'Ekki' Beier
  - Distribution Ray Tracing, by Marc Levoy
  - Ray Tracing, Antialiasing, and What To Do Instead, Steven Demlow
  - Yet Another Illumination Ph.D., by George Drettakis
- 

## **RTNv7n2: February 2.01, 1994**

- Introduction
- Ray Tracing Roundup
- A Note on SPD Platform/Compiler Results (SPARC II), by David Hook
- OORT - Object Oriented Ray Tracer, by Nicholas Wilt [and Eric Haines]
- Partial Evaluation Applied to Ray Tracing, by P.H. Andersen
- Comparison of Ray Traversal Methods, by Erik Jansen
- Sphere Tessellation//Gamma Correction, by Olin Lathrop
- Computational Geometry On-Line Bibliography, by Bill Jones
- Summary of Advanced Rendering Papers from Eurographics '93, by Erik Jansen
- RAT, Another Ray Tracer, by Tom Wilson
- Fast Raytracing via SEADS, by John Chapman and Wilfrid Lefer
- Parallel Ray Tracing Schemes, by Rainer Menes
- Notes on Parallel Ray Tracing, by Hsiu Lin and Sam Uselton
- Parallel Texturing, by Jon Leech and Brian Corrie
- Mapping Texture on a Sphere, by Ron Capelli
- Computational Geometry in C, by Joseph O'Rourke
- Programming for Graphics Files in C and C++, by John R Levine
- Point in Polygon, the Quick Answer, by Wm. Randolph Franklin and Eric Haines
- Elib and NetNews Information, by Eric Haines
- CFP: 5th Eurographics Workshop on Rendering, by Georgios Sakas
- CFP: 5th EG workshop on Animation & Simulation, by Gerard Hegron
- Morphology Digest, by Henk Heijmans

- Position of the Sun, by Joe Cychosz
- 

## **RTNv7n1: February 2, 1994**

- Introduction
  - Quick Book Reviews, by Eric Haines
  - Ray Tracing Roundup
  - 3D Artist Magazine Information, by Tim Riley
  - Updated Graphics CD-ROM from Knowledge Media, by Paul A. Benson
  - The Desert Isle List, by Amanda Osborne
  - Raytrace Utilities List for DOS (and Windows), by Amanda Osborne
  - Brief Reviews of a Bunch of Useful PC Stuff, by Tim Lister
  - Tree and Plant Image Generation, by Phil Drinkwater and Jason Weber
  - POV-Ray 2.0 Released, by Dave Buck
  - JPEG Texture Maps, by Petri Nordlund
  - ArchiCAD Model Translator, and other notes, by Paul D. Bourke
  - New Radiance Version Available, by Greg Ward
  - Previewer Program for Radiance, by Greg Ward
  - IRT Solid Modeller Version 4.0, by Gershon Elber
  - Object-Oriented Graphics: GOOD 0.50, by Ekkehard Beier
  - GFX News, by Eric Hsiao
  - New Wavefront Listserv, by George H. Otto
  - Optimized POV 2.0 version, by Peter K. Campbell
  - Errata for "Adventures in Ray Tracing", by Alfonso Hermida
- 

## **RTNv6n3: September 28, 1993**

- Introduction
- New People
- Ray Tracing Roundup
- Free Ray Tracer Summary, compiled by Eric Haines
- Ray Tracer Races, Round 2, by Eric Haines
- An Enhanced Standard Procedural Databases Package, by Eric Haines
- Shadows from Refractive Objects, by Steven Collins
- Shadows Through Transmitters, by Eric Haines, Greg Ward, Alexander Enzmann, Stephen Coy, and David Hook
- Faster Bounding Volume Hierarchies, by Brian Smits and Eric Haines



- Simple Sun Position Program, by James Ashton
  - Sphere and Cylinder/Cone/Disc/Annulus Definition, by Eric Haines
  - Ray Tracing Roundup
  - Simple Sphere Tessellation, by Mike Castle
  - Syndesis CD-ROM Review, by Eric Haines
  - Ray Tracing Related Abstracts from the Proceedings of Graphics Interface '93
  - Ray Tracing Papers in the First Bilkent Computer Graphics Conference, ATARV-93, Ankara, Turkey, July 1993
  - Fourth Eurographics Workshop on Rendering, Paris, France June 14-16, by Francois Sillion
  - Gamma Correction Frequently Asked Questions, by Graeme Gill
  - 3D Studio Rendering, by Chris Williams
  - Ray-Bezier Patch Intersection, by Chuck McKinley, Max Froumentin
  - Turbulence and Noise, by Ken Perlin
  - Ray Tracing Research Ideas, by Klaus Lisberg Kristensen & Christian Gautier
  - POV-Ray Utilities, by Dan Farmer
  - SPD Platform/Compiler Results, by David Hook
- 

## **RTNv6n2: July 1, 1993**

- Introduction
- New People
- Ray Tracer Races, Round 2, by Eric Haines
- Simple, Fast Triangle Intersection, part II, by John Spackman
- Review: \_Photorealism and Ray Tracing in C\_ (and others), by Eric Haines
- Comments on Various Ray Tracing Speedups, by Andrew Woo
- Errata to \_Photorealism and Ray Tracing in C\_, compiled by Eric Haines
- InterChange Plus Model/Texture Data CD-ROM, by John Foust
- Ray Tracing Roundup

### =====Net News Cullings=====

- Re: Intersection Between a Line and a Polygon (UNDECIDABLE??), by Allen B
  - Obfuscated Postscript Ray Tracer, by Takashi Hayakawa
- 

## **RTNv6n1: January 27, 1993**

- Introduction
- New People, New Places, etc
- An Easily Implemented Ray Tracing Speedup, by Steve Worley



- Color Storage, by Greg Ward
- Bounding Areas for Ray/Polygon Intersection, by Steve Worley and Eric Haines
- Simple, Fast Triangle Intersection, by Chris Green and Steve Worley
- Ray Tracing Roundup
- Spectrum Overview, edited by Nick Fotis
- Comments on the Glazing Trick, by Eric Haines

=====Net News Cullings=====

- Announcing the ACM SIGGRAPH Online Bibliography Project, by Stephen Spencer
  - A Brief History of Blobby Modeling, by Paul Heckbert
  - Cool Raytracing Ideas, Karen Paik
  - Optical Effects and Accuracy, by Sam Uselton
  - Map Projections Reference Book, by Mike Goss
  - A Brief Review of Playmation, by Chris Williams
  - PV3D Quick Review, by David Anjo
  - Bounding Volumes (Sphere vs. Box), by Tom Wilson
  - Raytracing Swept Objects, by Mark Podlipec
  - Ray Transformation, by Kendall Bennett
- 

[\*\*RTNv5n3: September 2, 1992\*\*](#)

- Introduction
  - Intersection Between a Line and a Polygon (UNDECIDABLE??), by Dave Baraff, Tom Duff
  - Fastest Point in Polygon Test, by Aladdin Nassar, Philip Walden, Eric Haines, Tom Dickens, Ron Capelli, Sundar Narasimhan, Christopher Jam, and (last but not least) Stuart MacMartin
  - Polygon Intersection via Barycentric Coordinates, by Peter Shirley
  - Many-Sided Polygon Intersection, by Eric Haines, Benjamin Zhu
  - Code for Point in Polygon Intersectors, by Eric Haines
- 

[\*\*RTNv5n2: August 26, 1992\*\*](#)

- Introduction - SIGGRAPH roundtable, etc
- New People, New Addresses, etc
- BSP Traversal Errata, by Kelvin Sung
- The Art of Noise, by Steve Worley
- Ray Tracing Roundup, by Eric Haines
- Graphics Gems III Residency Mask errata, by Joe Cychosz
- Any Future for Efficiency Research?, by Eric Haines

- NuGrid Update, by Mike Gigante

=====Net News Cullings=====

- Spline Patch Ray Intersection Routines, by Sean Graves
  - Xdart, from Mark Spychalla
  - 4D Space Viewing Software, by Steve Hollasch
  - BYU Modelling and Visualization Package - Free Demo, by Stacey D. Son
  - New Radiosity Program Available, by Guy Moreillon
  - Optics Lab Information, by Anthony Siegman
  - Ray Tracing (or Casting) Applications, by Tom Wilson
  - Goofy (?) Idea, Gary McTaggart
  - Constant Time Ray Tracing, by Tom Wilson, Eric Haines, Brian Corrie, and Masataka Ohta
  - VLSI for Ray Tracing, by Kenneth Cameron, George Kyriazis, Peter De Vijt, Jon Leech, Sam Uselton, Thierry Priol, and "Dega Dude"
  - The Moon Revisited, by Tom Duff
  - Soft Shadow Ideas, compiled by Derek Woolverton
  - Book Announcement: Multiprocessor Methods for Computer Graphics Rendering, by Scott Whitman
- 

**RTNv5n1: July 10, 1992**

- Introduction - SIGGRAPH roundtable, etc
  - New People, New Addresses, etc
  - Texturing Parameterization, by Haakan "Zap" Andersson
  - NuGrid results, by Mike Gigante
  - Recursive Ray Traversal, by Erik Jansen and Wim de Leeuw, Response by Kelvin Sung
  - Ideal Grid/Object Densities, by Dan Gehlhaar, Marc Andreessen
  - BVH Traversal Results, by Nicholas Wilt
  - Ray Tracing Roundup, by Eric Haines
  - Mail Based 3D File Server, by Bob Lindabury
  - Imagine That, by Steve Worley
  - Correct Roots for Torus Intersection, by Haakan "Zap" Andersson
  - Information on Taos Parallel Processor, by Paul Wain
  - The Glazing Trick, by Haakan "Zap" Andersson
  - Bug in Ray-Convex Polyhedron Intersector in Graphics Gems II, Eric Haines
- 

**RTNv4n3: November 18, 1991**

- Introduction
  - New People, Address Changes, etc
  - ElectroGig Free Software Offer
  - Spectrum: A Proposed Image Synthesis Architecture, by Andrew Glassner
  - Spline Intersection, Texture Mapping, and Whatnot, by Rick Turner
  - Satellite Image Interpretation, by Andy Newton
  - Material Properties, by Ken Turkowski
  - New Library of 3D Objects Available via FTP, by Steve Worley
  - Object Oriented Ray Tracing Book
  - New and Updated Ray Tracing and Radiosity Bibliographies
  - DKBTrace 2.12 Port to Mac, by Thomas Okken
  - Graphics Gems II Source Code
  - Radiance Digest Archive, by Greg Ward
  - Model Generation Software, by Paul D. Bourke
  - Rayshade 4.0 Release, Patches 1 & 2, and DOS Port, by Craig Kolb and Rod Bogart
  - RayShade Timings, by Craig Kolb
  - RayShade vs. DKBtrace Timings, by Iain Dick Sinclair
  - PVRay Beta Release, by David Buck
  - Vort 2.1 Release, by Eric H. Echidna
  - BRL-CAD 4.0 Release, by Michael J. Muuss and Glenn M. Gillis
- 

## **RTNv4n2: July 15, 1991**

- Introduction - SIGGRAPH get-together, etc
- New People, Address Changes, etc
- Ray Tracing Related FTP sites, compiled by Eric Haines
- Ray Tracing, the way I do it, by Haakan 'Zap' Andersson
- More Thoughts on Anti-Aliasing, by John Woolverton
- Spatial Measures for Accelerated Ray Tracing, by John Spackman
- Barcelona Workshop Summary, by Arjan Kok
- Book Announcement, from Stuart Green
- Spiral Scene Generator, by Tom Wilson
- An Announcement From The 'Paper Bank' Project, by Juhana Kouhia
- Radiance 1.4 via FTP, by Greg Ward
- Proceedings of Graphics Interface '91 Availability, by Irene Gargantini
- NFF Previewers, by Bernie Kirby, Patrick Flynn, Mike Gigante, Eric Haines
- RayTracker Demos Available, by Jari Kahkonen

- RayTracker Info, by Zap Andersson
- 

## **RTNv4n1: March 1, 1991**

- Introduction
- New People, Address Changes, etc
- Ray Tracing Abstract Collection, by Tom Wilson
- Report on Lausanne Hardware Workshop, by Erik Jansen
- New Version of SPD Now Available, by Eric Haines
- Teapot Timings, by John Spackman
- The Very First Point in Polygon Publication, by Chris Schoeneman
- The Acne Problem, by Christophe Schlick
- Shirley Thesis Available via Anonymous FTP, by Pete Shirley
- Some Thoughts On Anti-aliasing, by Zap Anderssen, Eric Haines
- New VORT Release, and the VORT Chess Set, by Eric H. Echnida, David Hook
- Best (or at least Better) of RT News, by Tom Wilson
- At Long Last, Rayshade v4.0 is Available for Beta-testing, by Craig Kolb
- Parallel Ray Tracer, by Kory Hamzeh, Mike Muuss, Richard Webb
- Distributed DKB, by Jonas Yngvesson
- Quadrangle Tessellation, by Christophe Schlick
- New Release of Radiance Software & Bug Fixes, by Greg Ward
- Radiance Timings on a Stardent, by Eric Ost
- RTSYSTEM Fast Ray Tracer, by Patrick Aalto
- DKBTrace Beta Available, by David Buck
- "Computer Graphics" 2nd Edition Corrections, Software, etc by Brown Emailer
- Papers which are currently available from the Net via anon. FTP, J. Kouhia

### =====Net News Cullings=====

- Ray-Cylinder Intersection Tutorial, by Mark VandeWettering
- C++ Fractal and Ray Tracing Book, by Fractalman
- Ray/Spline Intersection Reference, by Spencer Thomas
- Superquadric Intersection, by Rick Speer, Michael B. Carter
- Comments on Interval Arithmetic, by Mark VandeWettering, Don Mitchell
- Platonic Solids, by Alan Paeth
- Moon Problems, by Ken Musgrave, Pete Shirley
- Ken gave another problem with rendering the moon:
- Shadow Testing Simplification, by Tom Wilson
- SIMD Parallel Ray Tracing, by George Kyriazis, Rick Speer

- Rayscene Animator, by Jari K{hk|nen [sic]
  - SIPP 2.0 3d Rendering Package, by Jonas Yngvesson and Inge Wallin
  - 3DDDA Comments, by John Spackman
  - Radiosity Implementation Available via FTP, by Sumant
  - Dirt, by Paul Crowley
  - Thomson Digital Images University Donation Program, by Michael Takayama
  - A Brief Summary of Ray Tracing Related Stuff, Angus Y. Montgomery
- 

## **RTNv3n4: October 1, 1990**

- Introduction
- New People, Address Changes, etc
- Photorealism, and the color Pink (TM), from Andrew Glassner
- DKBTrace v2.0 and Ray Tracing BBS Announcement, by David Buck, Aaron Collins
- Article Summaries from Eurographics Workshop, by Pete Shirley
- Convex Polyhedron Intersection via Kay & Kajiya, by Eric Haines
- New Radiosity Bibliography Available, by Eric Haines
- A Suggestion for Speeding Up Shadow Testing Using Voxels, by Andrew Pearce
- Real3d, passed on by Juhana Kouhia, "Zap" Andersson
- Utah Raster Toolkit Patch, by Spencer Thomas
- NFF Shell Database, by Thierry Leconte
- FTP list update and New Software, by Eric Haines, George Kyriazis

### =====Net News Cullings=====

- Humorous Anecdotes, by J. Eric Townsend, Greg Richter, Michael McCool, Eric Haines
  - Graphics Interface '91 CFP
  - Parametric Surface Reference, by Spencer Thomas
  - Solid Light Sources Reference, by Steve Hollasch, Philip Schneider
  - Graphics Gems Source Code Available, by Andrew Glassner, David Hook
  - Graphics Gems Volume 2 CFP, by Sari Kalin
  - Foley, Van Dam, Feiner and Hughes "Computer Graphics" Bug Reports, by Steve Feiner
  - Radiosity via Ray Tracing, by Pete Shirley
  - Algorithm Order Discussion, by Masataka Ohta, Pete Shirley
  - Point in Polygon, One More Time..., by Mark Vandewettering, Eric Haines, Edward John Kalenda, Richard Parent, Sam Uselton, "Zap" Andersson, and Bruce Holloway
  - Quadrant Counting Polygon In/Out Testing, by Craig Kolb, Ken McElvain
  - Computer Graphics Journals, by Juhana Kouhia
-

## **RTNv3n3: July 13, 1990**

- Introduction
- Ray Tracing Roundtable Announcement
- New People, Address Changes, etc
- Jarke van Wijk Thesis Availability, by Erik Jansen
- New Name For "Distributed Ray Tracing", by Paul Heckbert et al
- NFF Files from Thierry Leconte
- RADIANCE Software Available, by Greg Ward
- Rayshade Updates & SPD Bug, by Craig Kolb
- New Version of Vort Ray Tracer, by David Hook
- Graphics Interface '90, by Eric Haines
- Real3d Review, Haakan "ZAP" Andersson
- Bits From a Letter by David Jevans
- On Antialiasing, & Light and Such, by Haakan "ZAP" Andersson

### =====Net News Cullings=====

- Summary: Uniform Random Distribution of Points on Sphere's Surface, Marshall Cline
- Ray Tracing & Radiosity, by Frank Vance, Mark VandeWettering
- Ray-Tracing the Torus, by Prem Subrahmanyam, Bob Webber
- Need Help on Superquadrics, by Wayne McCormick, Robert Skinner
- Ray Tracing Penumbra Shadows, Prem Subrahmanyam
- Ray with Bicubic Patch Intersection Problem, Wayne Knapp, John Peterson, Lawrence Kesteloot, Mark VandeWettering, Thomas Williams
- Rendering Intersecting Glass Spheres, by John Cristy, Craig Kolb
- DKBPC Raytracer, by Tom Friedel
- New release of IRIT solid modeller, by Gershon Elber
- Easier Use of Ray Tracers, by Philip Colmer, Mark VandeWettering, Jack Ritter
- Raytracer Glass, by F. Ken Musgrave, Michael A. Kelly
- Ray Intersection with Grid, by Alasdair Donald Robert McIntyre, Rick Speer
- Database for DBW-Render, by Prem Subrahmanyam

---

## **RTNv3n2: March 20, 1990**

- Introduction
- New People, Address Changes, etc
- FTP Site List, by Eric Haines
- RayShade Posting and Patches and Whatnot, by Craig Kolb

- Common Rendering Language, by Eric Haines
- Avoiding Re-Intersection Testing, by Eric Haines
- Torus Equation Correction, by Pat Hanrahan
- "Introduction to Ray Tracing" Shading Model Errata, by Kathy Kershaw
- Comments on Last Issue, by Mark VandeWettering
- An Improvement to Goldsmith/Salmon, by Jeff Goldsmith
- Fiddling with the Normal Vector, by H. 'ZAP' Anderson
- A Note on Texture Sampling, by Eric Haines
- Unofficial MTV Patches, by Eric Haines

=====Net News Cullings=====

- OFF Databases, by Randi Rost
- VM\_pRAY is now available, by Didier Badouel
- Superquadrics, by Prem Subrahmanyam, Patrick Flynn, Ron Daniel, and Mark VandeWettering
- Graphics Textbook Recommendations, by Paul Heckbert, Mark VandeWettering, and Kent Paul Dolan
- Where To Find U.S. Map Data, by Dru Nelson
- References for Rendering Height Fields, Mark VandeWettering
- RayShade 3.0 Released on comp.sources.unix, by Craig Kolb
- Bibliography of Texture Mapping & Image Warping, by Paul Heckbert
- More Texturing Functions, by Jon Buller
- Ray/Cylinder Intersection, by Mark VandeWettering
- C Code for Intersecting Quadrics, by Prem Subrahmanyam
- Parallel Ray Tracing on IRISs, collected by Doug Turner

---

**RTNv3n1: January 2, 1990**

- Introduction
- New People
- Archive Site for Ray Tracing News, by Kory Hamzeh
- $K_s + T > 1$ , by Craig Kolb and Eric Haines
- Quartic Roots, and "Intro to RT" Errata, by Larry Gritz and Eric Haines
- More on Quartics, by Larry Spence
- Question: Kay and Kajiya Slabs for Arbitrary Quadrics? by Thomas C. Palmer
- Ambient Term, by Pierre Poulin
- Book Reviews on Hierarchical Data Structures of Hanan Samet, by A. T. Campbell, III
- Comparison of Kolb, Haines, and MTV Ray Tracers, Part I, by Eric Haines
- Raytracer Performance of MTV, by Steve Lamont



- BRL-CAD Ray Tracer Timings, by Gavin Bell
- BRL-CAD Benchmarking and Parallelism, by Mike Muuss

=====Net News Cullings=====

- Rayshade Patches Available, by Craig Kolb
- Research and Timings from IRISA, by Didier Badouel
- Concerning Smart Pixels, by John S. Watson
- Input Files for DBW Render, by Tad Guy
- Intersection with Rotated Cubic Curve Reference, by Richard Bartels
- Needed: Quartz surface characteristics, by Mike Macgirvin
- Solution to Smallest Sphere Enclosing a Set of Points, by Tom Shermer
- True Integration of Linear/Area Lights, by Kevin Picott

---

[RTNews9](#) v2n8: October 27, 1989

- Introduction
- Tracing Tricks, edited by Eric Haines Ambient Light Efficiency Schemes Bounding Volume Hierarchy Octree Bounding Box Intersection Spline Surface Intersection Acknowledgements Bibliography

---

[RTNews8](#) v2n7: October 13, 1989

- Introduction
- New People and Address Changes
- Solid Surface Modeler Information, by Eric Haines
- Minimum Bounding Sphere Program, by Marshall Levine
- Parallelism & Modeler Info Request, by Brian Corrie

=====Net News Cullings=====

- Ray Tracer Available, by Craig Kolb
- Source from Roy Hall's Book, by Tim O'Connor
- More on Texture Mapping by Spatial Position, by Paul Lalonde
- Procedural Bump-mapping Query, by Prem Subrahmanyam
- Ray Tracer Performance on Machines, by Gavin A. Bell, Steve Lamont
- Projective Mapping Explanation, by Ken "Turk" Turkowski
- Intersection Calculation Problem Request, Jari Toivanen
- Mathematical Elements for Computer Graphics - Call for Errata, by David Rogers
- Raytracing on NCUBE Request, by Ping Kang Hsiung
- Intersection Between a Line and a Polygon (UNDECIDABLE??), by Dave Baraff, Tom Duff



## [RTNews8](#) v2n6: September 20, 1989

- Introduction
- New People and Address Changes
- Q&A on Radiosity Using Ray Tracing, by Mark VandeWettering and John Wallace
- Dark Bulbs, by Eric Haines
- MTV Ray Tracer Update and Bugfix, by Mark VandeWettering
- DBW Ray Tracer Description

### =====Net News Cullings=====

- Wanted: Easy Ray/Torus Intersection, by Jochen Schwarze
  - Polygon to Patch NFF Filter, by Didier Badouel
  - Texture Mapping Resources, by Eric Haines, Prem Subrahmanyam, Ranjit Bhatnagar, and Jack Ritter
- 

## [RTNews7](#) v2n5: August 29, 1989

- Introduction
- A SIGGRAPH Report, by Eric Haines
- Ray Tracing Poll, from the roundtable discussion
- \_An Introduction to Ray Tracing\_, Announcement and Errata
- \_Graphics Gems\_ Call for Contributions, by Andrew Glassner
- New People and Address Changes
- Bugs in MTV's Ray Tracer, by Eric Haines
- Bug in SPD, from Pete Segal
- Solid Textures Tidbit, by Roman Kuchkuda
- Sundry Comments, by Jeff Goldsmith
- Texture Mapping Question, by Susan Spach

### =====Net News Cullings=====

- Ray Traced Image Files, Prem Subramanyan
  - Image Collection, by Paul Raveling
  - MTV-Raytracer on ATARI ST - precision error report, by Dan Riley
  - Question on Ray Tracing Bicubic Parametric Patches, by Robert Minsk
- 

## [RTNews7](#) v2n4: June 21, 1989

- Introduction
- Hardcopy News

- New (and Used?) People
- Minimum Bounding Sphere, continued, by Jack Ritter
- Comments on "A Review of Multi-Computer Ray-Tracing", by Thierry Priol
- Query: Dataflow architectures and Ray Tracing, by George Kyriazis

=====Net News Cullings=====

- Re: Pixar's noise function, by Jon Buller
  - Re: DBW\_render for SUN 3? by Tad Guy
  - Re: Steel Colors, by Eugene Miya
  - Dirty Little Tricks, by Jack Ritter
  - Obfuscated Ray Tracer, by George Kyriazis
  - Contents of FTP archives, skinner.cs.uoregon.edu, by Mark VandeWettering
- 

[\*\*RTNews7\*\*](#) v2n3: May 12, 1989

- Introduction
  - New People, by Carl Bass, Paul Wanuga
  - QRT Ray Tracer (and five other Amiga Ray Tracers), by Steve Koren
  - New Version of MTV Ray Tracer, by Mark VandeWettering
  - Re: Ray Traced Bounding Spheres, by Earl Culham
  - Noise and Turbulence Function Code, Pascal and C, by Jon Buller, William Dirks
- 

[\*\*RTNews6\*\*](#) v2n2: February 20, 1989

- Introduction, by Eric Haines
  - New Subscribers, by Turner Whitted, Mike Muuss
  - The BRL CAD Package, by Mike Muuss
  - New Book: \_Illumination and Color in Computer Generated Imagery\_, Roy Hall (Eric Haines)
  - Uniform Distribution of Sample Points on a Surface
  - Depth of Field Problem, by Marinko Laban
  - Query on Frequency Dependent Reflectance, by Mark Reichert
  - "Best of comp.graphics" ftp Site, by Raymond Brand
  - Notes on Frequency Dependent Refraction
  - Sound Tracing
  - Laser Speckle
- 

[\*\*RTNews6\*\*](#) v2n1: January 6, 1989

- Introduction, by Eric Haines
  - New Members, by David Jevans, Subrata Dasgupta, Darwin Thielman, Steven Stadnicki, Mark Reichert
  - Multiprocessor Visualization of Parametric Surfaces, by Markku Tamminen, comments from many others
  - Miscellany, by K.R.Subramanian, David F. Rogers, Steven Stadnicki, Joe Smith, Mark Reichert, Tracey Bernath
  - Supersampling Discussion, by David Jevans, Alan Paeth, Andrew Woo, Loren Carpenter
  - Distributed Ray Tracer Available, by George Kyriazis
  - Ray Tracing Program for 3b1, by Sid Grange
  - Map Archive, by Gary L. Crum
  - Index of Back Issues, by Eric Haines
- 

## [RTNews5](#) v1n11: November 4, 1988

- Intro
  - New People
  - Ray/Triangle Intersection with Barycentric Coordinates, by Rod Bogart, Jeff Arenberg
  - Transforming normals, by David F. Rogers
  - 2D box-test, by Jack van Wijk
  - Re: Neutral File Format, by Jeff Goldsmith
  - RT and Applications, by Cary Scofield
  - Re: Goldsmith and Eyes, by K.R.Subramanian
  - Wood Textures, by Rod Bogart
  - Shadows, Mirrors, and "Virtual Lighting", by Steve Stadnicki
  - Re: Basics of Raytracing, by David Jevans
  - Re: What is Renderman Standard?, by Steve Upstill
  - Free On-Line Computer Graphics References
  - Latest Mailing List, Short Form, by Eric Haines
- 

## [RTNews5](#) v1n10: October 3, 1988

- Intro
- New Addresses and People
- Bitmap Stuff, by Jeff Goldsmith
- More Comments on Kay/Kajiya
- Questions and Answers (for want of a better name)
- More on MTV's Public Domain Ray Tracer (features, bug fixes, etc)

- Neutral File Format (NFF), by Eric Haines
- 

## [RTNews4](#) v1n9: September 11, 1988

- Intro
  - Capsule Autobiographies, by more new people
  - The Continuing Saga of MTV's Raytracer, by Mark VandeWettering
  - Public Domain Ray Tracer Q & A, by Mark VandeWettering
  - Public Domain Ray Tracer Utilities, by Tom Vijlbrief
  - Sorting Unnecessary on Shadow Rays for Kay/Kajiya? by Eric Haines and Mark VandeWettering
  - Summary of Replies to Vectorizing Ray-Object Intersection Query, by Tom Palmer
  - The Ray Tracer I Wrote, by George Kyriazis
  - New Bitmaps Library Available, Jef Poskanzer
- 

## [RTNews4](#) v1n8: September 5, 1988

- Intro
  - Capsule Biographies
  - SIGGRAPH '88 RT Roundtable Summary, by Paul Strauss and Jeff Goldsmith
  - Commercial Ray Tracing Software and SIGGRAPH, by Eric Haines and others
  - A Letter, by Jeff Goldsmith
  - Best of USENET
  - Postscript Ray Tracer, John Hartman and Paul Heckbert
- 

## [RTNews3](#) v1n7: June 20, 1988

- Introduction
  - New People and Addresses
  - Non-article on RenderMan and Dore', by Eric Haines
  - Ray Tracing Bibliography Update, by Paul Heckbert and Eric Haines
  - Commercial Ray Tracing Software, by Eric Haines
  - Benchmarks, by Jeff Goldsmith
- 

## [RTNews3](#) v1n6: April 6, 1988

- RT News, Hardcopy Form, by Andrew Glassner
- New People

- Question for the Day, by Rod Bogart
  - Re: Linear-time Voxel Walking for BSP, by Erik Jansen
  - Some Thoughts on the Theory of RT Efficiency, by Jim Arvo
  - Automatic Creation of Object Hierarchies for Ray Tracing, by Eric Haines
  - Best of USENET
- 

## [RTNews2](#) v1n5: March 26, 1988

- Intro, Eric Haines
  - Mailing list changes and additions: Kuchkuda, Lorig, Rekola
  - More on shadow testing, efficiency, etc., Jeff Goldsmith
  - More comments on tight fitting octrees for quadrics, Jeff Goldsmith
  - LINEAR-TIME VOXEL WALKING FOR OCTREES, Jim Arvo
  - Efficiency Tricks, Eric Haines
  - A Rendering Trick and a Puzzle, Eric Haines
  - PECG correction, David Rogers
- 

## [RTNews2](#) v1n4: March 8, 1988

- Surface acne, by Eric Haines
- Goldsmith/Salmon hierarchy building
- Efficiency tricks followup, by Ohta, Goldsmith, Haines

=====Net News Cullings=====

- Spatial Subdivision, by Ruud Waij, Paul Heckbert, Andrew Glassner
- 

## [RTNews2](#) v1n3: March 1, 1988

- Mailing list updates
  - Another Dore' article
  - Teapot in a football stadium, by Glassner, Arvo, Haines
  - Efficiency Tricks, by Jeff Goldsmith
  - More book recommendations, by Jeff Goldsmith
  - Bug for the day, by Eric Haines
  - A pet peeve, by Jeff Goldsmith
- 

## [RTNews2](#) v1n2: February 15, 1988

- Dore'
- 

## [RTNews1](#) v1n1: January 15, 1988

- Introduction
  - Subdivision and CSG, by Erik Jansen
  - Spline surface rendering, and what's wrong with octrees, by Eric Haines
  - Top ten hit parade of computer graphics books, by Eric Haines
  - Normal vectors and octree relies, by Olin Lathrop and Eric Haines
  - Subspaces and simulated annealing, by Jim Arvo
- 

## [RTNews1](#) the start: Sometime after Siggraph '87

- Introduction
  - SPD and NETLIB, by Eric Haines
  - Spline Surfaces, by John Peterson, Haines, Goldsmith, Kay
  - Abnormal Normals, by Eric Haines
- 

[Eric Haines](#) / [erich@acm.org](mailto:erich@acm.org)

# Real-Time Rendering

This is the Web site for the book *Real-Time Rendering*, by [Tomas Möller](#) and [Eric Haines](#), 512 pages, from [A.K. Peters Ltd.](#), \$49.95, ISBN 1-56881-101-2.

Book related resources include:

- The [Corrections](#) area.
- The [Bibliography](#) of the book, with linked resources.
- The [Preface](#) and [Chapter Overview](#) of the book.
- A [portal page](#) for real-time web resources Eric uses frequently.
- The first draft of our [book recommendation list for real-time computer graphics](#); we would appreciate your comments.
- Sample sections of the book in PDF and Postscript - see the table below.
- [Fantasy Graphics League](#)



[click to see larger image](#)

You can order the book from [Amazon](#), [Barnes & Noble](#), [Amazon UK](#), or direct from [A.K. Peters](#) (or in the UK, [orders@plymbridge.com](mailto:orders@plymbridge.com)), among others. Check the [DealTime](#) or the [Click the Button](#) searchers for various deals.

The first printing lists this web site as <http://www.acm.org/tog/resources/RTR/>, but this page and others are now mirrored at <http://www.realtimerendering.com>. Thanks to all who bought this and other books by clicking through to [Amazon](#) and [B&N](#); money earned by doing so has paid for this site's costs for the next few years.

Here are the book chapters and links to resources (in column-major order):

<a href="#">Introduction</a>	<a href="#">Texturing</a>	<a href="#">Polygonal Techniques</a>	<a href="#">The Future (resources)</a>
<a href="#">Rendering Pipeline</a>	<a href="#">Special Effects</a>	<a href="#">Intersection Testing</a>	<a href="#">Linear Algebra</a>
<a href="#">Transforms</a>	<a href="#">Speed-Up Techniques</a>	<a href="#">Collision Detection</a>	<a href="#">Trigonometry</a>
<a href="#">Visual Appearance</a>	<a href="#">Pipeline Optimization</a>	<a href="#">Graphics Hardware</a>	<a href="#">Bibliography</a>

The following sample book sections are beta versions; the final drafts are similar but with a few improvements and corrections.

Book Section	Adobe PDF	PDF in zip	Postscript
Table of Contents	<a href="#">205K</a>	<a href="#">72K</a>	<a href="#">503K</a>
Image-Based Rendering and Lens Flares	<a href="#">290K</a>	<a href="#">145K</a>	<a href="#">4,118K</a>



Gouraud Bump Mapping	<a href="#">389K</a>	<a href="#">181K</a>	<a href="#">1,433K</a>
Ray/Box and Ray/Triangle Intersection	<a href="#">464K</a>	<a href="#">178K</a>	<a href="#">611K</a>
Index	<a href="#">621K</a>	<a href="#">242K</a>	<a href="#">1,008K</a>

[Gamasutra](#) has published an excerpt from the book, [Section 7.1.5: Occlusion Culling](#). [GameDev.Net](#) has an HTML version of the [Image-Based Rendering and Lens Flares](#) section available online. GIG has an interview with us both about [how we wrote the book](#) and an excerpt about [transforming normals](#).

Since we are constantly adding new links (and deleting stale ones), [the newest additions are shown in this color, fading to this color, then this color](#), then to black, over a period of a few months. If you want to know whenever this page changes, you could use [Spyonit.com](#) to alert you.

## Introduction

The [ACM Transactions on Graphics site](#) has links to free [computer graphics code](#) and [literature](#) on the Web. There are other good links sites, including [Technomagi's](#), [Magic Software's](#), and [Frédo Durand's](#).

[Game Developer magazine's site](#) has tutorials and other information on real-time programming techniques, including computer graphics articles. Related to this publication is [Gamasutra](#), which also has many resources.

For many useful links, see [Karim Ratib's page](#). The Applications.Publications subdirectory here has links to many online versions of published articles.

[Sample chapters are available](#) [for the new book](#) [Computer Graphics Using OpenGL](#) by F.S. Hill, Jr.

The [Exploratory](#) at Brown University has a wide range of computer graphics applets. These cover areas such as geometry, color science, image processing, and hierarchy of transformations. Patrick Min at Princeton has also made [Java applets](#) to aid in understanding computer graphics concepts.

[Intel has a short course](#) on the basics of 3D interactive rendering. [HyperGraph](#) covers the basics on many topics within computer graphics. An [extensive reference](#) for many elements of computer graphics is available from the University of Waterloo. A source for quick summaries of various graphics algorithms is [The White Flame Page of Learning](#). [Zed3d](#), an entire book on introductory and intermediate graphics (with code), is available online for free. It overlaps with a number of topics our book covers, with a different view of them.

There are a number of online glossaries of terms. [Exaflop](#) has one with some illustrations, [Olin Lathrop's book](#) gives a more general one, Apple has a [Mac oriented one](#), [Webopedia](#) a short general one, [Mondo Labs](#) a modeling package oriented one, and [Nintendo](#), [3D Gaming](#), and [GCS Extreme](#) have games-oriented graphics glossaries. [Schorsch has a glossary of lighting design terms available online](#).



Some books which have come out recently and may be of interest:

- [Game Engine Design: A Practical Approach to Real-Time Computer Graphics](#), by Dave Eberly ([his code](#) is online)
- [Game Programming Gems](#), by Mark DeLoura
- [3D Games, Volume 1: Real-time Rendering and Software Technology](#), by Alan Watt and Fabio Polcarpo ([their Fly3D SDK](#) is online)
- [3D Graphics Programming: Games and Beyond](#), by Sergei Savchenko
- [Mathematics for Computer Graphics Applications](#), by Michael Mortenson

## Rendering Pipeline

Karsten Isakovic maintains an all-encompassing [3D engines list](#), many with source code available. One free source engine worth pointing out is [Mesa](#), an OpenGL clone which runs on almost everything and even has some accelerator support. SGI themselves have released a [sample OpenGL implementation](#) to the open source community, to [encourage Linux driver creation](#).

Sim Dietrich has a presentation on [Guard Band Clipping](#), an accelerator assisted clipping scheme which will see more use in hardware. More on guard band clipping can be found in this [extremely silly interview](#).

Ronen Zohar has an article about [polygon clipping](#) and optimizing it on the Pentium III using SSE instructions.

An excellent article on [pure software rendering](#), with all its tricks and pitfalls, is detailed by Charles Bloom.

## Transforms

The [Portable Game Library](#) includes a [Simple Geometry library](#).

A number of chips have support for [vertex skinning](#) (a.k.a. vertex blending). The basic idea of this technique is to be able to transform patches which connect relatively rigid elements (e.g. patches at shoulders or knees) by using sets of matrices to transform the patches. The basic idea is to interpolate among the matrix transform results by using weights based upon each vertice's initial location. This allows a single triangle to deform (stretch like skin) and keep continuity with the rigid elements, as each vertex is given a different weight. More at [NVIDIA's site](#) and [ATI's site](#).

[Code](#) for rotating from one vector to another rapidly using quaternions (as described in the book) is available online. [If you need a further explanation of quaternions, you might try this article.](#)

Patrick Min at Princeton has made a number of graphics tutorial Java applets available on the Web. The code is also available. The applets include:

- [Two dimensional transform experimenter](#)
- [Three dimensional viewing](#)

The [Exploratory](#) has many applets related to linear algebra and transforms.

[Paul Bourke's site](#) includes information on geometry and projection, and includes many basic geometric operations as well as more advanced (e.g. [anamorphic projections](#)). [Flipcode](#) has a [3D geometry primer online](#).

An excerpt from our book about transforming normals is [available at GIG](#), and includes source code for computing the adjoint.

[Dave Eberly's site](#) has useful papers and code on a wide variety of geometric operations, including quaternion interpolation. Nick Bobick also has a nice article on [quaternion rotation and interpolation](#).

The [Graphics Gems](#) book series contains a number of good articles on transformations. The table of contents is online, and the code from the books is free to download.

Sample chapters from the book [The Geometry Toolbox](#) are available for download. One chapter is about affine maps in 3D, the other about curves. The affine transforms chapter is definitely worth reading if you want to build up your intuition and mathematical understanding of translation, projection, etc. The curves chapter discusses Bézier curves in depth.

The book [Numerical Recipes](#) is available online, and contains information on topics such as linear and spline interpolation (and much else).

For mathematical definitions, turn first to [Eric Weisstein's World of Mathematics](#), one of the true Wonders of the Web ([Sadly, this is no more, but may return, so the link is left here](#)).

## Visual Appearance

[Pellucid](#) is a Java applet that allows you to try out the VRML lighting model, which is similar to the lighting model covered in this chapter. A more elaborate lighting model applet can be found at [Patrick Min's site](#). Source is available for both applets. The [Exploratory](#) has applets concerning lighting models, color science, and signal processing.

The [webreference site](#) has a number of tutorials on lighting and other three dimensional graphics topics. Many of the tutorials are for software users, but there is some good general introductory material here.

The paper [Phong Shading at Gouraud Speed](#) gives a way of simplifying Phong interpolation so that it is faster.

An excellent [non-photorealistic rendering](#) (NPR) resources page has been put together by Craig Reynolds. Amy and Bruce Gooch have a [nicely organized NPR page](#), in anticipation of their [upcoming book](#) on the subject. Adam Lake's (wonderful) [paper on real-time NPR techniques](#) is available online. Jeff Lander has an [executable and sample code](#) showing one technique for NPR. Sim Dietrich has a



presentation on other [NPR techniques using the GeForce](#). Some evocative images using sphere mapping for realtime NPR effects can be found on [Kenneth Hoff III's page](#). More can be found on [Intel's page](#). Ramesh Raskar [silhouette edge paper](#) and [source code](#) gives techniques for generating these without needing connectivity or database information.

Scott R. Nelson's [antialiased line code](#) is available for download. Here is a comparison generated using his program of lines improperly drawn with [gamma=1.0](#) (note the severe roping and Moiré patterns) and properly drawn with [gamma=2.2](#). Note that you must view these files with a 2.2 gamma display system (e.g. on a PC).

[Poynton's color technology site](#) talks about gamma correction, color spaces, and proper monitor settings. It includes a number of papers available for download. The [CGSD site](#) also has some good [documents explaining gamma](#). [Robert Berger's page](#) also has a good quick explanation of gamma correction, as well as a monitor test.

Stanford is working on a project exploring techniques for [programmable shading](#); code is included and you can write your own shaders. Slides from their SIGGRAPH 2000 talks are available. SGI has been investigating hardware-accelerated programmable shading, as described in [Interactive Multi-Pass Programmable Shading](#). The [compiler software](#) is available for IRIX. The paper [Illuminating Micro Geometry Based on Precomputed Visibility](#) is about using graphics hardware capabilities to render bump maps with more realistic shading models and including some global illumination interreflection effects. A number of [publications by Kautz and others](#) are available with techniques to simulate a variety of materials by using environment mapping.

For information on the physical basis of illumination functions, see [NIST's optical reflectance](#) pages. For spectral reflectance and emission data, see Glassner's [Principles of Digital Image Synthesis](#) Web site. Errata for the book is also available at this site.

Microsoft's DirectX 8 has support for vertex and pixel shading (essentially generalized multitexturing) on the graphics accelerator; see their [DirectX 8 documentation](#) and [NVIDIA's developer site](#). Vertex shading gives an assembly language in which you can program how a vertex is shaded by the graphics accelerator. While software can shade and modify vertices in any way desired before being sent to the accelerator, accelerated transform and lighting is normally more limited. Vertex shading allows this part of the hardware's pipeline to be programmed. Pixel shading is a way of programming how various texture stages are combined. Different hardware supports different numbers of texture stages and how they can interact. See [NVIDIA's texture combiners documentation](#) for an example of one implementation.

## Texturing

The invaluable SIGGRAPH 99 course notes for *Advanced Graphics Programming Techniques Using OpenGL* and *Lighting and Shading Techniques for Interactive Applications* are [available online](#). These contain an incredible amount of information on shading, texturing, and special effects.

[NVIDIA's Developer Relations Site](#) includes demos, white papers, and presentations on stencil buffering,

texturing, anisotropic lighting, paraboloid mapping, vertex blending, and much more.

ATI's Radeon line includes paraboloid mapping and 3D textures, among other capabilities - see their [Pixel Tapestry page](#) for many interesting examples. [ATI's Radeon page](#) includes tutorials and demo code for projective textures, bump mapping, shadows, and many other topics.

Gamasutra has an extensive articles on [mipmapping techniques](#), [multitexturing](#), [bump mapping](#), and using [procedural textures](#) in conjunction with hardware; [there is more on this at Intel's site](#). The [bump mapping](#) article in particular is complementary to our book, as it includes extensive coverage of the environment map bump method (a technique we are surprised to find graphics chip makers now starting to include in their offerings). An article on [dot-product bump mapping using OpenGL](#) lays out the specific commands needed, has excellent tutorial images, and provides demo code for download. Another overview of bump mapping techniques can be found on [PVR-NET](#). To give a brief summary of who is doing what bump mapping technique:

- Embossing: any chip with multitexturing, and 3DLabs' Permedia 3 has a dedicated embossing unit.
- Environment bump mapping: ATI Radeon, Matrox G400, Permedia 3, and Glaze 3D.
- Dot product bump mapping: NVIDIA GeForce and GeForce2, ATI Radeon, VideoLogic PowerVR-250, and Permedia 3.

[Egerter's Power Render](#) has a number of demos showing off the Matrox G400's environment bump mapping capability. Even if you don't have a G400, there are QuickTime movies of the effect.

[Mark Kilgard's SGI site](#) has much useful information on texture mapping techniques and OpenGL. Examples for many of these are included in his [OpenGL GLUT \(GL Utilities Toolkit\) source code](#).

Some fascinating applications of texture mapping can be seen on [Angus Dorbie's site](#).

[S3TC texture compression](#) has become a standard part of DirectX, and is called [DXTn texture compression](#).

The PowerVR architecture's [vector quantization](#) texture compression scheme works by maintaining a code book of small image tiles. [Microsoft has licensed NVIDIA's Volume Texture Compression \(VTC\) technology](#) for 3D textures for inclusion into DirectX.

Heckbert has written a worthwhile [Survey of Texture Mapping](#) and a more in-depth report, [Fundamentals of Texture Mapping and Image Warping](#).

Two important elements in forming mipmaps are good filtering and gamma correction. The common way to form a mipmap level is to take each 2x2 set of pixels and average them to get the mip value. This is fraught with peril: first, the filter used is then a box filter, one of the worst filters possible. Better is to use a [Gaussian filter](#) or similar, and it is not all that much more complicated to code. By ignoring gamma correction, the overall perceived brightness of the mipmap level will be different than the original texture: as you get farther away from something and the uncorrected mipmaps kick in, the object will usually look darker.

A great [history of reflection mapping](#) is available from Paul Debevec's site. Some normally difficult to



obtain early papers and videos can be found here. [High dynamic range environment map image data is also available at Paul's site, along with 8 bit/channel spherical map images.](#) [Gamasutra has two articles on performing refractive mapping.](#)

An article on [solving problems with sphere mapping while using DirectX](#) is available at Gamasutra.

[Chris Hecker](#) has written extensively on perspective correct texture mapping. The [good-looking textured light-sourced bouncy fun smart and stretchy page](#) also has a little on this topic, as well as how to do real-time Phong shading and bump mapping.

A Web site discussing [anisotropic reflections](#) done using texture maps also has an article on the topic available for download. Many other interesting applications of texture mapping are discussed at [Paul Haeberli's site](#). Another site showing some excellent results from using multipass rendering with BRDF maps is the [University of Waterloo reflection models page](#). The report linked from here details how to use OpenGL to perform such surface shading. Two papers from SIGGRAPH 99 describe using more elaborate BRDF models, one by [Heidrich and Seidel](#) and another by Cabral, Olano, and Nemec on [SGI's ClearCoat technology](#). [Stanford and Berkeley's Appearance Models course reading list](#) is a good place to start for information on material appearance.

SIGGRAPH 99 Proceedings has a paper on higher quality anisotropic texture mapping in hardware, called the [FELINE system](#). [This article is similar to the approach published by Andreas Schilling, Gunter Knittel, and Wolfgang Strasser, "Texram: A Smart Memory for Texturing", IEEE Computer Graphics Applications, Vol. 16, No. 3, May 1996 \(a reference we missed\), and Barkans \[p. 116-7 of our book\].](#)

Rendering to a texture is an operation supported by DirectX 7 (though not necessarily by hardware). It permits many of the effects explained in our book. [Kim Pallister's article](#) on Gamasutra and longer version on [Intel's site](#) describe the specifics. [Offscreen rendering in general is described in Brian Paul's course notes.](#)

There are many sites with free textures out there. Some good places to start are [Axem](#), [Avalon](#), [The New Graphics BBS](#), and [3D Cafe's texture site list](#). Axem has, among their 800+ textures, trees with cutouts.

An [annotated set of links](#) to information on procedural texture generation is available online. Kim Pallister discusses [how to generate clouds on the fly](#) using procedural texturing.

## Special Effects

The [interactive rendering SIGGRAPH 99 course notes](#) are an incredible free resource covering all sorts of graphics techniques. Though OpenGL is the API used in examples, these documents are more about algorithms than APIs. Highly recommended.

[The pear demo used in our book is available for download.](#)

A variety of [demo programs and movies](#) are available for download. These show off a number of special effects supported by [Egert's Power Render](#) graphics engine.

Masaki Kawase has a [demo \(including source\)](#) for NVIDIA cards of projective texturing, shadow mapping, reflection, anisotropic filtering, and other effects. Ron Frazier has a [number of articles \(and code\)](#) discussing how NVIDIA's register combiners can be used to give complex shading, lighting, and shadowing effects. Mark Kilgard's [SGI site](#) is a treasure trove of useful tips on real-time rendering, including how to create [lens flares](#), [shadows and reflections](#), [particle systems](#), and much more. Mark's discussion and demo code for [stencil buffered shadows](#) (a.k.a. volumetric shadows) is now available on the [OpenGL organization site](#). An [article on Gamasutra](#) also covers this topic in detail. [Using stencil buffers in Direct3D](#) is discussed by Peter Kovach. NVIDIA also has a number of talks on special effects on their [Developer Relations](#) page. [ATI has demos and images on their developer sample pages](#). [Adam Moranvanszky has a page showing some real-time soft shadows made by blurring the projective texture after creating it.](#)

For fast 2D sprite display, read about [Compiled Sprites](#). The open-source [Allegro](#) game programming library implements many sprite-related functions (see the [Gamasutra article](#) for more about this package). [OpenPTC](#) is an open source library for quick blitting to the screen; Windows, X11, and Java versions are available.

Jed Lengyel's [The Convergence of Graphics and Vision](#) is a good quick introduction and overview of research done in image-based techniques. Andrew Glassner has more on [Chicken Crossing](#), a film made using image layers on a Talisman simulator.

Some excellent [QuickTime VR](#) panoramas (including the one shown in the book) are available for viewing on [Ken Turkowski's page](#).

To get a flavor of image based rendering, see the [Virtual Camera site's demos](#). Though there is little going on here algorithmically, it shows the power of capturing a scene from many angles at once.

Particle systems code can be found in the [Game Developer magazine code site](#) in the [July 1998 archive](#), and at [Dman's site](#). John van der Burg discusses [data structures for particle systems](#) on Gamasutra. A cool [Java applet](#) shows particle systems in action. David McAllister has a [particle API](#) available, along with some nice screen shots and links. [PyroTechnics](#) is a free OpenGL-based package for firework simulation.

To see the original soft reflection and transparency images by Paul Diefenbach, visit [his site](#). There are other interesting experiments in using graphics accelerators for global illumination here, as well as his thesis on the subject and a shorter article.

Michael McCool presents a [hybrid scheme](#) of using shadow maps to derive shadow volume boundaries. His [lecture notes](#) are also available. Udeshi and Hansen present an overview of shadowing techniques and their own hybrid method in [Towards interactive photorealistic rendering of indoor scenes: A hybrid approach](#). Zhang presents a [method of 3D image warping](#) to transform shadow maps to eye space. Herf and Heckbert's [soft shadow technique](#) paper and images are available online. Heckbert maintains a (slightly dated) [page on shadowing techniques](#). Charles Bloom has a useful "how to" guide for [generating shadow textures](#). Intel has information about [implementing shadow volumes](#). [Heidrich et alia's paper](#) [Soft](#)



[\*Shadow Maps for Linear Lights\*](#) uses two shadow maps and edge detection to generate polygons for approximating the penumbra.

A software-only [Java applet](#) that shows real-time mirror reflection works surprisingly quickly on fast machines.

Tutorials on some special effects can be found at the [good-looking textured light-sourced bouncy fun smart and stretchy page](#). Articles are found in the "Graphics" and "Models" areas near the bottom of the page; topics covered include particle systems and projective shadows.

ATI's Radeon includes a modified shadow buffer technique, using a hardware assisted [priority buffer](#). Better would be to call it an ID buffer; Hourcade's algorithm (see our book) is supported in hardware to allow quick identification of whether a pixel is in shadow.

The trailing accumulation buffer technique for motion blur can be seen in a [Java applet](#) online.

3dfx's [T-buffer](#) is like a parallel accumulation buffer, consisting of a set of 4 or 8 image & Z buffers (or more, memory permitting), each of which can be rendered to simultaneously. There is a mask which determines where a triangle gets sent, i.e. you can send it to one or more buffers at the same time, as desired. On the back-end is some video logic which combines the set of buffers to display a single, averaged image. So, for motion blur and depth of field you send down each triangle a number of times, changing the mask as you change the position or view. Triangles which do not need these effects can be sent just once, to all buffers. The T-buffer's real strength is for anti-aliasing, as there is some logic (or driver software?) which jitters the triangles in screen space per buffer, e.g. the first buffer has a jitter of say (0,0), the second buffer (0.5,0), third (0.5,0.5), fourth (0,0.5) [Better is to sample with something like (0,0.25),(0.5,0),(0.75,0.5),(0.25,0.75) or similar rotated grid, which gives more vertical and horizontal resolution and so gives more levels of antialiasing for nearly vertical or horizontal edges, which usually look the worst]. So you can send a single triangle and have it sent in parallel, at slightly different offsets, to the set of buffers. Combining on the back end gives a (somewhat) antialiased image (ignoring problems from doing regular sampling, box filtering, and not doing gamma correction - still, hey, it's a LOT better than doing nothing). The cool thing is that anti-aliasing can be done by default for old games without needing any programming changes, it's just a matter of setting the driver to be in anti-aliasing mode. 3dfx's [VSA-100 chipset](#) (codename Napalm) can be used in parallel to perform T-buffering.

There is also a demo (with source) which works on many hardware accelerators showing accumulation buffer techniques, called the [T-Bluffer](#) - very nice! Another chip that appears to have a similar technology is [Glaze3D](#). NVIDIA and ATI offer antialiasing by using supersampling, and a generic term for this feature is [full-screen antialiasing \(FSAA\)](#).

The [Crystal Space renderer](#), which is free and portable, has special effects like halos, shadows, reflections, and more. [EasyGen3D](#) is another open source 3D renderer. The [FOCUS renderer](#) has a number of special effects and advanced features, as does the [SparkLight 3D/Engine](#). [More engines can be found at DemoNews](#). [Genesis3D](#) is a commercial rendering engine that has an open source license. [WildTangent](#) has purchased Genesis3D and developed an interesting system which runs DirectX in a browser window, and is developing an inexpensive game development platform. The [Unreal engine](#), [LithTech engine](#), [Egert's Power Render engine](#), [RenderWare3](#), and [Quake engines](#) are some

commercial renderers with many special effects. An interesting offshoot from gaming engines is that [the Unreal engine is getting used for interactive architectural walkthroughs](#). Commercial general purpose game engines include [NetImmerse](#) (now with [Gamecube support](#)), [SN Systems](#), [Check Six](#), and [3DGM](#). [Anfy 3D](#) is a small, fast 3D renderer in Java. [Shout3D](#) is another impressive real-time 3D Java system. Plutonium Software has an impressive Java demo game, [Burning Metal 3D](#). Worth repeating, the [3D engines list](#) covers over 500 graphics engines, breaking them down in various ways, such as capabilities, platforms, source, etc.

A fast [Java applet](#) for voxel terrain rendering can be run online. [OliveVoxel](#) is another Java voxel renderer which is pretty impressive and runs online. [Animatek](#) has some interesting demos of [fire](#), [particle effects](#), and a fascinating [voxel rendering method](#) for characters. This voxel rendering method inspired the authors of a SIGGRAPH 2000 paper on [surfels](#), in which a model is converted to surface elements. [QSplat](#) is another rendering technique in the same vein. To see a terrain voxel rendering algorithm in action, try the demos of various [Novalogic](#) PC games. Novalogic has patented their latest voxel technology (patent #6,020,893). There is a [description of Outcast's voxel engine](#) online, from a GDC talk. An excellent, [in-depth tutorial on voxel rendering](#) has been written by Alex Champandard, who also has an experimental terrain voxel renderer, [terraVox](#). [DDG](#) is an open source toolkit for graphics development, and includes a terrain rendering engine. Another article and C++ code for a voxel renderer can be found at the [Scriptorium](#). An article called [Voxel Texturing](#), on rendering heightfields rapidly and directly, is available online. An article on [procedural textures for terrain generation](#) discusses generating heightfields using turbulence equations. A [video of Mark Kilgard's talk at GDC on advanced hardware techniques](#) is available at [Gamasutra](#).

For volume rendering information, look at the [ACM TOG software page](#) for some leads. The [OpenGL Volumizer](#) has some interesting technical information on how SGI is approaching the volume visualization problem.

The [PC demo scene](#) is an interesting phenomenon. Programmers, primarily from Europe, make elaborate, fast, amazing programs which display artistic computer graphic animations with sound. See [Demoo](#), [scene.org](#), [Hornet Archive](#), daily news at [Demoscene.org](#), and check the newsgroup [comp.sys.ibm.pc.demos](#); new graphics algorithms sometimes surface here first, months before hitting the mainstream. For one connoisseur's selection of demos, see [Trixter's list](#). Various [real-time ray tracing demos](#) have also been collected in one spot.

An interesting phenomenon is [Machima](#), making movies using real-time rendering engines to generate or display the frames.

The [OpenGL Challenge](#) is a weekly programming contest, often involving special effects or interesting modeling tricks.

Paul Heckbert has a collection of old but valuable news posts on [accurate polygon edging techniques](#), including code examples.



# Speed-Up Techniques

ATI's Radeon includes an interesting speedup called [Hyper Z](#). When rendering a polygon, the screen is split into tiles (e.g. 8x8 pixels or similar). Before any pixel Z-buffer testing is performed for a polygon and a tile, the highest z-value in the tile is compared to the lowest z-value for the polygon. If the polygon is farther back than the tile's highest value, then it cannot be visible and so individual pixel testing can be avoided. ATI claims a 20% fill-rate performance gain. Time is also saved by being able to sometimes avoid clearing the Z buffer.

[A Compact Method for Backface Culling](#) is an interesting discussion of the subject. It is not terribly compact ("Vector position;" is a repetition of vertex data), but trades a little extra memory for increased speed.

Incredibly, [Cosmo3D](#) and [Optimizer](#) are free for download from the SGI site. Optimizer has a number of speed-up techniques as part of it. A white paper on [Optimizer](#) gives an overview of its capabilities.

Bretton Wade's [BSP FAQ](#) is the perfect place to start for binary space partitioning algorithm information. Some [applets](#) for visualizing spatial indexing schemes are available online.

Michael Abrash has an [illustrated account of how the Quake rendering engine works, excerpted from his Black Book](#); he also has a [shorter outline version](#). This engine uses an interesting mix of BSP-trees and Z-buffering. Many other [Quake programming related papers](#) are available, and the [source code for the first Quake](#) is now available for download. A [port of Quake to D3D](#) is available.

Assarsson and Möller have written a paper on [optimized view frustum culling](#). Hoff presents [a number of papers](#) on this subject.

There are some white papers about a variety of speed-up techniques at [Numerical Design's site](#).

A fascinating, readable overview of the tricks and techniques used in making flight simulators is available online. Carl Mueller's "Architectures of Image Generators for Flight Simulators" is available in [postscript form](#) from [UNC Chapel Hill's Technical Reports library](#).

[Seth Teller's publications page](#) [has a number of articles on occlusion culling and portals, including his thesis](#). [Charles Bloom](#) has a number of demos and articles on practical implementation of portals, LOD techniques, etc.

If you use SGI's Performer software, you may be interested in David Luebke's [pfPortals](#) extension, which does portal and cell culling.

The [Crystal Space renderer](#), which is free and portable, uses portals in its efficiency scheme. The [flipcode site](#) has a set of tutorials on [portals](#) [BSP trees](#), and [related schemes](#) (and [more](#)).

Hybrid Holding sells [Umbra](#), a system for accelerating visibility and occlusion testing. The [manual](#) includes a 130 page section near the end which details visibility theory and the algorithms used in their library.

Anselmo Lastra's talk of [\*All the Triangles in the World\*](#) touches on the use and blending of different rendering techniques such as impostors, level of detail, etc.

For information on multiresolution modeling (LODs and more), see [Garland and Heckbert's site](#). There are links here to papers, free and commercial software, and much else. Intel's [Multiresolution Mesh Technology page](#) shows a simulation of popping. [Sven Tech](#) also has a multi-resolution geometry SDK, with downloadable demo (which is fascinating to watch in wireframe mode) and other information.

The [STRIPE algorithm](#) code, papers, and results are available online. This algorithm produces near optimal triangle strips. However, it's been said that the code is a bit crufty. Another approach to [stripification](#) is available from David Kormann, and includes a demo. According to David, it is easy to implement and much faster than STRIPE, though STRIPE usually has better results. [Simple code from Neal Tringham](#) (based on [Brad Grantham's code](#)) is available for stripification. [Pierre Terdiman](#) also has an article about stripifying. [Martin Isenburg](#) has done research on compressed transmission of mesh data along with stripification information.

Michael Wimmer has a somewhat dated, but mostly complete [list of graphics APIs and scene graphs](#). The [Hoops3D application framework](#), a professional scene graph system used in CAD applications, is now open source and free for personal use on Linux systems. Ur Studios has announced [GEL \(Graph Evaluation Language\)](#), an open-source scene graph API which supports multi-user distributed processing. [Intrinsic Graphics](#), formed by some of the designers of SGI's Performer, has announced a graphics development system for games with Performer-like capabilities and more. [Coin](#) is an open source retained mode scene graph library based on Open Inventor. The [Portable Game Library](#) includes a [Simple Scene Graph library](#) built on OpenGL.

## Pipeline Optimization

[Chris Hecker](#) has written a number of excellent articles on compilers and speed-up techniques for the PC. For example, his [More Compiler Results, and What To Do About It](#) article shows how the simple operation of transforming a set of three vectors by a matrix could be made 3 times faster by trying different forms of the same code. Haim Barad presented an [optimized matrix library \(with source\) for the Pentium III](#). For extensive coverage of how compilers optimize source code, see the [Nullstone site](#).

The [3D Studio MAX R2 display architecture](#) page is a fascinating case study, showing how a wide range of hardware is accommodated in making a modeler's user interface fast. It talks about pipeline optimization issues, software vs. hardware acceleration, Direct3D vs. OpenGL vs. [Heidi](#), and much else of interest. A must read. Though a little dated, the [R1 display architecture](#) is also worth reading.

An [overview of 3D instruction sets on CPUs](#) is presented by Jonathan Hirshon, with links to relevant sites. An article on [cleaning memory and partial register stalls](#) discusses assembly level optimizations for Intel processors. [Intel's MMX site](#) has many practical articles and code for using MMX commands for graphics and other areas. Haim Barad et al. have an [interesting article](#) on MMX programming which also has a method for Intel chips for fast float to long conversion. Michael Herf has a solid article about



floating point precision control and in-depth coverage of fast float to int conversion. Michael has other [cute optimization tricks](#) on his page. Rob Wyatt has a practical overview of the [Pentium III architecture](#). Baker and Pallister discuss [optimizing games](#) on the Pentium III. Zohar and Barad discuss the use of SSE (Katmai) instructions in [Implementing a 3D SIMD Geometry and Lighting Pipeline](#), and Gross has an article on [Pentium III prefetch optimizations](#). The slides and other materials for the SIGGRAPH 2000 courses [Aggressive Performance Optimizations for 3D Graphics](#) and [Developing Efficient Graphics Software](#) are available online.

The [processor pack for Visual C++ 6.0](#) provides support for 3DNow! and SSE instruction sets. Sean Palmer has written an open source [matrix and vector library](#) called OpenXL, meant for use with OpenGL (though usable independently). The interesting part of this library is that, on a PC, it detects the CPU type and tailors the library accordingly (e.g. SSE and 3DNow! command sets are supported). This should not be confused with [Khronos' OpenML](#), which is an API to be developed for integrating graphics, video, and audio. AMD's [3DNow! technology](#) speeds a number of common graphics operations. [Dr. Dobb's Journal](#) has an [article about optimizing for 3DNow!, with the code listings available online](#).

See the [Graphics Gems](#) series for code for quicker square roots and inverse square roots (search the [tables of contents](#) for "square"). On Intel processors, sqrt() takes about 70 cycles, and reciprocal square root 109 cycles through the FPU (according to [Huddy's Scalability talk](#), which has some other good code optimization techniques). Richard Huddy of NVIDIA has provided their [improved fast square root code](#). It avoids the odd/even exponent headaches of Lalonde's *Graphics Gems* approach by using the low order bit of the exponent as a part of the look up table, too.

Gamasutra has an article about using [cache memory](#), something that can make a huge difference in performance.

Memory allocation can be one of the more expensive operations in some rendering systems. A [public domain memory allocator](#) by Doug Lea may improve your program's performance.

If you are working with particular hardware, check the manufacturer's site. For example, the [MIPS site](#) is great for MIPS users. SGI's publicly available [technical publications](#) are a useful collection of tutorials and articles. This library includes [OpenGL on Silicon Graphics Systems](#), which has chapters on optimization, and the [MIPSpro Compiling and Performance Tuning Guide](#).

Here are the Web sites for various useful tuning programs: [VTune](#) and [IPEAK Graphics Performance Tool \(GPT\)](#). Intel also has an [optimization newsletter](#). Another good code profiling tool is [NuMega's TrueTime](#) package.

## Polygonal Techniques

[Narkhede and Manocha's polygon tessellator code](#) in *Graphics Gems V* has been improved to handle holes. O'Rourke has a [tessellator](#) available online, from his (wonderful) book *Computational Geometry in C*, but it is mostly for educational purposes. [Held](#) has a paper on the current state of polygon triangulation research, as well as his own solution.

For file format information, start at [Wotsit's Format](#). For translating various file formats, Keith Rule's [Crossroads](#) package is free, includes source, and works under MS Windows. It does not have consolidation support as discussed in the book, though it could be used as a framework for adding such. [3DWin](#) is also free, converts many formats, and also does smoothing of surfaces (no source available, though). If you need models, you might check the free [Avalon](#) collection or [3D Cafe](#). There are also some file format documents here, as well as many translators, textures, graphics FAQs, and more.

Nate Robins has an interesting online document on [surface smoothing](#). Gavin Bell describes a bit more about how to get the [normals to point outwards](#), along with sample code.

There are a number of papers summarizing simplification research to date. Summaries by [Garland and Heckbert](#), [Krus et al.](#), [Erikson](#), and [Hadwiger](#) are all online. O'Rourke has a summary of recent work in his [Computational Geometry Column 33](#).

For information on multiresolution modeling and simplification, [Garland and Heckbert's site](#) has links to papers, free and commercial software, bibliography, and more. Garland's [QSlim](#) is one of the fastest algorithms for simplification. Hoff has [a number of tools](#) available for viewing and converting Garland's meshes. [Melax's free demo](#) does a good job of showing decimation and geomorphs in action. Commercial versions include [Realax's RXpolyred](#) decimator, [Systems in Motion's Rational Reducer](#), [IntegrityWare's POPLib](#), and [Raindrop Geomagic's Decimate](#). [A summary page for commercial decimators is also available](#). [Optimizer](#) includes a number of model decimators and is free for download; also see the [white paper](#) on it.

[Hoppe's site](#) has a number of papers on simplification and related topics. [More technical papers and demos can be found at Jan Svarovsky's site, Lindstrom's site, and Jon Cohen's site.](#)

Gabe Kruger's tutorial on [Bézier spline surfaces](#) is a practical introduction to these surfaces, as is [Mark DeLoura's article](#) on bicubic Bézier surfaces and [Sharp's article](#) on Hermites and Béziars. Justin Reynen also wrote an [introduction to Béziars](#), as has [Hugo Elias](#).

We mention NURBS in the book; a generalization of spline surfaces likely to have hardware support in the future is the [subdivision surface](#). Brian Sharp has two excellent articles on subdivision surfaces: [one on the theory](#), [another on implementation](#). [Aaron Lee has an article on subdivision surfaces and progressive meshing](#). Dean Macri has an article on using NURBS in real-time applications at [Gamasutra's site](#) and a longer version on [Intel's site](#). Mark DeLoura presents a good introduction to bicubic bezier surfaces. The [source code](#) for the new book [An Introduction to NURBS](#) is available online. There is an entire [NURBS manipulation library](#) under GPL. [ATI has information on their version of DirectX 8 N-Patch \(Bezier\) tessellation.](#)

The [Virtual Terrain Project](#) has useful information about terrain storage and rendering, as well as source code. Examples of terrain simplification can be found at [Röttger and Heidrich's site](#). Also note the commonly used alternating quad-split strategy to break up patterning in the heightfield (i.e. the diagonals for splitting quads alternate in a checkerboard fashion). Another scheme by Duchaineau is the [ROAM](#) method, which is [explained in depth](#) (and with sample code) by Turner. Ranalli gives a scheme of using



[ROAM with portals](#). Thatcher Ulrich has an article, demo, and source code for [using LOD techniques](#) for terrain models. Seumas McNally uses [binary triangle trees](#) for terrain data management. [Peter Lindstrom](#) has extensively researched terrain simplification and LOD techniques.

A number of companies are working on polygonal model compression, simplification, and streaming over the web: [Metastream](#), [Wild Tangent](#), [Cycore](#), [3D Groove](#), and [RealityWave](#). Metastream's [Metastream3](#) interactive renderer is very impressive, an all-software renderer with antialiasing and expressive soft shadows, along with a number of other features. The [Web3D Consortium](#), in charge of VRML standards, continues on.

## Intersection Testing

We created a [3D Object Intersection page](#), giving references and pointers to code for a wide variety of object/object intersection tests.

Möller and Trumbore's [ray-triangle intersection paper](#) is available online, as is their [code](#). [Extensive testing of variants](#) was also done by Möller.

Code for [triangle-triangle intersection](#) is available online (with some small improvements from the algorithm described in our book). Code for an OBB/OBB overlap test is available in the [RAPID collision detection package](#).

The [Graphics Gems](#) book series covers a number of ray/object intersections (e.g. polygon, box, polyhedron, quadric, cylinder). See the [category list](#) online; code is also available for download. Another source for ray/object intersection routines is ray tracer code; see the [ray tracer list](#) at ACM TOG for places to look.

[Dave Eberly's site](#), has code for many object/object intersection tests.

Held has published a paper about [ERIT](#), which deals with a wide range of intersection tests. Source code is available from the author on request.

[Gregory et al.](#), in their paper *Fast and Accurate Collision Detection...*, have what looks to be a fast algorithm for ray/AABB and ray/OBB testing, for when you just want to know whether the ray hits and do not care about the distance.

Miguel Gomez has an [article on a variety of collision intersection tests](#) for spheres and boxes. Kenny Hoff gives an [in-depth analysis](#) of the fast plane/box intersection test we present in the book.

## Collision Detection

Stan Melax presents a practical technique (used in MDK2) for [using BSP trees for collision detection](#) with a variety of different sized and shaped objects, with little additional memory cost.

Jeff Lander has written a good set of articles about [polygon collisions](#) (though really more on point in

polygon), [collision detection using AABB's and separating planes](#), and [collision response](#). Nick Bobic's article is a good place to go next for more on [collision detection techniques](#). Thatcher Ulrich's [loose octrees](#) concept gives a useful data structure for dynamic collision detection, as the structure allows O(1) insertion and deletion of objects. A fuller version of this article is in [Game Programming Gems](#).

[Ming Lin's paper collection](#) has many articles on collision detection. Gamasutra has her [GDC 99 lecture on collision detection](#) available in video online. For what to do when two objects collide, read Brian Mirtich's [Rigid Body Contact](#) paper.

A number of collision detection packages are available on the Web. These include source, and most have limitations on commercial reuse.

- Collision Detection Packages [from UNC Chapel Hill](#): [RAPID](#) (based on OBB), [V-Collide](#), [I\\_COLLIDE](#), and [PQP](#).
- [SOLID](#) - Software Library for Interference Detection. Under GNU license.
- [Q\\_Collide](#) - an improved I\_COLLIDE.
- [V-clip](#) - a low level object collision library.
- [SWIFT](#) - Some preliminary results indicate that it is faster than I-COLLIDE and V-CLIP, and more robust than I-COLLIDE.

Related to collision detection, [Qhull](#) implements the Quickhull algorithm for finding convex hulls quickly. There is a [Java applet](#) which shows various convex hull algorithms in action. The [Stony Brook Algorithm Repository](#) has convex hull and other code in its computational geometry section.

## Graphics Hardware

Many of the papers at the [Eurographics/SIGGRAPH Graphics Hardware Workshop 2000](#) are [available on the web](#). A basic [overview of new graphics hardware features](#) can be found at Gamespot, as well as an interesting [look at future features and desires](#). A less game-centric article by the same author on these themes can be [read on ZDNet](#). A more up-to-date and technically astute overview of new features is a part of a [GeForce 2 GTS analysis](#) at Tom's Hardware Guide.

Sony's Playstation2 is cool, so 16 PS2's together is 16 times as cool. Enter [Sony's GSCube](#) (more at [Gamasutra](#)). [Saddam Hussein](#) [evidently thinks this is a good idea, too](#). Nintendo's [GAMECUBE](#) system should give the PS2 a run for its money. The [XBox](#) looks to be hot, too, including having a unified memory architecture. [A pro-XBox comparison with the PS2 is available from Xbox Magazine](#). To track game consoles, try [Next Generation's site](#). Comparison charts for the newer and announced consoles can be found at [msxbox](#) and [Gamespy](#).

Interestingly, ArtX, who designed Nintendo's GAMECUBE graphics hardware, [was purchased by ATI](#) for \$400 million. [3dfx bought Gigapixel for \\$186 million](#). Other reorganizations include [S3's graphics division being bought by Via Technologies](#) for \$323 million, and Intergraph's graphics unit, Intense3D, [was purchased by 3Dlabs](#) for \$25 million plus. [The biggest news: 3dfx purchased by NVIDIA](#). [3Dlabs](#)



has also announced it is getting involved in [3D graphics for embedded systems](#).

Dreamcast's PowerVR2 specs can be examined at the [Sega Otaku](#). This system has an architecture which allows it to have a lower fill rate and to do true transparency easily; one person's theory of how this [tiling architecture](#) works is available. Unfortunately for this architecture, [Sega is getting out of the hardware market](#). For more on tiling architecture chips, see [PVR-NET](#). Imagination Technologies' low price [Kyro chip](#) uses a PowerVR tiling architecture, giving good performance while consuming less memory bandwidth.

One opinion is that the polys/sec rate is now almost irrelevant for consoles. [Brad Wardell](#) notes that the extra polygons provided by the Playstation 2 provide relatively little, since images are at television resolution. He feels that antialiasing would be more productive than simply higher numbers of polygons. This problem is analogous to displaying textures without mipmapping: if the polygons are all tiny and vary considerably in shading, in animation the object will "sparkle" and look poor. That said, extra polys/sec can still be used up with multipass techniques for much higher shading quality.

Interestingly enough, a [GeForce can be converted to a Quadro](#) with a soldering iron, 5 minutes, and nerves of steel.

Another view of future hardware capabilities is the [PixelFusion](#) chip. By using embedded DRAM with massively parallel computing, 3 gigaflops of processing power can be used for graphics computations. It allows such effects as Phong shading, true bump mapping, and programmable shaders to be used. Another interesting architecture (from what little information can be gleaned about it) is [Glaze3D](#). The [VolumePro](#) architecture for volume rendering is another interesting direction for rendering. Some researchers have been looking at designs for [implementing procedural solid texturing in hardware](#). [Stanford's WireGL project](#) is exploring cluster rendering systems using a tiled screen approach.

Some solid technical information about how one modern hardware architecture implements antialiasing and transparency is presented in the [Z3 paper](#) by Jouppi and Chang. It also has an excellent summary of antialiasing techniques.

An in-depth technical report on the [SGI Infinite Reality architecture](#) is available online, as is [Nintendo 64 architecture](#) information. A rambling, yet interesting, discussion of hardware techniques is available at [Beyond 3D](#).

A place for latest research developments in hardware accelerated rendering is in the [SIGGRAPH 2000 papers](#); there is an entire session on this topic.

The [Stanford FLASH project](#) is researching hardware for data parallel rendering algorithms.

Sam Paik has created a links site for information on [research and development of embedded DRAM](#). A comparison of [DDR vs. Rambus memory](#) is available online. [Intel has a good summary of the AGP interface](#). For hardcore semiconductor news, try [the EE Times](#).

Roy Latham outlines 12 features he considers important for high-end simulation work in a brief, readable [4 page article](#).

Dominic Filion has written a good Web article on [triple buffering](#) and related subjects, as has [Paul Hsieh](#). Michael Bacik discusses handling different color depths and screen sizes in his [Run-Time Pixel Format Conversion](#) article.

Fairchild and Wyble wrote a [report on the characterization of an Apple flat panel LCD display](#). It discusses the relation of voltage to luminance for LCDs and how Apple made the LCD display compatible with traditional Apple CRTs. A fascinating property of LCDs is that [subpixel resolution rendering](#) is possible; this is the underlying technology for Microsoft's ClearType.

An article by Joshua Walrath on [how many frames per second we can perceive](#) discusses how 60 FPS is still perceptible, but 72 FPS is sufficient.

For more on how perspective correct textures are done, see [Chris Hecker's articles](#). An example of the errors caused by not correcting for perspective when texturing can be seen in [the spinning head](#).

The [OpenGL GLUT \(utilities\) source code](#) includes a red-blue stereo demo program.

For Mac developers, [develop magazine archives](#) are maintained at the [MacTech magazine](#) site.

[Linux3d](#) is a good jump-off point for 3D graphics on Linux. Initiatives such as [DRI](#) allow direct access to graphics hardware.

[Tom's Hardware](#) has a good article on what affects benchmarking tests. Particularly interesting is the increasing effect of memory bandwidth. For benchmarking, the [Viewperf/glperf benchmarks](#) by the OpenGL Performance Characterization (OPC) subcommittee of the Graphics Performance Characterization (GPC) committee is good for OpenGL-based hardware. Ziff-Davis's [3D WinBench 99](#) is a reasonable benchmark for Windows9x machines. [Gamespot](#) and Mercury Research gives 3D and 2D benchmark [test results](#). Mad Onion provides [3DMark 2000](#), a benchmark which includes [transform & lighting testing](#). Computer Games' [CG VidMark](#) tests by averaging the results of 3DMark 2000 and the speeds of 3 games.

[An interview with John Carmack about programming on consoles vs. PCs has some interesting insights \(parts one and two\).](#)

PC and console chipset manufacturer sites include [NVIDIA/GPU 3D/RIVA 3D/Planet RIVA/RIVA Zone/RIVA Extreme](#), [ATI Tech](#), [Matrox/Matrox Users Resource Centre](#), [S3/S3 Planet](#), [3Dlabs](#), [Intense3D](#), [Glaze3D](#), [Stellar Semiconductor](#), and [VideoLogic](#), some of which also include information of more general use in programming.

For pure PC graphics accelerator coverage, try [Sharky Extreme](#), [Tom's Hardware Guide](#), [AnandTech](#), [HARDOCP](#), [HotHardware](#), [IXBT Hardware](#), [Beyond 3D](#), [3D Gaming](#), [FastGraphics](#), [Operation 3D](#), [3D Maniac](#), [Tweak3D](#), [Björn's 3D World](#), [3DHardware](#), [Planet Hardware](#), [Hardware Central](#), [Gamers Depot](#), [FiringSquad](#), and [GamePC](#), to name too many.



# The Future (resources)

For a wide variety of predictions of the future of computer graphics, see Jon Peddie Associates' [Visions 2000 and Beyond](#) collection. [Jon Peddie Associates](#) is an influential analyst of the computer graphics market, and provides some useful news online. A related analyst group is [MicroDesign Resources](#). Two electronic news magazines which include material of relevance to real-time rendering are [The WAVE Report](#) (searchable from the [3D Links](#) site), and David Duberman's [Spectrum Report](#). Both are worthwhile, and both are free. The [3D Direct Newsletter](#) is also good; issues are not sent to you, rather, you are notified when a new issue is out. [Computer Graphics World](#) tracks commercial developments in the computer graphics industry in general, though it's easier to just read the magazine. Trade magazines such as [CTW](#) are good for tracking the computer games and entertainment industries. Other analysis of gaming hardware and related topics can be found at [Smokezine](#). Michael Herf has some [cogent comments](#) on some capabilities 3D hardware is lacking.

One interesting trend is the idea of representing surfaces by splats instead of triangles. With triangles getting smaller, the cost of sending vertices and setting up triangles becomes greater than using other representations. [QSplat](#), [surfels](#), and [image based approaches](#) become feasible.

[SIGGRAPH's bibliography searcher](#) now also has links to publications, code, and more. [Frédo Durand's](#) site is a good jump-off point for publications and other research related topics; his [list of links to SIGGRAPH 99 papers online](#) is particularly useful. More useful yet, a [SIGGRAPH 2000 papers](#) site is being maintained by Tim Rowley. [A few more articles and other graphics related resources](#) can be found at FAQSYS. The ACM TOG's [Resources](#) section has pointers to much other material and research labs. See this page's [Bibliography](#) section for more information about where to get articles online.

Usenet news has 2% pure gold, information you cannot find anywhere else. The newsgroup [comp.graphics.algorithms](#) is just what it sounds like, and often has interesting threads. [Much of the gold](#) has been collected in one spot by Steve Hollasch. The newsgroups [comp.games.development.programming.algorithms](#) and [comp.games.development.programming.misc](#) have good material about real-time programming. If you have a specific topic in mind, [DejaNews](#) is a good way to search Usenet news for it.

The [GDAlgorithms mailing list](#) is a superb place for information on interactive computer graphics. Search their [archives](#) for information on all sorts of topics.

**The home for the most current graphics FAQs is the [Ohio State FAQ Collection](#).** A wide range of graphics FAQs are collected at the Avalon site. The [Exaflop](#) site has a nicely formatted [FAQ for comp.graphics.algorithms](#). This site also has some articles on real-time rendering and a glossary.

[Gamasutra's programming area](#) has a wide variety of easily approachable articles on many facets of real-time rendering. Definitely recommended.

The [Game Developer magazine code site](#) is an excellent collection of source code and demonstration programs illustrating various techniques. This magazine is free to qualified readers, and is highly

recommended. [Jeff Landers' site](#) has his code and executables distributed by *Game Developer*, plus additional notes and screen shots. [Programmers Heaven](#) has resources and links to programming information on a wide variety of topics.

**Links to some conferences of interest:** [numerous SIGGRAPH conferences](#) such as [SIGGRAPH](#) and [I3D](#), [Eurographics](#), and the [Game Developers Conference](#).

Some book reviews are available at the [flipcode site](#). An old but still useful list of [book recommendations](#) is available from Brian Hook. [Akbar A.](#) also has brief reviews of many books. Old trade paperback books on a wide variety of topics, including game programming, are available for free in their entirety at [ITKnowledge](#).

Borland has made their [C++ compiler free for download](#).

For an overview of the computational and resource costs of *A Bug's Life*, see [Larry Gritz's talk](#) at the [Workshop on Rendering, Perception, and Measurement](#). Also from this workshop, Anselmo Lastra provided an [interesting chart by John Poulton](#) showing how graphics accelerators have been beating Moore's Law over the years. Turned Whitted has a talk also concerning the future: [The Rendering Problem Part II: Architectures](#).

An open question is what is the primitive of the future: polygons, fragments, voxel splats, or something else? One man's answer: [ASCII](#).

An argument given that ray tracing will in the long term dominate rendering is because its performance is  $\log(n)$  vs. the  $n$  that hidden surface algorithms use, where  $n$  is the number of surfaces. While waiting for ray tracing to take over the world, check out the [real-time ray tracing demos](#) (with a pulsing beat, to boot). Of course, these were written for 386's and whatnot.

**Available on the web is** [a succinct summary of Direct3D and OpenGL and the differences between the two](#). For a brief (and biased) history of OpenGL and the API wars, see [Microsoft and 3D Graphics](#).

## OpenGL

A handy resource is all the [OpenGL reference documentation](#) online and hyperlinked (there are a number of variants on this, e.g. [Sun's offering](#); search [Google](#) on "glGet" for more). An older version of the entire [Red Book](#) is also available online. **The idea behind OpenGL extensions is described at SGI's site.** The best sources for OpenGL information are [OpenGL.org](#) and [SGI's OpenGL site](#). For other lists of OpenGL resources, see [Mason Woo's](#) or [Karim Ratib's page](#) (look under Applications/Computer Graphics/OpenGL). A good way to learn OpenGL is to use it; [Nate Robins's tutorials](#) are an excellent starting place. There are other interesting code samples here, as well as the SIGGRAPH 97 course notes on [OpenGL and Window System Integration](#), all about using OpenGL from MS Windows. Mark Kilgard's [GLUT \(GL Utilities Toolkit\)](#) is another good way to try out and experiment with code for many advanced features in OpenGL, and provides a basic platform independent windowing API for OpenGL. For testing the speed of an OpenGL feature, try the [isfast and pdb](#) packages. [Neal Tringham's "Supposedly useful stuff for OpenGL Game Developers" site](#) actually does have useful stuff; well worth



a visit.

To see what makes OpenGL tick, take a look at [Mesa](#) or [SGI's sample implementation](#). SGI's implementation is not meant to be fast, but is useful for understanding what various commands do. Mesa's is more practical, and the (poorly named) [Utah-GLX project](#) is actively working on drivers for Mesa.

There are OpenGL subsets being ported to the Palm: [miniGL](#) and [tinyGL](#) (in related news, Microsoft announced it is adding [DirectX capabilities to Windows CE](#)).

An interesting overview of OpenGL related happenings (as well as some details about Microsoft's per pixel microcode shading feature in DX8) can be found at [Gamasutra](#).

The OpenGL newsgroup is [comp.graphics.api.opengl](#). SGI-specific OpenGL questions can be posted to [comp.sys.sgi.graphics](#). [A popular OpenGL game development mailing list can be found on egroups](#).

Mason Woo has an [errata site](#) for the *OpenGL Programming Guide*. [Code examples from this book](#) and other OpenGL related documents and executables are also available. [Druid's GL Journal](#) has a number of interesting articles and links for OpenGL. A tutorial on [OpenGL texture objects](#) is available from Gamasutra's site. Tom Hubina has an [OpenGL FAQ](#) for game developers.

[GLSetup](#) detects the graphics card and installs the matching OpenGL drivers. [Glean](#) is a free OpenGL conformance test suite. Sean Palmer has a number of [open source libraries](#) meant for use with OpenGL. These include an optimized (by CPU) matrix/vector math library, audio, input devices, etc.

A large number of IHVs has announced the formation of the [Khronos Group SIG](#) to develop [OpenML](#), an API for integrating OpenGL graphics with video and audio.

If for some reason you have only DirectX hardware support (or a lousy OpenGL driver) and want to use OpenGL, get [Alt.software's driver](#).

## Direct3D

[DirectX 8](#) is the newest release available. Highlights include vertex and pixel shading, higher order primitives (i.e. spline patches), 3D textures, multisampling support, point sprites for particle systems. The newsgroups to read are [microsoft.public.win32.programmer.directx](#) and [microsoft.public.directx](#).

DirectX sites include [DirectX eXperience](#), [DirectxFaq](#), [DirectX Workshop](#). Also see the games programming sites that follow for DirectX articles and code. [CodeGuru](#) has some useful DirectX information and much else on Windows programming. *The book [Advanced 3D Game Programming with DirectX 7.0](#) has a [web site](#) associated with it, with demos, links, and other materials.*

A roadmap of the [DirectX Graphics Driver Architecture](#) outlines the future for DirectX, [GDI+](#), etc. [RenderIt 3D!](#) is a free DirectX immediate mode wrapper. A [basic tutorial on DirectX \(sans D3D\)](#) is available at EastCoastGames.

# Games Programming

The [International Game Developers Association](#) is a group for helping game developers in a wide variety of ways. If you are interested in getting involved in this area professionally, you may wish to read [one veteran's view](#). [Grandmaster B.](#) (Brian Hook) explains it all at this site. Lots of interesting, chewy tidbits here, including interviews with top games programmers.

There are many sites related to computer games programming, including (in approximate order of usefulness) [Game Developer magazine's site](#) and the related [Gamasutra site](#), [GameDev.net](#), [Flip Code](#), [Game Developers.com](#), [GIG](#), [Inverse](#), [The International Game Developers Network](#), [Games Programming Magazine](#), [Games++](#), [Programmer's Lair](#), [Xtreme Games LLC](#), and [Opifex](#). For even more sites and resources, see the nicely organized [Game Development Search Engine](#).

If you are a games developer, you will want to apply for a [free subscription](#) to *Game Developer* magazine.

To see what games graphics programmers are thinking about, check their [plan files](#). John Carmack of [id Software](#) is one of the longest running plan files, and has much interesting material in it.

The *Game Developer* magazine's [Frontline Awards](#) are a reasonable list of some of the best tools for creating and programming games.

Scalability is an important concept: how do you make your game run well on a wide range of machines? A [Gamasutra article](#) goes over the basics and talks about some ways to fulfill this goal. More information on this subject is available at [Intel's site](#).

The slidesets from [three GDC 99 talks](#) are available from Nihilistic, including one on game programming for windows.

The results of Stanford's graphics course [video game competition](#) are available for download.

A somewhat dated [list of games developer books](#) with short reviews is available online, though it has not been updated lately. Another [list of books](#) is available at the [GameJobs site](#), which also has some articles on interviewing, etc.

Scitech's [MGL](#) is an open source 3D engine for games development.

The source code for [Doom](#) has been released under the Gnu General Public License, and has been extended with OpenGL support and more. This could be the [3D killer app](#) for system administration. Better yet, [source code for the first Quake](#) has also been released under GPL. Bungie's [Marathon 2](#), a first person game for the Mac from 1995, has been made open source.

For the latest news and demos of 3D games, see [3D Files](#). Another fun demo/screensaver/eye-candy site is [DemoNews](#). Other 3D gaming news sites include the [Adrenaline Vault](#) and [Blue's News](#). For too much eye candy, see [Screenshots.net](#). Console and computer game news, see [UGO](#). Announcements, hiring/firing news, and other industry news can be found at [FGN Online](#). Actually, there are many more



such sites, go [Yahooing](#) for them...

For a nicely annotated set of graphics related links, see the [good-looking textured light-sourced bouncy fun smart and stretchy links page](#). If you want to search through (sometimes stale) links all day long, check the [Linkomania graphics](#) and [games programming](#) pages. **Then try the [Game Development Search Engine](#)**. Then go visit [this one](#).

## Linear Algebra

Also see the [Transforms](#) section.

Excerpts from the [CRC Standard Mathematical Tables and Formulas](#) are available on the Web. A huge amount of information on mathematics in general is available in Eric Weisstein's [World of Mathematics](#), much of which is available online (**Sadly, this is no more, but may return, so the link is left here**). His other [Treasure Troves](#) are pretty great, too.

The [Exploratory](#) has many tutorial applets related to linear algebra and transforms. These applets are good for building an intuition and understanding of various topics.

## Trigonometry

Trig formulas, tables, and other mathematical reference material can be found at [Dave's Math Tables](#).

## Bibliography

[Karim Ratib's page](#) (look under Applications/Computer Graphics) has lists and links to many journal articles available online (see his *Publications* subdirectory). A small collection of papers is located at the [Graphics Papers](#) site. [IEEE Computer Graphics and Applications](#) has issues from 1995 to the present available online to members. The [ACM Digital Library](#) is a paid service offering ACM proceedings and journals electronically, back to the 1980's; it is searchable for free by anyone. A yearly subscription is available (a particular bargain for students) or articles can be purchased individually. The site can also be searched and browsed by non-subscribers. WPI's ["Advanced Topics in Computer Graphics"](#) page has some interesting material. Under "Presentation Summaries" there are many paper summaries on a wide variety of rendering related topics.

SIGGRAPH's [Bibliography Database Search](#) is focused on computer graphics references. A general computer science reference search engine [has references to many other computer graphics and computer science related sources](#).



Thanks to [ACM TOG](#) for providing a home for these web pages.

---

webslaves: [Eric Haines](#) / [erich@acm.org](mailto:erich@acm.org)



# journal of graphics tools

*published by*  
[A K Peters, Ltd.](#)

The *journal of graphics tools* is a quarterly journal whose primary mission is to provide the computer graphics research, development, and production community with practical ideas and techniques that solve real problems. We aim to bridge the gap between new research ideas and their use as tools by the computer graphics professional.

There are several other journals in computer graphics which together provide a good forum for introducing new and seminal research. However, rarely can the ideas described in these journals be applied in practice without a great deal of experimentation and experience.

*jgt* provides a forum for the presentation of the useful techniques that emerge as new research ideas mature. In addition, we provide a forum for novel ideas and elegant research results that are perhaps too "small" for the heavyweight research journals, but that are nonetheless of significant use to practitioners.

*Contents of current issue: Volume 5 Number 2*

[Accelerating "Intelligent Scissors" Using Slimmed Graphs.](#) Kevin Chun-Ho Wong, Pheng-Ann Heng, and Tien-Tsin Wong.

[Three-Dimensional Deformation Using Directional Polar Coordinates.](#) Xiaogang Jin and Y. F. Li.

[An Anisotropic Phong BRDF Model.](#) Michael Ashikhmin and Peter Shirley.

[Using Graphics Hardware to Speed Up Your Visibility Queries.](#) Laurent Alonso and Nicolas Holzschuch.

*Papers to appear in upcoming issues:*

"Generating Random Points in a Tetrahedron" C. Rocchini and P. Cignoni.

"One-Dimensional Resampling with Inverse and Forward Functions" G. Wolberg.

"A Simple Recursive Tessellator for Adaptive Surface Triangulation" A. J. Chung and A. J. Field.

*Complete bibliography of jgt papers:*

[Alphabetically](#) by authors.

[Chronologically](#) by issue.

[Indexed](#) by topic.

[Search](#) the paper abstracts and web pages.

*Information for authors:*

[Call for papers](#)

[Paper submission information](#)

[Referee's review form](#)

[Instructions for publication](#)

*General information:*

[Subscription information](#)

[Back volumes, single issues, and reprints](#)

[Masthead](#): Editorial board, publisher, etc.

[Comments, suggestions, and other feedback.](#)

*(Last updated: 03 January 2001)*

---

The *jgt* web site is provided courtesy of [ACM](#).



# Ray Tracing Bibliography

A bibliography of ray tracing related references is available at this site. The bibliographic references are in *refer* format. You can obtain it as a [ZIP](#) file or just the **ray.refer** file itself as [text](#). Please report any missing references or errors to [Eric Haines](#).

You can [search](#) this bibliography online.

A somewhat dated (last updated in 1993) but still useful resource from Tom Wilson is his set of ray tracing abstracts, giving not only article references but also the abstracts. You can download the [ZIP](#) file of all of abstracts or the individual Latex files of [abstracts](#), [newer abstracts](#), or his [read me](#) file.



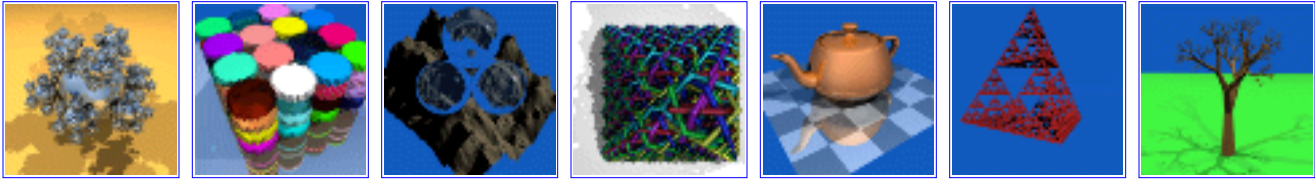
go to [ACM TOG literature related resources page](#)

---

Last change: *September 2, 1999*

# Standard Procedural Databases

by [Eric Haines](#) et al.



*click on an image to see a full size rendering  
(rendered with [POV-Ray 3.1](#))*

You can [download the latest version of the SPD](#) (currently 3.13). This software package is not copyrighted and can be used freely (for example, [WCT2POV](#), a good 3D file converter for Windows, uses SPD's libvec.c graphics library). All source is in K&R vanilla C (though ANSI headers can be enabled) and has been used on many systems. A MacOS version can be found on [Eduard Schwan's SPD page](#).

For a newer set of more realistic environments for benchmarking ray tracers (or renderers in general), see [BART: A Benchmark for Animated Ray Tracing](#). The focus is software that generates an animated set of frames for a ray tracer to render. These scenes use an NFF-like language (AFF), and the authors provide a number of tools for parsing and visualization.

This software is meant to act as a set of basic test images for ray tracing algorithms. The programs generate databases of objects which are fairly familiar and "standard" to the graphics community, such as the teapot, a fractal mountain, a tree, a recursively built tetrahedral structure, etc. I originally created them for my own testing of ray tracing efficiency schemes. Since their first release other researchers have used them to test new algorithms. In this way, research on algorithmic improvements can be compared in a more standardized fashion. If one researcher ray-traces a car, another a tree, the question arises, "How many cars to the tree?" With these databases we may be comparing oranges and apples, but it's better than comparing oranges and orangutans. Using these statistics along with the same scenes allows us to compare results in a more meaningful way.

Another interesting use for the SPD has been noted: debugging. By comparing the images and the statistics with the output of your own ray tracer, you can detect program errors. For example, "mount" is useful for checking if refraction rays are generated correctly, and "balls" (a.k.a. "sphereflake") can check for the correctness of eye and reflection rays.

The images for these databases and other information about them can be found in **A Proposal for Standard Graphics Environments**, *IEEE Computer Graphics and Applications*, vol. 7, no. 11, November 1987, pp. 3-5. See *IEEE CG&A*, vol. 8, no. 1, January 1988, p. 18 for the correct image of the tree database (the only difference is that the sky is blue, not orange). The teapot database was added later.

The [Neutral File Format](#) (NFF) is the default output format from SPD programs. This format is trivial to parse (if you can use sscanf, you can parse it), and each type of object is defined in human terms (e.g. a cone is defined by two endpoints and radii). The basic shapes supported are polygon and polygon patch (normal per vertex), cylinder, cone, and sphere. Note that there are primitives supported within the SPD

which are not part of NFF, e.g. heightfield, NURBS, and torus, so more elaborate programs can be written. If a format does not support a given primitive, the primitive is tessellated and output as polygons.

Other output formats are supported:

- POV-Ray 1.0
- POV-Ray 2.0 to 2.2
- POV-Ray 3.1
- Polyray 1.4 to 1.6
- Vivid 2.0
- QRT 1.5
- Rayshade 4.0.6
- RTrace 8.0.0
- Art 2.3 (from Vort)
- RenderMan RIB
- AutoCAD DXF [object data only]
- Wavefront OBJ format (polygons only)
- RenderWare RWX script file
- Apple 3DMF
- VRML 1.0
- VRML 2.0

Alexander Enzmann receives most of the credit for creating the various file format output routines, along with many others who contributed.

There are also reader programs for the various formats. Currently the following formats can be read and converted:

- NFF
- DXF (just 3DFACEs)
- OBJ

This makes the NFF format a nice, simple language for quickly creating models (whether by hand or by program), as any NFF file can be converted to many different formats. Warnings:

- The conversions tend to be verbose in many cases (e.g. there is currently no code in place to group polygons of the same material into polygon mesh primitives used in some formats).
- No real tessellation of polygons is done when needed for conversion, all that happens are that polygon fans are created.
- You might find the images you obtain are mirror reversed with some formats (e.g. VRML 2.0 files).

The [Graphics Gems V](#) code distribution has a simple z-buffer renderer by Raghu Karinithi, using NFF as the input language.

On hashing: a sore point in mount.c, the fractal mountain generator, has been its hashing function. Mark

VandeWettering has provided a great hashing function by [Bob Jenkins](#). To show what a difference it makes, check out images of models made with the [original hash function](#) with a large size factor, [replacement hash function](#) I wrote (still no cigar), and [Jenkins' hash function](#).

For more information on the SPD, see the README.txt file included in the distribution.

---

## Research Works using SPD

Timing comparisons for the various scenes using a wide variety of free software ray tracers are summarized in *The Ray Tracing News*, [3\(1\)](#) (many), [6\(2\)](#), [6\(3\)](#), [8\(3\)](#), and [10\(3\)](#). Here are some research works which have used the SPD to benchmark their ray tracers (please let [me](#) know of others):

- Kay, Timothy L. and James T. Kajiya, "Ray Tracing Complex Scenes," *Computer Graphics* (SIGGRAPH '86 Proceedings), **20**(4), Aug. 1986, p. 269-78.
- Arvo, James and David Kirk, "Fast Ray Tracing by Ray Classification," *Computer Graphics* (SIGGRAPH '87 Proceedings) **21**(4), July 1987, p. 55-64. Also in Tutorial: Computer Graphics: Image Synthesis, Computer Society Press, Washington, 1988, pp. 196-205. Predates SPD, uses recursive tetrahedron.
- Subramanian, K.R., "Fast Ray Tracing Using K-D Trees," Master's Thesis, Dept. of Computer Sciences, Univ. of Texas at Austin, Dec. 1987. Uses balls, tetra, tree.
- Fussell, Donald and K.R. Subramanian "Fast Ray Tracing Using K-D Trees," Technical Report TR-88-07, Dept. of Computer Sciences, Univ. of Texas at Austin March 1988. Uses balls, tetra, tree.
- Salmon, John and Jeffrey Goldsmith "A Hypercube Ray-Tracer," *Proceedings of the Third Conference on Hypercube Computers and Applications*, 1988. Uses balls and mountain.
- Bouatouch, Kadi and Thierry Priol, "Parallel Space Tracing: An Experience on an iPSC Hypercube," ed. N. Magnenat-Thalmann and D. Thalmann, *New Trends in Computer Graphics* (Proceedings of CG International '88), Springer-Verlag, New York, 1988, p. 170-87. Uses balls.
- Priol, Thierry and Kadi Bouatouch, "Experimenting with a Parallel Ray-Tracing Algorithm on a Hypercube Machine," *Eurographics '88*, Elsevier Science Publishers, Amsterdam, North-Holland, Sept. 1988, p. 243-59. Uses balls.
- Devillers, Olivier, "The Macro-Regions: an Efficient Space Subdivision Structure for Ray Tracing," *Eurographics '89*, Elsevier Science Publishers, Amsterdam, North-Holland, Sept. 1989, p. 27-38, 541. (revised version of Technical Report 88-13, Laboratoire d'Informatique de l'Ecole Normale Supérieure, Paris, France, Nov. 1988). Uses balls, tetra.
- Priol, Thierry and Kadi Bouatouch, "Static Load Balancing for a Parallel Ray Tracing on a MIMD Hypercube," *The Visual Computer*, **5**(1/2), March 1989, p. 109-19. Uses balls.
- Green, Stuart A. and D.J. Paddon, "Exploiting Coherence for Multiprocessor Ray Tracing," *IEEE Computer Graphics and Applications*, **9**(6), Nov. 1989, p. 12-26. Uses balls, mount, rings, tetra.
- Green, Stuart A. and D.J. Paddon, "A Highly Flexible Multiprocessor Solution for Ray Tracing," *The Visual Computer*, **6**(2), March 1990, p. 62-73. Uses balls, mount, rings, tetra.
- Dauenhauer, David Elliot and Sudhanshu Kumar Semwal, "Approximate Ray Tracing,"

- Proceedings of Graphics Interface '90*, Canadian Information Processing Society, Toronto, Ontario, May 1990, p. 75-82. Uses balls, gears, tetra.
- Badouel, Didier, Kadi Bouatouch, Thierry Priol, "Ray Tracing on Distributed Memory Parallel Computers: Strategies for Distributing Computations and Data," *SIGGRAPH '90 Parallel Algorithms and Architecture for 3D Image Generation course notes*, 1990. Uses mountain, rings, teapot, tetra.
  - Green, Stuart A., *Parallel Processing for Computer Graphics*, MIT Press/Pitman Publishing, Cambridge, Mass./London, 1991. Uses balls, mount, rings, tetra.
  - Subramanian, K.R. and Donald S. Fussell, "Automatic Termination Criteria for Ray Tracing Hierarchies," *Proceedings of Graphics Interface '91*, Canadian Information Processing Society, Toronto, Ontario, June 1991, p. 93-100. Uses balls, tetra.
  - Fournier, Alain and Pierre Poulin, "A Ray Tracing Accelerator Based on a Hierarchy of 1D Sorted Lists," *Proceedings of Graphics Interface '93*, Canadian Information Processing Society, Toronto, Ontario, May 1993, p. 53-61. Uses balls, gears, tetra, tree.
  - Simiakakis, George, and A. Day, "Five-dimensional Adaptive Subdivision for Ray Tracing," *Computer Graphics Forum*, **13**(2), June 1994, p. 133-140. Uses balls, gears, mount, teapot, tetra, tree.
  - Klimaszewski, Krzysztof and Thomas W. Sederberg, "Faster Ray Tracing Using Adaptive Grids," *IEEE Computer Graphics and Applications* **17**(1), Jan/Feb 1997, p. 42-51. Uses balls.
  - Nakamaru, Koji and Yoshio Ohno, "Breadth-First Ray Tracing Utilizing Uniform Spatial Subdivision," *IEEE Transactions on Visualization and Computer Graphics*, **3**(4), Oct-Dec 1997, p. 316-328.
  - Müller, Gordon and Dieter W. Fellner, "[Hybrid Scene Structuring with Application to Ray Tracing](#)," *Proceedings of International Conference on Visual Computing (ICVC'99)*, Goa, India, Feb 1999, pp. 19-26. Uses balls, lattice, tree.
  - Havran, Vlastimil and Filip Sixta "[Comparison of Hierarchical Grids](#)," *Ray Tracing News*, **12**(1), June 25, 1999. Uses all spd. [Additional statistics are available at this site](#)
  - Havran, Vlastimil, "[A Summary of Octree Ray Traversal Algorithms](#)," *Ray Tracing News*, **12**(2), December 21, 1999. Uses all spd. [Additional statistics are available at this site](#)
- 

## Graphics Benchmarks

Various graphics hardware comparisons are available online. The [Graphics Performance Characterization](#) group has been around for a long time. [Gemini Technology Corporation](#) has begun an effort to benchmark graphics hardware accelerators and share the results with the public.

---



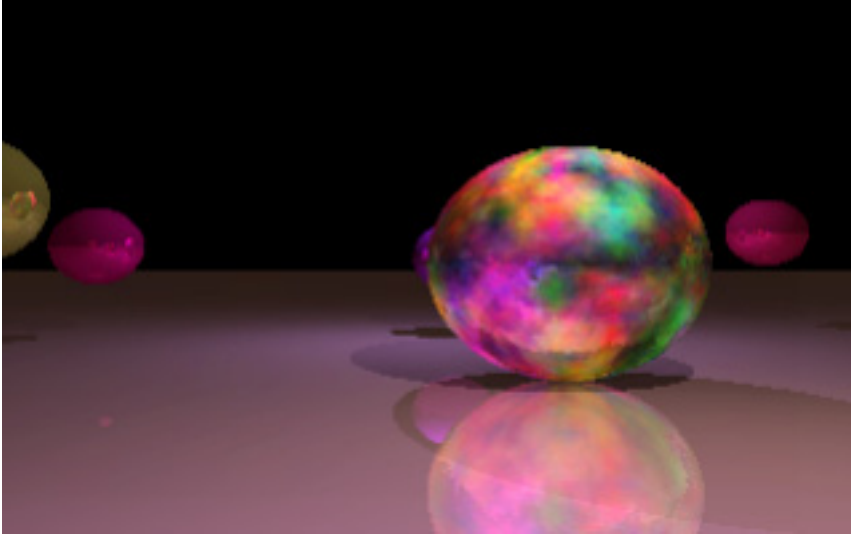
back to [TOG Software](#)

---

[Eric Haines](#), On-Line Editor / [erich@acm.org](mailto:erich@acm.org)

Last change: *November 19, 2000*

# The Realtime Raytracing Realm



*from the "I feel like I could" demo*

The so-called demos (having little in common with program demonstrations) are a particular kind of realtime calculated animation, a phenomenon particularly alive in Europe (one could say, in Finland). Among the many graphical effects and techniques, raytracing is especially suited to the traditional 64k size format, and since Mfx released Transgression in late 1995 a few more raytracing intros have been a part of various programming competitions. The most common realtime raytraced objects we've seen are definitely quadrics, but recently also quartics have been used. Reflections are common, and sometimes you'll see CSG. Raytracing intros can be sometimes disturbing, because of the huge size of filtered pixels (and usually also for that swell north-european techno music, provided you own a Gravis Ultrasound soundcard, for years the standard in demos), but some of them are real technical masterpieces, keeping alive the myth of hacking so misunderstood nowadays.

[rtrtdemo.zip](#) - contains all of the demos below:

Title	Author(s)	Type	Year
<a href="#">Chrome</a>	Tomcat/Abaddon	4k	1995
<a href="#">Transgression</a>	Mfx	64k	12/1995
<a href="#">Chrome 2</a>	Tomcat/Abaddon	4k	8/1996
<a href="#">Transgression 2</a>	Mfx	64k	1/1996
<a href="#">Ah!</a>	Pulse	64k	3/1997
<a href="#">Uh!</a>	Pulse	64k	3/1997
<a href="#">Gamma</a>	Mfx	64k	6/1997
<a href="#">Just like antani</a>	Bug2Fix	64k	7/1997
<a href="#">I feel like I could</a>	Spinning Kids	64k	9/1997
<a href="#">Sink</a>	Pulse	64k	12/1997
<a href="#">Gamma 2</a>	Mfx	64k	12/1997



<a href="#">Jive 2</a>	Sublogic	64k	4/1998
<a href="#">Sviluppo insostenibile</a>	Spinning Kids	64k	12/1998
<a href="#">Taint</a> (and <a href="#">old version</a> )	Spinning Kids	64k	3/1999
<a href="#">(Independent) Love Bong</a>	Contrast	64k	1/1999
<a href="#">Rubicon</a>	Suburban	64k	3/1999
<a href="#">Ray of light</a>	Magic and Schatz	4k	4/1999
<a href="#">joLLy apoLLo</a>	INF	64k	7/1999
<a href="#">Dallas</a>	Sublogic & Blocc	64k	9/1999
<a href="#">Rotation hyperboloid</a>	Muhmac/Freestyle	4k	10/1999
<a href="#">Slumpism</a>	Pathos	64k	10/1999
<a href="#">Rubicon2</a>	Suburban	64k	2/2000
<a href="#">Heaven7</a>	Exceed	64k	5/2000
<a href="#">Transgression 3</a>	Mfx	64k	6/2000
<a href="#">Dast</a>	Power Rangers	64k	7/2000
<a href="#">Fresnel 2</a>	Kolor	68k	1/2001

Other demos with ray tracing parts, too large to include here directly as downloads:

- Gateways, by [Trauma](#).
- [Naturesuxx](#), by FAN.

Try, for example, running Gamma2 from Mfx in high resolution mode and discover the real computational power of your processor. Sometimes people don't even believe raytracing can be done at the speed of 10 or more frames per second: i.e. Iflic by Spinning Kids didn't get a good rating at TheItalianGathering'97 because many thought reflections there were just plain environment mapping... likewise the text files accompanying Mfx's Gamma and Gamma2 are a clear example of the same atmosphere around this Finnish demogroup that started endless discussions.

All intros available on this page should run under pure Ms-Dos (reboot Windows 9x with Alt+f4 and the Ms-Dos option), and sometimes configuration switches are available (see the text files included). Also useful (often necessary) for truecolor video modes is a Vesa driver like Scitech's Display Doctor (see <http://www.scitechsoft.com>) since not all video cards and Vesa programs like each other. Normally to hear the music you'll need a Gravis Ultrasound soundcard (now out of production).

A mailing list specifically for discussing real-time ray tracing now exists; [sign up](#) if you are interested. [Archives](#) are also available. The [FAQ](#) for this group is also available online.

For more information about the demoscene try <http://www.scene.org>, [Scenet](#), [Hugi](#), the outdated-but-nicely-presented [Hornet Archive](#), (or [my links page](#)). There are code tidbits and comments at [Thomas Germano's site](#).



Many thanks to Eric Haines for hosting this page, and all the raytracing coders who made it possible, and those who keep discussing the thing on IRC (yes, one day I might think my intro is ready for release...).

Regards,

Piero Foscari

---

Officium research - CG department - [wally@italymail.com](mailto:wally@italymail.com)

<http://www.geocities.com/ResearchTriangle/Lab/2838/index.html>

---

maintainer: [Eric Haines](#) / [erich@acm.org](mailto:erich@acm.org) Last change: *January 22, 2001*

# pellucid

VRML 2.0 shading model applet

by Eric Haines, erich@acm.org, copyright (c) 1997

version 1.1 3/13/97

{ You may freely use and redistribute this code as long as you keep intact the author and copyright information in pellucid.java }

This applet simulates the [VRML illumination model](#) given a default view of a sphere, a default directional light with direction [ -1 -1 -1 ], and a default material. Also, we have set the ambientIntensity to 1 for the light (by default it is 0) so that the Material's ambientIntensity immediately has an effect. The lighting and material fields can be modified, as well as the background color and the gamma correction factor for the monitor. Shading is computed on a pixel by pixel basis (so, for example, the specular highlights will be tighter and more precise than those normally seen using Gouraud interpolation, which is what all VRML browsers currently use).

No error checking is done by this application; the short answer is "read the spec" for the range of values, but in actuality it's pretty easy to characterize: except for the light's direction and gamma, all values should be in the 0.0 to 1.0 range. Even with this limitation it is easy to create scenes where the computed color is outside the 0.0 to 1.0 display range; this applet clamps each individual RGB channel (vs. uniformly scaling each channel by the largest value) so that the values are within this range. When "show clamp" is set, all pixel results which had to be clamped are instead turned red for easy identification.

Note that the view in the applet is "from infinity", while the corresponding pellucid.wrl file simply moves the eye back and tightens the field of view, so the views will not be identical. Also, the sphere here is not tessellated, while spheres in VRML browsers are turned into polygons. Therefore, do not expect to do a pixel by pixel comparison with your VRML browser - the major goal of this program is to help browser writers using APIs which are different than OpenGL to be able to see how their approximation of the VRML illumination fares under various parameter combinations. This should help us all reach some level of convergence in rendering similar materials and lighting conditions similarly.

See the [International Color Consortium](#) page for some notes on gamma correction, especially the sRGB paper (near the bottom of their page), also see [Poynton's page](#). For PC monitors gamma is an average of 2.2 (according to the [sRGB white paper](#)). To compare stored image values across platforms, set the gamma value to 1.0. However, VRML is concerned with how the users perceive worlds viewed on the screen, so I give a default value of 2.2 in order to show how the results should look on a gamma-corrected browser on a PC. Just to hit you over the head: gamma matters, and the sooner browser writers and authoring systems deal with

it the better we'll all be. End of agenda.

Note that CosmoPlayer and Live3D on the PC are the only PC browsers which attempts to simulate gamma correction (they do this by boosting the material contributions on the front end; gamma should be corrected on the back end but most rendering APIs do not support this). Note that gamma is important: if you design your world so that it looks good on a PC browser which does no gamma correction, your materials will tend to look washed out on a SGI or Mac, which do some gamma correction. This is a long term issue which is being addressed by the [Color Fidelity workgroup](#) (contact: Mark Callow, [msc@sgi.com](mailto:msc@sgi.com)), but is something to keep in mind.

An interesting idea is for each browser company to put into Java code their approximation model and so have their own pellucid application which shows how they render a given scene. All they would have to do is edit the code to show their simplifications or translations of various fields, assuming they know their API's illumination algorithm.

There is a [white paper](#) by Alan Norton about how he used Renderware to approximate the VRML illumination model.

code history:

3/6/97 - created 1.0

3/13/97 - released 1.1: added clamping, erased sphere during recompute, changed defaults to be more interesting and useful.

---

## Downloads

View the [pellucid Java source](#), or download [pellucid.zip](#), which contains all relevant files (java, classes, html, wrl).

A [VRML world file](#) which has the default pellucid program settings (you'll have to modify the pellucid.wrl file by hand to match whatever you see in the applet - sorry, Java applets can't write to your disk).

---

Eric Haines / [erich@acm.org](mailto:erich@acm.org)

Last change: March 14, 1997

# Computer Graphics Applets

Various institutions have written Java applets which teach fundamental concepts in computer graphics. This site lists applets we have found; let us know if you find or write any others.

The [Brown Exploratory](#) has a large number of applets for teaching computer graphics.

A [ray tracing tutorial](#) is available from Daniel Gould at Brown University.

Patrick Min of Princeton has a number of [tutorial applets](#) on the web. He has kindly made the [source code](#) available.

[Pellucid](#) is a Java applet showing how the VRML lighting model works (a simple Phong based lighting model). Source is available.

---

Eric Haines / [erich@acm.org](mailto:erich@acm.org)

# Index to Ray Tracing News

The following is an index to some of the articles in [Ray Tracing News](#), an electronic newsletter edited by [Eric Haines](#). I created the index primarily to organize the articles from RTN devoted to the topic of spatial data structures for optimizing ray tracing, but I've also included some other articles that looked interesting to me. The index was created for [a course I'm teaching](#).

Each article is listed just once, with title, author(s), and in some cases, my summary or comments on the article in *italics*. I can't vouch for the accuracy of the information in these articles. RTN articles generally have quality somewhere between that of a SIGGRAPH paper and a comp.graphics.algorithms posting. But that's a wide range!

Please let me know of errors or mis-categorizations you find here.

- [Paul Heckbert](#) (and updated by [Eric Haines](#))

---

## Categories:

- [Spatial Data Structures for Optimizing Ray Tracing](#)
  - [Optimizing Ray Tracing, General](#)
  - [Ray-Surface Intersection](#)
  - [Ray Tracing, General](#)
  - [Soft Shadows](#)
  - [Texture Mapping](#)
  - [Antialiasing](#)
  - [Radiosity and Global Illumination](#)
  - [Modeling](#)
  - [Miscellaneous](#)
  - [Book Reviews](#)
  - [Ray Tracing Resource Roundups](#)
- 

## Spatial Data Structures for Optimizing Ray Tracing

- [Book Reviews on Hierarchical Data Structures of Hanan Samet](#), by A. T. Campbell, III, January 2, 1990
- [Any Future for Efficiency Research?](#) by Eric Haines, August 26, 1992
- [3D Point Hashing](#), by Steve Worley, January 18, 1996

## ● Grids (Uniform, Hierarchical, and Nonuniform)

- [Ideal Grid/Object Densities](#), by Dan Gehlhaar, Marc Andreessen, July 10, 1992
- **Uniform Grids**
  - [3DDDA Comments](#), by John Spackman, March 1, 1991  
(*uniform grid traversal*)
  - [Comparison of Ray Traversal Methods](#), by Erik Jansen, February 2.01, 1994  
(*results on grid method*)
  - [Fast Raytracing via SEADS](#), by John Chapman and Wilfrid Lefer, February 2.01, 1994  
(*grids and nested grids*)
  - [Additional Notes on Nested Grids](#), by Kris Klimaszewski, Andrew Woo, Frederic Cazals, and Eric Haines, December 2, 1997
  - [Recursive Grids and Ray Bounding Box Comments and Timings](#), by Andrew Woo, December 2, 1997
  - [Comparison of Hierarchical Grids](#), by Vlastimil Havran and Filip Sixta, June 25, 1999
  - [Quicker Grid Generation via Memory Allocation](#), by Eric Haines, June 25, 1999
- **Non-Uniform Grids**
  - [NuGrid results](#), by Mike Gigante, July 10, 1992  
(*nonuniform grids*)
  - [NuGrid Update](#), by Mike Gigante, August 26, 1992

## ● Adaptive Trees (Octrees, k-d, BSP)

- **Octrees**
  - [Response to the "teapot in a football stadium" Problem](#), by Andrew Glassner, March 1, 1988  
(*hybrid of octree & bounding volume*)
  - [Linear-time Voxel Walking for Octrees](#), by Jim Arvo, March 26, 1988  
(*nice article; see also similar ideas by Erik Jansen*)
  - [Octree](#), edited by Eric Haines, October 27, 1989
  - [Octrees and Whatnot](#), by Steve Worley and Eric Haines, May 16, 1995  
(*best way to build octree hierarchy, also about HBV*)
  - [Octree Neighbor Finding](#), by Andrew Glassner, Francois Sillion, and Paul Heckbert
  - [Octree Traversal and the Best Efficiency Scheme](#), by Ben Hutchison, Eric Haines, Hanan Samet, and Erik Jansen
  - [A Summary of Octree Ray Traversal Algorithms](#), by Vlastimil Havran, December 21, 1999
  - [Additional Octree Traversal Notes](#), by John Spackman, Erik Jansen, and Joel Welling, December 21, 1999

- **Binary Space Partitioning (BSP) Trees**

- [Re: Linear-time Voxel Walking for BSP](#), by Erik Jansen, April 6, 1988  
(*good, see also Arvo's octree article above*)
- [Recursive Ray Traversal](#), by Erik Jansen and Wim de Leeuw, July 10, 1992  
(*concludes that recursive traversal of BSP data structure is faster than incremental*)
- [BSP Traversal Errata](#), by Kelvin Sung, August 26, 1992

- **Hierarchical Bounding Volumes (HBV's)**

- [Bounding Volume Hierarchy](#), edited by Eric Haines, October 27, 1989
- [Bounding Box Intersection](#), edited by Eric Haines, October 27, 1989
- [BVH Traversal Results](#), by Nicholas Wilt, July 10, 1992
- [Bounding Volumes \(Sphere vs. Box\)](#), by Tom Wilson, January 27, 1993
- **Tight Bounding Spheres**
  - [Minimum Bounding Sphere, continued](#), by Jack Ritter, June 21, 1989  
(*fast algorithm to find a near-optimal sphere around  $n$  points*)
  - [Minimum Bounding Sphere Program](#), by Marshall Levine, October 13, 1989
  - [Solution to Smallest Sphere Enclosing a Set of Points](#), by Tom Shermer, January 2, 1990  
(*references to computational geometry papers that find smallest sphere*)
- **Goldsmith/Salmon: Automatic Creation of Bounding Volume Hierarchies**
  - [Goldsmith/Salmon Hierarchy Building](#), by Jeff Goldsmith, March 8, 1988
  - [Automatic Creation of Object Hierarchies for Ray Tracing](#), by Eric Haines, April 6, 1988  
(*lengthy*)
  - [Re: Goldsmith and Eyes](#), by K.R.Subramanian, November 4, 1988  
(*optimization for eye rays*)
  - [An Improvement to Goldsmith/Salmon](#), by Jeff Goldsmith, March 20, 1990
  - [Faster Bounding Volume Hierarchies](#), by Brian Smits and Eric Haines, September 28, 1993  
(*using randomization and multiple trials to find the best clustering, useful for both Goldsmith/Salmon and Kay/Kajiya*)
- **Kay/Kajiya: Intersection of Slabs Method**
  - [Sorting Unnecessary on Shadow Rays for Kay/Kajiya?](#) by Eric Haines and Mark VandeWettering, September 11, 1988
  - [More Comments on Kay/Kajiya](#), by Jeff Goldsmith, Eric Haines, October 3, 1988
  - [Question: Kay and Kajiya Slabs for Arbitrary Quadrics?](#) by Thomas C. Palmer, January 2, 1990
  - [Convex Polyhedron Intersection via Kay & Kajiya](#), by Eric Haines, October 1, 1990



## ● Other, Fancier Spatial Data Structures

Partial list:

- [Constant Time Ray Tracing](#), by Tom Wilson, Eric Haines, Brian Corrie, and Masataka Ohta, August 26, 1992
- [Ray Classification for Animation](#), by Matt Quail

## ● Mailboxes and Other Miscellaneous Speedups for Spatial Data Structures

- [Avoiding Re-Intersection Testing](#), by Eric Haines, March 20, 1990  
(*mailboxes*)

## ● Public Ray Tracing Software

Partial list:

- [At Long Last, Rayshade v4.0 is Available for Beta-testing](#), by Craig Kolb, March 1, 1991
- [POV-Ray](#), August 2, 1995
- [Free Radiosity Renderer \(inc. C++ source code\)](#), by Ian Ashdown, January 27, 1997

## ● Benchmarking Ray Tracers

- [Comparison of Kolb, Haines, and MTV Ray Tracers, Part I](#), by Eric Haines, January 2, 1990  
(*using grids, HBV*)
- [Comments on Last Issue](#), by Mark VandeWettering, March 20, 1990  
(*MTV ray tracer switched to Goldsmith/Salmon*)
- [New Version of SPD Now Available](#), by Eric Haines, March 1, 1991  
(*"Standard Procedural Databases" for benchmarking ray tracers*)
- [Ray Tracer Races, Round 2](#), by Eric Haines, July 1, 1993
- [An Enhanced Standard Procedural Databases Package](#), by Eric Haines, September 28, 1993
- [SPD Platform/Compiler Results](#), by David Hook, September 28, 1993
- [Recursive Grids and Ray Bounding Box Comments and Timings](#), by Andrew Woo, December 2, 1997
- [BART: A Benchmark for Animated Ray Tracing](#), by Jonas Lext, Ulf Assarsson, and Tomas Moller, July 16, 2000
- [Global Illumination Test Scenes](#), by Brian Smits and Henrik Wann Jensen, July 16, 2000

## ● Comparison of Spatial Data Structures and Ray Tracers

- [Poll: What efficiency schemes have you used?](#), August 29, 1989
- [Algorithm Order Discussion](#), by Masataka Ohta, Pete Shirley, October 1, 1990  
(*computational complexity*)



- [Efficiency Tricks](#), by Eric Haines, March 26, 1988  
*(little optimizations for ray tracing)*
- [Efficiency Schemes](#), edited by Eric Haines, October 27, 1989  
*(similar to the previous)*
- [Comments on Various Ray Tracing Speedups](#), by Andrew Woo, July 1, 1993
- [Faster Refraction Formula, and Transmission Color Filtering](#), by Xavier Bec, January 21, 1997
- [Z-buffer and Raytracing](#), by Sam Paik, December 21, 1999
- [Importance for Ray Tracing](#), by Per H. Christensen, December 21, 1999
- [Realtime Techniques for Ray Tracing](#), by Ádám Nagy, Eric Haines, Russel Simmons, Paul Kahler, Factory, and Bruno Schödlbauer, July 16, 2000

- [A Suggestion for Speeding Up Shadow Testing Using Voxels](#), by Andrew Pearce, October 1, 1990
- [Shadow Testing Simplification](#), by Tom Wilson, March 1, 1991
- [Fast Shadow Testing Bibliography](#), by Slawomir Kilanowski, June 26, 1997

[http://www.acm.org/tog/resources/RTNews/html/rtn\\_index.html](http://www.acm.org/tog/resources/RTNews/html/rtn_index.html) (5 of 11) [2/21/2001 9:56:35 AM]

- [Simple, Fast Triangle Intersection](#), by Chris Green and Steve Worley, January 27, 1993
- [Simple, Fast Triangle Intersection, part II](#), by John Spackman, July 1, 1993
- [Point in Polygon, the Quick Answer](#), by Wm. Randolph Franklin and Eric Haines, February 2, 1994  
(*short code*)
- [A Storage Trick for 3D Polygons](#), by Eric Haines, July 6, 1994  
(*save 4 or 8 bytes per vertex*)
- [Detecting Points on the Edge of a Polygon](#), by Eric Haines, June 26, 1997
- [Rendering Unflat Polygons](#), by Eric Haines, June 26, 1997
- [Even Faster Crossings Test](#), by Philip Brown, December 2, 1997
- [Cyrus-Beck as a Ray/Polygon Tester](#), by David Rogers, December 2, 1997
- [Origins of Point In Polygon, Take 10...](#), by Neil Stewart, July 11, 1998
- [Intersecting a Ray and a Triangle with Plücker Coordinates](#), by Ray Jones, July 16, 2000

## ● Quadric Surfaces

- [Primitive/Box Overlap Testing](#) by Ruud Waaj, Paul Heckbert, Andrew Glassner, March 8, 1988  
(*which voxels should list a given primitive?*)
- [More Comments on Tight Fitting Octrees for Quadrics](#), by Jeff Goldsmith, March 26, 1988
- [C Code for Intersecting Quadrics](#), by Prem Subrahmanyam, March 20, 1990

## ● Tori and Quartic Root-Finding

- [Quartic Roots and Tori](#), January 2, 1990
- [Correct Roots for Torus Intersection](#), by Haakan "Zap" Andersson, Joe Cychosz, July 10, 1992
- [Faster Ray-Torus Intersection](#), by Eric Haines, July 6, 1994

## ● Parametric Surfaces (e.g. Bicubic Patches)

- [Spline Surface Rendering, and What's Wrong with Octrees](#), by Eric Haines, January 15, 1988  
(*how to choose octree subdivision criteria for parametric surfaces*)
- [Spline Surface Intersection](#), edited by Eric Haines, October 27, 1989
- [Ray with Bicubic Patch Intersection Problem](#), Wayne Knapp, John Peterson, et al, July 13, 1990
- [Spline Patch Ray Intersection Routines](#), by Sean Graves, August 26, 1992
- [Ray Tracing Procedural & Parametric Surfaces](#), by Mark VandeWettering, Stephen Westin, and Matt Pharr, December 21, 1999

## ● Implicit Surfaces

- [A Short Note on Kalra and Barr's Algorithm](#), by Andrei Sherstyuk, July 11, 1998
- 

# Ray Tracing, General

- [Ray Transformation](#), by Kendall Bennett, January 27, 1993  
(*generating eye rays*)
  - [Shadows from Refractive Objects](#), by Steven Collins, September 28, 1993
  - [Shadows Through Transmitters](#), by Eric Haines, Greg Ward, Alexander Enzmann, Stephen Coy, and David Hook, September 28, 1993
  - [Obfuscated Postscript Ray Tracer](#), by Takashi Hayakawa, July 1, 1993  
(*this is insane!*)
  - [Distribution Ray Tracing](#), by Marc Levoy, July 6, 1994  
(*email to Levoy's graphics class*)
  - [Interacting with Ray Traces](#), by Bert Peers, January 18, 1996
  - [Total Internal Reflection](#), by Eric Haines, Greg Ward, and Chris Larson, January 18, 1996
  - [Ray Tracing, What is it Good For?](#) by Arijan Siska, January 21, 1997
  - [Progressive Ray Tracing and Fast Previews](#), by Eric Haines, January 21, 1997
  - [More on Rendering Bugs](#), by Eric Haines, June 26, 1997
  - [Raytracker Tricks](#), by Hakan "Zap" Andersson, December 2, 1997
  - [Mirror Reflectivity](#), by Greg W. Larson, December 2, 1997
  - [Recent Ray Tracing European Conference Papers](#), July 11, 1998
  - [Attenuation in Water](#), by Bretton Wade and Ian Ashdown, July 11, 1998
  - [Raytracing Gloss/Translucency](#), by Peter Eastman, Matt Pharr, and Stephen Westin, December 21, 1999
  - [Fastest Way to Generate Eye Rays in Ray Tracing](#), by Samuel Paik, July 16, 2000
  - [Volumetric Rendering](#), by Paul Kahler, July 16, 2000
  - [Recent Papers](#), summarized by Eric Haines and Vlastimil Havran, July 16, 2000
- 

# Soft Shadows

- [True Integration of Linear/Area Lights](#), by Kevin Picott, January 2, 1990
- [Ray Tracing Penumbra Shadows](#), Prem Subrahmanyam, July 13, 1990
- [Soft Shadow Ideas](#), compiled by Derek Woolverton, August 26, 1992

- [Fast Soft Shadows](#), by Steve Worley, January 21, 1997
  - [Great Shadow Hack](#), summarized by Eric Haines, June 25, 1999
- 

## Texture Mapping

- [Projective Mapping Explanation](#), by Ken "Turk" Turkowski, October 13, 1989
  - [A Note on Texture Sampling](#), by Eric Haines, March 20, 1990  
(*texture filtering*)
  - [Getting Rid of the Divide](#), by Frank Compagner and Bob Pendleton, January 18, 1996
  - [Synthetic Textures and Genetic Algorithms](#), by Steve Worley, January 21, 1997
  - [The Curse of the Monkey's Paw](#), by Eric Haines, June 26, 1997
  - [Lifting the Monkey's Paw Curse](#), by Jeff Goldsmith, December 2, 1997
- 

## Antialiasing

- [Antialiasing Issues](#), by Arijan Siska, October 13, 1994
  - [Raytracker Tricks](#), by Hakan "Zap" Andersson, December 2, 1997
- 

## Radiosity and Global Illumination

- [Radiosity via Ray Tracing](#), by Pete Shirley, October 1, 1990
  - [Simple Databases Available For Rendering \(Global Illumination\)](#), by Peter Shirley, July 6, 1994
  - [Hierarchical Techniques for Glossy Global Illumination](#), by Per Christensen, January 18, 1996
  - [What's Wrong with Monte-Carlo Methods?](#) by Nguyen D.C. (JuHu), Pete Shirley, Stephen Westin, and Eric Veach, June 26, 1997
  - [European Graphics Tour](#), by Nelson Max, June 26, 1997
  - [Declassified CIA BRDFs now available](#), by Greg Ward Larson, December 2, 1997
- 

## Modeling

- [Correcting Normal Direction](#), by Gavin Bell, October 13, 1994
- [Eccentricity Effects in Blobs](#), by Alfonso Hermida, December 2, 1997

- [Polygon Shrinking](#), by Dave Rusin and Jeff Erickson, July 11, 1998
  - [Correcting Normals on "Flipped" Polygons](#), by Kev, Duncan Colvin, Steve Baker, John Nagle, Dennis Jiang, Alejo Hausner, and Eric Haines, July 11, 1998
- 

## Miscellaneous

- [Order of Rendering and Fast Texturing](#), by Bruno Levy, Dan Piponi and Bernie Roehl, May 16, 1995  
(*per pixel cost of Gouraud, Phong, texture mapping*)
  - [Cells and Portals Resources](#), by Randy Stiles, January 27, 1997
  - [Plücker Coordinates](#), by Jeff Erickson, December 2, 1997
  - [Testing a Matrix for Uniform Scaling](#), by Charles Iliya Krempeaux and Eduardo Marreiros, December 2, 1997
  - [Plücker Coordinate Tutorial](#), by Ken Shoemake, July 11, 1998
  - [Info on REYES Algorithm](#), by Robert Speranza and Tom Duff, July 11, 1998
  - [What's Mesa?](#) by Brian Paul, July 11, 1998
  - [Multithreading Mesa](#), by John Stone, July 11, 1998
  - [On Transforming Planes](#), by Steve Hollasch, July 16, 2000
- 

## Book Reviews (alphabetic)

- [Advice to Authors](#), by Anonymae  
(*advice on writing a book*)
- [Good Computer Graphics Books](#), by Brent McPherson
- [3D Computer Animation](#), by John Vince
- [3D Graphic File Formats: A Programmers Reference](#), by Keith Rule
- [3D with HOOPS](#), by William Leier and Jim Merry
- [Advanced Rendering and Animation Techniques: Theory and Practice](#), by Alan and Mark Watt
- [Adventures in Ray Tracing](#), by Alfonso Hermida
- [The Algorithm Design Manual](#), by Steven Skiena
- [Computational Geometry in C](#), by Joseph O'Rourke
- [Computer Graphics](#), by Don Hearn and Pauline Baker
- [CRC Handbook of Discrete and Computational Geometry](#), edited by Jacob Goodman and Joseph O'Rourke

- [Create Stereograms on Your PC, by Dan Richardson](#)
  - [Encyclopedia of Graphics File Formats, by James Murray & William vanRyper](#)
  - [The Geometry Toolbox, by Gerald Farin and Dianne Hansford](#)
  - [Graphical Treasures of the Internet, by Bridget Mintz](#)
  - [Graphics File Formats, by David Kay and John Levine](#)
  - [Graphics Gems IV, edited by Paul Heckbert](#)
  - [Graphics Programming with Direct3D, by Robert Glidden](#)
  - [Hidden Images: Making Random Dot Stereograms, by Bob Hankinson and Alfonso Hermida](#)
  - [An Introduction to Implicit Surfaces, edited by Jules Bloomenthal](#)
  - [An Introduction to Ray Tracing, edited by Andrew Glassner](#)
  - [The Inventor Mentor, by Josie Wernecke](#)
  - [Jim Blinn's Corner, by Jim Blinn](#)
  - [Making Movies on Your PC, by David Mason and Alexander Enzmann](#)
  - [Numerical Recipes, by William Press et al.](#)
  - [Object-Oriented Ray Tracing in C++, by Nicholas Wilt](#)
  - [Photorealism and Ray Tracing in C, by Christopher Watkins, Stephen Coy, and Mark Finlay](#)
  - [Photorealistic Rendering in Computer Graphics, edited by Pere Brunet and Frederik Jansen](#)
  - [Practical Ray Tracing in C, by Craig Lindley](#)
  - [Procedural Elements for Computer Graphics, 2nd edition, by David Rogers](#)
  - [Principles of Digital Image Synthesis, by Andrew Glassner](#)
  - [Programming for Graphics Files in C and C++, by John Levine](#)
  - [Radiosity: A Programmer's Perspective, by Ian Ashdown](#)
  - [Radiosity and Global Illumination, by Francois Sillion and Claude Puech](#)
  - [Ray Tracing Creations, by Drew Wells and Chris Young](#)
  - [Rendering with Radiance: The Art and Science of Lighting Visualization, Greg Ward Larson and Rob Shakespeare](#)
  - [Stereogram Programming Techniques, by Christopher Watkins & Vincent Mallette](#)
  - [Texturing and Modeling: A Procedural Approach, edited by David Ebert](#)
  - [Tricks of the Graphics Gurus, by Dick Oliver, Scott Anderson, James McCord, Spyro Gumas, and Bob Zigon](#)
-

# Ray Tracing Resource Roundups

- [December 21, 1999](#)
- [June 25, 1999](#)
- [July 11, 1998](#)
- [December 2, 1997](#)
- [June 26, 1997](#)
- [January 21, 1997](#)
- [January 18, 1996](#)
- [August 2, 1995](#)
- [May 16, 1995](#)
- [January 23, 1995](#)
- [October 13, 1994](#)
- [July 14, 1994](#)
- [July 6, 1994](#)
- [February 2.01, 1994](#)
- [February 2, 1994](#)
- [September 28, 1993](#)
- [July 1, 1993](#)
- [January 27, 1993](#)
- [August 26, 1992](#)
- [July 10, 1992](#)

---

Index created by [Paul Heckbert](#), 5 Oct. 1996

Updated by [Eric Haines](#), 16 July 2000

# Index of

## /pubs/tog/resources/RTNews/text

<a href="#">Name</a>	<a href="#">Last modified</a>	<a href="#">Size</a>	<a href="#">Description</a>
<a href="#">[DIR]</a> <a href="#">Parent Directory</a>	26-Apr-1999 15:08	-	
<a href="#">[TXT]</a> <a href="#">RTN_TOC.txt</a>	16-Jul-2000 23:47	43k	
<a href="#">[TXT]</a> <a href="#">RTNews1.txt</a>	26-Jun-1997 13:48	73k	
<a href="#">[TXT]</a> <a href="#">RTNews2.txt</a>	06-Oct-1998 10:52	74k	
<a href="#">[TXT]</a> <a href="#">RTNews3.txt</a>	21-Apr-1998 12:32	47k	
<a href="#">[TXT]</a> <a href="#">RTNews4.txt</a>	21-Apr-1998 12:32	70k	
<a href="#">[TXT]</a> <a href="#">RTNews5.txt</a>	21-Apr-1998 12:32	91k	
<a href="#">[TXT]</a> <a href="#">RTNews6.txt</a>	21-Apr-1998 12:32	85k	
<a href="#">[TXT]</a> <a href="#">RTNews7.txt</a>	21-Apr-1998 12:32	83k	
<a href="#">[TXT]</a> <a href="#">RTNews8.txt</a>	21-Apr-1998 12:32	80k	
<a href="#">[TXT]</a> <a href="#">RTNews9.txt</a>	21-Apr-1998 12:32	19k	
<a href="#">[TXT]</a> <a href="#">RTNv10n1.txt</a>	21-Apr-1998 12:32	53k	
<a href="#">[TXT]</a> <a href="#">RTNv10n2.txt</a>	21-Apr-1998 12:32	79k	
<a href="#">[TXT]</a> <a href="#">RTNv10n3.txt</a>	26-Oct-1999 15:21	93k	
<a href="#">[TXT]</a> <a href="#">RTNv11n1.txt</a>	15-Jul-1998 10:19	92k	
<a href="#">[TXT]</a> <a href="#">RTNv12n1.txt</a>	21-Dec-1999 21:38	47k	
<a href="#">[TXT]</a> <a href="#">RTNv12n2.txt</a>	31-May-2000 10:06	107k	
<a href="#">[TXT]</a> <a href="#">RTNv13n1.txt</a>	17-Jul-2000 19:51	53k	
<a href="#">[TXT]</a> <a href="#">RTNv3n1.txt</a>	21-Apr-1998 12:32	51k	
<a href="#">[TXT]</a> <a href="#">RTNv3n2.txt</a>	21-Apr-1998 12:32	97k	
<a href="#">[TXT]</a> <a href="#">RTNv3n3.txt</a>	21-Apr-1998 12:31	78k	
<a href="#">[TXT]</a> <a href="#">RTNv3n4.txt</a>	21-Apr-1998 12:30	112k	
<a href="#">[TXT]</a> <a href="#">RTNv4n1.txt</a>	21-Apr-1998 12:30	125k	
<a href="#">[TXT]</a> <a href="#">RTNv4n2.txt</a>	21-Apr-1998 12:31	54k	



<a href="#">[TXT]</a> <a href="#">RTNv4n3.txt</a>	21-Apr-1998	12:32	44k
<a href="#">[TXT]</a> <a href="#">RTNv5n1.txt</a>	21-Apr-1998	12:32	32k
<a href="#">[TXT]</a> <a href="#">RTNv5n2.txt</a>	21-Apr-1998	12:31	90k
<a href="#">[TXT]</a> <a href="#">RTNv5n3.txt</a>	21-Apr-1998	12:32	39k
<a href="#">[TXT]</a> <a href="#">RTNv6n1.txt</a>	12-Nov-1998	16:04	76k
<a href="#">[TXT]</a> <a href="#">RTNv6n2.txt</a>	11-Jan-2001	23:11	57k
<a href="#">[TXT]</a> <a href="#">RTNv6n3.txt</a>	21-Apr-1998	12:30	116k
<a href="#">[TXT]</a> <a href="#">RTNv7n1.txt</a>	21-Apr-1998	12:31	79k
<a href="#">[ ]</a> <a href="#">RTNv7n2.ps</a>	21-Dec-1996	11:13	160k
<a href="#">[TXT]</a> <a href="#">RTNv7n2.txt</a>	21-Apr-1998	12:31	66k
<a href="#">[TXT]</a> <a href="#">RTNv7n3.txt</a>	21-Apr-1998	12:31	67k
<a href="#">[TXT]</a> <a href="#">RTNv7n4.txt</a>	21-Apr-1998	12:31	67k
<a href="#">[TXT]</a> <a href="#">RTNv7n5.txt</a>	21-Apr-1998	12:30	90k
<a href="#">[TXT]</a> <a href="#">RTNv8n1.txt</a>	21-Apr-1998	12:31	90k
<a href="#">[TXT]</a> <a href="#">RTNv8n2.txt</a>	21-Apr-1998	12:30	91k
<a href="#">[TXT]</a> <a href="#">RTNv8n3.txt</a>	21-Apr-1998	12:31	53k
<a href="#">[TXT]</a> <a href="#">RTNv9n1.txt</a>	21-Apr-1998	12:31	72k
<a href="#">[TXT]</a> <a href="#">RTNv9n2.txt</a>	12-Nov-1998	16:04	50k
<a href="#">[ ]</a> <a href="#">rtn_text.zip</a>	11-Jan-2001	23:12	1.2M

---

Apache/1.3.12 Server at www1.acm.org Port 80

# A Shaft Culling Tool

[Eric Haines](#)  
[Autodesk, Inc.](#)  
Ithaca, New York  
[erich@acm.org](mailto:erich@acm.org)

## Abstract:

Shaft culling is a means to accelerate the testing of visibility between two objects. This paper briefly describes an algorithm for shaft culling and various implementation options. The code and test harness for the algorithm is available online.

## Source Code:

Download the shaft culling code and a test harness, bundled into a zip archive: [shaft.zip](#)

---

[Return to \*jgt\* home page](#)

# Fast, Low Memory Z-Buffering when Performing Medium-Quality Rendering

Eric Haines

[3D/EYE](#)

Ithaca, New York

[erich@acm.org](mailto:erich@acm.org)

Steven Worley

Worley Laboratories

[steve@worley.com](mailto:steve@worley.com)

## Abstract:

This article presents algorithms which both improve performance and decrease memory costs when using a Z-buffer for medium-quality rendering. The crux of the method is to perform rendering in two passes; the first quickly renders only Z-depth values, the second does all shading calculations. This method allows the reuse of memory used to store the Z-depths and colors, as only one of these two values is needed at any given moment for any given pixel. It also eliminates all unnecessary shading/shadowing/texturing calls, which typically take the majority of computation time in medium-quality algorithms.

---

[Return to \*jgt\* home page](#)

# Ray Tracing News

*"Light Makes Right"*

September 20, 1989

Volume 2, Number 6

Compiled by [Eric Haines](#) [erich@acm.org](mailto:erich@acm.org). Opinions expressed are mine.

All contents are copyright (c) 1989, all rights reserved by the individual authors

Archive locations: anonymous FTP at <ftp://ftp-graphics.stanford.edu/pub/Graphics/RTNews/>, [wuarchive.wustl.edu:/graphics/graphics/RTNews/](http://wuarchive.wustl.edu:/graphics/graphics/RTNews/), and many others.

You may also want to check out [the Ray Tracing News issue guide](#) and the [ray tracing FAQ](#).

---

## Contents:

- [Introduction](#)
- [New People and Address Changes](#)
- [Q&A on Radiosity Using Ray Tracing](#), by Mark VandeWettering and John Wallace
- [Dark Bulbs](#), by Eric Haines
- [MTV Ray Tracer Update and Bugfix](#), by Mark VandeWettering
- [DBW Ray Tracer Description](#)

=====Net News Cullings=====

- [Wanted: Easy Ray/Torus Intersection](#), by Jochen Schwarze
  - [Polygon to Patch NFF Filter](#), by Didier Badouel
  - [Texture Mapping Resources](#), by Eric Haines, Prem Subrahmanyam, Ranjit Bhatnagar, and Jack Ritter
- 

## Introduction

There are a lot of new people, with some interesting fields of study. There's been a lot of talk about texture mapping and the DBW ray tracer on the net. This discussion will probably continue into next issue, but I felt Jack Ritter's posting a good way to end it for now. I've also been toying with texturing again, making my version of "Mount Mandrillbrot" (fractal mountain with everyone's favorite beastie textured onto it), which some clever person invented at the University of Waterloo (I think) some years ago (does anyone know who?). There are also other useful snippets throughout.

However, one major reason that I'm flushing the queue right now is that the node "hpfers" is disappearing off the face of the earth. So, please note my only valid address is the "wrath" path at the top of the issue. Thanks!



back to [contents](#)

---

## New People and Address Changes

Panu Rekola, [pre@cs.hut.fi](mailto:pre@cs.hut.fi)

To update my personal information in your files:

Surface mail: Panu Rekola  
Mannerheimintie 69 A 7  
SF-00250 Helsinki, Finland  
Phone: +358-0-4513243 (work), +358-0-413082 (home)  
Email: [pre@cs.hut.fi](mailto:pre@cs.hut.fi)  
Interests: illumination models, texture mapping, parametric surfaces.

You may remove one of the names you may have in the contact list. Dr. Markku Tamminen died in the U.S. when he was returning home from SIGGRAPH. How his project will go on, is still somewhat unclear.

---

Andrew Pearce, [pearce@alias](mailto:pearce@alias)

I wrote my MS thesis on Multiprocessor Ray Tracing, then moved to Alias where I sped up Mike Sweeney's ray caster. I've just completed writing the Alias Ray Tracer using a recursive uniform subdivision method (see Dave Jevans paper in Graphics Interface '89, "Adaptive Voxel Subdivision for Ray Tracing") with additional bounding box and triangle intersection speed ups.

Right now, I'm fooling around with using the guts of the ray tracer to do particle/object collision detection with complex environments, and particle/particle interaction with the search space reduced by the spatial subdivision. (No, I don't use the ray tracer to render the particles.)

In response to Susan Spach's question about mip mapping, we use mip maps for our textures, we get the sample size by using a "cone" size parameter which is based on the field of view, aspect ratio, distance to the surface and angle of incidence. For secondary rays this size parameter is modified based on the tangents to the surface and the type of secondary ray it is (reflection or refraction). This may be difficult to do if you are not ray tracing surfaces for which the tangent information is readily available (smooth shaded polygonal meshes?).

- Andrew Pearce  
- Alias Research Inc., Toronto, Ontario, Canada.  
- [pearce%alias@csri.utoronto.ca](mailto:pearce%alias@csri.utoronto.ca) | [pearce@alias.UUCP](mailto:pearce@alias.UUCP)  
- ...{allegro,ihnp4,watmath!utai}!utcsri!alias!pearce

---

Brian Corrie, [bcorrie@uvicctr.uvic.ca](mailto:bcorrie@uvicctr.uvic.ca)

I am a graduate student at the University of Victoria, nearing the completion of my Masters degree. The topic of my thesis is producing realistic computer generated images in a distributed network environment. This consists of two major research areas: providing a distributed (in the parallel computing sense) system for ray tracing, as well as a workbench for scene description, and image manipulation. The problems that need to be addressed by a system for parallel processing in a distributed loosely coupled system are quite different than those addressed by a tightly coupled parallel processor system. Because of the (likely) very high cost of communication in a distributed processing environment, most parallel algorithms currently used are not feasible (due to the high overhead). The gains of parallel ray tracing in a distributed environment are: the obvious speedup by bringing more processing

power to bear on the problem, the flexibility of distributed systems, and the availability of the resources that will become accessible as distributed systems become more prominent in the computer community.

Whew, what a mouthful. In a nutshell, I am interested in: ray tracing in general, parallel algorithms, distributed systems for image synthesis (any one know of any good references), and this new fangled radiosity stuff.

---

Joe Cychosz

Purdue University CADLAB  
Potter Engineering Center  
W. Lafayette, IN 47906

Phone: 317-494-5944  
Email: [cychosz@ecn.purdue.edu](mailto:cychosz@ecn.purdue.edu)

My interests are in supercomputing and computer graphics. Research work is Vectorized Ray Tracing. Other interests are: Ray tracing on MIMD tightly coupled shared memory machines, Algorithm vectorization, Mechanical design processes, Music synthesis, and Rendering in general.

---

Jerry Quinn  
Department of Math and Computer Science  
Bradley Hall  
Dartmouth College  
Hanover, NH 03755  
[sunapee.dartmouth.edu!quinn](mailto:sunapee.dartmouth.edu!quinn)

My interests are currently ray tracing efficiency, parallelism, animation, radiosity, and whatever else happens to catch my eye at the given moment.

---

Marty Barrett - octrees, parametric surfaces, parallelism.  
[mlb6@psuvm.bitnet](mailto:mlb6@psuvm.bitnet)

Here is some info about my interests in ray tracing:

I'm interested in efficient storage structures for ray tracing, including octree representations and hybrid regular subdivision/octree grids. I've looked at ray tracing of parametric surfaces, in particular Bezier patches and box spline surfaces, via triangular tessellations. Parallel implementations of ray tracing are also of interest to me.

---

Charles A. Clinton  
Sierra Geophysics, Inc.  
11255 Kirkland Way  
Kirkland, WA 98033 USA  
Email: ...!uw-beaver!sumax!ole!steven!cac  
Voice: (206) 822-5200  
Telex: 5106016171

I am doing scientific visualization of 3D seismic data. To see the kind of work that I am doing, check out:

'A Rendering Algorithm for Visualizing 3D Scaler Fields'  
Paolo Sabella  
Schlumberger-Doll Research  
Computer Graphics, Vol. 22, Number 4 (SIGGRAPH '88 Conference Proc.)  
pp 51-58

In addition, I try to keep up with ray-tracing and computer graphics in general. I occasionally try my hand at doing some artistic ray-tracing. (I would like to extend my thanks to Mark VandeWettering for distributing MTV. It has provided a wonderful platform for experimentation.)

---

Jochen Schwarze

I've been developing several smaller graphics packages, e.g. a 3D visualization of turning parts etc. Now I'm implementing the 2nd version of a ray tracing program that supports modular programming using a description language, C++ vector analysis and body class hierarchy, CSG trees, texture functions and mapping, a set of body primitives, including typeface rendering for logos, and a network ipc component to allow several cpu's to calculate a single image.

My interests lie - of course :- ) - in speedup techniques, and the simulation of natural phenomena, clouds, water, etc. Just starting with this.

Jochen Schwarze	Domain: <a href="mailto:schwarze@isaak.isa.de">schwarze@isaak.isa.de</a>
ISA GmbH, Stuttgart, West Germany	UUCP: <a href="mailto:schwarze@isaak.uucp">schwarze@isaak.uucp</a>
	Bang: ...!uunet!unido!isaak!schwarze
	S-Mail: ISA GmbH
	c/o Jochen Schwarze
	Azenberstr. 35
	7000 Stuttgart 1
	West Germany



back to [contents](#)

---

## Q&A on Radiosity Using Ray Tracing, by Mark VandeWettering and John Wallace

From Mark VandeWettering:

I am currently working on rewriting my ray tracer to employ radiosity-like effects. Your paper (with Wallace and Elmquist) is very nice, and suggests a really straightforward implementation. I just have a couple of questions that you might be able to answer.

When you shoot energy from a source patch, it is collected at a specific patch vertex. How does this energy get transferred to a given patch for secondary shooting? In particular, is the vertex shared between multiple patches, or is each vertex only in a single patch? I can imagine the solution if each vertex is distinct, but have trouble with the case where vertices are shared. Any quick insights?

The only other question I have is: HOW DO YOU GET SUCH NICE MODELS TO RENDER? [We use ME30, HP's Romulus based solids modeler - EAH]

Is there a public domain modeling package that is available for suns or sgi's that I can use to make more sophisticated models? Something cheap even?

[The BRL modeler and ray tracer runs on a large number of machines, and they like having universities as users - see [Vol.2 No. 2](#) (archive 6). According to Mike Muuss' write-up, some department in Princeton already has a copy.

The Simple Surface Modeler (SSM) works on SGI equipment. It was developed at the Johnson Space Center and, since they are not supposed to make any money off it, is being sold cheap (?) by a commercial distributor. COSMIC, at 404-542-3265, can send you some information on it. It also runs on a Pixel Machine (which is what I saw it running on at SIGGRAPH 88), though I don't believe support for this machine will be shipped. It's evidently not shipping yet (red tape - the product is done), but should be "realsoonnow". More information when I get the abstract. Does anyone else know any resources?]

---

Reply from John Wallace:

Computing the patch energy in progressive radiosity using ray tracing:

Following a step of progressive radiosity, every mesh vertex in the scene will have a radiosity. Energy is not actually collected at the mesh vertices. What is computed at each vertex is the energy per unit area (radiosity) leaving the surface at that location. The patch radiosity is the average energy per unit area over the patch. Finally, the patch energy is the patch radiosity times the patch area (energy per unit area times area).

The vertex radiosities can be considered a sampling of the energy per unit area at selected points across the patch. To obtain the average energy per unit area over the patch, take the average of the vertex radiosities. This assumes that the vertices represent uniform sub-areas of the patch. This is not necessarily true, and when it is not a more accurate answer is obtained by taking an area weighted average of the vertex radiosity. The weight given to a vertex is equal to the area of the patch that it represents. In our work we used a uniform mesh and weighted all vertices equally.

It doesn't matter whether vertices are shared by neighboring patches, since we're talking about energy per unit area. Picture four patches that happen to all share a particular vertex. The energy per unit area leaving any of the patches at the vertex is not affected by the fact that other patches share that vertex. If we were somehow collecting energy at the vertex, then it would have to be portioned out between the patches.

Once the patch radiosity is know, the patch energy is obtained by multiplying patch radiosity times patch area.



back to [contents](#)

---

## Dark Bulbs, by Eric Haines

An interesting idea mentioned to me by Andrew Glassner was the concept of "darkbulbs" in computer graphics. This idea is a classic joke technology, in which the darkbulb sucks light out of an area. For example, if you want to sleep during the daytime, you simply turn on your negative 100 watt dark bulb and your bedroom is flooded in darkness. Andrew noted that this technology is entirely viable in computer graphics, and would even be useful in obtaining interesting results.

I happened to mention the idea to Roy Hall, and he told me that this was an undocumented feature of the Wavefront



package! Last year Wavefront came out with an image of two pieces of pottery behind a candle, with wonderful texturing on the objects. It turns out that the artist had wanted to tone down the brightness in some parts of the image, and so tried negative intensity light sources. This turned out to work just fine, and the artist mentioned this to Roy, who, as an implementer of this part of the package, had never considered that anyone would try this and so never restricted the intensity values to be non-negative.



back to [contents](#)

## MTV Ray Tracer Update and Bugfix, by Mark VandeWettering

[this was extracted by me from personal mail, with parts appearing on comp.graphics - EAH]

As was recently pointed out to me by Mike Schoenborn, the cylinder code in the version of the MTV raytracer is broken somewhat severely. Or at least it appeared to be so, what actually happens is that I forgot to normalize two vectors, which leads to interesting distortions and weird looking cylinders. Anyway, the bug is in cone.c, in the function MakeCone(). After the vectors `cd -> cone_u` and `cd -> cone_v` are created, they should be normalized. A context diff follows at the end of this. This makes the SPD "tree" look MUCH better. (And all this time I thought it was Eric's fault :-)

This bugfix will be worked into the next release, and I should also update the version on [cs.uoregon.edu](http://cs.uoregon.edu) SOMETIME REAL SOON NOW (read, don't hold your breath TOO anxiously). Hope that this program continues to be of use... :-)

Somebody has some texture mapping code that they are sending me, I will probably try to integrate it in before I make my next release. I am also trying to get in spline surfaces, but am having difficulty to the point of frustration. Any recommendations on implementing them?

```
*** ../tmp/cone.c          Fri Aug 25 20:25:52 1989
--- cone.c                Fri Aug 25 21:31:04 1989
*****
*** 240,247 ****
--- 240,251 ----
        /* find two axes which are at right angles to cone_w
        */

+
+   VecCross(cd -> cone_w, tmp, cd -> cone_u) ;
+   VecCross(cd -> cone_u, cd -> cone_w, cd -> cone_v) ;
+
+   VecNormalize(cd -> cone_u) ;
+   VecNormalize(cd -> cone_v) ;

        cd -> cone_min_d = VecDot(cd -> cone_w, cd -> cone_base) ;
        cd -> cone_max_d = VecDot(cd -> cone_w, cd -> cone_apex) ;
```



back to [contents](#)

## DBW Ray Tracer Description

[A ray tracer that has been mentioned in these pages (screens?) before is DBW. Not having an Amiga and not being able to deal with "zoo" files, I never got a copy. Prem Subrahmanyam has now made it available via anonymous FTP from [geomag.gly.fsu.edu](http://geomag.gly.fsu.edu) in /pub/pics/DBW.src. Output is four bits for each channel.

The original program was written by David B. Wecker, translating from a Vax 11/750 to the Amiga, with a conversion to Sun workstations by Ofer Licht ([ofer@gandalf.berkeley.edu](mailto:ofer@gandalf.berkeley.edu)). - EAH]

Below is an excerpt from the documentation RAY.DOC:

The RAY program knows how to create images composed of four primitive geometric objects: spheres, parallelograms, triangles, and flat circular rings (disks with holes in them). Some of the features of the program are:

Diffuse and specular reflections (with arbitrary levels of gloss or polish). Rudimentary modeling of object-to-object diffusely reflected light is also implemented, that among other things accurately simulates color bleed effects from adjacent contrasting colored objects.

Mirror reflections, including varying levels of mirror smoothness or perfection.

Refraction and translucency (which is akin to variable microscopic smoothness, like the surface of etched glass).

Two types of light sources: purely directional (parallel rays from infinity) of constant intensity, and spherical sources (like light bulbs, which cast penumbral shadows as a function of radius and distance) where intensity is determined by the inverse square law.

Photographic depth-of-field. That is, the virtual camera may be focused on a particular object in the scene, and the virtual camera's aperture can be manipulated to affect the sharpness of foreground and background objects.

Solid texturing. Normally, a particular object (say a sphere) is considered to have constant properties (like color) over the entire surface of the object, often resulting in fake looking objects. Solid texturing is a way to algorithmically change the surface properties of an object (thus the entire surface area is no longer of constant nature) to try and model some real world material. Currently the program has built in rules for modelling wood, marble, bricks, snow covered scenes, water (with arbitrary wave sources), plus more abstract things like color blend functions.

Fractals. The program implements what's known as recursive triangle subdivision, which creates all manners of natural looking surface shapes (like broken rock, mountains, etc.). The character of the fractal surface (degree of detail, roughness, etc.) is controlled by parameters fed to the program.

AI heuristics to complete computation of a scene within a user specified length of time. [???



back to [contents](#)

===== USENET cullings follow =====

## **Wanted: Easy Ray/Torus Intersection, by Jochen Schwarze**

What I want to do is to turn a path consisting of line and arc segments around an axis and then ray-trace the generated turning part. The rotated line segments produce cylinders or cones that are easy to intersect with a ray, whereas the arcs produce tori. To evaluate the intersection of the ray with a torus I'd have to numerically solve a polynomial equation of fourth degree.

Does anybody know a way that avoids solving a general fourth- degree equation? Perhaps something that respects torus geometry and allows to split the equation into two quadric ones? Any other fast way to do it?

Thanks very much.

Jochen Schwarze  
ISA GmbH, Stuttgart, West Germany

Domain: [schwarze@isaak.isa.de](mailto:schwarze@isaak.isa.de)  
UUCP: [schwarze@isaak.uucp](mailto:schwarze@isaak.uucp)  
Bang: ...!uunet!unido!isaak!schwarze



back to [contents](#)

---

## Polygon to Patch NFF Filter, by Didier Badouel

This is a new filter program for NFF databases, it converts polygons (p) into patches (pp) computing normal vector for vertices.

---

Didier BADOUEL	<a href="mailto:badouel@irisa.fr">badouel@irisa.fr</a>
INRIA / IRISA	Phone : +33 99 36 20 00
Campus Universitaire de Beaulieu	Fax : 99 38 38 32
35042 RENNES CEDEX - FRANCE	Telex : UNIRISA 950 473F

---

[Code removed. Find it at [cs.uoregon.edu](http://cs.uoregon.edu) or write him - EAH]



back to [contents](#)

---

## Texture Mapping Resources, by Eric Haines, Prem Subrahmanyam, Ranjit Bhatnagar, and Jack Ritter

From: Eric Haines

Robert Minsk had a question about how to do inverse mapping on a quadrilateral. This was my response:

For the inverse bilinear mapping of XYZ to UV, see p. 59-64 of "An Introduction to Ray Tracing", edited by Andrew Glassner, Academic Press (hot off the press). Tell me if you find any bugs, since I need to send typos to AP. This same info is in the "Intro to RT" SIGGRAPH course notes from 1987 & 1988, with one important typo fixed (see old issues of the Ray Tracing News to find out the typo).

An excellent discussion of the most popular mappings (affine, bilinear, and projective), and for a discussion on why to avoid simple Gouraud interpolation, get a copy of Paul Heckbert's Master's Thesis (again, hot off the press), "Fundamentals of Texture Mapping and Image Warping". It's got what you need and is also a good start on sampling/filtering problems. Order it as Report No. UCB/CSD 89/516 (June 1989) from

Computer Science Division  
Dept of Electrical Engineering and Computer Sciences  
University of California  
Berkeley, California 94720

It was \$5.50 when I ordered mine. Oh, I should also note: it has source code in C for most of the algorithms described in the text.

From: [prem@geomag.fsu.edu](mailto:prem@geomag.fsu.edu) (Prem Subrahmanyam)  
Newsgroups: comp.graphics  
Subject: Re: Texture mapping  
Organization: Florida State University Computing Center

I would strongly recommend obtaining copies of both DBW\_Render and QRT, as both have very good texture mapping routines. DBW uses absolute spatial coordinates to determine texture, while QRT uses a relative position per each object type mapping. DBW has some really interesting features, like sinusoidal reflection to simulate waves, a turbulence-based marble/wood texture based on the wave sources defined for the scene. It as well has a brick texture, checkerboard, and mottling (turbulent variance of the color intensity). Writing a texture routine in DBW is quite simple, since you're provided with a host of tools (like a turbulence function, noise function, color blending, etc.). I have recently created a random-color texture that uses the turbulence to redefine the base color based on the spatial point given, which it then blends into the object's base color using the color blend routines. Next will be a turbulent-color marble texture that will modify the marble vein coloring according to the turbulent color. Also in the works are random color checkerboarding (this will require a little more thought), variant brick height and mortar color (presently they are hard-wired), the list is almost endless. I would think the ideal ray-tracer would be one that used QRT's user-definable texture patches which are then mapped onto the object, as well as DBW's turbulence/wave based routines. The latter would have to be absolute coordinate based, while the former can use QRT's relative position functions. In any case, getting copies of both of these would be the most convenient, as there's no reason to reinvent the wheel.

---

From: [ranjit@grad1.cis.upenn.edu](mailto:ranjit@grad1.cis.upenn.edu) (Ranjit Bhatnagar)  
4211 Pine St., Phila PA 19104  
Newsgroups: comp.graphics  
Subject: Re: Texture mapping by spatial position  
Organization: University of Pennsylvania

The combination of 3-d spatial texture-mapping (where the map for a particular point is determined by its position in space rather than its position on the patch or polygon) with a nice 3-d turbulence function can give really neat results for marble, wood, and such. Because the texture is 3-d, objects look like they are carved out of the texture function rather than veneered with it. It works well with non-turbulent texture functions too, like bricks, 3-d checkerboards, waves, and so on. However, there's a disadvantage to this kind of texture function that I haven't seen discussed before: as generally proposed, it's highly unsuited to \_animation.\_ The problem is that you generally define one texture function throughout all of space. If an object happens to move, its texture changes accordingly. It's a neat effect - try it - but it's not what one usually wants to see.

The obvious solution to this is to define a separate 3-d texture for each object, and, further, \_cause the texture to be rotated, translated, and scaled with the object.\_ DBW does not allow this, so if you want to do animations of any real complexity with DBW, you can't use the nice wood or marble textures.

This almost solves the problem. However, it doesn't handle the case of an object whose shape changes. Consider a sphere that metamorphoses into a cube, or a human figure which walks, bends, and so on. There's no way to keep the 3-d texture function consistent in such a case.

Actually, the real world has a similar defect, so to speak. If you carve a statue out of wood and then bend its limbs around, the grain of the wood will be distorted. If you want to simulate the real world in this way and get animated objects whose textures stay consistent as they change shape, you have to use ordinary surface-mapped (2-d) textures. But 3-d textures are so much nicer for wood, stone, and such! There are a couple of ways to get the best of both

worlds: [I assume that an object's surface is defined as a constant set of patches, whether polygonal or smooth, and though the control points may be moved around, the topology of the patches that make up the object never changes, and patches are neither added to or deleted from the object during the animation.]

- 1) define the base-shape of your object, and `_sample its surface_` in the 3-d texture. You can then use these sample tables as ordinary 2-d texture maps for the animation.
- 2) define the base-shape of your object, and for each metamorphosized shape, keep pointers to the original shape. Then, whenever a ray strikes a point on the surface of the metamorphed shape, find the corresponding point on the original shape and look up its properties (i.e. color, etc.) in the 3-d texture map. [Note: I use ray-tracing terminology but the same trick should be applicable to other techniques.]

The first technique is perhaps simpler, and does not require you to modify your favorite renderer which supports 2-d surface texture maps. You just write a preprocessor which generates 2-d maps from the 3-d texture and the base-shape of the object. However, it is susceptible to really nasty aliasing and loss of information. The second technique has to be built into the renderer, but is amenable to all the antialiasing techniques possible in an ordinary renderer with 3-d textures, such as DBW. Since the notion of 'the same point' on a particular patch when the control points have moved is well-defined except in degenerate cases, the mapping shouldn't be a problem -- though it does add an extra level of antialiasing to worry about. [Why? Imagine that a patch which is very large in the original base-shape has become very small - sub-pixel size - in the current animated shape. Then a single pixel-sized sample in the current shape could be mapped to a large part of the original - using, for instance, stochastic sampling or analytic techniques.]

If anyone actually implements these ideas, I'd like to hear from you (and get credit, heh heh, if I thought of it first). I doubt that I will have the opportunity to try it.

---

From: [ritter@versatc.UUCP](mailto:ritter@versatc.UUCP) (Jack Ritter)  
Organization: Versatec, Santa Clara, Ca. 95051

[Commenting on Ranjit's posting]

It seems to me that you could solve this problem by transforming the center/orientation of the texture function along with the object that is being instantiated. No need to store values, no tables, etc. The texture function must of course be simple enough to be so transformable.

Example, wood grain simulated by concentric cylindrical shells around an axis (the core of the log):

Imagine the log's center line as a half-line vector, (plus a position, if necessary), making it transformable. Imagine each object type in its object space, BOLTED to the log by an invisible bracket. As you translate and rotate the object, you also sling the log around. But be careful, some of these logs are heavy, and might break your teapots. I use only natural logs myself.

Jack Ritter, S/W Eng. Versatec, 2710 Walsh Av, Santa Clara, CA 95051  
Mail Stop 1-7. (408)982-4332, or (408)988-2800 X 5743  
UUCP: {ames,apple,sun,pyramid}!versatc!ritter



back to [contents](#)

---

Eric Haines / [erich@acm.org](mailto:erich@acm.org)

# Ray Tracing News

*"Light Makes Right"*

October 27, 1989

Volume 2, Number 8

Compiled by [Eric Haines erich@acm.org](mailto:erich@acm.org). Opinions expressed are mine.

All contents are copyright (c) 1989, all rights reserved by the individual authors

Archive locations: anonymous FTP at <ftp://ftp-graphics.stanford.edu/pub/Graphics/RTNews/>, [wuarchive.wustl.edu:/graphics/graphics/RTNews/](http://wuarchive.wustl.edu:/graphics/graphics/RTNews/), and many others.

You may also want to check out [the Ray Tracing News issue guide](#) and the [ray tracing FAQ](#).

---

## Contents:

- [Introduction](#)
  - [Tracing Tricks](#), edited by Eric Haines
    - [Ambient Light](#)
    - [Efficiency Schemes](#)
    - [Bounding Volume Hierarchy](#)
    - [Octree](#)
    - [Bounding Box Intersection](#)
    - [Spline Surface Intersection](#)
    - [Acknowledgements](#)
    - [Bibliography](#)
- 

## Introduction

I've decided to pass on an article published in the SIGGRAPH '89 "Introduction to Ray Tracing" course notes. It's something of a "best of the Ray Tracing News" compendium of ideas. Since the notes are not easy for everyone to access, and the article probably will not be printed elsewhere, I thought it worthwhile to reprint here.



back to [contents](#)

---

## Tracing Tricks, edited by Eric Haines

[previous discussion of topic](#)



Over the years I have learnt a variety of tricks to increase the performance and image quality of my ray tracer. It's almost a cliché that today's successful trick is tomorrow's established technique. Photorealistic computer graphics is, after all, concerned with figuring out shortcuts and approximations for rendering various physical phenomena, i.e. tricks.

For whatever reason, many of the tricks mentioned here are not common knowledge. Some have been published (and sometimes overlooked), some have been discussed informally and have never made it into research papers, and others seem to have appeared out of nowhere. It's most likely that there are tricks that are commonly known that have not percolated over to me yet.

When possible, I have tried to give appropriate references or attributions; if not attributed, the ideas are my own (I think!). My apologies if I have overlooked anyone. Only references that do not appear in the book's "Ray Tracing Bibliography" are included at the end of this article. For more general rendering hacks, see [Whitted85], which originally inspired me to attempt to pass on some ideas from my bag of tricks.



back to [contents](#)

---

## Ambient Light

One common trick (origins unknown) is to put a light at the eye to do better ambient lighting. Normally if a surface is lit by only ambient light, its shading is pretty crummy. For example, a non-reflective cube totally in shadow will have all of its faces shaded the exact same shade. All edges disappear and the cube becomes a hexagonal blob. The light at the eye gives the cube definition. Note that a light at the eye does not need shadow testing: wherever the eye can see, the light can see, and vice versa. However, this trick can lead to various artifacts, e.g. there will always be a highlight near the center of every specular sphere in the image.



back to [contents](#)

---

## Efficiency Schemes

There are any number of efficiency schemes out there, including Bounding Volume Hierarchy, Octree, Grid, and 5D Ray Classification. See Jim Arvo's section of the book for an excellent overview of all of these. The most important conclusion is that any efficiency scheme is better than none. Even on the simplest scenes an efficiency scheme will help execution. For example, in one test scene with only ten objects, using an efficiency scheme made the job take only one third the time. Grid subdivision is probably the quickest to implement, though the others are not that much harder.

While at the University of Utah, John Peterson and Tom Malley actually implemented Whitted/Rubin, Kay/Kajiya, and an octree scheme, and found that all three schemes were within 10-20% of each other speedwise. In an informal survey at SIGGRAPH '88, the BV Hierarchy, Octree, Grid and 5D schemes all had about the same number of users (all the 5D users were from Apollo; on the other hand, 5D is the new kid on the block).

There are a number of techniques I have found to be generally useful for all efficiency schemes.

- 1) When shadow testing, keep the opaque object (if any) which shadowed each light for each ray tree node. Try this object immediately during the next shadow test at that ray tree node. Odds are that whatever shadowed your last intersection point will shadow again. If the object is hit you can immediately stop testing because the light is not seen. This was first published in [Haines86].

- 2) When shadow testing, save transparent objects for later intersection. Only if no opaque object is hit should the



transparent objects be tested. The idea here is to avoid doing work on transparent filters when in fact the light does not reach the surface.

3) Don't calculate the normal for each intersection. Get the normal only after all intersection calculations are done and the closest object for each node is known. After all, each ray can have only one intersection point and one normal. Saving intermediate results is worthwhile for some intersection calculations, which are then used if the object is actually hit. This idea was first mentioned in [Whitted85]. Similarly, other calculations about the surface can be delayed, such as (u,v) location, etc.

4) One other idea (which I have not tested) is sorting each intersection list by various criteria. Most efficiency schemes have in common the idea of lists of objects to be tested. For a given list, the order of testing is important. For example, all else being equal, if a list contained a spline surface and a polygon, I would want to test the polygon first since it is usually a quicker intersection test. Given an opaque object and a bounding box in a list, I probably want to test the opaque object first when doing shadow testing, since I want to find any intersection as soon as possible. If two polygons are on the list, I probably want to test the larger one first, as it is more likely to cast a shadow or give me a maximum depth (see next section). There are many variations on this theme and at this point little work has been done on these possibilities.



back to [contents](#)

## Bounding Volume Hierarchy

I have a strong bias towards this scheme since it handles a wide variety of object sizes, types, and orientations in a robust fashion. Other schemes will often be faster, but this scheme has the fewest crippling pathological cases (e.g. a grid subdivision scheme is useless whenever most of the objects fall into one grid box). I favor the automatic bounding volume generation technique described by [Goldsmith87].

I have found a number of tricks to speed up hierarchy traversal, most of which are simple to implement. Some of the ideas can also be useful for other efficiency schemes.

1) Keep track of the closest intersection distance. Whenever an object is hit, keep its distance as the maximum distance to search. During further intersection testing use this distance to cut short the intersection calculations: if an object or bounding box is beyond the maximum distance, no further testing of it needs to be done. Note that for shadow testing the distance to the light provides an initial maximum.

2) When building a ray intersection tree, keep the ray tree which was previously built. For each ray tree node, intersect the object in the old ray tree, then proceed to intersect the bounding box/object tree. By intersecting the old object first you can usually obtain a good maximum distance immediately, which can then be used to aid trick #1.

3) When shooting rays from a surface (e.g. reflection, refraction, or shadow rays), get the initial list of objects to intersect from the bounding volume hierarchy. For example, a ray beginning on a sphere must hit the sphere's bounding volume, so include all other objects in this bounding volume in the immediate test list. The bounding volume which is the parent of the sphere's bounding volume must also automatically be hit, and its other children should automatically be added to the test list, and so on up the object tree. Note also that this list can be calculated once for any object, and so could be created and kept around under a least-recently-used storage scheme. Another advantage of this scheme is that nearby neighbors of the object are tested for shadowing first. These neighbors are more likely to cast a shadow on the point than any random object. I first saw this trick used in Weghorst and Hooper's ray tracer at Cornell's Program of Computer Graphics.

4) Similar to trick #3, the idea is simply to do the same list making process for the eye position. Check if the eye position is inside the topmost node of the hierarchy. If it is, check the children which are boxes. Continue to check

and unwrap until you are left with a list of objects to intersect. Again, the idea is to avoid wasting time shooting a ray against boxes which you know must be hit.

For light sources, since the farthest endpoint of the ray is also known, it can also be used to open some boxes early on. The tradeoff here, however, is that for shadow testing we want to find any intersection we can. Wasting time opening boxes near the light or ray origin might be better spent trying to find an intersection as fast as possible.

5) An improvement to trick #3 is also to use trick #4 to open more boxes initially. You work up the hierarchy opening all parent boxes; any children of the parent (except the original one, of course) are then tested against the ray position. However, this can be done only when the trick is done on the fly, since the ray's origin will change.

Kay & Kajiya's hierarchy scheme [Kay86] is about the best overall traversal method. However, Jeff Goldsmith and others note that if you do use Kay & Kajiya's heapsort on bounding volumes in order to get the closest, don't bother to do it for illumination rays. In shadowing, you don't care about the closest intersection, but just whether anything blocks the light.



back to [contents](#)

## Octree

There are a few tips on accessing and moving through the octree. Olin Lathrop and others have pointed out that there is a faster method than Glassner's hashing scheme for finding which octree voxel contains a point. Quickest is to simply transform the point into the octree's (0,0,0) to (1,1,1) space. Say you allow your octree a maximum of eight levels. This means you'll want to convert from world coordinates to eight bits in X, Y, and Z. For example, if the octree box went from 3.0 to 6.0 along the X axis in world space, you would convert 5.25 into the binary fraction 0.11000000 (which is equal to 0.75 decimal, which is where 5.25 lies between 3.0 and 6.0). The most significant bit of each binary fraction for each coordinate is then combined and used to access the correct topmost octree voxel (i.e. 0 through 7, similar to Glassner's scheme). The next-most significant three bits are then stripped off, and the corresponding subordinate octree voxel is accessed, on down until a leaf voxel is found.

In practice, each octree voxel notes whether it is a parent of further voxels or is a leaf and contains a list of objects to hit. If it is a parent, it stores a list of 8 pointers to its subordinate octrees, with null pointers meaning that the subordinate octree is empty; otherwise, a list of objects is used.

One problem with building octrees is deciding when enough is enough. You want to subdivide an octree voxel if the number of objects is too many, but you may find that these further subdivisions do not gain you anything. Olin Lathrop has an interesting method for octree subdivision. First, the biggest win is to subdivide on the fly. Never subdivide anything until you find there is a demand for it (this same idea was used by Arvo and Kirk [Arvo87] in their 5D efficiency scheme). His subdivision criteria are, in order of precedence:

- 1) Do not subdivide if subdivision generation limit is hit.
- 2) Do not subdivide if a voxel contains less than X objects (These first two criteria were first proposed in [Glassner84]). Olin uses X=1.
- 3) Do not subdivide if less than N rays passed through this voxel, but did not hit anything. Olin uses N=4.
- 4) Do not subdivide if  $M \cdot K \geq N$ , where M is the number of rays that passed through this voxel that did hit something, and K is a parameter you chose. Olin uses K=2, but suspects it should be higher. This step seeks to avoid subdividing a voxel that may be large, but has a good history of producing real intersections anyway. Keep in mind

that for every ray that did hit something, there are probably shadow test rays that did not hit anything. This can distort the statistics, and make a voxel appear less "tight" than it really is, hence the need for larger values of  $K$ .

Another possible criterion is to base the subdivision generation limit (criterion 1) on the number of objects in the octree. If you had, say, 6 objects and 5 of them are clustered tightly together, you may find your octree reaching its maximum depth without the subordinate octrees actually splitting up the 5 objects. These octree voxels are useless, costing extra time and memory. They could be avoided by setting the limit based on the total number of objects. I use something along the lines of the depth limit being equal to  $\log_2$  of (number of objects).

Andrew Glassner has a better method to avoid this problem. When you subdivide a voxel, look at its children. If only one child is non-empty, replace the original voxel with its non-null child. Do this recursively until the subdivision criterion is satisfied. He does this in his spacetime ray tracer, and the speedup can be large. Note that this scheme means adding a field to the octree structure to identify what level in the hierarchy it represents.

An idea to speed octree traversal was first mentioned to me by Andrew Glassner and later by Mike Kaplan. The idea is to place a pointer on each face of each octree voxel. If a voxel's face is next to a larger or same size voxel, a pointer to this neighbor is stored. If the voxel face's neighbors are smaller, then the face pointer is set to point at the bordering voxel of the same size (which is these neighbors' common parent). If there are no neighbors (i.e. the face is on the exterior), a null pointer is stored.

When a ray exits a voxel, the voxel face is accessed and the next voxel found directly. This voxel may have to be descended, but this trick saves having to descend the octree from the top.

Mike Kaplan independently arrived at a similar method, in which he stores quadtrees at the faces so that he can immediately access the next voxel and avoid any descent altogether.



back to [contents](#)

---

## Bounding Box Intersection

The fastest method in general is Kay and Kajiya's slab intersection method [Kay86]. As they point out, precompute as much as you can for the ray, i.e. check whether the ray is parallel to any of the axes, and for whichever axes it is not, computing the multiplicative inverse of the ray direction vector. However, there are other tests which can actually improve the performance of the box intersector. For example, I have found that for my particular ray tracer, if we first do a quick inside-outside test with the ray origin and the box, the overall box intersection time goes down (for a related trick, which is something of a preprocess version of this method, see #4 under "Bounding Volume Hierarchy").



back to [contents](#)

---

## Spline Surface Intersection

There are three camps on this question: the numerical analysts, the polygon meshers, and the synthesists (who do a little of each). The following comments are distilled from John Peterson's thoughts on the subject. The analytic methods are often slow, and there are many nightmares involved in finding roots of two equations (see section 9.6 of [Press86]). To find the quadrilaterals, John recommends subdividing the surfaces by using the Oslo Algorithm, due to its generality [Bartels87, Sweeney86]. Easiest to implement is simply subdividing the surface by a given step size, then ray tracing the mesh of polygons produced (throwing these polygons into an octree or grid efficiency scheme is recommended). Another method is to use adaptive subdivision with a quadtree structure, checking a flatness criteria

to see whether a given quadrilateral should be subdivided into four sub-quadrilaterals.

Peterson's subdivision criterion is to use a bounding box around each quad generated, subdividing until the box is smaller than a certain number of pixels. A drawback of this method is that it does not elegantly handle objects that are part of the scene yet do not appear in the viewing frustum (e.g. if the teapot is only seen in a mirror, we cannot get a good sense of how much to subdivide it). Snyder and Barr [Snyder87] have some good recommendations on this process, and they use the change in the tangent vector between the quad's points as a subdivision criterion. This article also points out other pitfalls of tessellation and of rendering polygons with a normal at each vertex.

If adaptive techniques are used, one problem to guard against is cracking. Say there are two adjacent quadrilaterals, and one has been subdivided into four smaller quads. Something must be done along the seam between the two large quadrilaterals, as normally the subdivision point between the two common vertices will not lie on the large, undivided quad. If rendered from some angle, there will be a noticeable crack between the large quad and the two neighboring smaller quads.



back to [contents](#)

---

## Acknowledgements

This article owes a large debt to Andrew Glassner, who began "The Ray Tracing News," an informal journal for ray tracing researchers to share ideas. He has kept the hardcopy version moving along, while I have had the pleasure of running the electronic edition. Most of the ideas given a personal attribution in this article are from contributions to the "News", and I thank all those who have contributed to it over the years. Finally, my thanks to Kells Elmquist and Andrew Glassner for their comments on this paper.



back to [contents](#)

---

## Bibliography

[Bartels87] Bartels, Richard H., John C. Beatty, Brian A. Barsky, An Introduction to Splines for Use in Computer Graphics, Morgan Kaufmann, Los Altos, California, 1987.

[Press88] Press, William H. et al, Numerical Recipes in C, Cambridge University Press, Cambridge, 1988.



back to [contents](#)

---

Eric Haines / [erich@acm.org](mailto:erich@acm.org)

# Ray Tracing News

*"Light Makes Right"*

October 13, 1994

Volume 7, Number 5

Compiled by [Eric Haines erich@acm.org](mailto:erich@acm.org). Opinions expressed are mine.

All contents are copyright (c) 1994, all rights reserved by the individual authors

Archive locations: anonymous FTP at <ftp://ftp-graphics.stanford.edu/pub/Graphics/RTNews/>, [wuarchive.wustl.edu:/graphics/graphics/RTNews/](http://wuarchive.wustl.edu:/graphics/graphics/RTNews/), and many others.

You may also want to check out [the Ray Tracing News issue guide](#) and the [ray tracing FAQ](#).

---

## Contents:

- [Introduction](#)
- [New People](#)
- [Ray Tracing Roundup](#)
- [Recent Ray Tracing Papers](#)
- [Rumor Mill](#)
- [AERO Animation/Simulation System version 1.5.1](#), by Thomas Braeunl
- [A Brief Review of \[an old\] AERO](#), by Dave Negro
- [Photon Tracing](#), by Chris Thornborrow and Greg Ward
- [Faster Than POV-RAY 2.1](#), by Dieter Bayer
- [Z Buffer Based Rendering Program](#), by Raghu Karinithi
- [Gossamer](#), a Free Macintosh VR/3D Renderer, by Jon Blossom
- [Antialiasing Issues](#), by Arijan Siska
- [Microcosm](#), by Abe Megahed of Cosmic Software
- [Fisheye Lens Distortion](#), by Greg Ward
- [Optical Ray Tracers](#)
- [Correcting Normal Direction](#), by Gavin Bell
- [Graphics Gems IV Table of Contents](#), by Paul Heckbert
- [Beyond Graphics Gems](#), by Paul Heckbert
- [Radiosity vs. Ray Tracing](#), by Rico Tsang
- [ACM SIGGRAPH Online Bibliography Updated](#), by Frank Kappe
- [How to be Notified of New POV Releases](#)
- [PoVSB Windows-based Modeler v0.85](#), by Jeff Hauswirth

- [Porting Rayshade](#), PBM, etc from Unix to DOS, by Mike Castle
  - [REYES & Patents](#), William C. Archibald
  - [Going from AutoCAD and 3DS into Ray Tracing](#), by Sean Ross
  - [Computer Lego Modeling](#), by Paul Gyugyi
  - [Rowe's Ray Tracing World BBS](#), by Harry Rowe
  - [On Using BSP trees](#), by Benton Jackson
  - [Books about Commercial Renderers](#), by Don Lewis, Jimbo and Yury German
- 

## Introduction

Another SIGGRAPH has come and gone, and it was a pretty fun one. Getting soaked time and again along with other allegedly intelligent people by the whales at the SeaWorld reception was definitely memorable. In many ways this conference was better than previous years. The proceedings and course notes were also provided on CD-ROM (some headaches on some machines, but that's the norm). Being able to move course to course was wonderful (even with the headache of getting to and from Stouffer's, which was no fault of the conference organizers). The addition of the technical sketches sessions and parallel paper sessions were both plusses to me. Note that there are a number of great resources on the various CD-ROMs, e.g. the latest release of Radiance (since there was a paper on the system in the proceedings this year).

The most noticeable technology for me was Apple's QuickTime VR. Nothing earth-shattering, but something which looks to be popular. Essentially you can move from view to view of an environment. You have a panoramic view of wherever you're located, so you can turn your head, zoom in on something, and even look up and down a bit: there's a nice illusion that you're interacting with the environment but without having more than a single panoramic view for a location. The software does all the distortion correction on the fly. The first product that is out using this technique is a tour of the Enterprise (TNG) on CD-ROM, and I suspect you'll see a lot of Apple's technique in upcoming CD-ROM tours and adventure games (e.g. rumor has it the Myst guys were looking at the technique).

So what's this have to do with ray tracing? Well, ray tracing is a natural for rendering panoramic views (Ken Musgrave has a Gem on how to do this in "Graphics Gems III"). Scanline images can be used and glued together and then warped, but ray traced images give you the distortion you need for free and at the sampling rates you need where you need them.

This is yet another catch-up issue, where the bits and pieces left in the queue since April are gathered up and spewed out now. If anything, have the material molder in my files for a few months meant I've culled a bit more than usual, leaving a denser filling with a firm yet flakey crust. Food imagery aside, there *are* some worthwhile bits in this issue: AERO, a demo of Microcosm, a new z-buffer renderer, and so on. Next issue will have more juicy stuff (and I really do hope to get it out before the Millenium arrives), including an announcement of a non-Pixar-produced shareware RenderMan implementation!



back to [contents](#)

---

## New People

# Werner Purgathofer - radiosity, ray tracing, visualization, color  
# Technical University of Vienna  
# Institute of Computer Graphics  
# Karlsplatz 13 / 186-2  
# A-1040 Wien / Austria



# +43 (1) 58801 4548

alias [purgathofer@cg.tuwien.ac.at](mailto:purgathofer@cg.tuwien.ac.at)

I am a professor of computer graphics, my main job consisting of teaching, administration and research. My group consists of about 12 people, including a technician and a secretary. We did a lot of ray tracing some years ago, including distributed ray tracing (see e.g. Eurographics 86), now we are concentrating on radiosity issues. As (forward) ray tracing becomes more and more important here, we are almost back where we started!



back to [contents](#)

---

## Ray Tracing Roundup

Thought folks might be interested in the On-Line Ray-Tracing Bibliography, available over the World Wide Web via URL:

<http://www.cm.cf.ac.uk/Ray.Tracing/>

Even those without Web clients like Mosaic can check this out by telneting to one of these sites:

If you're near:	Telnet to:
Switzerland:	<a href="http://info.cern.ch">info.cern.ch</a> or 128.141.201.74
Kansas, USA:	<a href="http://ukanaix.cc.ukans.edu">ukanaix.cc.ukans.edu</a> (login as www)
New Jersey, USA:	<a href="http://www.njit.edu">www.njit.edu</a> (login as www)
Israel:	<a href="http://vms.huji.ac.il">vms.huji.ac.il</a> or 128.139.4.3 (login as www)
Slovakia:	<a href="http://sun.uakom.cs">sun.uakom.cs</a> (slow link; use only from nearby)
Hungary:	<a href="http://fserv.kfki.hu">fserv.kfki.hu</a> (slow link, login as www)
Finland:	<a href="http://info.funet.fi">info.funet.fi</a> or 128.214.6.100

Mark Maimone ([Mark.Maimone@A.GP.CS.CMU.EDU](mailto:Mark.Maimone@A.GP.CS.CMU.EDU))

[This is Ian Grimstead's doing, and "bibliography" is a misnomer - it's a great place to start looking for all sorts of ray tracing and rendering resources, with links to just about everywhere. Really, I could just list this site each issue and not have to list most of the other resources that I do. -EAH]

---

The FAQ for comp.graphics.algorithms are available at:

<http://www.cis.ohio-state.edu/hypertext/faq/usenet/graphics/algorithms-faq/faq.html>

[ftp://rtfm.mit.edu/pub/usenet-by-group/news.answers/graphics/algorithms-faq](http://rtfm.mit.edu/pub/usenet-by-group/news.answers/graphics/algorithms-faq)

Also available at: [ftp://wuarchive.wustl.edu/graphics/graphics/mail-lists/comp.graphics.algorithms](http://wuarchive.wustl.edu/graphics/graphics/mail-lists/comp.graphics.algorithms)

FYI, all usenet FAQ's are available with Mosaic via: <http://www.cis.ohio-state.edu/hypertext/faq/usenet/top.html>

Jon Stone ([jdstone@ingr.com](mailto:jdstone@ingr.com))

---

A computer graphics related FTP list [maintained by Nick Fotis and myself. -EAH] is kept at:

[wuarchive.wustl.edu](http://wuarchive.wustl.edu/graphics/graphics/ray/GraphicsFTP.txt): /graphics/graphics/ray/GraphicsFTP.txt

George Kyriazis ([kyriazis@esd.sgi.com](mailto:kyriazis@esd.sgi.com))

---

I've been playing around with HTML and the World-Wide Web lately, and I have created hypertext versions of the three latest RTNews issues, with more to come when I get the time and energy to do it...

RTNews is at

<http://www.lysator.liu.se:7500/users/ture/rtnews.html>

I also have some general graphics stuff at .../graphics.html, and my own presentation at .../ture.html

Ture Persson ([ture@lysator.liu.se](mailto:ture@lysator.liu.se))

---

## Texture Collections

: I'm looking into different texture collections on CD-ROMs. Are there any that are absolutely awesome, or others that should be avoided (i.e. not worth the money)? "Accents" looks pretty nice (in reviews, etc), and also Pixar's \$99 offer for "One Twenty Eight" CD-ROM texture collection looks good.

I just finished writing an article on this very topic for Digital Video Magazine. It should be in the September issue, and it covers about twenty or so packages for the Amiga, Mac and PC, most of them on CD-ROM. The Pixar set you mentioned is very good, as are Autodesk's Texture Universe and Texture City. Check the article out for more info on a bunch of packages. Hope this is helpful.

[Digital Video Magazine is at 603-924-0100 in Peterborough, NH. -EAH]

Dave Thomas ([dthomas@bbx.basis.com](mailto:dthomas@bbx.basis.com))

Moving Pixels

(Client, looking at image of a sphere on a monitor: "Can we make it rounder?")

---

Recent Uploads to <ftp.cica.indiana.edu>

File: desktop/gcad110.zip Date: 940311 Desc: GammaCAD v1.10 - Full featured

File: programr/3dlib30a.zip Date: 940315 File: programr/3dlib30b.zip Date: 940315 Desc: C++ & Pascal 3D graphic animation

---

As a new feature, the fractal FAQ has some links for use with the World Wide Web. It can be accessed with a program such as xmosaic at <http://www.cis.ohio-state.edu/hypertext/faq/usenet/fractal-faq/faq.html>

Ken Shirriff ([shirriff@sprite.berkeley.edu](mailto:shirriff@sprite.berkeley.edu))

---

New version of RTrace 8.4.0

There is a new version of the RTrace ray-tracing package (8.4.0) at [asterix.inescn.pt](http://asterix.inescn.pt) [192.35.246.17] in directory



pub/RTrace. Check the README file.

Antonio Costa, Comp. Graphics/CAD ([cgcad@bart.inescn.pt](mailto:cgcad@bart.inescn.pt))

---

## New Game Programming FTP-Place

I have the great pleasure to officially announce the new (pc) game programming oriented ftp-place [teeri.oulu.fi](ftp://teeri.oulu.fi). It's currently an old sun3, has max 20 user limit, is located in Finland (Europe) and has about 40M more or less game programming oriented material online + lots more to come.

[teeri.oulu.fi:/pub/msdos/programming](ftp://teeri.oulu.fi/pub/msdos/programming)

(uploads to [teeri.oulu.fi:/incoming](ftp://teeri.oulu.fi/incoming) with a .txt file, please)

There are graphics libs, sound libs, hardware tech specs, file format specs, drawing utils, music composing utils, sample game source, some carefully selected FAQs, file format converters, all 3 known Game Programmers Magazines, is the official home && dist site for Game Programmers Encyclopedia etc. etc.

The PC Games Programmers Encyclopedia 1.0 (PC-GPE) is in [/pub/msdos/programming/gpe/pcgpe10.zip](ftp://pub/msdos/programming/gpe/pcgpe10.zip)

Jouni Miettunen ([jon@stek.oulu.fi](mailto:jon@stek.oulu.fi))

[Another place to look for PC related graphics source code is: [ftp.uwp.edu: /pub/msdos/demos/programming/source-EAH](ftp://ftp.uwp.edu/pub/msdos/demos/programming/source-EAH)]

---

## PCFormat (UK) RTPics on cover CD

Just thought UK readers might like to know that this month, PC Format magazine have launched a copy which comes with a CD rather than the usual disks. On it are over 150 ray-traced images, most of which are pretty stunning and in at least 640x480 (GIFs though :-), so only 256 colours). The ray-traced trees generated with LParser are particularly gobsmacking! Even 'Frosty' is on there, but compared to the rest, I found it to be pretty average (sorry Dan, but I think some of your others are MUCH better).

IMHO, it's well worth a look, and Pc Format can be reached at [format@compulink.cix.co.uk](mailto:format@compulink.cix.co.uk) if you want to try & persuade them to put more ray-tracing stuff on the disc in future (such as source files, image data etc).

PS: The Bowling.GIF has incredible motion blur (dunno what it was done on though!)

William Turner ([wturner@acorn.co.uk](mailto:wturner@acorn.co.uk))

---

## Geombib Reminder

Otfried Schwarzkopf has set up an experimental server on the world-wide web for the geometry biblio; you can try it by pointing your browser (e.g. mosaic for xwindows) at

<http://www.cs.ruu.nl/people/otfried/html/geombib.html>

The server can be used to do remote searches on the bibliography (it has biblook under the hood). When the entry has been annotated with the new URL field, the server will recognize this and allow you to download the network object pointed to, e.g. a Postscript copy of a techreport. (You can find some examples by looking for "ftp" in "any" field.) If you have comments or suggestions regarding the server, please send them to [otfried@cs.ruu.nl](mailto:otfried@cs.ruu.nl).

If you wish to join the computational geometry bibliography mailing list, send email with body of "subscribe geombib" to [majordomo@cs.usask.ca](mailto:majordomo@cs.usask.ca). Beware that mail sent to [geombib-request@cs.usask.ca](mailto:geombib-request@cs.usask.ca) gets a form letter from majordomo but not a subscription.

[You can FTP the bib from [cs.usask.ca](http://cs.usask.ca): /pub/geometry/geombib.tar.Z -EAH]

Bill Jones ([jones@skdad.usask.ca](mailto:jones@skdad.usask.ca))

---

Animation FTP Site

[toe.cs.berkeley.edu](http://toe.cs.berkeley.edu) in directory /pub/multimedia/mpeg/movies

has some quite good mpeg streams including the morphing sequence from Michael Jackson's 'Black and white' video.

Jim Marsden ([ee\\_c456@vulture.dcs.kingston.ac.uk](mailto:ee_c456@vulture.dcs.kingston.ac.uk))

---

I have recently completed version 1.1 of RenderCAD-PRO, a modeling and ray-tracing application for the Macintosh. Its easy graphic user interface can be used to generate photo-realistic single images and multi-image animations. A free demo (requires Mac w/ FPU co-pro) of this \$45 package is available. Please send info requests to...

Paul Rybarczyk ([prybarc@heartland.bradley.edu](mailto:prybarc@heartland.bradley.edu))

---

If you have direct Internet access, you can learn about BRL-CAD via:

ftp to <ftp.arl.mil>, directory /brl-cad

Mosaic/WWW to the following URL:

<http://www.brl.mil/software/brlcad/index.html>

Lee A. Butler ([butler@ARL.MIL](mailto:butler@ARL.MIL))

---

The current version of the Stereogram FAQ is in the HTML file at <http://www.cs.waikato.ac.nz/~singlis/sirds.html> or can be FTPed from: [katz.anu.edu.au](http://katz.anu.edu.au) [150.203.7.91] : /pub/stereograms (which also has a lot of other SIRDS related information)

[Stereograms are those things you see in the mall (trade name "Hollusion") that you try to stare at and defocus and see the dinosaur or whatever in 3D. I could never see the effect until I learned about the techniques listed in the FAQ. Fun illusion, and it turns out to be very easy to implement as a post process to a ray tracer - I believe people have already done it for POV and Rayshade. Also, at the listed FTP site there is a ray tracer called raysis which is devoted to this task. To do this stuff yourself you can try to decode the FAQ (it's a tad cryptic when it comes to actually saying what to do for SIS), buy the new book \_Create Stereograms on Your PC\_ by Dan Richardson, Waite Group Press, or wait an issue: I hope to write up the 20-30 lines of code I used to add it to my renderer for the heck of it. -EAH]

---

Raytech is a BBS specialising in ray tracing. It is located in northern Scotland, and their telephone number is 44-862-832020.

Adam Fulcher ([afulcher@cix.compulink.co.uk](mailto:afulcher@cix.compulink.co.uk))

Just about everything that always gets asked over and over again on rec.games.programmer is covered in the latest Dr. Dobbs: circle, ellipse, line, texture mapping, landscape generation, and even ray-tracing. So before you ask get that episode (#216 July 1994 ddj).

Ben ([sbc5a@helga6.acc.Virginia.EDU](mailto:sbc5a@helga6.acc.Virginia.EDU))

---

## RT Ray Tracer

I have implemented a CSG Ray Tracer with lots of primitives, textures, bounding volumes in C (but I've no multi-processing/wire-frame stuff).

My Ray Tracer is called RT and is available on [ftp-os2.nmsu.edu:os2/2\\_x/graphics/rt.zip](ftp-os2.nmsu.edu:os2/2_x/graphics/rt.zip). It comes with full portable C source and uses the GBM module for file I/O (found in same directory as gbmsrc.zip). I will refresh it with a version less sensitive to rounding errors sometime soon.

Andy Key ([ak@hursley.ibm.com](mailto:ak@hursley.ibm.com))

---

## Equation for a Cow, er Frog

There are some datasets at:

<http://george.lbl.gov/ITG.html>

including a rasterization of a frog that was made by freezing, slicing, and scanning. (They planned to do an MRI scan, but frogs don't scan well.)

There's also MRI data for an orange, a tomato, a pumpkin, and a rat.

Roger Critchlow ([rec@arris.com](mailto:rec@arris.com))

---

The WorldToolKit User's Group (SIG-WTK) electronic archive is located at

[artemis.arc.nasa.gov](ftp.artemis.arc.nasa.gov): /sig-wtk

This anonymous ftp site serves as the software and documents library for the SIG-WTK. It is intended to serve the needs of active WTK users and other parties interested in WorldToolKit.

The site is organized within the ~ftp/sig-wtk as follows:

incoming/	for anonymous contributions
models/	3D objects in various formats (NFF, DXF, etc.)
textures/	textures in various formats (SGI, Targa, etc.)

[Note that the NFF format referred to is Sense8's NFF format, not mine. -EAH]

Terry Fong ([terry@ptolemy.arc.nasa.gov](mailto:terry@ptolemy.arc.nasa.gov))

---

Wavefront site:

<http://wavefront.wti.com>

(WWW only?)

Kevin Bjorke ([bjorke@pixar.com](mailto:bjorke@pixar.com))

---

The author of TextureSynth (Joshua Jeffe) has written a new texture generating application called TextureScape, which uses postscript shapes as the basis of really cool, resolution-independent textures. It can also be used to create animated textures for use in multimedia. TextureSynth is published by Specular International (call 1-800-433-SPEC for more info).

Peter E. Lee ([lee@cs.umass.edu](mailto:lee@cs.umass.edu))

---

3D Studio utils uploaded to Avalon

I have uploaded 3D Studio IPAS routines and utilities to [avalon.chinalake.navy.mil](http://avalon.chinalake.navy.mil) (129.131.1.225) /pub/utls/3ds/

Each file is accompanied by a .txt file. I didn't write, haven't used, don't know anything else about these utilities. Just passing them on.

Pat Kane ([prkane@nyx.cs.du.edu](mailto:prkane@nyx.cs.du.edu))

---

Also at [avalon.chinalake.navy.mil](http://avalon.chinalake.navy.mil), in the pub/misc directory, are all sorts of interesting PC software, including the fabled TrueSpace Caligari demo version (doesn't save/restore; ts\_demo.zip), a Real3D demo (which is supposed to be tough to use without the documentation), and other interesting things.

---

Acoustic Simulation

Renkus-Heinz, a speaker manufacturer from California, distributes EASE (Electro Acoustic Simulation for Engineers), and EASE Jr. Both run on MS-Dos systems. Ease also supports aurilization using a companion program called EARS and a DSP board. Ease allows you to calculate reverb time, impulse response, frequency response of systems, ray tracing, and perform sound system design. Ease Jr allows reverb time and sound system design calculations. Both programs allow wireframe room models to be built in the program, or importation of 3-D DXF files.

Jay Paul ([jaypaul1@aol.com](mailto:jaypaul1@aol.com))

[it had the magic words "ray tracing" in there, so I included it. -EAH]

---

Blob Sculptor Update

We have released Blob Sculptor for Windows 1.0. It has the same functionality as the DOS version. Ron Praver ported the DOS version to Windoze. [look around on [ftp.povray.org](http://ftp.povray.org), I think it's in there somewhere. -EAH]

Blob Sculptor, ver 2.0 for \_DOS\_ is almost ready to be released. The new functionality includes:

- \* better user interface

- \* camera can be interactively located
- \* support for multiple blobs, i.e. components within a blob interact with each other but blobs don't interact with each other
- \* sphere, cones and cylinder shaped components
- \* faster preview mode

Version 2.0 has been beta tested for some time and already has produced some awesome images. For example, Edgar L. Ibarra has produced various images including cartoon-like characters: dog, candle holder (a la Beauty and the Beast), and a baby. You can find these at CompuServe's GraphDev forum. He renders the objects with Imagine 3.0.

I've been working on a spline blob component. The component consists of 3 points which are created in 3D space. A spline is used to interpolate them and a radius is defined to give the object a bent cylinder look. The component also permits the modification of a tension parameter which gives the user more versatility without the need to use more points. Uses for this, you ask? well, if you want to create the fingers in a hand you don't need to use spheres or 2 or more cylinders to do the job...just 1 spline blob will do the job. Think of a spline blob as a generalized cylinder or playdough in the shape of a cylinder which can be bent.

Alfonso Hermida ([AFANH@STDVAX.GSFC.NASA.GOV](mailto:AFANH@STDVAX.GSFC.NASA.GOV))

---

Radiance Related  
=====

### SIGGRAPH Proceedings Distribution

Other than a paper about the Radiance system in the SIGGRAPH '94 Proceedings, the entire Radiance system is included on the Proceedings CD ROM. It's in /papers/ward/rad2r4.tz (uncompress and read .tar, I assume).

Eric Haines

---

### Radiance vs. POV

I have found Radiance to be comparable in speed to POV. Once the octree is generated the tracing times are even shorter than POV. It has very efficient pruning and the tracing times go up less than linearly with the increase in the number of polygons in a scene. And besides - the results are MUCH better in terms of realistic-looking lighting and better-looking surfaces. In my opinion Radiance is the best ray-tracing package on any platform - I use it on an Amiga. The main problem with Radiance is lack of a good modeler and the many other support programs that are available for POV.

Alison Colman ([acolman@magnus.acs.ohio-state.edu](mailto:acolman@magnus.acs.ohio-state.edu))

---

POV Related  
=====

Andy Wardley ([abw@dsbc.icl.co.uk](mailto:abw@dsbc.icl.co.uk)) notes:

I've just uploaded geodome1.zip to uniwa which is the utility for creating the geodome data models.

Non-PC users will be glad to hear that it *\*does\** include source.

--

Harry Rowe ([Harry.Rowe@wedowind.meaddata.com](mailto:Harry.Rowe@wedowind.meaddata.com)) comments:

Like others, I would just like to say GREAT scene! The sky is one of the best I've seen. I scarfed the GIF and your utility from uniwa and placed it on my BBS (Rowe's Ray Tracing World BBS (513) 866-8181). It will be on my custom 600 meg Ray Tracing/Graphics CD coming this fall

---

The POV-Utilities 2.0 for the Mac have hit the streets. The package is now available on CompuServe, Go GRAPHDEV, Lib 8, POVUTL.SIT. It is also available via anonymous ftp on the internet, from the Australian secondary official POV-Ray site, <ftp.uwa.edu.au> (and <ftp.povray.org> for N.American users, I assume), under pub/povray/utilities/. As always, feedback and bug reports are encouraged.

Also look for my shareware System 7 "MooVer" utility, that converts a series of (POV-Ray animated?) PICT files into a QuickTime movie with a simple drag-n-drop. It should also be showing up on uniwa, and is on CIS/GRAPHDEV/Lib 6/MOOVER.SIT.

Eduard Schwan ([71513.2161@CompuServe.COM](mailto:71513.2161@CompuServe.COM))

---

### New Povray Clipart Available

I've uploaded Povray V2.0 files to [avalon.chinalake.navy.mil](http://avalon.chinalake.navy.mil). They can be found at pub/objects/pov. They were all converted from \*.obj files found on [avalon.chinalake.navy.mil](http://avalon.chinalake.navy.mil) in pub/objects/obj.

Keith D. Rule ([keithr@tekig7.pen.tek.com](mailto:keithr@tekig7.pen.tek.com))

[many are the free Viewpoint models, plus a few Star Trek thingies. -EAH]

---

### POV Utility Information on the WWW

I have just set up a WWW page for POV utilities, containing information about a few popular ones, and also allowing you to download them directly from the page. For most people that should be faster than the various ftp sites, as well as having the advantage that you actually get some info on what the utility is about.

The URL is <http://www.ifi.uio.no/~mariusw/pov/utilities.html>

Please give me feedback and point me to utilities you think should be on the pages.

Marius Ibenhardt Watz ([mariusw@ifi.uio.no](mailto:mariusw@ifi.uio.no))

---

Some utilities of note: MORAY (1.5 of course !), SUDS, CTDS, POVCAD, TGA2GIF, TTG, DXF2POV and RAYLATHE, just to mention a few.

[from the POV newsletter]

---

### Faces for POV (Frosty):

There was a thread a while back about Dan Farmer's Frosty picture and how we wanted the .POV file. I think in fact most people wanted the face data, and on someone's advice on this group I got hold of the face1.3ds, face2.3ds and face3.3ds files from [avalon.chinalake.navy.mil](http://avalon.chinalake.navy.mil) and converted them into .pov form with a utility also on that ftp server. The utility for conversion is something like 3dspov18.zip [source for conversion from 3DS is included. -EAH]

The results are good, although they come out as 230-320k files that might fill up too much space, and take a while to parse. The faces are more like masks than real faces, but do the job so long as expressions aren't needed.

Mr. Jonathan H. S. Peterson ([zctyjhp@ucl.ac.uk](mailto:zctyjhp@ucl.ac.uk))

---

## Parallel POV-Ray using PVM

I have written such a thing (somehow). It works like this:

A server is started, that takes all parameters and stuff. The server divides the final picture in several stripes. The server spawns some children, that each calculate one stripe (10 pixels high). For this, each child starts povray with +SL and +EL and writes the result to a file. This file is passed back to the server. The server collects all files.

After all stripes have been calculated (a list of calculated stripes is maintained, so you can abort and restart the process), all stripes are collected into a final picture.

The results are impressive. texture1.pov-demo is rendered in 1-2 minutes at 1280x1024 pixels.

Aaron Digulla ([sorry, no address])

---

## Rayshade Related

=====

Craig Kolb has moved to Stanford, following his advisor Pat Hanrahan, and so has pretty much bowed out of maintaining Rayshade. In his place others are attempting to glue together all the additions and bug fixes over the years and get out a Rayshade 4.1. More news as it happens. Other than this effort, there has been an explosion of WWW stuff for Rayshade.

---

Scattered notes from Craig Kolb:

There's an experimental version of a rayshade homepage available on the Web:

<http://www.cs.princeton.edu/grad/cek/rayshade>

Thanks to Jelle van Zeijl and Stuart Warmink for providing documents, images, and encouragement to set this up.

As some of you might have noticed, I've reorganized and expanded the rayshade FTP archive a bit. If you have anything that you'd like to add (new primitives, textures, whatever), drop me a line.

Hundreds of surface definitions for Rayshade are available in:

<ftp://ftp.princeton.edu/pub/Graphics/rayshade/Contrib/Libraries/surfdefs.rh.Z>

Stephen Peter's excellent notes on using rayshade are now available via the rayshade homepage (<http://www.cs.princeton.edu/grad/cek/rayshade/>), and in source form on the rayshade FTP archive as Contrib/Docs/raynotes.tar.gz.

Craig Kolb ([cek@Princeton.EDU](mailto:cek@Princeton.EDU))

---

## Z Buffer (Range Map) Extension



To get this extension, ftp [vacation.venari.cs.cmu.edu](ftp://vacation.venari.cs.cmu.edu) (128.2.209.207), don't chdir, and use binary mode and download zbuf.tar. You can get documentation updates (a patch file and a new file) in the same place: file zbuf-doc.tar.

Mark Maimone ([Mark.Maimone@A.GP.CS.CMU.EDU](mailto:Mark.Maimone@A.GP.CS.CMU.EDU))

---

The RayShade docs done by Jelle are up as URL:

<http://www.msi.umn.edu/miscdocs/Rayshade/index.html>

-- Matt Hughes ([hughes@msi.umn.edu](mailto:hughes@msi.umn.edu))

The Rayshade users' manual can now be reach at MIT via two URLs:

<http://web.mit.edu/afs/athena/activity/c/cgs/lib/html/rayshade/index.html> and

<http://www.mit.edu:8001/activities/cgs/rayshade/index.html>

-Fred, Director, MIT Computer Graphics Society

(<http://www.mit.edu:8001/activities/cgs/mitcgs.html>)

---

For what it's worth, I've included a pointer to Rayshade in the Computer Vision Home Page:

<http://www.cs.cmu.edu:8001/afs/cs/project/cil/ftp/html/vision.html>

Mark Maimone ([Mark.Maimone@A.GP.CS.CMU.EDU](mailto:Mark.Maimone@A.GP.CS.CMU.EDU))

---

I've made a sample page of the standard Rayshade images (and just a few more :-) available to the Net. There are several ways to get at them:

AFS filesystem: cd /afs/cs.cmu.edu/misc/rayshade/all\_mach/omega/doc/Examples/

WWW with icons: [http://www.cs.cmu.edu:8001/afs/cs/misc/rayshade/all\\_mach/omega/doc/Examples/rayimages.html](http://www.cs.cmu.edu:8001/afs/cs/misc/rayshade/all_mach/omega/doc/Examples/rayimages.html)

WWW without icons: [http://www.cs.cmu.edu:8001/afs/cs/misc/rayshade/all\\_mach/omega/doc/Examples/rayimgtxt.html](http://www.cs.cmu.edu:8001/afs/cs/misc/rayshade/all_mach/omega/doc/Examples/rayimgtxt.html)

anonymous FTP: ftp [ftp.cs.cmu.edu](ftp://ftp.cs.cmu.edu) login: anonymous passwd: email cd

/afs/cs/misc/rayshade/all\_mach/omega/doc/Examples/



back to [contents](#)

---

## Recent Ray Tracing Papers

[This is a list of some recent research papers related to ray tracing from some of the lesser known conferences and journals. -EAH]

---

[This paper is interesting in that it is the first independent testing of Arvo and Kirk's 5D space subdivision ray tracing scheme. -EAH]



G. Simiakakis and A. Day  
Five-dimensional Adaptive Subdivision for Ray Tracing  
Computer Graphics Forum, volume 13, number 2  
(Special Issue: Rendering), pp. 133-140, June 1994.

[George Simiakakis can be reached at [gns@sys.uea.ac.uk](mailto:gns@sys.uea.ac.uk)]

---

Fifth Eurographics Workshop on Rendering  
13-15 June 1994  
Darmstadt, Germany

RAY TRACING (Chair: Erik Jansen ([fwj@uticg.twi.tudelft.nl](mailto:fwj@uticg.twi.tudelft.nl))) [Summaries courtesy of Erik Jansen] [Really this session is more for global illumination via rays. There were many other global illumination papers here. -EAH]

Adaptive Splatting for Specular to Diffuse Light Transport  
Steve Collins

For a two-pass algorithm an adaptive light pass is used that deposits the power carried by rays as 'splats' of energy flux on the surfaces using a Gaussian distribution kernel. The kernel of the splat is adaptively scaled according to the 'ray wavefront' convergence or divergence, thus resolving sharp intensity gradients in regions of high wavefront convergence and smooth gradients in areas of divergence.

Rayvolution, An Evolutionary Ray Tracing Algorithm  
Markus Beyer, N. Sander, Brigitta Lange

In order to increase the statistical efficiency of Monte Carlo integration of the rendering equation, evolutionary algorithms are applied to optimize the sample distribution. An initial population of rays evolves towards an optimal sample ray distribution by application of generic operators and selection mechanisms. As a result we achieve an implicit stratification and a better convergence towards the actual value.

Bidirectional Estimators for Light Transport  
Eric Veach, Leonidas Guibas

Monte Carlo methods do not suffer from the artifacts and limitations that standard radiosity methods have to address, but have another well-known artifact: noise. A major source of noise is bright indirect light. We can construct different light paths between the light sources and the eye either starting from the eye or from the light sources, or starting from both meeting halfway. The different paths leads to a partitioning of the rays. Each partition gives us a different unbiased estimator. Combining these estimators may lead to near-optimal variance reduction.

A New Raytracing Architecture Robert E. Bacon, John A. Gerth, V. Alan Norton, James Kajiya

[withdrawn, unfortunately; mentioned because it sounded interesting. -EAH]

---

Computer Graphics International '94  
Royal Melbourne Institute of Technology  
Melbourne, Australia, June 27 - July 1, 1994

Rendering and Display  
Kenjiro Takai Miura

Ray Tracing Gregory-type Patches  
W. Lamotte, K. Elens, F. van Reeth and E. Flerackers  
A Parallel Ray Tracing Algorithm Using Hierarchical Bounding Volumes

Kerry Gigante ([kerry@cgl.citri.edu.au](mailto:kerry@cgl.citri.edu.au))

---

The Fourth Eurographics Workshop on Animation and Simulation was held in Barcelona, Spain September 4-5, 1993.

[This workshop had many papers with interesting titles, but here's the one with those magic words "raytracing":]

Combining computer animation and video using bluebox raytracing M. Zeiller Technical University of Vienna - Austria.

Sabine Coquillart ([Sabine.Coquillart@inria.fr](mailto:Sabine.Coquillart@inria.fr))

---

Abstract from the Sharp Technical Journal, #57 Nov 1993.

The Sharp Technical Journal is a Japanese language publication which contains English abstracts for most of the articles.

Optimization of Illumination System for LC Projector Using Newly Developed "Reverse Ray Tracing Method"

Takashi Shibatani, Hiroshi Nakanishi, Hiroshi Hamada

A new illumination simulation technique using "Reverse ray tracing method" has been developed as a design tool for an illumination system of an LC projector with a microlens array. In this method, rays are traced from an illuminated plane to a light source in opposite direction compared to the conventional method. This method have an advantage that the numbers of rays to be traced is much smaller than in the case of conventional method for required accuracy, and it presents 2-dimensional angular distribution of the incident rays, which affect the efficiency of the microlens array.

An illumination system for a single-panel LC projector with microlens array with a newly developed correction lens and an aspherical mirror is optimized to improve brightness of a screen employing this simulation method.

Dr. David K. Kahaner ([kahaner@cs.titech.ac.jp](mailto:kahaner@cs.titech.ac.jp))

[Copies of previous reports written by Kahaner can be obtained using anonymous FTP from host [cs.arizona.edu](http://cs.arizona.edu), directory japan/kahaner.reports.]



back to [contents](#)

---

## Rumor Mill

[A new feature - this is simply snippets of tantalizing information from the net that I saw no follow-up about. If anyone has any details about these items, let us all know. -EAH]

John T. Chapman ([jtc1@cornell.edu](mailto:jtc1@cornell.edu)) notes:

By the way - there's a CD-ROM rendering/ray-tracing tutorial offered through Tiger Software and others. It supposedly has stuff on Ray Dream Designer, along with other programs. Has anyone seen this? Is it for RDD 3.0 or earlier? Is it worth the money? (There is apparently two versions - a 'sampler' at about \$19 or so and the full version at about \$90 or so.)

Vareck Bostrom ([bostrov@CSOS.ORST.EDU](mailto:bostrov@CSOS.ORST.EDU)) says:

Didn't they have a Intel Paragon doing 1 frame/sec 1024x768 or so ray tracing at supercomputing '93? I didn't see it myself, but my friends that did said it was impressive.

---

NURBS Rendering, by Henrik Wann Jensen

Jouko Vuoskoski ([jvuoskos@suca01.cern.ch](mailto:jvuoskos@suca01.cern.ch)) wrote:

```
: 1) What is the "normal" method to do ray-tracing with  
: B-splines (or NURBS)?
```

```
: 2) How can I do it fast (with less accuracy)?
```

```
: I was thinking to ray-trace with facets from the control  
: polygon of the B-spline. Halving the knot intervals I could  
: get more accuracy. Is this right approach?
```

You are right. The easiest way and also fastest (often stated) is to create a facet representation of the NURBS-surface and halving the knot intervals is the way to do it. I know there is some source-code floating around - try archie and search for nurbs.

[I vaguely recall there being some spline surface intersection code around and about, but does anyone have any hard info? -EAH]



back to [contents](#)

---

## **AERO Animation/Simulation System version 1.5.1, by Thomas Braeunl** **([braunl@hermes.informatik.uni-stuttgart.de](mailto:braunl@hermes.informatik.uni-stuttgart.de))**

AERO is an X-window based tool for simulation and visualization of rigid-body systems. AERO contains a 3D scene editor for designing simple blocks world scenes. Objects may be placed in space, linked to each other, and forces may be exerted onto them.

In animation mode, the simulation of the scene entered is carried out in real time (depending on scene complexity) displaying 3D wire frames. Also, a flag can be set to generate scene description files for each time point as input files for a ray tracing program, producing photorealistic output.

In both modes, wire frame and ray tracing scenes, the generation of stereo images is possible. In the case of real time wire frames a red-green representation of the scene is rendered, which can be viewed with red-green glasses. For the ray tracing output, a binocular pair of scene descriptions is generated for each time step.

AERO can be used for exploring the physical laws of mechanics and also for generating realistic computer animations. AERO is available free of charge as public domain software. Software and documentation can be copied via "anonymous ftp" over the Internet.

The address of our server is:

<ftp.informatik.uni-stuttgart.de> (currently 129.69.211.2)

The directory is:

pub/AERO

[Note that AERO 1.5.1 now outputs POV-Ray 2.2 format. I believe AERO is mirrored on wuarchive, though check version numbers with archie. This system looked to be pretty cool - you can simulate physically based phenomena. There's also a paper about the system and the algorithms used. Admittedly the primitive set is limited (spheres and boxes), but Olli Vinberg ([vinberg@cc.helsinki.fi](mailto:vinberg@cc.helsinki.fi)) pointed out that you can run the simulation, output the results, and substitute the primitives with your own complex objects. -EAH]



back to [contents](#)

---

## A Brief Review of [an old] AERO, by Dave Negro ([dln2@cornell.edu](mailto:dln2@cornell.edu))

My room-mate is running Linux and I learned about AERO last [school] year and made him install it. Haven't had any time to play with it this year, though. I can tell you some of the things that I remeber from last year.

- 1) The simulations were great! I found the emulation of real life was great and could be used to make some great animations.
- 2) Editing was a little awkward but then again I wasn't able to fully get accustomed to it myself. There was of course the standard primitives (Sphere, Cone, box etc)
- 3) Textures were lacking. The selection and editing of textures left a lot to be desired.
- 4) A separate .pov file is created for each frame! There is no way to make an include or anything for static objects. Thus editing the files would be nearly impossible to do with each frame!
- 5) Adding things to the scene but not with the tools in the AERO editor would again be extremely difficult.
- 6) There were still some things that needed to be worked on like the open dialog and save. Biggies in my opinion.

Well, that is all I can remember for the moment. But let me remind you that I have not touched the program in a year, so I don't know if has changed at all, and I wouldn't totally trust my memory either.



back to [contents](#)

---

## Photon Tracing, by Chris Thornborrow ([ct@opal.epcc.ed.ac.uk](mailto:ct@opal.epcc.ed.ac.uk)) and Greg Ward ([greg@pink.lbl.gov](mailto:greg@pink.lbl.gov))

Chris Thornborrow writes: OK onto the questions. I am writing a raytracer which does the following:

- a) Ordinary raytracing
- b) Distributed raytracing for lighting
- c) Path tracing (Kajiya)
- d) Photon tracing

Now I need to be able to bias random vectors to do these \*efficiently\* but I am unsure how to do this. First off then, to generate a random direction vector from a point, I generate a random vector in the unit cube and reject those that lie outside the unit sphere. This has a 50rejection rate. Can I do better without approximations (I know about generating a 0..1 element and then a random angle).

The problem above appears compounded for a random vector within some solid angle of another vector. Currently I generate a random vector (as above) and then reject ones out with the solid angle (using a dot product). This is awful. Very small angles can be approximated by generating within a cone but larger (say 20 solid degrees) give obviously

biased distributions this way.

Now my real problem. Given an incident angle of a photon (ray) and the surface normal and the BRDF, how can I generate a random reflection/refraction angle that done many times would give the proper distribution defined by the BRDF. In other words, how can I bias the angle of perfect reflection in some pseudo-random manner, so that a great number of samples of those directions have the same distribution as the BRDF ?

Obviously this is trivial for perfectly diffuse surfaces, its the others I worry about :-). I'd like to do this without rejection testing - is this even possible without rejection testing ?

---

Greg Ward answers:

Let me start by saying that I'm not the world's foremost expert on Monte Carlo sampling, but I have written a ray-tracer that does some of what you're asking. I assume you have read some books on Monte Carlo sampling already, such as Rubenstein's 1981 treatise, "Simulation and the Monte Carlo Method" (Wiley, NY). Professor Pete Shirley of Indiana University has also written a fair amount on Monte Carlo methods in ray tracing, though all I have in front of me is something from his 1992 course notes on Global Illumination (course 18 that year), which I'm not sure you can find easily. (I seem to remember that someone put a bunch of Monte Carlo examples in a Graphics Gems book. It would have to be GG III, since that's the only one I don't have.)

The trick is to compute a cumulative probability function and invert it. In many cases, this can be done analytically. In the case of the random direction vector, it can even be solved by inspection. You need only generate two random angles, altitude and azimuth, and convert it back to a Cartesian vector. Rejection sampling is unnecessary. Likewise, for the vector in the solid angle of another vector, you can limit your polar angle to the cone you have selected, 20 degrees in your example.

Arbitrary BRDF's are another matter. Rejection sampling is the most general method, but there is a more efficient way to go. As you probably know, many BRDF's are highly peaked, and using rejection sampling means you may have to test hundreds of ray directions before you get one that isn't rejected. The more efficient approach is to generate a cumulative distribution table and invert it. I haven't done this myself, so excuse me if I'm a little foggy on the details:

1. Select a resolution limit for the polar and azimuthal angles in your ray direction calculation, somewhere around 5 degrees should be good. (This is the only real limitation to this technique.)
2. Create a 3-dimensional table of real numbers. The first dimension is the number of polar angles, the second dimension is the number of azimuthal angles, and the third dimension is the product of the two, i.e. `float cuml_prob[N][M][N*M];`
3. For each reflected polar and azimuthal angle, do the following:
  - 3a. For each incident polar and azimuthal angle, compute the BRDF times the cosine of the polar angle times the sine of the polar angle and add it to a running total. Store this sum at the appropriate point in the table created in step 2.
  - 3b. Once done, your sum should equal the total reflectance, which should be less than one (but greater than 0!) if you have a valid BRDF. If it doesn't, you might want to report an error, but you can proceed with the calculation regardless.

4. You have now filled a 6.5 Mbyte table, and this completes the initialization phase. (Storing this sucker to disk wouldn't be a bad idea.)

Now, when you have to compute a ray direction, you look up the appropriate table for this reflection angle, and:

5. Compute a uniformly-distributed random number between 0 and 1 or 0 and the maximum value in this cumulative table, depending on whether the distribution is properly normalized and if you are accounting for reflectance with a multiplier or with pure Monte Carlo.
6. Perform a binary search to find the value in the table that is closest to your random number, and determine the corresponding polar and azimuthal incident angles. (This is the inversion step in this algorithm.)
7. Jitter your sample uniformly within your 5 degree tolerance (or whatever you picked in step 1) to get the final ray direction.
8. Trace that ray!

Obviously, the above algorithm could benefit from some refinement. Firstly, the precomputed table ends up being quite large. The bigger your angle tolerance, the smaller it will be, but I can never see it being very compact. You can store the whole thing to disk, and just read in the section appropriate to the reflection angle in hand when the time comes, and this will save on your memory costs at the expense of one or two disk accesses. Also, the binary search (step 6) is fast but not free, and it may pay to invert your table directly by computing azimuth and altitude angles corresponding to two random variables. This would require a little more thought, but I think it could solve both of these problems. The only catch is that you still want some way of jittering the final result so as to avoid sampling a discrete set of directions, and the distance between adjacent points in the table will vary with such a scheme. (Shouldn't be a problem, though.)

Has anyone implemented something along these lines who could help us out? Pete?



back to [contents](#)

---

## Faster Than POV-RAY 2.1, by Dieter Bayer ([dieter@cip.e-technik.uni-erlangen.de](mailto:dieter@cip.e-technik.uni-erlangen.de))

[This bent my mind: someone actually implemented my light buffer algorithm? Astounding! -EAH]

A modified, sped-up and unofficial version of the Persistence of Vision Ray-Tracer Version 2.2.

Faster than POV-Ray (FTPOV-Ray) speeds up calculation of images by using some kind of direction cubes for primary rays (the vista buffer) and shadow rays (the light buffer) at the cost of additional preprocessing time and greater memory usage. The bounding slab hierarchy used by POV-Ray is projected onto the viewing plane and each point light source a priori. Thus the number of ray/slab-tests is reduced. Furthermore some modules have been modified to eliminate unnecessary calculations and automatic bounding has been improved.

The modified source code in the archive FTPV21S.ZIP may only be distributed together with this text and the file POVLEGAL.DOC that is part of the official POV-Ray package. To use the source code you'll need the original POV-Ray 2.2 distribution.

**WARNING!!!** If you use FTPOV-Ray you'll do it at your own risk! And don't forget that the POV-Ray team isn't

responsible for this version.

I have uploaded the source code and executables for MS-DOS and OS/2 2.x to <ftp.informatik.uni-oldenburg.de> and [wuarchive.wustl.edu](http://wuarchive.wustl.edu) (/pub/graphics/graphics/ray/pov/pov-dkb-archive/incoming).



back to [contents](#)

---

## Z Buffer Based Rendering Program, by Raghu Karinithi ([raghu@cerc.wvu.edu](mailto:raghu@cerc.wvu.edu))

[I tried this out and got some funky lighting problems (perhaps fixed by now), but it seemed essentially sound. It's nice to see a z-buffer for a switch (lots harder to write than a basic ray tracer). Hey, it reads NFF files, so I like it ;-> -EAH]

We have developed a Z bufffer based rendering program in the West Virginia University. The key features of this program are:

(a) Reads a variant of the standard Neutral File Format (NFF) as input (b) Outputs a 24 bit color TARGA file (c) Uses accurate fixpoint arithmetic in all its calculations (d) Works in X windows environment. We have tested it on Sun Sparc machines.

This software is available via anonymous ftp. The software includes two sample files, including the teapot. We would appreciate any feedback you can give us on this software. We are quite impressed by the quality of the images produced. Enjoy the use of this utility!

FTP from 157.182.44.36: pub/sources/ZRendv1.tar.Z

Below is a short description of the software.

Raghu Karinithi  
Department of Statistics and Computer Science  
Concurrent Engineering Research Center  
West Virginia University  
Morgantown, WV 26506-6506  
Voice: (304) 293-7226  
Fax: (304) 293-7541  
Email: [raghu@cs.wvu.edu](mailto:raghu@cs.wvu.edu) OR [raghu@cerc.wvu.edu](mailto:raghu@cerc.wvu.edu)

### Description:

Often times, I have seen a question on the net for an accurate Z buffer rendering implementation producing 24 bit color output along with facilities for viewing the same. We needed one ourselves as a benchmark for our work on parallel rendering. This document describes a program that we developed for this purpose. We have leveraged a number of public domain utilities and defacto standards in our development, and hence this program does not have any fundamental algorithms, it integrates existing ones effectively.

This describes briefly our implementation of the Z buffer rendering algorithm. The rendering pipeline has the following steps:

1. Reading the input file.
2. Computing the light intensity at the vertices. Later, Gouraud shading is used for interpolating the light intensity.
3. Computing the view transformation matrix.
4. Applying the view transformation to all the vertices
5. Rasterization, and writing to framebuffer.



## 6. Writing framebuffer to a TARGA file.

Subsequently, one can display the TARGA file, by invoking a separate utility (included with this package).

This rendering program leverages of several public domain utilities. The matrix library is from the SPHIGS package from Brown University. The view transformation matrix also is computed using the code from the SPHIGS package. Rasterization is based on fixed point arithmetic. Fixed point arithmetic library is from the directory "accurate\_scan" from Graphics Gems III contributed by Kurt Fleischer. The rasterization code used in this program is a modification of some code from the same source. We compute 24 bit color (R,G,B) at each pixel and write it to a TARGA file. The functions to do write in TARGA file format are from the "LUG" library due to Raul Rivero. The program "sx11" to display a TARGA file is also from the same library. We have extracted these two utilities from the LUG library and provided them in the subdirectories "tga" and "sx11".

Input File Format: The file format we use is a modification of the Neutral File Format (NFF) described in the Standard Procedural Database (SPD) of Eric Haines. The modified NFF is described in the file called NFF in this directory. One can also look at the sample files. In the NFF file format, perspective projection is assumed.



back to [contents](#)

---

## Gossamer, a Free Macintosh VR/3D Renderer, by Jon Blossom ([jonbl@microsoft.com](mailto:jonbl@microsoft.com))

Gossamer 2.0, a real-time 3D walkthrough engine for the Macintosh, is now available - for FREE!

Gossamer is a very fast, general 3D polygon rendering engine. This version radically improves speed over version 1.1, fixes a number of bugs, and focuses on a more coherent user interface for "experimenters."

Version 2.0 also supports object and world files based on the Rend386 PLG and WLD formats. The Gossamer 2.0 package includes a number of sample worlds and objects ported directly from the PC to the Macintosh, and there are others available on bulletin boards, electronic services, and ftp sites all over.

The package is available for anonymous ftp from <ftp.apple.com> in the pub/VR/graphics.systems directory.

If you have a Macintosh and are interested in 3D graphics, this will be well worth your time.

The system requires a 68020 or better processor Mac running system 7 or system 6.x with 32-bit color QuickDraw. A color display is nice but not necessary.

If you like Gossamer 2.0, please let me know!



back to [contents](#)

---

## Antialiasing Issues, by Arijan Siska ([arijan@kette.fer.uni-lj.si](mailto:arijan@kette.fer.uni-lj.si))

I want to get some attention to the (in my opinion) a very serious issue: antialiasing.

When a raytracer with a finite resolution (obviously a resolution cannot be infinite) renders an image jagged edges appear in picture (or in case of an animation, when difference between two consecutive frames is large (when objects move fast), animation tends to flicker).

These type of problems can be solved by two approaches:

- supersampling: is a very very expensive method, since it involves much extra (often unneeded) calculation, but generally it gives very good results (image quality).



- adaptive sampling: implies that there exists some criterion function, upon which you can decide whether you should subdivide further to gain a more precise answer or you should be satisfied with the approximation you already have. Note that in general there is no way to make this decision function absolutely correct. The best counterexample that I can think of is a thin wire fence in front of uniform background. If you use textures with rapid changes it can happen that you subdivide (to the maximum level) 800f the pixels, which is effectively supersampling, but slower (you need to evaluate criterion function all the time). I don't see how some kind of preprocessing could help you with criterion function, since all those reflections refractions .... are there.

---

Steven C. Demlow ([demlow@cis.ohio-state.edu](mailto:demlow@cis.ohio-state.edu)) replies:

I played around with a variety of AA approaches in both my own ray tracer and the public domain POV ray tracer. I tried various derivations and combinations of supersampling, stochastic jittering, adaptive methods, and filtering. For the animation project I was working on I ended up using 4x4 stochastically jittered grids on each pixel and a tent filter that extended into the (cached, of course) grids of neighboring pixels. I've been working on using a hexagonal grid instead of a rectangular one (this gets tricky when you try to overlap the arbitrarily-sized filtering grids of adjacent pixels). The hex grid supposedly provides close to an optimal ray distribution for AA purposes.

It sounds like you would do well to add filtering to your ray tracer - it can help a lot.

The conclusion I came to, after eight months of rendering on a bunch of HP7?0s, was that ray tracing is not the way to go to generate a lot of high- quality images. :) I love ray tracing but there are reasons that it sees little use in production environments.

---

Andy Key ([ak@hursley.ibm.com](mailto:ak@hursley.ibm.com)) replies:

I implemented Whitted Adaptive Supersampling as my anti-aliasing mechanism. Really, the test of seeing whether to subdivide is tiny compared to the cost of spawning a ray. I support un-anti-aliased, and whitted. I was going to add unconditional-supersampling to improve the case when I have fine detail textures, but I am not sure it will buy me a noticeable difference if I do.



back to [contents](#)

---

## Microcosm, by Abe Megahed of Cosmic Software ([cosmic@world.std.com](mailto:cosmic@world.std.com))

[I include this here as there is a free demo and images available. It's a nice system, allowing a wide range of interactive techniques with ray tracing, and last I heard they were adding support for native graphics packages (e.g. Starbase on HP's), multiple windows, and picking - they're up to version 1.4 or so by now.

This package was also the easiest system I ever dealt with for rendering on a network (not that I've dealt with a lot, but...): I simply started up the daemons, let the software know which computers to use (HP workstations, in my case), and let the software suck up resources and render at great speeds. Impressively painless. -EAH]

Microcosm, Version 1.2, New Product Announcement / Free Demonstration Version

A new version of Microcosm is being released for many popular platforms including:

- \* IBM PC / MS-DOS
- \* DEC / Alpha
- \* DEC / MIPS

- \* HP 9000 Series 700 (Snake)
- \* Sun Sparkstation
- \* IBM RS 6000

Microcosm offers the highest quality rendering features such as ray tracing, texture mapping, bump mapping, and Phong shading as well as interactive animation capabilities. A free demo version and over 100 example description files of animations, simulations, interactive environments, objects, and pictures is available via anonymous FTP.

What is Microcosm?

The Microcosm system is composed of a set of advanced 3D graphics routines which is controlled by a simple, high level interpreted programming language called SMPL for 'Simulation / Modeling Programming Language'.

This gives even the novice programmer the power and flexibility to easily create graphics applications which could only previously be done with great difficulty and expense in low level languages such as C. Since the language is interpreted, you may change things and immediately re-run your graphics program without waiting for the computer to re-compile it. It's fun and easy - like programming an Apple II on steroids!

Most computer graphics intensive programs spend at least 95% of their time in the rendering, so using the interpreter to control the renderer doesn't slow the graphics down by more than 50% over a fully compiled program.

Features of Microcosm

Microcosm incorporates a number of totally new and innovative rendering algorithms in a well integrated system. Some interesting features are as follows:

Rendering Features:

- \* Shading Language
- \* Surface Mapping
- \* Procedural Textures
- \* Real-Time Soft Shadows, Reflections, and Transparency / Refraction
- \* Interactive Coarse Ray Tracing
- \* Clean Line (Silhouette) Rendering Modes
- \* Shaded Rendering Modes
- \* Ray Tracing
- \* A Choice of Projections
- \* Hierarchy
- \* Stereo Rendering

Modeling Features:

- \* Procedural Modeling
- \* Fractals
- \* Variety of Primitives
- \* Extensibility

Programming Features:

- \* Easy-to-Read Syntax
- \* Procedural, Block Structure
- \* Flexible Parameter Passing
- \* Built-in Data Types
- \* Smart Arrays
- \* Nested Comments

The demonstration version of Microcosm can be obtained by anonymous ftp at: [dpls.dacc.wisc.edu](http://dpls.dacc.wisc.edu):

pub/graphics/mcm/mcm\_demo.zip. The demo version runs for 1 minute before quitting and also is unable to save files, so it can be used only to preview simple renderings or animations. A number of finished renderings are available in mcm/images to illustrate the advanced rendering capabilities of Microcosm. Also provided free are about 100 example description files to demonstrate the capabilities of Microcosm.

Please contact us for specific details about these or other platforms. Any other information, including price and/or ordering information can be obtained as follows:

Cosmic Software Corp.  
1413 Mound St.  
Madison, Wisconsin  
53711  
[cosmic@world.std.com](mailto:cosmic@world.std.com)  
Tel: (608) 259-1776  
FAX: (608) 233-4995



back to [contents](#)

---

## Fisheye Lens Distortion, by Greg Ward ([greg@pink.lbl.gov](mailto:greg@pink.lbl.gov))

Almost all wide angle photographic lenses have a bit of this, but in a rendering program it is easier to avoid.

In a perfect perspective image, straight lines will always be straight, no matter how wide the viewing angle becomes. Viewing angles can never be equal or greater than 180 degrees, however, because the arc tangent blows up. A full view angle close to the maximum will cause the center of the image to all but disappear, and the view will be dominated by what surrounds the viewpoint. (Try it, I guarantee you won't like it.)

In order to get a view of 180 degrees or greater, it is necessary to adopt some sort of perspective distortion. Most commercial fisheye lenses use a distortion that causes distances from the center of the image to be proportional to the geometric angle from the central line of sight. Another type of lens creates a view equivalent to taking a hemispherical image and projecting it onto a plane, those such lenses are much harder to come by.

Another poster (Benjohn Barnes) mentioned that these effects are easier to achieve in a ray tracing program, and he's probably right. I didn't have too much trouble implementing these view types in Radiance, and they come in quite handy. I can even render a 360 degree fisheye view, where the surround of the circular image is actually a single point. Neat trick.

P.S. When it comes right down to it, we all have eyes evolved from fishes.



back to [contents](#)

---

## Optical Ray Tracers

[A common question on sci.optics is what ray tracers there are for lens design. Here are some cullings. -EAH]

Mark Butterworth ([markb@hpcss01.cup.hp.com](mailto:markb@hpcss01.cup.hp.com) - address now invalid)

Instead of driving this subject in to the ground, try the library at your university and see if they have:

Modern Optical Engineering by Warren J Smith (he taught me) Applied Optics and Optical Engineering by Kingslake volume 3 and volume 8 are good for this subject

Warren Smith just published a new one (1992) that is a collection of lens designs. I forget the title, something like Optical Engineering: a resource manual. Published by McGraw Hill.

The lenses in this book are also available on a disk library for the Genesee ray tracing program. Warren designed alot of the lenses used by Panavision, just to mention one.

---

On the commercial side, an excellent package is ORA's CODE V (that's a Roman numeral 5). It's highly regarded and we've been happy with it. They'll let you have a free trial account on their dial-up VAX.

Optical Research Associates is in Pasadena,CA at (818) 795-9101

J. Sallay ([sallay@scubed.com](mailto:sallay@scubed.com))

---

In article ([ps7Mmch.argil@delphi.com](mailto:ps7Mmch.argil@delphi.com)), argil@delphi.com says:

>Can anyone suggest a good commercial ray tracing program for general  
>optical design. We are planning on spending a reasonable amount, like  
>up to \$1500. We have only experience with much more expensive packages.

I don't think you can beat Zemax in this price range. See their ad in Photonics Spectra (p. 19 of the May '94 issue) or Laser Focus World (p. 7 of May). You can also send email to [focussoft@crl.com](mailto:focussoft@crl.com) They will send a free demo disk which is fully functional except that it won't save lenses. They plan to post it to ftp servers as soon as they get a readme done. Caution: Focusoft is moving to Arizona and turning into Focus Software. They hope to be operational by 1 June. The new phone numbers are: voice: 602/749-5646 fax: 602/749-0987. (Ken got tired of the California government.)

Steve Eckhardt ([skeckhardt@mmm.com](mailto:skeckhardt@mmm.com))

---

If you have access to Mathematica (Wolfram Research Inc), there is a Notebook called LensLab, available at [MathSource.wri.com](http://MathSource.wri.com) in directory pub/Applications/Engineering/Other, with the item # 0204-343. It is a decent ray tracing program with an extensive set of functions for lenses, mirrors, prisms etc.. It is relatively slow, unless you run Mathematica on a fast Workstation, Quadra, Pentium,... machine.

Rudolf Oldenbourg ([rudolfo@mbl.edu](mailto:rudolfo@mbl.edu))

---

A good optical tracer is the new Mac version of Beam, from Stellar Software, PO Box 10183, Berkeley, CA 94709, Fax 510-845-2139. I helped with the beta testing, so I know it's good.

A. David Beach ([d.beach@irl.cri.nz](mailto:d.beach@irl.cri.nz))

---

For a freeware optics ray tracer, try irt52.zip in /pub/irt at herx1.colorado.edu. Use anonymous ftp to get this software. It is written for MS-DOS window and you need an unzip program to unzip it.

Djamshid Navabi ([djamshid@lasa.com](mailto:djamshid@lasa.com))

I've tried it: can't get past the demo example without a GPF that causes Windows to quit to DOS the next time I try to run irt. If I then try to run Windows it tells me something gruesome like "invalid command processor" or some such. I'm trying to compile a more responsible set of actions to send to the author but I wanted to mention it here to see if any one else has had the experience or if it may be my system.

Neal E. Tornberg ([tornberg@netcom.com](mailto:tornberg@netcom.com))[back to contents](#)**Correcting Normal Direction, by Gavin Bell ([gavin@krypton.engr.sgi.com](mailto:gavin@krypton.engr.sgi.com))**

[There are a lot of polygonal models out there with inconsistent surfaces. The problem is this: you're given some random set of polygons and you want to consistently orient them outwards so that your ray tracer or z-buffer can use backface culling (i.e. ignore visible backfaces) and so speed performance. Consistent orientation, that is, getting all the faces to be in clockwise orientation, is fairly simple: once you have the mesh of the object, each shared edge must be traversed once in both directions. So if two polygons share an edge and traverse it in the same direction, then the order of vertices of one of the polygons is reversed. Continue the process until all shared edges are traversed in both directions or until you get stuck (e.g. a moebius strip doesn't work). The next part is to know which side of the polygon set is the outside. This can certainly be done manually, but here's a method of doing it automatically. -EAH]

[someone (sorry, the name was deleted in the posting) writes:]

```
>> The user picks
>> a "Seed polygon" and I recursively "grow" a surface starting at
>> that polygon, a neighboring (child) polygon is considered to be flipped
>> if the child traverses the two shared vertices in the same
>> direction as the parent.
```

[someone else (sorry, name deleted) writes:]

```
>If you are not able to interactively pick such a seed polygon, there is an
>automatic way, but it takes time to write. The idea is an extension of a
>point in polygon algorithm. Basically, to find if a point is within a
>polygon, you can extend a ray from that point to infinity, if the ray crosses
>an even number of edges of the polygon, the point is outside (where 0 is
>defined to be even). If it crosses an odd number, it is inside.
```

[I think the concept here was to take a vertex and see which side was outside by ray tracing. -EAH]

A better method:

Consistently orient all of the polygons (you need to know which edges are shared to do this, and this isn't possible for some objects-- e.g. mobius strips, Klein bottles, etc).

Now, choose the point 'P' in the middle of the bounding box of the object. For each triangle in the object, compute a signed volume for the tetrahedron formed by the triangle and P. Arrange your calculation so that the area will be positive if P is left-hand-side of the triangle and negative if P is on the right-hand-side of the triangle. (if your triangle is ABC, then doing  $(AB \times BC) \cdot P$  will have this property).

Add up all of the volumes. If the result is positive, then the normals are oriented 'outside'. If the result is negative, then the normals are oriented 'inside'. If the result is zero or very close to zero, then the object is flat or has just as many concave parts as convex parts.

This will always work for completely enclosed objects, and does the right thing for surfaces-- it chooses the orientation that marks the surface 'most convex'. It works for self-intersecting objects.

Here's the code I use:

(If you have Inventor 1, this is in the 'ivnorm' code:)

```

int i, j;

int total_v = 0;
SbVec3f average(0.0, 0.0, 0.0);

for (j = 0; j < length(); j++)
{
    Face *f = (*this)[j];
    if (f->degenerate) continue;

    for (i = 0; i < f->nv; i++)
    {
        average += verts[f->v[i]];
        ++total_v;
    }
}
average /= (float) total_v;

float result = 0.0;

for (j = 0; j < length(); j++)
{
    Face *f = (*this)[j];
    if (f->degenerate) continue;

    for (i = 1; i < f->nv-1; i++)
    {
        SbVec3f v1 = verts[f->v[0]] - average;
        SbVec3f v2 = verts[f->v[i]] - average;
        SbVec3f v3 = verts[f->v[i+1]] - average;

        float t = (v1.cross(v2)).dot(v3);
        if (f->orientation == Face::CCW)
        {
            result += t;
        }
        else if (f->orientation == Face::CW)
        {
            result -= t;
        }
        else
        {
            assert(0);
        }
    }
}
return result > 0;

```



back to [contents](#)

# Graphics Gems IV Table of Contents, by Paul Heckbert ([ph@cs.cmu.edu](mailto:ph@cs.cmu.edu))

[I trimmed out the author names for the sake of space. Incidentally, I've found Gems IV to be subtly useful - don't judge the articles just by their titles! For example, "Detecting Intersection of a Rectangular Solid and a Convex Polyhedron" by Ned Greene is actually about a method for determining if a bounding box is visible within a viewing frustum, a very handy tool. -EAH]

Below is the table of contents for "Graphics Gems IV". This table also serves as an index to the code in the FTP collection. Note that every article has text that appears in the book but not in the FTP archive, and some articles contain no C or C++ code.

file or directory	book chapter	chapter title and author
-----		
	I	POLYGONS AND POLYHEDRA
centroid.c	I.1	Centroid of a Polygon
convex_test/	I.2	Testing the Convexity of a Polygon
ptpoly_weiler/	I.3	An Incremental Angle Point in Polygon Test
ptpoly_haines/	I.4	Point in Polygon Strategies
delaunay/	I.5	Incremental Delaunay Triangulation
vert_norm/	I.6	Building Vertex Normals from an Unstructured Polygon List
	I.7	Detecting Intersection of a Rectangular Solid and a Convex Polyhedron
collide.c	I.8	Fast Collision Detection of Moving Convex Polyhedra
	II	GEOMETRY
	II.1	Distance to an Ellipsoid
dist_fast.c	II.2	Fast Linear Approximations of Euclidean Distance in Higher Dimensions
outcode/	II.3	Direct Outcode Calculation for Faster Clip Testing
sph_poly.c	II.4	Computing the Area of a Spherical Polygon
	II.5	The Pleasures of 'Perp Dot' Products
	II.6	Geometry for N-Dimensional Graphics
	III	TRANSFORMATIONS
arcball/	III.1	Arcball Rotation Control
	III.2	Efficient Eigenvalues for Visualization
inv_fast.c	III.3	Fast Inversion of Length- and Angle-Preserving Matrices
polar_decomp/	III.4	Polar Matrix Decomposition
euler_angle/	III.5	Euler Angle Conversion
	III.6	Fiber Bundle Twist Reduction
	IV	CURVES AND SURFACES
data_smooth/	IV.1	Smoothing and Interpolation with Finite Differences
	IV.2	Knot Insertion using Forward Differences
	IV.3	Converting a Rational Curve to a Standard Rational Bernstein-Bezier Representation
curve_isect/	IV.4	Intersecting Parametric Cubic Curves by Midpoint Subdivision
patch_conv.C	IV.5	Converting Rectangular Patches into Bezier Triangles
nurb_polyg/	IV.6	Tessellation of NURB Surfaces



	IV.7	Equations of Cylinders and Cones
implicit.c	IV.8	An Implicit Surface Polygonizer
	V	RAY TRACING
	V.1	Computing the Intersection of a Line and a Cylinder
ray_cyl.c	V.2	Intersecting a Ray with a Cylinder
vox_traverse.c	V.3	Voxel Traversal along a 3D Line
multi_jitter/	V.4	Multi-Jittered Sampling
minray/	V.5	A Minimal Ray Tracer
	VI	SHADING
	VI.1	A Fast Alternative to Phong's Specular Model
	VI.2	R.E versus N.H Specular Highlights
	VI.3	Fast Alternatives to Perlin's Bias and Gain Functions
	VI.4	Fence Shading
	VII	FRAME BUFFER TECHNIQUES
	VII.1	XOR-Drawing with Guaranteed Contrast
	VII.2	A Contrast-Based Scalefactor for Luminance Display
dyn_range/	VII.3	High Dynamic Range Pixels
	VIII	IMAGE PROCESSING
emboss.c	VIII.1	Fast Embossing Effects on Raster Image Data
coons_warp.c	VIII.2	Bilinear Coons Patch Image Warping
convolve.c	VIII.3	Fast Convolution with Packed Lookup Tables
thin_image.c	VIII.4	Efficient Binary Image Thinning using Neighborhood Maps
clahe.c	VIII.5	Contrast Limited Adaptive Histogram Equalization
mrsfoley.im	VIII.6	Ideal Tiles for Shading and Halftoning
	IX	GRAPHIC DESIGN
	IX.1	Placing Text Labels on Maps and Diagrams
graph_layout/	IX.2	Dynamic Layout Algorithm to Display General Graphs
	X	UTILITIES
trilerp.c	X.1	Tri-linear Interpolation
interp_fast.c	X.2	Faster Linear Interpolation
vec_mat/	X.3	C++ Vector and Matrix Algebra Routines
GraphicsGems.c	X.4	C Header File and Vector Library
	V	RAY TRACING



back to [contents](#)

## Beyond Graphics Gems, by Paul Heckbert ([ph@cs.cmu.edu](mailto:ph@cs.cmu.edu))

[From the Preface of Gems IV, which Paul posted to the net. -EAH]

In addition to the "Graphics Gems" series, there are several other good sources for practical computer graphics techniques. One of these is the column "Jim Blinn's Corner" that appears in the journal "IEEE Computer Graphics and Applications". Another is the book "A Programmer's Geometry", by Adrian Bowyer and John Woodwark (Butterworth's, London, 1983), which is full of analytic geometry formulas. A mix of analytic geometry and basic



computer graphics formulas is contained in the book "Computer Graphics Handbook: Geometry and Mathematics" by Michael E. Mortensen (Industrial Press, New York, 1990). Another excellent source is, of course, graphics textbooks.



back to [contents](#)

---

## Radiosity vs. Ray Tracing, by Rico Tsang ([csrico@cs.ust.hk](mailto:csrico@cs.ust.hk))

[I made a few minor fixes, but otherwise this was a nice cheat-sheet summary of the functional (vs. algorithmic) differences between meshed radiosity and classical ray tracing. -EAH]

### Radiosity

work best to model -

1. Area light sources
2. Diffuse reflections
3. Color bleeding
4. Soft shadows

### Limitations of Radiosity

=====

1. All surfaces are assumed to be perfect diffuse reflectors.
2. Specular reflections and transparent effects cannot efficiently modelled.
3. Time & storage consuming. For  $n$  surface patches in a scene, you need to calculate and store  $n^2$  form factors. If you use the 'progressive refinement' radiosity method or hierarchical radiosity, you can minimize the storage requirement for storing the form factors.

### Advantages of Radiosity

=====

1. The calculation of radiosities is view-independent. Once it does so, a view of the environment can be generated with relatively little effort (i.e. via gouraud interpolation) for any camera position.

==> provides the capability of interactive walkthrough

2. With the progressive refinement method, a geometrically correct view of the environment can be displayed almost immediately.



back to [contents](#)

---

## ACM SIGGRAPH Online Bibliography Updated, by Frank Kappe ([fkappe@iicm.tu-graz.ac.at](mailto:fkappe@iicm.tu-graz.ac.at))

This is to announce that the well-known ACM SIGGRAPH Online Bibliography (that holds some 16,000 references to computer graphics literature) has been updated. The ACM SIGGRAPH Online Bibliography Database was compiled by

Stephen N. Spencer ([spencer@cgrg.ohio-state.edu](mailto:spencer@cgrg.ohio-state.edu)) with help from various contributors.

The data set is available through the Hyper-G server of the Graz University of Technology either by WWW protocol:  
<http://www.tu-graz.ac.at/CSIGGRAPHbib>

or by Gopher protocol:

<gopher://gopher.tu-graz.ac.at/11SIGGRAPHbib>

or using a Hyper-G native client such as Harmony (for UNIX/X-Windows), which works best, of course. If you are interested in obtaining a copy of Harmony please look at <ftp://iicm.tu-graz.ac.at/pub/Hyper-G/Harmony>

for downloading instructions. People without clients may also try <telnet://info.tu-graz.ac.at>

to access the Hyper-G server of Graz University of Technology.

More information about the the ACM SIGGRAPH Online Bibliography Project can be reached from the URLs mentioned above.

[The bibliography can be FTP'ed from [siggraph.org](http://siggraph.org): /publications/bibliography -EAH]



back to [contents](#)

---

## How to be Notified of New POV Releases

The official POV distribution site is <ftp.uwa.edu.au>. This site has become quite active and holds many POV scene files and utilities as well as POV itself.

As an adjunct to this site, a mailer has been set up which permits file requests to be made by email, and which also manages a mailing list.

The site and the list itself is maintained by a member of the POV-Team [Chris Cason ([cjcason@yarrow.wt.uwa.edu.au](mailto:cjcason@yarrow.wt.uwa.edu.au))].

You can request that your name be placed on one or both of the POV mailing lists. These are not normal mailing lists in that you cannot submit messages for them ; only the POV-Team can. The lists are as follows -

ANNOUNCE	Used when new releases of POV are made. This includes POV-related items from the POV-Team, such as the soon-to-be released hypertext help system.
NEWS	Used to send new issues of POV-News and anything else relevant to POV, such as new publications, magazine articles, etc.

Volume on both of these lists will be extremely low ; perhaps only one message per month if even that. This is the easiest way for POV users to be informed of patches, bug fixes, new releases, new features, release dates for new versions and the actual release of new versions.

To join these lists, mail a message to -

[povmail@uniwa.uwa.edu.au](mailto:povmail@uniwa.uwa.edu.au)

The subject of the message is unimportant.

In the body of the message, place the commands -

```
JOIN ANNOUNCE      (to join the announce mailing list)
JOIN NEWS           (to join the news mailing list)
```

- or -

```
JOIN ALL            (to join both)
```

You should receive a reply indicating that your request was successful. If you don't, please try again. For more information, please mail the above address with a message containing the word -

HELP

on the first line.



back to [contents](#)

---

## PoVSB Windows-based Modeler v0.85, by Jeff Hauswirth ([jhauswir@carbon.denver.colorado.edu](mailto:jhauswir@carbon.denver.colorado.edu))

What's new-

```
Bezier Surfaces.
A help file.
```

PoVSB is now shareware at \$30. Everything is functional except saving the Bezier objects. Exporting Bezier objects to PoV is functional.

PoVSB is a Windows based modeler for the Persistence of Vision Raytracer. The goal of PoVSB is to allow users of PoV to quickly and easily design scenes in the Windows environment with true camera preview of the scene so no guess work is involved.

I would like to collect a bunch of PoVSB example files to include with PoVSB and to also make them available via ftp so other people can use them. If anyone is interested, just e-mail me your example files to include with PoVSB.

I have included two example files- teacup and satellite. Try rendering the satellite with and without the MB PoV option. With the MB option it was 6 times faster, if I remember right.

I would like to get some feedback from all the people with PoVSB, which I think is up near ~350. I want to know what new features you would like.

PoVSB has been tested on 386-SX-4M up to 486-66-16M. The 386-SX had trouble with loading the teacup example file. Other than that I have found no problems.

Features:

```
Four view of scene:
  3 ISO views, 1  Camera view
Objects supported:
  Sphere, Box, Plane, Cone, Cylinder, Torus,
  Height Fields, Bezier Surfaces.
```

CSG: Union, Merge, Intersection, Difference  
RAW: Only flat triangle output available (for now).

Interactive transformations of objects

Lights: Point, Spot

Multiple Layers

Textures: Customizable, add your own textures

How to get PoVSB:

ftp to: [vincent.iastate.edu](http://vincent.iastate.edu)

Username: anonymous.jhaus

Password: [YOUR\\_USERNAME@YOUR.EMAIL.HOST](mailto:YOUR_USERNAME@YOUR.EMAIL.HOST)

If you already have PoVSB you only need to get povsbXX.zip, Otherwise you need to get povsb.zip.



back to [contents](#)

---

## Porting Rayshade, PBM, etc from Unix to DOS, by Mike Castle ([mcastle@mcs213k.cs.umn.edu](mailto:mcastle@mcs213k.cs.umn.edu))

Amazingly enough Jasen M. Mabus said:

> Where can I find PBMtools, a preprocessor implementor, and CoProcessor  
> emulator for 386 in DOS executable?

I ported pbmplus using djgpp a couple years back. djgpp is a development environment by DJ Delorie that includes a homegrown dosextender called "go32", gcc/g++ compilers, flex, bison, assorted binary utilities (ar, ld, nm, etc), make, and so on. This environment is an eclectic mix of unixish and dosish calls, but is a very workable environment.

My port of pbmplus can be found on any simtel archive in graphics/pbmpl19d.zip (such as [oak.oakland.edu](http://oak.oakland.edu) in pub/msdos/graphics/pbm...).

djgpp can also be found in the djgpp dir (pub/msdos/djgpp on oak). You could scan cpp from the djgpp distribution, and have a working preprocessor.

I included the then current emu387 387 emulator from the djgpp package in my pbmplus port, however, that was some time ago. There are newer and better emulators that can be found in the djgpp archives. Of course, the emu387 and wmemu387 (another emulator compatible with go32, but done by someone else) will only work with go32 binaries.

The go32 included with my pbmplus port is out of date, and you may want to get the most recent go32.exe from the djgpp packages. you may also have to run a program called 'dpmifix' on the pbmplus binaries to get things working correctly (i'm not sure as i don't use dos much anymore, having switched to os/2, and so i'm not quite up on djgpp and stuff, and i don't even have my own copy of my pbmplus port anymore so....).

btw, i used the djgpp package to port rayshade quite some time ago, but as there was already a 386 port done using a commercial dos extender, never made it available. The port was very straight forward, and only required the normal flex/lex bison/yacc fixes (such as rayshades misuse of the yyline action), 8.3 file name fixups, and binary mode stdin/stdout. It worked very well, including support for RLE files (i ported the URT as well), cpp, and so on.



back to [contents](#)

## REYES & Patents, William C. Archibald ([billa@entropys.sps.mot.com](mailto:billa@entropys.sps.mot.com))

> Is REYES patented?

Pixar holds three (at last count) patents (including US Patent Number 5,025,400) on "Pseudo-random point sampling techniques in computer graphics". This includes claims that cover stochastic ray-tracing as well as what you are probably referring to by REYES.

> If so, what part of the algorithm?

Clearly stochastic sampling techniques as they apply to computer graphics.

> Is one allowed to implement the algorithm?

I believe that in the U.S.A. one is allowed to implement the algorithm for the purposes of studying it and trade-offs that may be involved. One is not allowed to redistribute any implementations without permission, as that is the very thing that patent law is set up to protect against.

[and in a separate note by the same author:]

In patent 5,025,400 the claims go on for 4.5 pages (9 columns) of legalese, and they seem to pretty much make claim to usage w/ respect to lens effects (depth of field), motion blur (temporal jitter), penumbra and other soft shadow effect, and just about every variation of jittering reflected rays of objects for any reason you might imaging. I've not been able to really pin down good, old-fashioned anti-aliasing anywhere in the claims though...



back to [contents](#)

---

## Going from AutoCAD and 3DS into Ray Tracing, by Sean Ross ([ross@CSOS.ORST.EDU](mailto:ross@CSOS.ORST.EDU))

Well, for starters, saying one raytracer is "better" than the others is guaranteed to cause a flame fest of amazing proportions, so I will just tell you about my experience so far and let you make your own determinations.

What I discovered, from following this group for a while and reading the "Mini-FAQ", is that there are quite a few different raytracers around with different capabilities. I found Vivid a bit rough to get started with, and therefore tried PoVray, it's been great so far. I find it easy to use because the language is fairly intuitive.

What has turned out to be easiest for me to create good raytracings with a fairly short learning curve is using the following method:

Use AutoCAD Release 12 with AME to create the initial objects. AME allows very easy creation of complex 3d objects using addition, subtraction, intersection, etc.

Export the finished objects into 3d Studio (I'm using ver 1) to set the camera and lighting. Assign surfaces to all objects with standard names like "surface1" etc.

(If you don't have access to AutoCAD and/or 3D Studio, try using Moray, it's shareware (\$50) and seems to be easy to use and well supported)

Use 3DS2POV program to convert the 3d studio file to a PoV file and include file.

Edit the resulting .pov file and replace 3d studio's surfaces with the much better surfaces included with pov. (I've also found it very simple to modify these surfaces to do almost anything.)

Render sample files using a small size (160 x 100) and adjust any settings you don't like in the .pov file.

Start the final rendering with high quality, anti-aliasing, and a large size, and plan on not using your system for a while if you're using a lowly 486DX266 w/8MB like me. My finished quality renderings generally take about 8 hours to process for a 2048 x 2048 image, and I consider these images to be fairly simple (not too many complex objects).

If the above doesn't sound like too much work, give PoVray a try, it's given me very good results with a negligible learning curve. I plan on trying out some of the others available, especially Radiance because of its radiosity features, but I haven't got around to it yet.

If anyone else has any suggestions on how I might improve upon the methods I described above, I would be greatly interested.

By the way, just one warning: after I started creating these images, I find myself dedicating a pretty good amount of time to it, I think it's pretty addictive, so be careful.



back to [contents](#)

---

## Computer Lego Modeling, by Paul Gyugyi ([paul@gyugyi.win.net](mailto:paul@gyugyi.win.net))

Check out the files on [earthsea.stanford.edu](http://earthsea.stanford.edu) in the ~ftp/pub/lego/cad/click and ~ftp/pub/lego/cad/rayshade directories. They include:

- 1) A library of high-detail shape definitions for Rayshade, a 3D modelling/ray tracing program (legolib.ray).
- 2) A definition of a language, LADEL, for specifying brick types and positions.
- 3) A compiler to convert LADEL files to rayshade input files that use the above library of brick definitions.
- 4) A new extension to the compiler that generates a 3D wire-frame image from a LADEL file. You can fly around and rotate it. It uses an Xwindows library that is similar to GL, called VOGL, and runs on UN\*X machines.

There is no documentation for most of this stuff, but if you want to try it out, email me any questions you have and I'll start creating a how-to.



back to [contents](#)

---

## Rowe's Ray Tracing World BBS, by Harry Rowe ([Harry.Rowe@wedowind.meaddata.com](mailto:Harry.Rowe@wedowind.meaddata.com))

phone: (513) 866 - 8181 v.32bis (Dayton, OH)

I have a new ray tracing BBS. No fees. It does require user to have Windows 3.1 and special client software which can be downloaded on the first call. It is Excalibur (tm) based and is a 100Windows GUI BBS. I only carry DOS/Windows tracers, but also have tons of objects, textures, converters, etc. I have most of what Compuserve GRAPHDEV has in the way of RT utilities. I primarily support Polyray v1.7 and Imagine 3.0 for DOS. We also have expert 3DS users.

[Harry has recently made a nice 500+ line summary guide to the major POV utilities, with 4-15 lines about each one. I won't put it here, but contact him for where it is (or if all else fails, contact me and I'll send you the old copy I have). -EAH]



back to [contents](#)

---

## On Using BSP trees, by Benton Jackson ([benton@fenriswolf.com](mailto:benton@fenriswolf.com))

[BSP trees are a nice technique for doing 3D and avoiding z-buffering in games. You partition a static environment with a binary-tree hierarchy of cutting planes, classifying everything in the node as in front of or behind the plane of some given polygon (and splitting things cut by the plane). Note that this is different than the BSP trees used in ray tracing, where the planes are independent of the polygons and are essentially a more flexible octree like structure. Worth understanding in general. -EAH]

>Do I have to build a new BSP tree for all visible polygons at each frame  
 >redraw? This is what I suspect, but I can't help thinking that there is  
 >something more I could do to improve calculation times if I had separate BSP  
 >trees for each shape.

I'm surprised that isn't in the FAQ. That's the whole point of BSP trees! Based on the current viewpoint, if it is in front of the root face, draw the back tree, then the face, then the front tree. This works since that face is guaranteed to be in front of everything on one side of the tree, and behind everything on the other side. Which way you go depends on the viewpoint. Cool, huh? Like this:

```
DrawBSP(tree B) {
    if (!B) return;
    if (in_front_of(viewpoint, B->face) {
        DrawBSP(B->back);
        DrawFace(B->face);
        DrawBSP(B->front);
    } else {
        DrawBSP(B->front);
        DrawFace(B->face);
        DrawBSP(B->back);
    }
}
```

This doesn't completely depth sort the faces. But you don't have to! You just need to ensure that a face that obscures another faces is drawn in the correct order. That's what this does.

If an object moves, then the tree has to be rebuilt. I think there is a way to merge BSP trees, but I don't know how. Anybody?



back to [contents](#)

---

## Books about Commercial Renderers, by Don Lewis, Jimbo and Yury German

3D Studio 3.0:

Don Lewis ([djlewis@ualr.edu](mailto:djlewis@ualr.edu)) writes:

Try Inside 3D STUDIO Release 3, comes with a CDROM of sample DXF's and images and maps. Listed as: ISBN 1-56205-075-3 price \$49.95 USA / \$65.95 CAN / L46.99 Net U.K. (inc of VAT)

About the CD:

Meshes, Texture maps, Bump maps, Graphics file format conversion utility, Special effects filters, Utilities for image viewing, printing, and conversion (windows and dos), An updated Autodesk animation player, Files for exercises in the book, etc ...



I like the book and a lot applies to 3DStudio 2.0 as well.

---

Jimbo ([Jimbo@agdesign.demon.co.uk](mailto:Jimbo@agdesign.demon.co.uk)) writes:

Inside 3D Studio:

-----

Author(s):        Steven D. Elliott  
                 Phillip L. Miller  
                 Gregory G. Pyros

ISBN: 1-56205-075-3

Publishers: New Riders Publishing.

An excellent book including a CD, unfortunately doesn't include the IPAS routines mentioned numerous times throughout.

3D Studio Applied:

-----

Author(s): Nancy Fulton

ISBN: 0-929870-24-7

Publishers: Advanstar Communications Inc.

---

Imagine system:

> Question: My uncle has this and would like to know  
> if there are any other references for IMAGINE that are a  
> bit easier to understand for those less technologically literate.

Yury German ([yury@bknight.jpr.com](mailto:yury@bknight.jpr.com)) writes:

Yes, there is a great book, it's called Understanding Imagine. Steve Worley is the author. Now the book is "Understanding Imagine 2.0" for the 2.0 version. In a few months the book will be out for "Understanding Imagine 3.0".

The first book was done with the Amiga computer in mind. The 3.0 book might be done for both computers. If you want you can email Steve at [worley@cup.portal.com](mailto:worley@cup.portal.com) for information.

The books are great with alot of humor and all the things needed to make manuals easy and understandable.



back to [contents](#)

---

Eric Haines / [erich@acm.org](mailto:erich@acm.org)



# Don Mosaic



[Here's the photomosaic of Don Greenberg](#)

Don Greenberg is the founder and head of the [Program of Computer Graphics](#) at Cornell University. 1998 marks the 25th anniversary of the program, so in honor of the occasion I assembled a mosaic image of various people who have passed through the lab. The lab web site has an [alumni page](#), if you want to know more.

A number of people sent digital images to Ben Trumbore for the Cornell reunion dinner at SIGGRAPH, many others I gleaned from the web, and a CFP (Call For Pixels) sent out to alumni brought in many more. A total of 119 photos were used in this photomosaic. Thanks to Ben for all his help and FTP space.

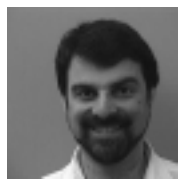
Eric Chen kindly donated time to create the FlashPix version of the image.

An excellent site for information about various photomosaic programs can be found at [William Leigh Hunt's website](#).

The perl program which made the photomosaic can be [downloaded](#) - read the top of the file for how to use it. It works only for grayscale images. You can get details on the gory [baling-wire-and-duct-tape story](#) of how I made the photomosaic of Don using free & shareware tools.

What follows is the table of the 121 images that were used to make up the [mosaic image](#) itself. "Times used" is the number of times the image is used in the mosaic, i.e. the number of "pixels" this image replaced. "Grayscale value" is the grayscale value (0-255) that the image represented (see the [story](#) for the shocking details about gamma correction, or lack thereof).

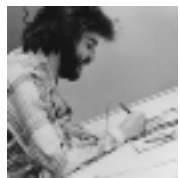
Image	Name	Times used	Grayscale value
	Alppay, Gun	75	79.03
	Arvo, Jim	13	187.63
	Baraff, David	14	102.17



Barsky, Brian - now

6

95.58



Barsky, Brian - using a tablet

40

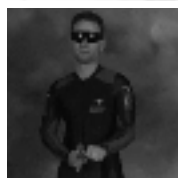
145.94



Barsky, Brian - then (from his Cornell ID)

22

170.13



Baum, Dan - FutureMan

86

49.66



Baum, Dan - then (prehistoric)

120

42.47



Berggren, Martin

13

135.06



Blocksom, Jon

13

136.80



Carey, Rikk

6

119.58



Chen, Eric

6

92.45



Cohen, Michael

6

122.40



Cohen, Michael - age 3

207

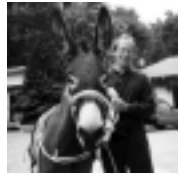
195.12



Cook, Rob

15

90.65



Corson-Rikert, Jonathan - and friend

6

89.18



Crane, Ted - doing Morris dance fooling

7

115.09



Crane, Ted - now (in Puerto Rico)

22

81.96



Crane, Ted - then

26

129.07



Dorsey, Julie

13

83.63



Dutre, Philip

8

115.99



Feibush, Eliot

6

82.48



Fernandez, Sebastian - about to be morphed

15

95.75



Ferwerda, Jim

6

91.47



Foo, Sing-Choog

21

148.41



Gelb, Dan

17

92.05



Gelb, Dan - 3D head scan

14

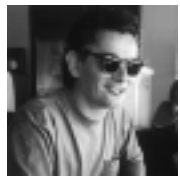
92.91



Georgiades, Priamos - clown guy

6

89.67



Georgiades, Priamos - now

30

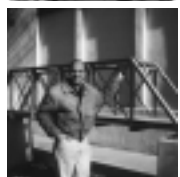
177.15



Georgiades, Priamos - then

11

95.42



Gerstle, Walter

31

98.18



Goral, Cindy

30

74.50



Greenberg, Don - ooo, he's his own pixel

22

105.14



Greger, Gene - old

6

156.38



Greger, Gene - doh! It's just a close up

12

136.05



Haber, Robert

14

113.55



Haines, Eric

8

160.19



Haines, Eric - 3D head scan

136

66.41



Hajjar, Jerry

8

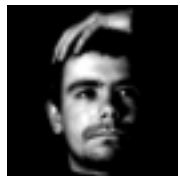
134.16



Hall, Roy

17

104.12



Hart, David

178

39.13



Hedelman, Harold - ride 'em

6

95.49



Hsieh, Patrick

14

128.71



Hubbard, Philip

6

92.90



Hubbard, Philip - jamming

15

103.42



Immel, Dave

10

151.81



Isaacs, Paul

14

151.43



Joblove, George

14

108.42



Joseph, Jonathan - flauting

7

152.27



Joseph, Jonathan - with flipper

6

82.53



Joseph, Jonathan - now

11

122.78



Joseph, Jonathan - then (in drag)

6

117.18



Kartch, Dan

338

231.59



Kershaw, Kathy - 1992

6

120.62



Kershaw, Kathy - glueing thesis pictures

15

173.54



Kershaw, Kathy - 1995 (wedding)

6

120.77



Kindlmann, Gordon - strung out

22

98.42



Kindlmann, Gordon - with shrubbery

6

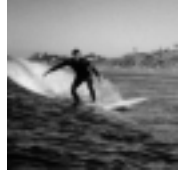
116.66



Kindlmann, Gordon

11

155.82



Kochevar, Peter

7

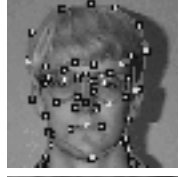
116.97



Kunz, Andrew

40

189.55



Kunz, Andrew - about to be morphed

38

106.16



Kunz, Andrew - morphed

6

104.76



Lafortune, Eric

6

103.42



Lengyel, Jed - now

23

86.69



Lengyel, Jed - then (Cornell photo ID)

24

86.72



Levoy, Marc

40

54.07



Lischinski, Dani

6

98.22



Lobb, Richard

41

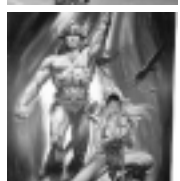
100.23



Lu, Wei

19

163.27



Lu, Wei - the heroic side

15

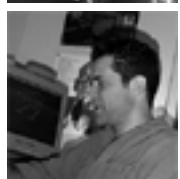
118.28



Lu, Wei - then

8

159.91



Malone, Michael

14

120.85





Marschner, Steve

6

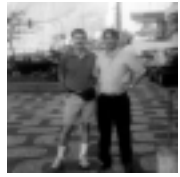
82.51



Martha, Luiz - at Rand Hall

6

114.83



Martha, Luiz - now, in Rio

32

109.22



Meyer, Gary

8

123.67



Monks, Michael

67

73.20



Monks, Michael - then, with family

17

84.74



Monks, Michael - then

6

89.33



Monks, Michael

16

90.26



Nall, Dan

7

160.08



Novins, Kevin

27

132.90



O'Connor, Tim

16

119.48



O'Connor, Tim - then, with family

6

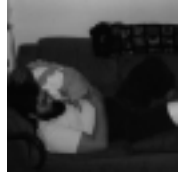
82.50



O'Connor, Tim - wild man

7

153.73



O'Connor, Tim - then

103

57.21



Pattanaik, Sumant

37

183.26



Peng, Liang

6

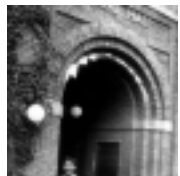
135.78



Perucchio, Renato

6

89.04



Rand Hall (with Luiz Martha's head)

18

93.97



Ramasubramanian, Mahesh

6

116.08



Reichert, Mark

59

56.98



Rushmeier, Holly

15

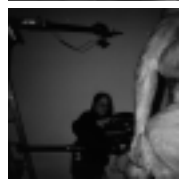
81.14



Rushmeier, Holly - measuring light at Rand Hall

6

134.69



Rushmeier, Holly - measuring light from the Pietà

26

54.82



Salesin, Dave

45

73.74



Schulman, Michael

861

29.64



Shaw, Erin

44

126.33



Sheldon, Hurf

17

167.31



Shirley, Pete

12

131.03



Sillion, Francois

6

115.78



Srivastav, Sanjeev

6

93.76



Stettner, Adam

6

105.15



Swenson, Dan

16

173.57



Theory Center

30

112.19



Toler, Corey

17

82.61



Torrance, Ken

23

89.01



Trumbore, Ben

6

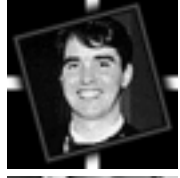
112.20



Verbeck, Chan

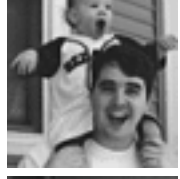
9

157.54

Wade, Bretton - from his [cool web page](#)

45

50.43



Wade, Bretton

11

122.32



Wallace, John

35

66.65



Wanuga, Paul

40

139.42



White, Donald

10

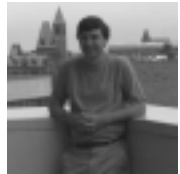
165.63



Wong, Eric

51

76.86



Zareski, David - at Cornell

14

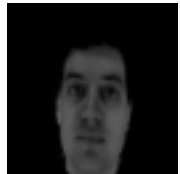
113.49



Zareski, David

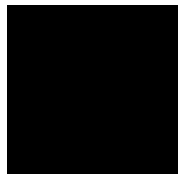
125

69.13

Zatz, Harold - [rotating 3D head scan](#)

391

17.61



pixel, black

30

0.00

pixel, white

271

255.00

## Gory Details

The idea for doing a photo mosaic occurred to me while editing [The Ray Tracing News](#). I had found a site in Norway with a photomosaic program called [Macbeth](#) available on the web. Also, the movie "The Truman Show" had a poster of Jim Carrey's face made using the same technique (this turned out to be made by [Runaway Technology](#)). It turns out the artist [Chuck Close](#) also uses this sort of technique (though taking it way beyond anything attempted here). Eventually it dawned on me that I had the perfect use for the concept...

Probably my greatest worry was finding images, and finding images that were light and dark enough to

cover the grayscale range. I also realized I'd have to bracket the image with pure white and black pixels (well, I could have shifted Don's image so that its minimum and maximum brightness levels fit into my grayscale gamut, but I didn't have a serious photo manipulation package available and didn't want to retouch Don's image anyway). Happily there were in the 79 photos I collected some fairly dark or light ones - enough images for a palette, but not so many that there would be a lot of palette collisions. I made some mistakes in collecting: I didn't realize Rich Coutts' picture wasn't him when he was young but was his son, for example. I was aiming for one picture per gray level - more than this would have complicated the process a fair bit. As it was, I had to cheat... but let's go back to the beginning.

I couldn't use the Macbeth software as I didn't run Unix (I'm in the Wintel world now). Instead, I used the following free or shareware packages (I paid for most of the shareware, that's how good they are) for Windows NT:

- [Netscape](#): for finding and saving images.
- [HyperSnap](#): for screen captures from Netscape.
- [Paint Shop Pro](#): for resampling and viewing images.
- [Image Alchemy](#): for batch conversion of images.
- [JASC Image Robot](#): for batch resizing and cropping of images.
- [Perl](#): for everything else that could not be done by packages.

After collecting, I found I couldn't view Rikk Carey's JPEG with Paint Shop Pro. I had to use HyperSnap to capture it directly from the screen.

Once the images were collected, they were viewed and borders trimmed off them (I missed a bit on Jerry Hajjar's picture). Any images larger than 640 x 480 were resized smaller than this, since the free, demo version of Image Alchemy is crippled to this size (otherwise it's mostly wonderful). All images were converted to grayscale GIFs by Image Alchemy, for simplicity. Then two different scripts (One for portrait, one for landscape. It was time consuming to have to use two different ones and have to separate the images by hand) were used to automatically resize and trim each image down to 64x64. A number of images had to then be redone by hand in Paint Shop Pro, as heads were cut off, etc. I picked a square image ratio because there were a fair number of landscape oriented shots; 64x64 because it would be easy to downsize if needed. In retrospect, this size was not ideal - in a number of images I was forced to chop off the top of the head.

In retrospect, if I had wanted to make more than one set of pixel images, I would have gone with [NetPBM](#), which would have allowed me to write my own filter in C. Not a good option if you're not a programmer, though.

I converted the pixel-images to PBM format so I could easily read them with a Perl script. I then read in the pixel-images, figured out the average grayscale of each, and so created a palette to use to render Don. It turned out there were some images which had the same average grayscale value - I simply cheated by pushing other entries up or down a level or two in order to resolve these collisions, so that each grayscale level would have only one image associated with it.

Are you still awake? Anyway, here was something interesting: I now had 81 images to use for pixels. What I wanted to do was take Don's original picture (which used all 256 grayscale levels) and dither it

down to this palette of 81. In Unix it would have been easy to do this, creating a palette to be read in by the Utah Raster Toolkit and applied to an image. In the Windows world this process is tough with the packages I had at hand. Both Paint Shop Pro and Image Alchemy profess to be able to apply a palette to an image, but both rot for grayscale images. I found this out after hand-editing a palette, grayscale-level by bloody grayscale-level, and applying it to Don's image. There were large gaps in how the palette was applied - I think the problem was that these programs do inverse color table lookup by building a 32x32x32 (5 bits) grid and mapping all locations to a palette entry. This is inappropriate for grayscale level work. What I had to do was write my own.

It turned out to be not that bad to do in Perl. I simply read in the palette and figured out how to map each grayscale level to the palette. Then I read Don's image in and did Floyd-Steinberg error-diffusion to get a new image - nice and simple for a grayscale image [looking back on it, I think the final mosaic had the error diffusion code off (oops) - see if you can find the row of Roy Halls in Don's right cheek]. I also wrote some perl programs to generate histograms and to check that the new image actually used all the palette entries (i.e. the images). This check was vital in detecting the limitations of Paint Shop Pro and Image Alchemy explained earlier.

The final step was to generate the mosaic itself. I was tempted to use the [gd](#) package, but it looked like a bit of work to compile and figure out. I finally ended out writing a Perl program that read in Don's image and the palette, then scanline by scanline filled in and wrote out an output pixel array with the images representing each input pixel. The uncompressed ascii output image file was around 60 megabytes. Happily, Paint Shop Pro was up to the task of reading and converting this file to GIF (still weighing in at 12 Meg). I then wrote a variant on the mosaic perl program that would create 4 separate tile images I could print on four separate sheets (Paint Shop Pro was not quite up to the task of accurately cropping an image this large, due to the GUI) - this took way too long to debug, but within an hour or two I was done. All in all the whole process took less than two days.

Once I finished, I realized I had made a mistake: I forgot to [gamma correct](#). Say you have two images you want to use as "pixels". One is gray, (128,128,128), the other is alternating pixel-wide lines of black and white. It turns out that perceptually the black and white band image will be much lighter on most monitors, due to gamma. You need to boost the gray image up considerably to match. I actually tried to fold this in, but the image looked darker. It wasn't until I left for SIGGRAPH that I realized that what I needed to do was transform the "pixel" images to linear space, figure out the average value, then use this value against a linearized version of Don's image to get a better match. {follow-up: I tried this later, but it didn't look as good for some reason.}

Another interesting concept that John Wallace first pointed out was that I could try to match larger sample areas of Don's image with the pixel images. For example, say Don's image was 256x256 originally. My method was to simply resize it to 64x64 and match the grayscale value of pixel images with the pixel values in Don's image. Instead, I could have looked at each area on Don's image as a 4x4 set of pixel values. Now I could compare each pixel image with this 4x4 subimage, resizing each pixel image for comparison purposes to 4x4. The image with, say, the least absolute difference for all pixels would be the closest match. For example, if the 4x4 sample on Don's right eye was dark on the left side, light on the right, then an image that was dark on the left and light on the right would be the best match.

At least, that's the theory. I tried this out, and what actually happened is that the low contrast images were favored over all others in most locations. If you think about it, the majority of subimages are slowly

changing gradients, which will favor low contrast images. The net effect was that about 30 of 79 images were never used at all, so I abandoned this technique. There are, I think, some interesting ways to make such matching techniques work better, especially if you allow the pixel images to be modified in brightness, contrast, or hue (for color mosaics). But, for now I'm done with the topic. Now, for the fiftieth anniversary I plan a holographic template with stereogram pairs in which you'll have to defocus your eyes while shaking your head back and forth in order to see it...

## The Saga Continues

After sending out a CFP (Call For Pixels), I received a lot more images to use in a new photomosaic with more alumni - a total of 119 photos were used. I wrote a new perl program which did all the steps in a single run and allowed a larger palette. It can be [downloaded](#) - read the top of the file for more information on how to use it.

---

Last change: *September 24, 1998* [Eric Haines](#) / [erich@acm.org](mailto:erich@acm.org)



# The Close Objects Buffer: A Sharp Shadow Detection Technique for Radiosity Methods

[A. C. Telea](#)

[Eindhoven University of Technology](#)

Eindhoven, The Netherlands

[alext@win.tue.nl](mailto:alext@win.tue.nl)

[C.W.A.M. van Overveld](#)

[Eindhoven University of Technology](#)

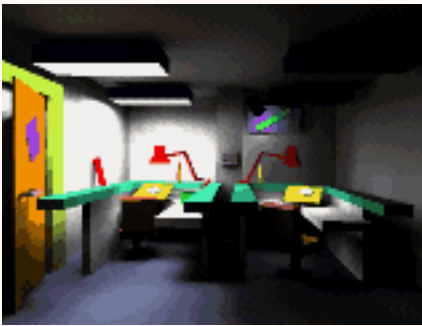
Eindhoven, The Netherlands

[wsinkvo@win.tue.nl](mailto:wsinkvo@win.tue.nl)

## Abstract:

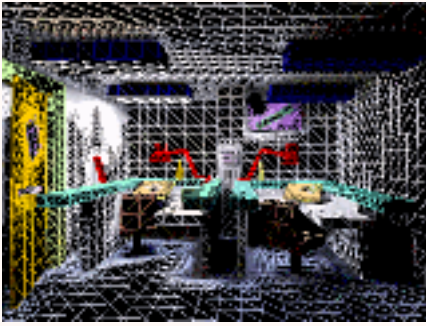
Detecting sharp illumination variations such as shadow boundaries is an important problem for radiosity methods. Such illumination variations are captured using a nonuniform mesh that refines the areas exhibiting high illumination gradients. Nonuniform meshing techniques like discontinuity meshing often rely on shadow casting, and as a result their application is computationally expensive. This paper presents a sharp shadow detection technique for radiosity tools based on the progressive refinement method. The presented technique offers good results (especially for capturing sharp shadows cast by small "detail" objects), is very simple to implement, has negligible time and space requirements, and integrates well with other adaptive subdivision strategies in a radiosity tool based on progressive refinement.

## Images:



Office room rendered with progressive refinement method, adaptive subdivision and close-objects buffer. Approx. 400 polygons, 3000 patches, 18000 subpatches. Rendering time: 2.5 minutes on a SGI INDY R4400, 300 refinement iterations.

[full image](#)



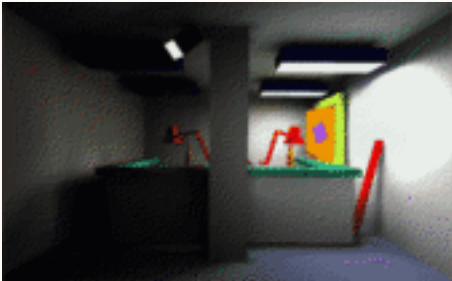
Same office room as above, view of the meshing created during the preprogressive refinement process.

[\*full image\*](#)



Another view of the office room.

[\*full image\*](#)



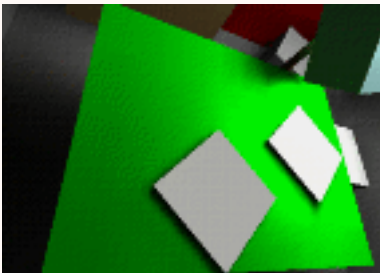
Back view of the office (notice the shadow cast by the red beam on the right wall).

[\*full image\*](#)



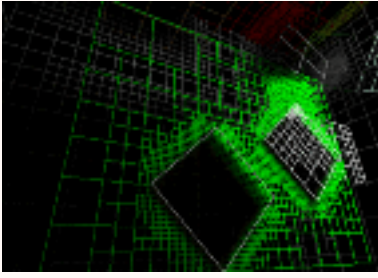
Hall with beams, rendered as a test for ray-tracing form factors. The image was rendered with 300 iterations.

[\*full image\*](#)



Close view of a table lit by a desk lamp. There are two sheets of paper on the table casting a sharp shadow on it. The close objects buffer was used to accurately capture these shadows.

[\*full image\*](#)



Discontinuity meshing generated by the close objects buffer for the scene depicted above. The subdivision proceeded along the papers' edges and the part of the table close to the light source. The mesh was constrained to be balanced.

[\*full image\*](#)

---

[Return to \*jgt\* home page](#)

Graphics Gems vol. 1-5  
Master Table of Contents  
(ver 1.1; June 22, 2000)

Derived from source material created by  
David Kirk, Paul Heckbert and Alan Paeth  
Notes: some titles have been truncated;  
page numbers must be rechecked.

=====  
Graphics Gems I (Andrew Glassner, Editor)  
=====  
Vol Txt Code Auth Title

1	Preface		
1	Introduction		
1	Mathematical Notation		
1	Pseudo-Code		
1	Contributors		
1	I. 2D Geometry		
1	3	Glassner	Useful 2D Geometry
1	12		Trigonometry Summary
1	13	Glassner	Useful Trigonometry
1	18	Paeth	Trigonometric Functions at Select Points
1	20	Goldman	Triangles
1	24 649	Turk	Generating Random Points in Triangles
1	29 651	Shapira	Fast Line-Edge Intersections on a Uniform Grid
1	37		Anti-Aliasing Summary
1	38	Thompson	Area of Intersection: Circle and a Half-Plane
1	40	Thompson	Area of Intersection: Circle and a Thick Line
1	43	Thompson	Area of Intersection: Two Circles
1	47	Thompson	Vertical Distance from a Point to a Line
1	49 654	Paeth	A Fast 2D Point-on-line Test
1	51 656	Shaffer	Fast Circle-Rectangle Intersection Checking
1	II. 2D Rendering		
1	57	Paeth	Circles of Integral Radius on Integer Lattices
1	61 657	Heckbert	Nice Numbers for Graph Labels
1	64 660	Cychosz	Efficient Gen. of Sampling Jitter Using Look-up Tables
1	75		Scan Conversion Summary
1	76 662	Morrison	Fast Anti-Aliasing Polygon Scan Conversion
1	84 667	Heckbert	Generic Convex Polygon Scan Conversion and Clipping
1	87 681	Heckbert	Concave Polygon Scan Conversion
1	92	Wallis	Fast Scan Conversion of Arbitrary Polygons
1	98		Line-Drawing Summary
1	99 685	Heckbert	Digital Line Drawing
1	101 686	Wyvill	Symmetric Double Step Line Algorithm
1	105 690	Thompson	Rendering Anti-Aliased Lines
1	107	Ritter	An Algorithm for Filling in 2D Wide Line Bevel Joints
1	114	Wallis	Rendering Fat Lines on a Raster Grid
1	121 694	Spoelder,Ullings	Two-Dimensional Clipping: Vector-Based Approach
1	129	G.Lee, Penk, Wallis	Periodic Tilings of the Plane on a Raster Grid
1	III. Image Processing		
1	143		Anti-Aliasing Filters Summary
1	144	Pavicic	Anti-Aliasing Filters that Minimize "Bumpy" Sampling
1	147	Turkowski	Filters for Common Resampling Tasks
1	166	Olsen	Smoothing Enlarged Monochrome Images
1	171 711	Paeth	Median Finding on a 3-by-3 Grid
1	176 713	Hawley	Ordered Dithering

1	179	Paeth	A Fast Algorithm for General Raster Rotation
1	196	Schumacher	Useful 1-to-1 Pixel Transforms
1	210	Thompson	Alpha Blending
1 IV. Frame Buffer Techniques			
1	215	Glassner	Frame Buffers and Color Maps
1	219	Paeth	Reading a Write-Only Write Mask
1	221 715	Morton	A Digital "Dissolve" Effect
1	233 718	Paeth	Mapping RGB Triples onto Four Bits
1	246	Heckbert	What Are the Coordinates of a Pixel?
1	249 719	Paeth	Proper Treatment of Pixels as Integers
1	257	Glassner	Normal Coding
1	265 720	Heckbert	Recording Animation for Progressive Temporal Refinement
1	270	Schumacher	1-to-1 Pixel xforms through Color-Map Manipulation
1	275 721	Heckbert	A Seed Fill Algorithm
1	278	Fishkin	Filling a Region in a Frame Buffer
1	285	Wallace	Precalculating for Fast Fills, Circles, and lines
1	287	Gervautz,Purgathofer	Color Quantization by Octree Quantization
1 V. 3D Geometry			
1	297	Glassner	Useful 3D Geometry
1	301 723	Ritter	An Efficient Bounding Sphere
1	304	Goldman	Intersection of Two Lines in Three-Space
1	305	Goldman	Intersection of Three Planes
1	306		Mapping Summary
1	307	Paeth	Digital Cartography for Computer Graphics
1	321 726	Bame	Albers Equal-Area Conic Map Projection
1	326		Boxes and Spheres Summary
1	327	Montani,Scopigno	Spheres-to-Voxels Conversion
1	335 730	Arvo	A Simple Method for Box-Sphere Intersection Testing
1 VI. 3D Rendering			
1	343 733	Wyvill	3D Grid Hashing Function
1	346	Hultquist	Backface Culling
1	348	M.Lee	Fast Dot Products for Shading
1	361	Thompson	Scanline Depth Gradient of a Z-Buffered Triangle
1	364	Glassner	Simulating Fog and Haze
1	366	Glassner	Interpretation of Texture Map Indices
1	376	Glassner	Multidimensional Sum Tables
1 VII. Ray Tracing			
1	385	Ritter	A Simple Ray Rejection Test
1	387	Ray-Object	Intersection Summary
1	388	Hultquist	Intersection of a Ray with a Sphere
1	390 735	Badouel	An Efficient Ray-Polygon Intersection
1	394	A.Woo	Fast Ray-Polygon Intersection
1	395 736	A.Woo	Fast Ray-Box Intersection
1	397	Pearce	Shadow Attenuation for Ray Tracing Transparent Objects
1 VIII. Numerical and Programming Techniques			
1	403		Root Finding Summary
1	404 738	Schwarze	Cubic and Quartic Roots
1	408 787	Schneider	A Bezier Curve-Based Root-Finder
1	416 743	Hook,McAree	Using Sturm Sequences to Bracket Roots of Polynomials
1	423		Distance Measures Summary
1	424 756	Lalonde,Dawson	A High-Speed, Low Precision Square Root
1	427 758	Paeth	A Fast Approximation to the Hypotenuse
1	432	Ritter	A Fast Approximation to 3D Euclidean Distance
1	434	Thompson	Full-Precision Constants
1	435	Thompson	Converting Between Bits and Digits
1	436	Wyvill	Storage-free Swapping

1	438	Glassner	Generating Random Integers
1	440	Ritter	Fast 2D-3D Rotation
1	442	Shoemake	Bit Patterns for Encoding Angles
1	443	759 Shaffer	Bit Interleaving for Quad- or Octrees
1	448	763 Fishkin	A Fast HSL-to-RGB Transform
IX. Matrix Techniques			
1	453	Thompson	Matrix Identities
1	455		Rotation Matrix Methods Summary
1	456	Thompson	Transforming Axes
1	460	Thompson	Fast Matrix Multiplication
1	462	Hultquist	A Virtual Trackball
1	464	765 Raible	Matrix Orthogonalization
1	465	Pique	Rotation Tools
1	470	766 Carling	Matrix Inversion
1	472	Goldman	Matrices and Transformations
1	476	770 Cychosz	Efficient Post-Concatenation of Transformation Matrices
X. Modeling and Transformations			
1	485	Greene	Transformation Identities
1	494	773 Turkowski	Fixed-Point Trigonometry with CORDIC Iterations
1	498	775 Maillot	Using Quaternions for Coding 3D Transformations
1	516	778 Cunningham	3D Viewing and Rotation Using Orthonormal Bases
1	522	Turkowski	The Use of Coordinate Frames in Computer Graphics
1	533	780 Wallis	Forms, Vectors, and Transforms
1	539	Turkowski	Properties of Surface-Normal Transformations
1	548	785 Arvo	Transforming Axis-Aligned Bounding Boxes
1	551		Constructing Shapes Summary
1	552	Hall	Defining Surfaces from Sampled Data
1	558	Hall	Defining Surfaces from Contour Data
1	562	Glassner	Computing Surface Normals for 3D Models
1	567	Bloomenthal	Calculation of Reference Frames along a Space Curve
XI. Curves and Surfaces			
1	575	Glassner	Planar Cubic Curves
1	579	Rasala	Explicit Cubic Spline Interpolation Formulas
1	585	Gomez	Fast Spline Drawing
1	587	Goldman	Some Properties of Bezier Curves
1	594	Wallis	Tutorial on Forward Differencing
1	604	Goldman	Integration of Bernstein Basis Functions
1	607	787 Schneider	Solving the Nearest-Point-on-Curve Problem
1	612	797 Schneider	An Algorithm for Automatically Fitting Digitized Curves
Appendix I: C Utilities			
1	629		Graphics Gems C Header File
1	633		2D and 3D Vector C Library
1	643		Memory Allocation in C
1	644		Two Useful C Macros
1	645		How to Build Circular Structures in C
1	646		How to Use C Register Variables to Point to 2D Arrays
Appendix II: C Implementations			
1	647		code starts
1	808		References
1	822		Index
1	833		last page

=====  
Graphics Gems II (James Arvo, Editor)  
=====

2	Foreword	
2	Preface	
2	Mathematical Notation	
2	Pseudo-Code	
2	Contributors	
2	I. 2D Geometry and Algorithms	
2	3	Introduction
2	5	Rokne The Area of a Simple Polygon
2	7 473	Prasad Intersection of Line Segments
2	10	Morrison Distance from a Point to a Line
2	14	Rokne An Easy Bounding Circle
2	17	Rokne The Smallest Circle Containing Two Circles
2	19	Rokne Appolonius' 10th Problem
2	25 477	Musgrave A Peano Curve Generation Algorithm
2	26 485	Voorhies Space-Filling Curves and a Measure of Coherence
2	31 487	Steinhart Scanline Coherent Shape Algebra
2	II. Image Processing	
2	49	Schumacher Image Smoothing and Sharpening by Discrete Convolution
2	57 502	Schumacher A Comparison of Digital Halftoning Techniques
2	72 509	Thomas,Bogart Color Dithering
2	78	Schumacher Fast Anamorphic Image Scaling
2	80	Ward Real Pixels
2	84 514	Yap A Fast 90-Degree Bitmap Rotator
2	86 516	Holt Rotation of Run-Length Encoded Image Data
2	89	Glassner Adaptive Run-Length Encoding
2	93	Paeth Image File Compression Made Easy
2	101	Max An Optimal Filter for Image Reconstruction
2	105	Schlag Noise Thresholding in Edge Images
2	107 525	Bieri,Kohler Area, Circumference, and Genus of a Binary Digital Image
2	III. Frame Buffer Techniques	
2	115	Introduction
2	116 528	Thomas Efficient Inverse Color Map Computation
2	126	X.Wu Statistical Computations for Optimal Color Quantization
2	134 536	Musgrave A Random Color Map Animation Algorithm
2	138	Hall,Lindgren A Fast Approach to PHIGS PLUS Pseudo Color
2	143	Paeth Mapping RGB Triples onto 16 Distinct Values
2	147 542	Martindale,Paeth Television Color Encoding and "Hot" Colors
2	159	Meyer An Inexpensive Method of Setting the Monitor White Point
2	163	Musgrave Some Tips for Making Color Hardcopy
2	IV. 3D Geometry and Algorithms	
2	169	Introduction
2	170	Goldman Area of Planar Polygons and Volume of Polyhedra
2	172	Shaffer Getting Around on a Sphere
2	174	Paeth Exact Dihedral Metrics for Common Polyhedra
2	179	Glassner A Simple Viewing Geometry
2	181 550	Bogart View Correlation
2	191	Glassner Maintaining Winged-Edge Models
2	202	Montani,Scopigno Quadtree/Octree-to-Boundary Conversion
2	219 563	Maillot 3-D Homogeneous Clipping of Triangle Strips
2	232 571	Thalmann,Thalmann,Minh InterPhong Shading
2	V. Ray Tracing	
2	245	Introduction
2	247 575	Haines Fast Ray-Convex Polyhedron Intersection
2	251 577	Cychosz Intersecting a Ray with an Elliptical Torus
2	257	Voorhies,Kirk Ray-Triangle Intersection Using Binary Subdiv.

2	264	Kirk,Arvo	Improved Ray Tagging for Voxel-Based Ray Tracing
2	267	Haines	Efficient Hierarchy Traversal in Ray Tracing
2	273	581 Pearce	A Recursive Shadow Voxel Cache for Ray Tracing
2	275	Pearce	Avoiding Incorrect Shadow Intersections for Ray Tracing
2	277	Lee,Uselton	A Body Color Model: Absorption through Translucent Media
2	283	Lee,Uselton	More Shadow Attenuation for Ray Tracing Translucent Objs

## 2 VI. Radiosity

2	293		Introduction
2	295	583 Chen	Progressive Radiosity with Provided Polygon Display Routines
2	299	Beran-Koehn,Pavicic	A Cubic Tetrahedral Hemi-Cube Algorithm
2	303	598 Tampieri	Fast Vertex Radiosity Update
2	306	Shirley	Radiosity via Ray Tracing
2	311	Sillion	Shadow Boundaries for Adaptive Meshing in Radiosity

## 2 VII. Matrix Techniques

2	319		Introduction
2	320	599 Thomas	Decomposing a Matrix into Simple Transformations
2	324	Goldman	Recovering the Data from the Transformation Matrix
2	332	Goldman	Transformations as Exponentials
2	338	Goldman	Matrices and Transforms: Shear and Pseudo-Perspective
2	342	603 K.Wu	Fast Matrix Inversion
2	351	Shoemake	Quaternions and 4x4 Matrices
2	355	606 Arvo	Random Rotation Matrices
2	357	608 Arvo	Classifying Small Sparse Matrices

## 2 VIII. Numerical and Programming Techniques

2	365		Introduction
2	366	Shoemake	Bit Picking
2	368	Shoemake	Faster Fourier Transform
2	371	610 Paeth,Schilling	Of Integers, Fields, and Bit Counting
2	377	Schlag	Geometric ... Interpolate Orientation with Quaternions
2	381	Paeth	The Joys of the Halved Tangent
2	387	612 Musial	An Integer Square Root Algorithm
2	389	Capelli	Fast Approximation to the Arctangent
2	392	613 Ritter	Fast Sign of Cross Product Calculation
2	394	Shoemake	Interval Sampling
2	396	615 Ward	A Recursive Implementation of the Perlin Noise Function

## 2 IX. Curves and Surfaces

2	405		Introduction
2	406	Moore,Warren	Least-Squares Approx. to Bezier Curves and Surfaces
2	412	Shoemake	Beyond Bezier Curves
2	417	Schlag	Curve Interpolation with Variable Control Point Approx.
2	420	Lindgren	Symmetric Evaluation of Polynomials
2	424	Seidel	Menelaus's Theorem
2	428	Seidel	Geometrically Continuous Cubic Bezier Curves
2	435	617 Musial	A Good Straight-Line Approximation of a Circular Arc
2	440	Paeth	Great Circle Plotting
2	446	X.Wu	Fast Anti-Aliased Circle Generation

## 2 Appendix I: C Utilities

2	455		Graphics Gems C Header File
2	458		2D and 3D Vector C Library -- Corrected and Indexed
2	467	Hollasch	Useful C Macros for Vector Operations

## 2 Appendix II: C Implementations

2	473		code starts
---	-----	--	-------------

2	621		References
2	635		Index



2 643 last page

=====  
Graphics Gems III (David Kirk, Editor)  
=====

3 Foreword  
3 Preface  
3 Mathematical Notation  
3 Pseudo-Code  
3 Contributors

3 I. Image Processing  
3 4 411 Moller fast bitmap stretching  
3 8 414 Schumacher general filtered image rescaling  
3 17 425 Schumacher optimization of bitmap scaling operations  
3 20 429 Bragg a simple color reduction filter  
3 23 Moore,Warren compact isocontours from sampled data  
3 29 432 Feldman generating iso-value contours from a pixmap  
3 34 Salesin,Barzel compositing black and white bitmaps  
3 36 Scofield 2-1/2-d depth of field simulation for computer animation  
3 39 441 Furman a fast boundary generator for composited regions

3 II. Numerical & Programming Techniques  
3 47 Introduction  
3 48 446 Hill IEEE fast square root  
3 49 448 Hill A simple fast memory allocator  
3 51 452 Hanson the rolling ball  
3 61 454 Rokne interval arithmetic  
3 67 458 Paeth fast generation of cyclic sequences  
3 77 Paeth a generic pixel selection mechanism  
3 80 Shirley nonuniform random point sets  
3 84 Goldman cross product in 4-dimensions and beyond  
3 89 460 Badouel,Wuthrich face connected line segment generation in n-d

3 III. Modeling and Transformations  
3 95 Introduction  
3 96 461 Morrison quaternion interpolation with extra spins  
3 98 Goldman decomposing projective transformations  
3 108 Goldman decomposing linear and affine transformations  
3 117 463 Arvo fast random rotation matrices  
3 121 Dana issues and techniques for keyframing transformations  
3 124 465 Shoemake uniform random rotations  
3 133 468 Elber interpolation using bezier curves  
3 137 472 Barr physically based superquadrics

3 IV. 2D Geometry and Algorithms  
3 163 Introduction  
3 164 478 VanAken,Simar a parametric elliptical arc algorithm  
3 173 480 Rosati a simple connection algorithm for 2d drawing  
3 182 487 Srinivasan a fast circle clipping algorithm  
3 188 491 Shaffer,Feustel exact computation of 2d intersections  
3 193 496 Miller joining two lines with a circular arc fillet  
3 199 500 Antonio faster line segment intersection  
3 203 Sevcici the problem of apollonius and other related problems

3 V. 3D Geometry and Algorithms  
3 213 Introduction  
3 215 Lopez-Lopez triangles revisited  
3 219 502 Chin partitioning a 3d convex polygon with an arbitrary plane  
3 223 511 Georgiades signed distance from point to plane

3	225	512	Salesin,Tampieri	grouping coplanar polygons into coplanar sets
3	231	517	Tampieri	newell's method for the plane equation of a polygon
3	233	519	Georgiades	plane to plane intersection
3	236	521	Voorhies	triangle-cube intersection
3	240	527	Wanger,Fusco	fast n-dimensional extent overlap testing
3	244	534	Moore	subdividing simplices
3	250		Moore	understanding simploids
3	256	536	Lischinski	converting bezier triangles into rectangular patches
3	262		Lindgren,Sanchez,Hall	curve tessellation criteria thru sampling
3	VI. Ray Tracing & Radiosity			
3	269			Introduction
3	271	538	Sung,Shirley	ray tracing with the BSP tree
3	275	547	Cychosz,Waggenpack	intersecting a ray with a quadric surface
3	284		Cychosz	... eliminate ray-object intersection calculations
3	288	551	Musgrave	a panoramic virtual screen for ray tracing
3	295	555	Trumbore	rectangular bounding volumes for popular primitives
3	301		X.Wu	a linear time simple bounding volume algorithm
3	307	562	Wang	correct direct lighting for distribution ray tracing
3	314	569	Bian	Hemispherical Projection of a triangle
3	318		Max,Allison	linear rad approx using vertex-to-vertex form factors
3	324	575	Beran-Koehn,Pavicic	delta form factor for cubic tetrahedral alg.
3	329	577	Tampieri	accurate form factor computation
3	VII. Rendering			
3	337			Introduction
3	338	582	Woo	the shadow depth map revisited
3	343	583	Cheng	fast linear color rendering
3	349	586	Cheng	edge and bitmask calculations for antialiasing
3	355	594	Grace	fast span conversion: unrolling short loops
3	358	597	Hollasch	progressive image refinement via gridded sampling
3	362	599	Fleischer,Salesin	poly. scan conversion using half-open intervals
3	366		Glassner	darklights
3	369		Glassner	anti-aliasing in triangular pixels
3	374	606	Snyder,Barzel,Gabriel	motion blur on graphics workstations
3	383		Arvo,Scofield	shader cache: a rendering pipeline accelerator
3	C Utilities			
3	393		Glassner	Graphics Gems C Header File
3	396		Glassner	2-D and 3-D Vector C Library -- Corrected and Indexed
3	405		Hollasch	Useful C Macros for Vector Operations
3	Appendix: C Implementations			
3	411			code starts
3	611			References
3	625			Index
3	631			last page

=====  
Graphics Gems IV (Paul Heckbert, Editor)  
=====

Below is the table of contents for "Graphics Gems IV". This table also serves as an index to the code in the FTP collection. Note that every article has text that appears in the book but not in the FTP archive, and some articles contain no C or C++ code.

file or	book	chapter title and author
directory	chapter	

-----

	I	POLYGONS AND POLYHEDRA
centroid.c	I.1	Centroid of a Polygon Gerard Bashein and Paul R. Detmer
convex_test/	I.2	Testing the Convexity of a Polygon Peter Schorn and Frederick Fisher
ptpoly_weiler/	I.3	An Incremental Angle Point in Polygon Test Kevin Weiler
ptpoly_haines/	I.4	Point in Polygon Strategies Eric Haines
delaunay/	I.5	Incremental Delaunay Triangulation Dani Lischinski
vert_norm/	I.6	Building Vertex Normals from an Unstructured Polygon List Andrew Glassner
	I.7	Detecting Intersection of a Rectangular Solid and a Convex Polyhedron Ned Greene
collide.c	I.8	Fast Collision Detection of Moving Convex Polyhedra Rich Rabbitz
-----		
	II	GEOMETRY
	II.1	Distance to an Ellipsoid John C. Hart
dist_fast.c	II.2	Fast Linear Approximations of Euclidean Distance in Higher Dimensions Yoshikazu Ohashi
outcode/	II.3	Direct Outcode Calculation for Faster Clip Testing Walt Donovan and Tim Van Hook
sph_poly.c	II.4	Computing the Area of a Spherical Polygon Robert D. Miller
	II.5	The Pleasures of 'Perp Dot' Products F. S. Hill, Jr.
	II.6	Geometry for N-Dimensional Graphics Andrew J. Hanson
-----		
	III	TRANSFORMATIONS
arcball/	III.1	Arcball Rotation Control Ken Shoemake
	III.2	Efficient Eigenvalues for Visualization Robert L. Cromwell
inv_fast.c	III.3	Fast Inversion of Length- and Angle-Preserving Matrices

Kevin Wu

polar_decomp/	III.4	Polar Matrix Decomposition Ken Shoemake
euler_angle/	III.5	Euler Angle Conversion Ken Shoemake
	III.6	Fiber Bundle Twist Reduction Ken Shoemake
-----		
	IV	CURVES AND SURFACES
data_smooth/	IV.1	Smoothing and Interpolation with Finite Differences Paul H. C. Eilers
	IV.2	Knot Insertion using Forward Differences Phillip Barry and Ron Goldman
	IV.3	Converting a Rational Curve to a Standard Rational Bernstein-Bezier Representation Chandrajit Bajaj and Guoliang Xu
curve_isect/	IV.4	Intersecting Parametric Cubic Curves by Midpoint Subdivision R. Victor Klassen
patch_conv.C	IV.5	Converting Rectangular Patches into Bezier Triangles Dani Lischinski
nurb_polyg/	IV.6	Tessellation of NURB Surfaces John W. Peterson
	IV.7	Equations of Cylinders and Cones Ching-Kuang Shene
implicit.c	IV.8	An Implicit Surface Polygonizer Jules Bloomenthal
-----		
	V	RAY TRACING
	V.1	Computing the Intersection of a Line and a Cylinder Ching-Kuang Shene
ray_cyl.c	V.2	Intersecting a Ray with a Cylinder Joseph M. Cychosz and Warren N. Waggenspack, Jr.
vox_traverse.c	V.3	Voxel Traversal along a 3D Line Daniel Cohen
multi_jitter/	V.4	Multi-Jittered Sampling Kenneth Chiu, Peter Shirley, and Changyaw Wang
minray/	V.5	A Minimal Ray Tracer Paul S. Heckbert
-----		
	VI	SHADING

VI.1	A Fast Alternative to Phong's Specular Model Christophe Schlick
VI.2	R.E versus N.H Specular Highlights Frederick Fisher and Andrew Woo
VI.3	Fast Alternatives to Perlin's Bias and Gain Functions Christophe Schlick
VI.4	Fence Shading Uwe Behrens

---

VII	FRAME BUFFER TECHNIQUES
VII.1	XOR-Drawing with Guaranteed Contrast Manfred Kopp and Michael Gervautz
VII.2	A Contrast-Based Scalefactor for Luminance Display Greg Ward
dyn_range/ VII.3	High Dynamic Range Pixels Christophe Schlick

---

VIII	IMAGE PROCESSING
------	------------------

emboss.c VIII.1	Fast Embossing Effects on Raster Image Data John Schlag
coons_warp.c VIII.2	Bilinear Coons Patch Image Warping Paul S. Heckbert
convolve.c VIII.3	Fast Convolution with Packed Lookup Tables George Wolberg and Henry Massalin
thin_image.c VIII.4	Efficient Binary Image Thinning using Neighborhood Maps Joseph M. Cychosz
clahe.c VIII.5	Contrast Limited Adaptive Histogram Equalization Karel Zuiderveld
mrsfoley.im VIII.6	Ideal Tiles for Shading and Halftoning Alan Wm Paeth

---

IX	GRAPHIC DESIGN
----	----------------

IX.1	Placing Text Labels on Maps and Diagrams Jon Christensen, Joe Marks, and Stuart Shieber
graph_layout/ IX.2	Dynamic Layout Algorithm to Display General Graphs Laszlo Szirmay-Kalos

---

X	UTILITIES
---	-----------

trilerp.c X.1	Tri-linear Interpolation Steve Hill
interp_fast.c X.2	Faster Linear Interpolation

Steven Eker

vec\_mat/            X.3        C++ Vector and Matrix Algebra Routines  
                                Jean-Francois Doue

GraphicsGems.c    X.4        C Header File and Vector Library  
GraphicsGems.h                Andrew Glassner and Eric Haines

=====  
Graphics Gems V (Alan Paeth, Editor)  
=====

Pg	Author(s)	Title
--	(Glassner)	(Forward)
--	(Paeth)	(Preface/Afterword)
1	I.	Algebra and Arithmetic
3	Herbison-Evans	Solving Quartics and Cubics for Graphics
16	Turkowski	Computing the Inverse Square Root
22	Turkowski	Fixed Point Square Root
25	Shoemake	Rational Approximation
33	II.	Computational Geometry
35	Van Gelder	Efficient Computation of Polygon Area and Polyhedron Volume
42	Carvalho/Cavalcanti	Point in Polyhedron Testing using Spherical Polygons
50	Glassner	Clipping a Concave Polygon
55	Hanson	Rotations for N-dimensional Graphics
65	Buckley	Parallelhedra and Uniform Color Quantization
72	Hill	Matrix-based Ellipse Geometry
78	Paeth	Distance Approximations and Bounding Polyhedra
89	III.	Modeling and Transformation
91	Alciatore/Miranda	The Best Least-Squares Line Fit
98	Hill/Roberts	Surface Models and the Resolution of n-Dim. Cell Ambiguity
107	Arata	Tri-cubic Interpolation
111	Miller	Transforming Coordinates from One Coordinate Plane to Another
121	Chin	A Walk Through BSP Trees
139	Blanc	Generic Implementation of Axial Deformation Techniques
147	IV.	Curves and Surfaces
149	Goldman	Identities for the Univariate, Bivariate Bernstein Basis Fcns
163	Goldman	Identities for the B-Spline Basis Functions
168	Turkowski	Circular Arc Subdivision
173	de Figueiredo	Adaptive Sampling of Parametric Curves
179	Ahn	Fast Generation of Ellipsoids
191	Bajaj/Xu	Sparse Smooth Connection between Bezier/B-Spline Curves
199	Gravesen	The Length of Bezier Curves
206	Miller	Quick and Simple Bezier Curve Drawing
210	Shoemake	Linear Form Curves
225	V.	Ray Tracing and Radiosity
227	Shene	Computing the Intersection of a Line and a Cone
232	Schlick/Subrenat	Ray Intersection of Tessellated Surfaces: Quad vs Triangle
242	Moller	Faster Raytracing using Scanline Rejection
258	Leipelt	Ray Tracing a Swept Sphere
268	Marton	Acceleration of Ray Tracing via Voronoi-diagrams
285	Zimmerman	Direct Lighting Models for Ray Tracing with Cylindrical Lamps
290	Feda	Improving Intermediate Radiosity using Directional Light
295	VI.	Halftoning and Image Processing

297	Purgathofer/Tobler/Geiler	Improved Threshold Matrices for Ordered Dithering
302	Wong/Hsu	Halftoning with Selective Precipitation & Adaptive Clustering
314	Eker	Faster Pixel-Perfect Line Clipping
323	Doue/Rubio	Efficient and Robust 2D Shape Vectorization
338	Hsu/Lee	Reversible Straight Line Edge Reconstruction
355	Sharma	Priority-based Adaptive Image Refinement
359	Cross	Sampling Patterns Optimized for Uniform Distribution of Edges

#### 365 VII. Utilities

367	Schlick	Wave Generators for Computer Graphics
375	Green/Hatch	Fast Polygon-Cube Intersection Testing
380	Bouma/Vanecek	Velocity-based Collision Detection
386	Vanecek	Spatial Partitioning of a Polygon by a Plane
394	Narkhede/Manocha	Fast Polygon Triangulation based on Seidel's Algorithm
398	Karinthi	Accurate Z-buffer Rendering
400	Paeth (et al)	A Survey of Graphics Libraries

Errata to \_Graphics Gems V\_, first edition, edited by Alan Paeth  
(awpaeth@okanagan.bc.ca), Academic Press 1995. Code available online in  
<http://www.graphicsgems.org/>

compiled by Eric Haines (erich@acm.org) from author and reader contributions

version 1.8  
date: 1/3/01

-----

Errors in the text:

The following proof changes might not appear in the book's 1st printing but  
are correct on the floppy disk and FTP mirror versions:

p. 85, bottom (code line) now reads:

```
...      if ((t = a - b) < 0) {a -= t; b += t; } }
                        ^   ^           ^   ^
('a','b' and '+','-' were transposed)
```

p. 86, top (code):

```
... + 16*d)/ ...
    ^^ replaces the ' 4' presently there
```

p. 153: formulas (ix) a and b are correct, but they would be better if they  
were written as:

(a)  $B_k^n(t) = \sum (-1)^{j-k} \text{Binomial}(j,k) \text{Binomial}(n,j) t^j$

(b) Similarly, replace  $\text{Binomial}(n,i,j) \text{Binomial}(n-i-j, k-i, l-j)$  by  
 $\text{Binomial}(k,i) \text{Binomial}(l,j) \text{Binomial}(n,k,l)$

p. 323: no cedilla in "Francois" in author's name (cp. p 405, bottom)

p. 327: Figure 5b has an expansion of 3 vertical lines. In 5a these are 5  
pixels high, in Figure 5b they incorrectly expand to 21 pixels high;  
these should be 20 pixels high.

p. 394: Atul Narkhede's email address is now atul@yamuna.asd.sgi.com

-----

The following are errors in the book's code listings (corrected in the online  
code at <http://www.graphicsgems.org/>). Note that some of the  
code listings online are different in minor and major ways from the code in  
the book.

Serious errors (ones your compiler cannot or may not catch):

ch1-4/rat.c - page 29, line 42, change

```
carry = t3&0xFFFF; lohi = (t3<<16)&0xFFFF;
to
carry = (t3>>16)&0xFFFF; lohi = t3&0xFFFF;
```

ch1-4/rat.c - page 31, line 11, change



```
ck = ck<<n + num/ak1;  
to  
ck = (ck<<n) + (num/ak1);
```

ch3-6/axd.\* - these files have been reworked to correctly match the macro library that they use. See the online version.

ch4-8/qbezier.c - line 76, change "if (k = 0)" to "if (k == 0)".

ch6-4/chainCode.C - line 70, add "int trueLength = strlen(code);" and change all "length" to "trueLength" in postProcess(). The old code loops through the whole allocated string instead of just the part with data.

ch7-1/wave.c - remove the definition "double a" from each of the routines Rwave, Twave, and Swave. The variable "a" is passed in and so should not be defined here.

ch7-5/misc.c - if you do not have the log2() function in your compiler, use:  
#define log2(x) (log((double)x)/log(2.0))

-----

Syntax errors (ones that are not usually harmful, or are easily caught):

There are various "lint" type errors in the text's and diskette's code which have been cleaned up in the FTP distribution. The only serious changes were to the axd.c code in ch3-6, as the code was out of sync with the macros it used from ch7-7/mactbox. The corrected code is in the FTP distribution.

-----

The following are typographical errors in the comments:

[none so far]

-----

END

=====  
Graphics Gems V (Alan Paeth, Editor)  
Table of Contents  
(ver 1.0; March 6, 1995)  
=====

Pg	Author(s)	Title
--	(Glassner)	(Forward)
--	(Paeth)	(Preface/Afterword)
1	I.	Algebra and Arithmetic
3	Herbison-Evans	Solving Quartics and Cubics for Graphics
16	Turkowski	Computing the Inverse Square Root
22	Turkowski	Fixed Point Square Root
25	Shoemake	Rational Approximation
33	II.	Computational Geometry
35	Van Gelder	Efficient Computation of Polygon Area and Polyhedron Volume
42	Carvalho/Cavalcanti	Point in Polyhedron Testing using Spherical Polygons
50	Glassner	Clipping a Concave Polygon
55	Hanson	Rotations for N-dimensional Graphics
65	Buckley	Parallelhedra and Uniform Color Quantization
72	Hill	Matrix-based Ellipse Geometry
78	Paeth	Distance Approximations and Bounding Polyhedra
89	III.	Modeling and Transformation
91	Alciatore/Miranda	The Best Least-Squares Line Fit
98	Hill/Roberts	Surface Models and the Resolution of n-Dim. Cell Ambiguity
107	Arata	Tri-cubic Interpolation
111	Miller	Transforming Coordinates from One Coordinate Plane to Another
121	Chin	A Walk Through BSP Trees
139	Blanc	Generic Implementation of Axial Deformation Techniques
147	IV.	Curves and Surfaces
149	Goldman	Identities for the Univariate, Bivariate Bernstein Basis Fcns
163	Goldman	Identities for the B-Spline Basis Functions
168	Turkowski	Circular Arc Subdivision
173	de Figueiredo	Adaptive Sampling of Parametric Curves
179	Ahn	Fast Generation of Ellipsoids
191	Bajaj/Xu	Sparse Smooth Connection between Bezier/B-Spline Curves
199	Gravesen	The Length of Bezier Curves
206	Miller	Quick and Simple Bezier Curve Drawing
210	Shoemake	Linear Form Curves
225	V.	Ray Tracing and Radiosity
227	Shene	Computing the Intersection of a Line and a Cone
232	Schlick/Subrenat	Ray Intersection of Tessellated Surfaces: Quad vs Triangle
242	Moller	Faster Raytracing using Scanline Rejection
258	Leipelt	Ray Tracing a Swept Sphere
268	Marton	Acceleration of Ray Tracing via Voronoi-diagrams
285	Zimmerman	Direct Lighting Models for Ray Tracing with Cylindrical Lamps
290	Feda	Improving Intermediate Radiosity using Directional Light
295	VI.	Halftoning and Image Processing
297	Purgathofer/Tobler/Geiler	Improved Threshold Matrices for Ordered Dithering
302	Wong/Hsu	Halftoning with Selective Precipitation & Adaptive Clustering
314	Eker	Faster Pixel-Perfect Line Clipping
323	Doue/Rubio	Efficient and Robust 2D Shape Vectorization
338	Hsu/Lee	Reversible Straight Line Edge Reconstruction

355	Sharma	Priority-based Adaptive Image Refinement
359	Cross	Sampling Patterns Optimized for Uniform Distribution of Edges
365	VII. Utilities	
367	Schlick	Wave Generators for Computer Graphics
375	Green/Hatch	Fast Polygon-Cube Intersection Testing
380	Bouma/Vanecek	Velocity-based Collision Detection
386	Vanecek	Spatial Partitioning of a Polygon by a Plane
394	Narkhede/Manocha	Fast Polygon Triangulation based on Seidel's Algorithm
398	Karinthi	Accurate Z-buffer Rendering
400	Paeth (et al)	A Survey of Graphics Libraries

Errata to \_Graphics Gems II\_, first edition, edited by Jim Arvo  
(arvo@cs.caltech.edu), Academic Press 1991. Code available online at  
<http://www.graphicsgems.org/>

compiled by Eric Haines (erich@acm.org) from author and reader contributions

version 1.13  
date: 1/3/01

-----

Errors in the text:

- p. xxix, middle of page: change "Rod G. Bogart (72)" to  
"Rod G. Bogart (72, 181)".
- p. 84, line 1: change "clockwise" to "counterclockwise".
- p. 85, line 1: change "counterclockwise" to "clockwise".
- p. 194, line 4: change "nil <- 0" to "int <- 0".
- p. 194, figure 3: change "a2.aF" to "e2.aF" and "a2.bF" to "e2.bF".
- p. 249, last sentence: change "less than 0, then" to "less than tnear, then".
- p. 340, Shear matrix: change " $-(Q.v)w$ " to " $-\tan(\phi)(Q.v)w$ ".
- p. 420, line 10: "Berstein" should read "Bernstein".

-----

The following are errors in the code listings (corrected in the online code at  
<http://www.graphicsgems.org/>). Note that many of the code listings online are  
different in minor and major ways from the code in the book (see Addenda, at  
the end, for new code not in the book).

Serious errors (ones your compiler cannot or may not catch):

- p. 456: Delete FLOOR and CEILING macros (they're more like truncations).  
Change ROUND macro to (i.e. add parentheses around "a"):  

```
#define ROUND(a) ((a)>0 ? (int)((a)+0.5) : -(int)(0.5-(a)))
```

  
Change SGN macro to (i.e. change positive condition result to "1"):  

```
#define SGN(a) ((a)<0 ? -1 : 1)
```
- p. 463, line 9: in V3Combine change last "result->y" to "result->z".
- p. 571, line 16: add missing macros:  

```
#define Trunc(v) ((int)floor(v))
```

```
/* normalize vector, return in pn */  
#define unity(v,pn) { double len ; \  
    len = sqrt((v).x*(v).x+(v).y*(v).y+(v).z*(v).z) ; \  
    (pn)->x=(v).x/len; (pn)->y=(v).y/len; \  
    (pn)->z=(v).z/len; }
```
- p. 572, line 27: change "struct epthbuf\_el {" to "struct depthbuf\_el {"
- p. 574, 4th line from bottom: delete "break;" statement in order to move to  
next pair of edges
- p. 576, line 28: change "if ( t < 0.0 )" to "if ( t < tnear )"

- p. 591, line 7: change `"*fp == 1.0;"` to `"*fp = 1.0;"`
- p. 599, bottom: add lines:
- ```
typedef struct {
    double x,y,z,w;
} Vector4;

Vector4 *V4MulPointByMatrix();
```
- p. 600, line 6: add lines:
- ```
#include <math.h>
#include "GraphicsGems.h"
#include "unmatrix.h"
```
- p. 600, line 22: change to `"Matrix4 pmat, invpmat, tinvpmat;"`
- p. 600, line 25: change to `"Point3 row[3], pdum3;"`
- p. 600, line 29: change
- ```
if ( pmat.element[3][3] != 0 )
to:
    if ( locmat.element[3][3] == 0 )
        return 0;
```
- p. 600, line 39: change `"pmat.element[4][3] = 1;"` to `"pmat.element[3][3] = 1;"`
- p. 601, line 4: change:
- ```
psol = *V4MulPointByMatrix(&prhs, &invpmat, &vdum4);
to:
    Transpose( &invpmat, &tinvpmat );
    V4MulPointByMatrix(&prhs, &tinvpmat, &psol);
```
- p. 602, line 5: change `"&row[2])"` to `"&row[2], &pdum3)"`
- p. 602, bottom: add the subroutines:
- ```
/* transpose rotation portion of matrix a, return b */
Matrix4 *TransposeMatrix4(a, b)
Matrix4 *a, *b;
{
    int i, j;
    for (i=0; i<4; i++)
        for (j=0; j<4; j++)
            b->element[i][j] = a->element[j][i];
    return(b);
}

/* multiply a hom. point by a matrix and return the transformed point */
Vector4 *V4MulPointByMatrix(pin, m, pout)
Vector4 *pin, *pout;
Matrix4 *m;
{
    pout->x = (pin->x * m->element[0][0]) + (pin->y * m->element[1][0]) +
              (pin->z * m->element[2][0]) + (pin->w * m->element[3][0]);
    pout->y = (pin->x * m->element[0][1]) + (pin->y * m->element[1][1]) +
              (pin->z * m->element[2][1]) + (pin->w * m->element[3][1]);
    pout->z = (pin->x * m->element[0][2]) + (pin->y * m->element[1][2]) +
              (pin->z * m->element[2][2]) + (pin->w * m->element[3][2]);
    pout->w = (pin->x * m->element[0][3]) + (pin->y * m->element[1][3]) +
              (pin->z * m->element[2][3]) + (pin->w * m->element[3][3]);
    return(pout);
}
```
- p. 604, line 7 from bottom: add code to check for singular matrix (near 0 determinant). Add code:
- ```
/* Is the submatrix A singular? */
if ((det_1 == 0.0) || (ABS(det_1 / (pos - neg)) < PRECISION_LIMIT)) {
```

```
    /* Matrix M has no inverse */  
    fprintf (stderr, "affine_matrix4_inverse: singular matrix\n");  
    return FALSE;  
}
```

-----

Syntax errors (ones your compiler or lint will catch):

- p. 458-466, throughout: replace "};" with "}" to make lint happy
- p. 464, gcd(): the variable "k" is set but not used - remove it
  
- p. 477, line 11 and p. 478, lines 15,18: change "coord" to "gcoord" and  
"last\_coord" to "glast\_coord" to avoid same name use for  
global and local variables
- p. 477, line 20: delete ", pos=0", since "pos" is not used in code.
- p. 483, v\_convert code: change "alpha" to "alph", since "alpha" is a global  
defined in "types.h"
  
- p. 543, 547, 548: comment out text after "#else" and "#endif" statements  
(some compilers don't like this)
  
- p. 550, line 21: change "int i, num\_iterations = 0;" to  
"int num\_iterations = 0 ;", as "i" is not used
- p. 551, 2nd line from bottom: delete line "Matrix3 \*vm;", as "vm" is not used
- p. 559, line 14: delete "int i;" declaration; "i" not used
  
- p. 573, line 15: delete "static" declaration from shadepoly
  
- p. 581 throughout: code is mostly for illustrative purposes; the RAY\_REC,  
LIGHT\_REC, TRIANGLE\_REC, cache, object, and voxel structures need  
definition. Add lines:  
    int i, hit ;  
    float shadow\_percent ;
  
- p. 588, 7th line from bottom: change "(j+0.5)" to "(dj+0.5)", or delete  
"dj = (double)j;" from previous line (dj currently is not used)  
2nd line from bottom: change "(k+0.5)" to "(dk+0.5)", or delete  
"dk = (double)k;" from previous line (dk currently is not used)
- p. 594, middle: the "const" declarations before the TSpectra definitions  
create assignment incompatibilities; replace with "static".
- p. 596, first line of InitParams: change "int i, n=0;" to "int i;", since  
"n" is not used.

-----

The following are typographical errors in the comments:

- p. 461, line 19: change "transpose matrix a," to "transpose rotation portion  
of matrix a,"
  
- p. 473, line 28: change "condititions:" to "conditions:"
  
- p. 514, line 2: change "clockwise" to "counterclockwise"
  
- p. 576, line 30: change "hit near face," to "hit far face,"
  
- p. 578, line 21: change "coefficents" to "coefficients"
  
- p. 584, line 12: change "nubmer" to "number"

p. 588, line 5: change "array" to "array"  
p. 590, line 10: change "radom" to "random"  
p. 595, line 24: change "quarilateral" to "quadrilateral"  
  
p. 610, line 2: change "varients" to "variants"

-----

#### Addenda:

Not include in the gem "Intersecting a Ray with an Elliptical Torus" is the following code which compensates for the order of the roots being transposed. It was not included since it would make the gem dependent on the behaviour of the root solver. Insert after the call to SolveQuartic (middle of p. 579):

```
/*      SolveQuartic returns root pairs in reversed order.      */  
if ( *nhits > 1 )  
    m = rhits[0]; u = rhits[1]; rhits[0] = u; rhits[1] = m;  
if ( *nhits > 3 )  
    m = rhits[2]; u = rhits[3]; rhits[2] = u; rhits[3] = m;
```

Also, note that the torus intersector may not handle degenerate surfaces (e.g. where the major radius < minor radius, or major radius < 0) as desired.

Xiaolin Wu's code for "Efficient Statistical Computations for Optimal Color Quantization" is available online in quantizer.c, though not in the book. The code originally distributed has been modified slightly, with the global variable "m2" renamed to "gm2" to avoid confusion with the local "m2" in some routines. The Var() function has had a fix to it, the absolute value of the result is now returned.

Code for Greg Ward's "Real Pixels" article is not in the book, but is included in the RealPixels subdirectory of the online code.

The code provided in the online distribution for Greg Ward's "A Recursive Implementation of the Perlin Noise Function" (noise3.c) has a variety of functions available (e.g. fractal noise), more than the code in the text.

Included in the distribution code is c\_format, a PostScript typesetting program for C source code (v0.6) by Dale Schumacher.

There is a tester provided for Alan Paeth and David Schilling's "Of Integers, Fields, and Bit Counting".

```
# Makefile for Graphics Gems II source
#
# Eric Haines, 9/92
#
# This make file will build "gemslib2.a" and a number of executables.
# Gemslib2 is built solely for debugging purposes -- it is not intended
# to be used as a library.
#
# Note that some of the gems need additional macros, functions, tables
# driving routines, etc. before they will compile or run properly.
# These include:
#
# Hilbert assumes GL is available
#
# hot needs write_pixel and read_pixel routines
#
# ran_ramp needs fb_init, fb_setmap and fb_done routines
#
# VoxelCache needs RAY_REC, LIGHT_REC, TRIANGLE_REC, cache, object, and voxel
#   structures. Code is mostly for illustrative purposes.
#
# radiosity code needs a fleshed out draw.c function
#
#
# C compiler flags
#
CFLAGS = -g
#
# Location of Graphics Gems library
#
LIBFILE = gemslib2.a
#
# Graphics Gems II Vector Library
#
VECLIB = GGVecLib.o

MFLAGS = "LIBFILE = ../$(LIBFILE)" "GENCFLAGS = $(CFLAGS)"

SHELL = /bin/sh

OFILES = FastUpdate.o GGVecLib.o InterPhong.o RayCPhdron.o \
  c_format.o hot.o inverse.o noise3.o quantizer.o \
  ran_ramp.o rotate.o rotate8x8.o sparse.o unmatrix.o xlines.o \
  Hilbert.o VoxelCache.o

DIRS = BitCounting Peano RealPixels dither intersect inv_cmap radiosity \
  viewcorr

ALL = c_format quantizer xlines hot ran_ramp Hilbert $(LIBFILE)

all: $(ALL)
    @for d in $(DIRS) ; do \
        (cd $$d ; $(MAKE) $(MFLAGS)) ;\
    done

$(LIBFILE): $(OFILES) $(VECLIB)
    ar rcs $(LIBFILE) $(OFILES) $(VECLIB)

Hilbert: Hilbert.o
    $(CC) $(CFLAGS) -o $@ Hilbert.o
```



```
c_format: c_format.o
        $(CC) $(CFLAGS) -o $@ c_format.o

hot:     hot.o
        $(CC) $(CFLAGS) -o $@ hot.o -lm

quantizer: quantizer.o
        $(CC) $(CFLAGS) -o $@ quantizer.o

ran_ramp: ran_ramp.o
        $(CC) $(CFLAGS) -o $@ ran_ramp.o

xlines: xlines.o
        $(CC) $(CFLAGS) -o $@ xlines.o

clean:

    @for d in $(DIRS) ; do \
        (cd $$d ; $(MAKE) $(MFLAGS) clean) ;\
    done
/bin/rm -f $(OFILES) $(VECLIB)
/bin/rm -f FastUpdate.o GGVecLib.o Hilbert.o InterPhong.o \
    RayCPhdron.o VoxelCache.o c_format.o hot.o inverse.o noise3.o \
    quantizer.o ran_ramp.o rotate.o rotate8x8.o sparse.o \
    unmatrix.o xlines.o \
    Hilbert c_format hot quantizer ran_ramp xlines \
    a.out core $(LIBFILE)

$(ALL): GraphicsGems.h
```

C Code From Graphics Gems II, Academic Press, Inc.

=====

This is a list of the files and directories containing the C code for Graphics Gems II. They are listed in order of their appearance in the book. Beside each file name is the Gem number, author's name, and Gem title to help you find what you're looking for. Unfortunately, not all of the code listed in the appendix of the book is here (for various reasons). The missing ones will be added as they turn up. Changes will be listed at the bottom of this file.

File or Directory -----	Gem Number -----	Author and Title of Gem -----
xlines.c	I.2	Mukesh Prasad, "Intersection of Line Segments"
Peano/ Makefile main.c mapply.c peano.c types.h	I.7	Ken Musgrave, "A Peano Curve Generation Algorithm"
Hilbert.c Coherence"	I.8	Douglas Voorhies, "Space-Filling Curves and a Measure of
dither/ dither.3 dither.c	II.3	Spencer W. Thomas and Rod G. Bogart, "Color Dithering"
RealPixels/ color.c color.h colrops.c header.c rasterfile.h ra_pr24.c resolu.c	II.5	Greg Ward, "Real Pixels"
rotate8x8.c	II.6	Sue-Ken Yap, "A fast 90-Degree Bitmap Rotator"
inv_cmap/ inv_cmap.3 inv_cmap.c	III.1	Spencer W. Thomas, "Efficient Inverse Color Map Computation"
quantizer.c Color Quantization"	III.2	Xiaolin Wu, "Efficient Statistical Computations for Optimal
ran_ramp.c	III.3	Ken Musgrave, "A Random Color Map Animation Algorithm"
hot.c and 'Hot' Broadcast Colors"	III.6	Dave Martindale and Alan Paeth, "Television Color Encoding

viewcorr/ matrix.c matrix.h viewcorr.c viewcorr.h viewfind.c	IV.5	Rod G. Bogart, "View Correlation"
InterPhong.c Shading"	IV.9	Nadia Magnenat Thalmann and Daniel Thanmann, "InterPhong
RayCPhdron.c	V.1	Eric Haines, "Fast Ray-Convex Polyhedron Intersection"
intersect/ Torus" inttor.c intsph.c	V.2	Joseph M. Cychosz, "Intersecting a Ray with an Elliptical
VoxelCache.c	V.6	Andrew Pearce, "A Recursive Voxel Cache for Ray Tracing"
radiosity/ README draw.c rad.c rad.h room.c	VI.1	Eric Chen, "Implementing Progressive Radiosity with User-Provided Polygon Display Routines"
FastUpdate.c	VI.3	Filippo Tampieri, "Fast Vertex Radiosity Update"
inverse.c	VII.5	Kevin Wu, "Fast Matrix Inversion"
rotate.c	VII.7	James Arvo, "Random Rotation Matrices"
sparse.c	VII.8	James Arvo, "Classifying Small Sparse Matrices"
BitCounting/ Bit Counting" bit32.c test.c	VIII.3	Alan W. Paeth and David Schilling, "Of Integers, Fields, and
noise3.c Function"	VIII.10	Greg Ward, "A Recursive Implementation of the Perlin Noise

#### Additional header files and utilities

-----

GraphicsGems.h	Andrew Glassner
GGVecLib.c	Andrew Glassner
vector.h	Steve Hollasch, "Useful C Macros for Vector Operations"
c_format.c code"	Dale Schumacher, "A PostScript typesetting program for C source

#### Changes

-----

10/02/91 Added ran\_ramp.c, rotate8x8.c, inverse.c, VoxelCache.c, and Peano  
directory.  
10/09/91 Added dither and inv\_cmap directories.  
11/05/91 Updated quantizer.c and sparse.c  
5/26/93 Errata changes [by Eric Haines]  
5/26/93 Added unmatrix.c VII.1 Spencer Thomas, "Decomposing a Matrix into  
Simple Transformations"  
8/25/93 Errata changes (SGN, ROUND, FLOOR, CEILING) [by Eric Haines]

From  
uupsi!psinntp!rpi!zaphod.mps.ohio-state.edu!swrinde!cs.utexas.edu!asuvax!ennews!enuxha.eas.asu.edu!hollasch  
Fri Aug 30 12:17:10 EDT 1991  
Path:  
eye!uupsi!psinntp!rpi!zaphod.mps.ohio-state.edu!swrinde!cs.utexas.edu!asuvax!ennews!enuxha.eas.asu.edu!hollasch  
>From: hollasch@enuxha.eas.asu.edu (Steve Hollasch)  
Newsgroups: comp.graphics  
Subject: Re: contents Graphics Gems II  
Summary: Here it is  
Message-ID: <1991Aug28.203807.23624@ennews.eas.asu.edu>  
Date: 28 Aug 91 20:38:07 GMT  
References: <1991Aug28.160122.9841@cv.ruu.nl>  
Sender: usenet@ennews.eas.asu.edu (Usenet)  
Reply-To: hollasch@enuxha.eas.asu.edu (Steve Hollasch)  
Followup-To: comp.graphics  
Organization: Arizona State University  
Lines: 243  
Nntp-Posting-Host: enuxha.eas.asu.edu

koen@cv.ruu.nl (Koen Vincken) writes:  
|Could someone post (or mail me) the contents of Graphics Gems II?  
|The book is still not available here, and I'm curious about the subjects  
|contained in Gems II (just titles should be sufficient).

Well, I figured that a lot of people would probably be interested in  
this, so I went ahead and typed it in. You folks had BETTER appreciate  
this! =^)

By the way, the dates for Graphics Gems III are as follows:

01-Oct-91: Submit first draft of gem and C code.  
01-Nov-91: Notification of acceptance and need for revision.  
01-Dec-91: Submit final manuscript, figures and C code.  
Mar-92: Review page proofs.  
24-Jul-92: Publication  
27-Jul-92: Celebration at SIGGRAPH '92.

For further information (and an author's packet), you can send  
e-mail to Jenifer Swetland at cdp!jswetland@labrea.stanford.edu.

The table of contents for Graphics Gems II follows my signature.

---

Steve Hollasch / Arizona State University (Tempe, Arizona)  
hollasch@enuxha.eas.asu.edu / uunet!mimsy!oddjob!noao!asuvax!enuxha!hollasch

Graphics Gems II  
edited by  
James Arvo

Copyright (c) 1991 by Academic Press, Inc.  
ISBN 0-12-064480-0

T A B L E O F C O N T E N T S

-----

Foreword By Andrew Glassner  
Preface  
Mathematical Notation  
Pseudo-Code  
Contributors

-----

I 2D GEOMETRY AND ALGORITHMS

Introduction

1. The Area of a Simple Polygon
2. Intersection of Line Segments
3. Distance from a Point to a Line
4. An Easy Bounding Circle

5. The Smallest Circle Containing the Intersection of Two Circles
6. Apollonius's 10th Problem
7. A Peano Curve Generation Algorithm
8. Space-Filling Curves and a Measure of Coherence
9. Scanline Coherent Shape Algebra

-----

## II IMAGE PROCESSING

### Introduction

1. Image Smoothing and Sharpening by Discrete Convolution
2. A Comparison of Digital Halftoning Techniques
3. Color Dithering
4. Fast Anamorphic Image Scaling
5. Real Pixels
6. A Fast 90-Degree Bitmap Rotator
7. Rotation of Run-Length Encoded Image Data
8. Adaptive Run-Length Encoding
9. Image File Compression Made Easy
10. An Optimal Filter for Image Reconstruction
11. Noise Thresholding in Edge Images
12. Computing the Area, the Circumference, and the Genus of a Binary Digital Image

-----

## III FRAME BUFFER TECHNIQUES

1. Efficient Inverse Color Map Computation
2. Efficient Statistical Computations for Optimal Color Quantization
3. A Random Color Map Animation Algorithm
4. A Fast Approach to PHIGS PLUS Pseudo Color Mapping
5. Mapping RGB Triples onto 16 Distinct Values
6. Television Color Encoding and "Hot" Broadcast Colors
7. An Inexpensive Method of Setting the Monitor White Point
8. Some Tips for Making Color Hardcopy

-----

## IV 3D GEOMETRY AND ALGORITHMS

### Introduction

1. Area of Planar Polygons and Volume of Polyhedra
2. Getting Around on a Sphere
3. Exact Dihedral Metrics for Common Polyhedra
4. A Simple Viewing Geometry
5. View Correlation
6. Maintaining Winged-Edge Models
7. Quadtree/Octree-to-Boundary Conversion
8. Three-Dimensional Homogeneous Clipping of Triangle Strips
9. InterPhong Shading

-----

## V RAY TRACING

### Introduction

1. Fast Ray-Convex Polyhedron Intersection
2. Intersecting a Ray with an Elliptical Torus
3. Ray-Triangle Intersection Using Binary Recursive Subdivision
4. Improved Ray Tagging for Voxel-Based Ray Tracing
5. Efficiency Improvements for Hierarchy Traversal in Ray Tracing
6. A Recursive Shadow Voxel Cache for Ray Tracing
7. Avoiding Incorrect Shadow Intersections for Ray Tracing
8. A Body Color Model: Absorption of Light through Translucent Media
9. More Shadow Attenuation in Ray Tracing Transparent or Translucent Objects

-----

## VI      RADIOSITY

### Introduction

1. Implementing Progressive Radiosity with User-Provided Polygon Display Routines
2. A Cubic Tetrahedral Adaptation of the Hemi-Cube Algorithm
3. Fast Vertex Radiosity Update
4. Radiosity via Ray Tracing
5. Detection of Shadow Boundaries for Adaptive Meshing in Radiosity

-----

## VII     MATRIX TECHNIQUES

### Introduction

1. Decomposing a Matrix into Simple Transformations
2. Recovering the Data from the Transformation Matrix
3. Transformations as Exponentials
4. More Matrices and Transformations: Shear and Pseudo-Perspective
5. Fast Matrix Inversion
6. Quaternions and 4x4 Matrices
7. Random Rotation Matrices
8. Classifying Small Sparse Matrices

-----

## VIII    NUMERICAL AND PROGRAMMING TECHNIQUES

### Introduction

1. Bit Picking
2. Faster Fourier Transform
3. Of Integers, Fields and Bit Counting
4. Using Geometric Constructions to Interpolate Orientation with Quaternions
5. A Half-Angle Identity for Digital Computation: The Joys of the Halved Tangent
6. An Integer Square Root Algorithm
7. Fast Approximation to the Arctangent
8. Fast Sign of Cross Product Calculation
9. Interval Sampling
10. A Recursive Implementation of the Perlin Noise Function

-----

## IX      CURVES AND SURFACES

### Introduction

1. Least-Squares Approximation to Bezier Curves and Surfaces
2. Beyond Bezier Curves
3. A Simple Formulation for Curve Interpolation with Variable Control Point Approximation
4. Symmetric Evaluation of Polynomials
5. Menelaus's Theorem
6. Geometrically Continuous Cubic Bezier Curves
7. A Good Straight-Line Approximation of a Circular Arc
8. Great Circle Plotting
9. Fast Anti-Aliased Circle Generation

-----

## C      APPENDIX I: C UTILITIES

Graphics Gems C Header File  
2D and 3D Vector C Library -- Corrected and Indexed  
Useful C Macros for Vector Operations

-----

## C      APPENDIX II: C IMPLEMENTATIONS

I.2	Intersection of Line Segments
I.7	A Peano Curve Generation Algorithm
I.8	Space-Filling Curves and a Measure of Coherence
I.9	Scanline Coherent Shape Algebra
II.2	A Comparison of Digital Halftoning Techniques
II.3	Color Dithering
II.6	A Fast 90-Degree Bitmap Rotator
II.7	Rotation of Run-Length Encoded Image Data
II.12	Computing the Area, the Circumference, and the Genus of a Binary Digital Image
III.1	Efficient Inverse Color Map Computation
III.3	A Random Color Map Animation Algorithm
III.6	Television Color Encoding and "Hot" Broadcast Colors
IV.5	View Correlation
IV.8	Three-Dimensional Homogeneous Clipping of Triangle Strips
IV.9	InterPhong Shading
V.1	Fast Ray-Convex Polyhedron Intersection
V.2	Intersecting a Ray with an Elliptical Torus
V.6	A Recursive Shadow Voxel Cache for Ray Tracing
VI.1	Implementing Progressive Radiosity with User-Provided Polygon Display Routines
VI.3	Fast Vertex Radiosity Update
VII.1	Decomposing a Matrix into Simple Transformations
VII.5	Fast Matrix Inversion
VII.7	Random Rotation Matrices
VII.8	Classifying Small Sparse Matrices
VIII.3	Of Integers, Fields and Bit Counting
VIII.6	An Integer Square Root Algorithm
VIII.8	Fast Sign of Cross Product Calculation
VIII.10	A Recursive Implementation of the Perlin Noise Function
IX.7	A Good Straight-Line Approximation of a Circular Arc

-----

References

Index



```

/*****
c_format -- a PostScript typesetting program for C source code (v0.6)

Released into the public domain, November 1990 by Dale Schumacher
email: <dal@syntel.mn.org> mail: 399 Beacon Ave., St. Paul MN 55104
*****/
```

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
```

```
static char psPreamble[] = "\
%!PS-Adobe-2.0\n\
/inch { 72 mul } def\n\
/pica { 12 mul } def\n\
/xOrg 114 def\n\
/xSize 32 pica def\n\
/yOrg 684 def\n\
/ySize 44 pica def\n\
/yDelta 9 def\n\
/Page 1 def\n\
/Line 1 def\n\
/FS {    % find and scale a font\n\
        findfont exch scalefont\n\
} bind def\n\
/Fonts [    % create an array of prescaled fonts\n\
        8 /Courier FS\n\
        8 /Courier-Bold FS\n\
        8 /Courier-Oblique FS\n\
] def\n\
/SL {    % set line position\n\
        /Line exch def\n\
        xOrg yOrg Line yDelta mul sub moveto\n\
} bind def\n\
/NL {    % newline\n\
        Line 1 add SL\n\
} bind def\n\
/R {    % use roman font\n\
        Fonts 0 get setfont\n\
} bind def\n\
/B {    % use bold font\n\
        Fonts 1 get setfont\n\
} bind def\n\
/I {    % use italic font\n\
        Fonts 2 get setfont\n\
} bind def\n\
/S {    % show\n\
        show\n\
} bind def\n\
";
```

```
char psFinal[] = "\
/BP {    % begin page\n\
        1 SL R\n\
} bind def\n\
/EP {    % end page\n\
        gsave\n\
            newpath\n\
                xOrg xSize add 0 moveto\n\
                0 11 inch rlineto\n\
                8.5 inch 11 inch lineto\n\
            showpage\n\
        endg\n\
} bind def\n\
";
```

```
                8.5 inch 0 lineto\n\
        closepath 1 setgray fill          % mask right margin\n\
    grestore\n\
    showpage\n\
} bind def\n\
";

char psDraft[] = "\
/DraftFont 72 /Helvetica-Bold FS def\n\
/BP {    % begin page\n\
    gsave\n\
        newpath\n\
        184 452 moveto\n\
        -30 rotate\n\
        DraftFont setfont\n\
        gsave \n\
            0.82 setgray (DRAFT) show\n\
        grestore \n\
        (DRAFT) false charpath 0 setlinewidth stroke\n\
    grestore\n\
    1 SL R\n\
} bind def\n\
/EP {    % end page\n\
    gsave\n\
        newpath\n\
            xOrg xSize add 0 moveto\n\
            0 11 inch rlineto\n\
            8.5 inch 11 inch lineto\n\
            8.5 inch 0 lineto\n\
        closepath 1 setgray fill          % mask right margin\n\
        0 setgray\n\
        newpath\n\
            xOrg yOrg moveto\n\
            xSize 0 rlineto\n\
            0 ySize neg rlineto\n\
            xSize neg 0 rlineto\n\
        closepath 0 setlinewidth stroke % nominal text box\n\
    grestore\n\
    showpage\n\
} bind def\n\
";

#define LPP      (60)          /* maximum allowable lines per page */

char title[256] = "THE TITLE OF YOUR GEM";
char author[256] = "YOUR NAME (as it appears on your gem)";
int draft = 0;      /* draft output mode switch */
int tabsize = 4;    /* horizontal tab spacing */
int page = 1;       /* current page of listing (one based) */
int line = 1;       /* current line on page (one based) */
int firstline = 18; /* first line of first page (one based) */

char *
ps_string(s)
char *s;
{
    static char ps_buf[512];
    char *p = ps_buf;
    int c;

    while(c = *s++) {
```

```
        if((c == '(') || (c == ')') || (c == '\\')) {
            *p++ = '\\';
        }
        if(isprint(c)) {
            *p++ = c;
        }
    }
    *p = '\\0';
    return(ps_buf);
}

void
start_listing(f)
FILE *f;
{
    line = firstline;
    fprintf(f, "%s%s\n", psPreamble, (draft ? psDraft : psFinal));
    fprintf(f, "%s start of listing\n");
    fprintf(f, "initgraphics\nBP\n");
    fprintf(f, "(%s) S NL\n", ps_string(title));
    fprintf(f, "(%s) S\n", ps_string(author));
    fprintf(f, "%d SL\n", firstline);
}

void
page_eject(f)
FILE *f;
{
    ++page;
    line = 1;
    fprintf(f, "EP\nBP\n");
}

void
new_line(f)
FILE *f;
{
    ++line;
    if(line > LPP) {
        page_eject(f);
    } else {
        fprintf(f, "NL\n");
    }
}

void
end_listing(f)
FILE *f;
{
    fprintf(f, "EP\n\n%% end of listing\n");
}

#define TK_EMPTY        (-1)
#define TK_TEXT          (0)
#define TK_WORD          (1)
#define TK_KEYWORD       (2)
#define TK_CSTART        (3)
#define TK_CEMPTY        (4)
#define TK_CTEXT         (5)
#define TK_CEND          (6)
#define TK_SEMPTY        (7)
```

```
#define TK_STEXT      (8)
#define TK_NEWLINE    (9)
#define TK_FORMFEED   (10)

static char *keywords[] = {
    "auto",           "break",           "case",           "char",
    "const",          "continue",      "default",        "do",
    "double",         "else",          "entry",          "enum",
    "extern",         "float",         "for",            "goto",
    "if",             "int",           "long",           "register",
    "return",         "short",         "signed",         "sizeof",
    "static",         "struct",        "switch",         "typedef",
    "union",          "unsigned",      "void",           "volatile",
    "while",
    NULL
};

};

int
iskeyword(s)
char *s;
{
    char **p;

    for(p = keywords; *p; ++p) {
        if(**p == *s && (strcmp((*p)+1, s+1) == 0)) {
            return(1);
        }
    }
    return(0);
}

static int spaces = 0;
static int pbchar = 0;
static int pbflag = 0;

int
push_back(f, c)
FILE *f;
int c;
{
    pbflag = 1;
    return(pbchar = c);
}

int
next_char(f)
FILE *f;
{
    static int col = 0;
    int c;

    if(pbflag) {
        pbflag = 0;
        return(pbchar);
    } else if(spaces > 0) {
        --spaces;
        ++col;
        return(' ');
    }
    do {
        c =getc(f);
    } while(c == ' ');
}
```

```
        if((c == EOF) || (c == '\\f') || (c == '\\n')) {
            if(c == '\\n') {
                col = 0;
            }
            return(c);
        } else if(c == '\\t') {
            spaces = (tabsize - (col % tabsize)) - 1;
            c = ' ';
            break;
        }
    } while(iscntrl(c));
    ++col;
    return(c);
}
```

```
char    token[256];
```

```
int
next_token(f)
FILE *f;
{
    static int state = TK_EMPTY;
    static int qchar = '\"';
    int c;
    char *tp;

    tp = token;
    while((c = next_char(f)) != EOF) {
        switch(state) {

        case TK_EMPTY:
            switch(c) {
                case '/':
                    c = next_char(f);
                    if(c == EOF) {
                        *tp++ = '/';
                        *tp = '\\0';
                        return(TK_TEXT);
                    } else if(c == '*') {
                        *tp++ = '/';
                        state = TK_CSTART;
                    } else {
                        *tp++ = '/';
                        state = TK_TEXT;
                    }
                break;
                case '\\n':
                    *tp++ = '\\n';
                    *tp = '\\0';
                    return(TK_NEWLINE);
                case '\\f':
                    *tp++ = '\\f';
                    *tp = '\\0';
                    return(TK_FORMFEED);
                case '\\':
                case '\"':
                    qchar = c;
                    state = TK_STEXT;
                    break;
                default:
                    state = (isalpha(c) || (c=='_'))

```

```

                                ? TK_WORD
                                : TK_TEXT;
                                break;
        }
        break;

case TK_TEXT:
    if(isalpha(c)
        || (c=='_' ) || (c=='/' )
        || (c=='"' ) || (c=='\\' ) || iscntrl(c)) {
        push_back(f, c);
        *tp = '\\0';
        state = TK_EMPTY;
        return(TK_TEXT);
    }
    break;

case TK_WORD:
    if(!(isalnum(c) || (c=='_' )
        || (c=='/' ) || (c=='"' ) || (c=='\\' )
        || iscntrl(c)) {
        push_back(f, c);
        *tp = '\\0';
        state = TK_EMPTY;
        return(iskeyword(token)
            ? TK_KWORD
            : TK_WORD);
    }
    break;

case TK_CSTART:
    if(c!='*') {
        push_back(f, c);
        *tp = '\\0';
        state = (c == '/') ? TK_CEND : TK_CEMPTY;
        return(TK_CSTART);
    }
    break;

case TK_CEMPTY:
    switch(c) {
        case '\\n':
            *tp++ = '\\n';
            *tp = '\\0';
            return(TK_NEWLINE);
        case '\\f':
            *tp++ = '\\f';
            *tp = '\\0';
            return(TK_FORMFEED);
        case '*':
            state = TK_CEND;
            break;
        default:
            state = TK_CTEXT;
            break;
    }
    break;

case TK_CTEXT:
    if((c=='*') || iscntrl(c)) {
        push_back(f, c);

```

```
        *tp = '\\0';
        state = (c == '*') ? TK_CEND : TK_CEMPTY;
        return(TK_CTEXT);
    }
    break;

case TK_CEND:
    if(c == '/') {
        *tp++ = '/';
        *tp = '\\0';
        state = TK_EMPTY;
        return(TK_CEND);
    } else if(c != '*') {
        push_back(f, c);
        *tp = '\\0';
        state = TK_CTEXT;
        return(TK_CTEXT);
    }
    break;

case TK_SEMPTY:
    switch(c) {
        case '\\n':
            *tp++ = '\\n';
            *tp = '\\0';
            return(TK_NEWLINE);
        case '\\f':
            *tp++ = '\\f';
            *tp = '\\0';
            return(TK_FORMFEED);
        default:
            if(c == qchar) {
                *tp++ = c;
                *tp = '\\0';
                state = TK_EMPTY;
                return(TK_STEXT);
            }
            state = TK_STEXT;
            break;
    }
    break;

case TK_STEXT:
    if(c == qchar) {
        *tp++ = c;
        *tp = '\\0';
        state = TK_EMPTY;
        return(TK_STEXT);
    } else if(c == '\\\\') {
        *tp++ = '\\\\';
        c = next_char(f);
        if(c == EOF) {
            *tp = '\\0';
            state = TK_EMPTY;
            return(TK_STEXT);
        }
    }
    if(iscntrl(c)) {
        push_back(f, c);
        *tp = '\\0';
        state = TK_SEMPTY;
    }
}
```

```
        return(TK_STEXT);
    }
    break;

}
*tp++ = c;
}
*tp = '\\0';
return(TK_EMPTY);
}

#define ROMAN_FONT      (0)
#define BOLD_FONT       (1)
#define ITALIC_FONT     (2)

void
list(fi, fo)
FILE *fi;
FILE *fo;
{
    int tk, ptk;
    int font = ROMAN_FONT;

    start_listing(fo);
    ptk = TK_EMPTY;
    while((tk = next_token(fi)) != TK_EMPTY) {
        switch(tk) {
            case TK_TEXT:
            case TK_WORD:
            case TK_STEXT:
                if(font != ROMAN_FONT) {
                    fprintf(fo, "R ");
                    font = ROMAN_FONT;
                }
                fprintf(fo, "(%s) S\\n", ps_string(token));
                break;
            case TK_KWORD:
                if(font != BOLD_FONT) {
                    fprintf(fo, "B ");
                    font = BOLD_FONT;
                }
                fprintf(fo, "(%s) S\\n", ps_string(token));
                break;
            case TK_CSTART:
                if(font != ROMAN_FONT) {
                    fprintf(fo, "R ");
                    font = ROMAN_FONT;
                }
                fprintf(fo, "(%s) S\\n", ps_string(token));
                break;
            case TK_CTEXT:
                if(font != ITALIC_FONT) {
                    fprintf(fo, "I ");
                    font = ITALIC_FONT;
                }
                fprintf(fo, "(%s) S\\n", ps_string(token));
                break;
            case TK_CEND:
                font = ROMAN_FONT;
                fprintf(fo, "R (%s) S\\n", ps_string(token));
                break;
        }
    }
}
```



```
        case TK_NEWLINE:
            if(((ptk == TK_NEWLINE) || (ptk == TK_FORMFEED))
                && (line < 2)) {
                break; /* ignore if at top of page */
            }
            if((ptk != TK_NEWLINE) || (line < (LPP - 6))) {
                new_line(fo);
                break;
            }
            /* FALL THRU */
        case TK_FORMFEED:
            if(line < 2) {
                break; /* ignore if at top of page */
            }
            page_eject(fo);
            break;
    }
    ptk = tk;
}
end_listing(fo);
}





void
usage()
{
    fprintf(stderr, "\
usage: c_format [-options] [-] [file.c]\n\
options:\n\
    -d                draft output mode\n\
    -t title          set listing title\n\
    -a author         set listing author\n\
    -x tabsize        set tab stops\n\
    -o output.ps      write to output file (default to stdout)\n\
");
    exit(1);
}

main(argc, argv)
int argc;
char **argv;
{
    FILE *f;
    register int c;
    register char *p;
    extern int optind;
    extern char *optarg;

    while((c = getopt(argc, argv, "dt:a:x:o:f:")) != EOF) {
        switch(c) {
            case 'd':
                draft = !draft;
                break;
            case 't':
                strcpy(title, optarg);
                break;
            case 'a':
                strcpy(author, optarg);
                break;
            case 'x':
                tabsize = atoi(optarg);
                break;
        }
    }
}
```

```
        case 'o':
            if(freopen(optarg, "w", stdout) == NULL) {
                perror(optarg);
                exit(1);
            }
            break;
        case 'f':
            c = atoi(optarg);
            if((c > 0) && (c < LPP)) {
                firstline = c;
            }
            break;
        case '?':
        default:
            usage();
    }
}
c = argc - optind;
if(c < 1) {
    if(isatty(fileno(stdin))) {
        usage();
    } else {
        list(stdin, stdout);
    }
} else if(c == 1) {
    p = argv[optind];
    if((p[0] == '-') && (p[1] == '\\0')) {
        list(stdin, stdout);
    } else {
        if(f = fopen(p, "r")) {
            list(f, stdout);
            fclose(f);
        } else {
            perror(p);
            exit(1);
        }
    }
} else {
    usage();
}
exit(0);
}
```

# Index of /pubs/tog/GraphicsGems/gemsii/intersect/

Name	Last modified	Size	Description
 <a href="#">_ Parent Directory</a>			
 <a href="#">_ Makefile</a>	29-Jun-00 08:14	1K	
 <a href="#">_ intsph.c</a>	29-Jun-00 08:14	1K	
 <a href="#">_ inttor.c</a>	29-Jun-00 08:14	3K	

```
/*
 * unmatrix.h - Definitions for using unmatrix
 *
 * Author:      Spencer W. Thomas
 *              University of Michigan
 */

/* The unmatrix subroutine fills in a vector of floating point
 * values.  These symbols make it easier to get the data back out.
 */

enum unmatrix_indices {
    U_SCALEX,
    U_SCALEY,
    U_SCALEZ,
    U_SHEARXY,
    U_SHEARXZ,
    U_SHEARYZ,
    U_ROTATEX,
    U_ROTATEY,
    U_ROTATEZ,
    U_TRANSX,
    U_TRANSY,
    U_TRANSZ,
    U_PERSPX,
    U_PERSPY,
    U_PERSPZ,
    U_PERSPW
};

typedef struct {
    double x,y,z,w;
} Vector4;

Matrix4 *TransposeMatrix4();
Vector4 *V4MulPointByMatrix();
```

Errata to \_Graphics Gems III\_, first edition, edited by David Kirk  
(dk@egg.gg.caltech.edu), Academic Press 1992. Code available online in  
<http://www.acm.org/tog/GraphicsGems>

compiled by Eric Haines (erich@acm.org) from author and reader contributions

version 1.14

date: 6/4/2000

-----

Errors in the text:

p. 67, equation 2.2 at bottom: should read "a := c - a;"

p. 201, just before "Timing Measurements" section: "corresponds to collinear line segments" should read "corresponds to parallel or collinear line segments".

p. 204, middle of page: the y-component of the center should be negative, so that it matches the equation two lines further down.

p. 234, final two equations need minus signs in the denominators:

$$P[u] = \frac{M[v]*Jd - N[v]*Id}{M[u]*N[v] - M[v]*N[u]}$$

$$P[v] = \frac{N[u]*Id - M[u]*Jd}{M[u]*N[v] - M[v]*N[u]}$$

p. 285, third paragraph, first sentence: change "with the complement of the residency mask" to "with the residency mask".

-----

The following are errors in the code listings (corrected in the online code at [princeton.edu:pub/Graphics/GraphicsGems/GemsIII](http://princeton.edu:pub/Graphics/GraphicsGems/GemsIII)). Note that many of the code listings online are different in minor and major ways from the code in the book (see Addenda, at the end, for new code not in the book).

Serious errors (ones your compiler cannot or may not catch):

p. 330, Equation (4): put a minus sign just before the 1/2pi term.

p. 394: Delete FLOOR and CEILING macros (they're more like truncations).  
Change ROUND macro to (i.e. add parentheses around "a"):  
#define ROUND(a) ((a)>0 ? (int)((a)+0.5) : -(int)(0.5-(a)))  
Change SGN macro to (i.e. change positive condition result to "1"):  
#define SGN(a) ((a)<0 ? -1 : 1)

p. 401, line 9: in V3Combine change last "result->y" to "result->z".

p. 422, third from last line: change "tmp->xsize" to "dst->ysize".

p. 425-427: bitbuf is used before being set; initialized bitbuf = 0 before use in the three scale\_bitmap\_\* routines.

p. 430, line 8: change "uranx" to "urany" in the "jittery" macro.

p. 438, after line 18, "dir = neighbor(...", add the following test:

```
/**      maybe done with contour      */  
  
if ( (x == start_x) && (y == start_y) && (chain != NULL) )  
    if (dir == chain->dir)  
        return(1);
```

and the code 20 lines later labelled "maybe done with contour"  
should be deleted (3 lines).

p. 474: the variable "rho" is not used, delete it.

p. 477, next to last two lines: call `sq_ellipsoid_tensor` with five "1."s and  
`sq_toroid_tensor` with six "1."s.

p. 498, lines 53-54: put "&" in front of `a1,b1,c1` and `a2,b2,c2`.

p. 499, lines 27-28: put "&" in front of `xa,ya` and `xb,yb`.

p. 500-1: change `COLLINEAR` to `PARALLEL` (i.e. when `f==0`, all we can conclude is  
that the two lines are parallel).

p. 521: Numerical instability can result during point-triangle intersection.

A better `SIGN3` macro is:

```
#define EPS 10e-5  
#define SIGN3( A ) \  
    ((A).x < EPS) ? 4 : 0 | ((A).x > -EPS) ? 32 : 0 | \  
    ((A).y < EPS) ? 2 : 0 | ((A).y > -EPS) ? 16 : 0 | \  
    ((A).z < EPS) ? 1 : 0 | ((A).z > -EPS) ? 8 : 0)
```

and the `point_triangle_intersection` test (p. 526) changes to:

```
return (sign12 & sign23 & sign31 == 0) ? OUTSIDE : INSIDE;
```

p. 525: change calls to `check_point`. The third argument to each call should  
switch `p1` and `p2`. For example, the first correction should be:

```
if (check_point(p1,p2,(.5-p1.x)/(p2.x-p1.x),0x3e) == INSIDE)  
    return(INSIDE);
```

p. 547, beginning: add line `#include <math.h>`

p. 570, line 49: change to `"major = V3New(0.0, 0.0, 1.0);"`

p. 580, line 19: change `"f+= acos(VDOT(s, t)) * VDOT(sxt, Ns);"` to  
`"f-= acos(VDOT(s, t)) * VDOT(sxt, Nr);"`, i.e. "Ns" becomes "Nr".  
"Ns" can be removed from the list of calling parameters for  
`"computeUnoccludedFormFactor()"`.

p. 606, beginning: add `#include <string.h>`

-----

Syntax errors (ones that are not usually harmful, or are easily caught):

p. 396-404, throughout: replace `"};"` with `"}"` to make lint happy

p. 402, `gcd()`: the variable "k" is set but not used - remove it

p. 411, top: for ANSI compatibility, add procedure declarations:

```
void RectStretch(long xs1,long ys1,long xs2,long ys2,long xd1,long yd1,  
    long xd2,long yd2) ;  
void Stretch(long x1,long x2,long y1,long y2,long yr,long yw) ;  
void CircleStretch(long SBMINX,long SBMAXX,long xc,long yc,long r) ;  
void Stretch2Lines(long x1,long x2,long y1,long y2,long yr1,long yw1,
```

long yr2, long yw2) ;

- p. 411-413: cast expressions inside "abs()" calls to type "(int)".
- p. 414-424: replace "ceiling" with "ceil" (on some machines)
- p. 416, lines 18-19: remove "char \*p;" and "int width, height;" declarations; unused.
- p. 420-421: replace "(\*filter)" with "(\*filterf)" to avoid redeclaration of "filter".
- p. 423, line 23: remove "register char \*p;" declaration; unused.
- p. 425, line 33 and p. 426, line 51: remove declaration of "i", as it is unused.
- p. 427, line 29: remove declarations for "\*p" and "\*q"; unused.
- p. 428, line 35: remove declarations for "i" and "j"; unused.
- p. 441, line 37: remove "nx" declaration; unused.
- p. 443, line 2: remove "xl, xr, yt, yb" declarations; unused.
- p. 443, lines 3-4: remove "k" and "ptb" declarations; unused.
- p. 472, line 19: remove "result" declaration; unused.
- p. 474, lines 17-18: remove "iworld" and "R" declarations; unused.
- p. 477, line 32: remove "x" declaration; unused.
- p. 476-477: note that sqellipsoidposn() and sqtoroidposn() are not particularly useful in their current forms; they do not return any values, e.g. x,y,z are computed but not returned.
- p. 489, line 50: cast "intsct" to "(void \*)" to make lint happy.
- p. 496, line 10: remove global "r, p1, p2, p3, p4" declarations.
- p. 496, line 10 and p. 499, throughout: change "v1" to "gv1" and "v1" to "gv2".
- p. 497, line 31: remove "xt" and "yt" declarations; unused.
- p. 498, line 27: remove "t" and "u" declarations; unused.
- p. 500, line 25: remove "temp" declaration; unused.
- p. 500, line 26: remove "x3lo,x3hi" and "y3lo,y3hi" declarations; unused.
- p. 513, line 17: add the declarations for the static routines:  
static void computePlaneEq() ;  
static Node \*insertPlaneEq() ;  
static int comparePlaneEqs() ;
- p. 522, line 5: remove "vect23, vect31" declarations; unused.
- p. 543, line 45: change "malloc" to "(StackPtr)malloc"
- p. 544, line 54: change "malloc" to "(BinNodePtr)malloc"
- p. 546, line 17: change "malloc" to "(BinNodePtr)malloc"
- p. 551-554: Note that this code uses definitions and subroutines that are found in Rayshade. Most are easy enough to understand and code (or use Steve Hollasch's vector macros on pages 405-407).
- p. 559, line 22: remove "tmp" declaration; unused.
- p. 570, line 9: remove "j" declaration; unused.
- p. 572, line 18: remove "a, b, c" and "x1, y1, z1" declarations; unused (i.e. replace line with "double d;".
- p. 573, lines 15 and 28: remove "i" declarations; unused.

p. 577, line 12: add the declarations for the static routines:  
static float computeFormFactor() ;  
static float computeUnoccludedFormFactor() ;  
static void splitQuad() ;  
static float quadArea() ;

p. 579, last line: remove "j" declaration, as "j" is not used.

p. 607, line 40: change to "memset((void \*)acc, ..." to make lint happy.

-----

The following are typographical errors in the comments:

p. 429, line 24: change "variables" to "variables"

p. 431, line 7: change "Cyshosz" to "Cychosz"

p. 443, line 54: change "segment" to "segment"

p. 454, line 3: change "architechture" to "architecture"

p. 458, line 21: change "consistancy" to "consistency"

p. 484, line 46: change "Inplementation" to "Implementation"

p. 493, lines 20 and 23: change "subsegements" to "subsegments"

p. 539, line 18: change "upto" to "up to"

p. 550, line 6: change "parrallel" to "parallel"

p. 556, line 48: change "miminum" to "minimum"

p. 572, line 7: change "homogenous" to "homogeneous"

p. 578, line 15: change "visiblity" to "visibility"

p. 606, line 32: change "initialized" to "initialized"

-----

Addenda:

There is test software (not in the text) for the "Fast n-Dimensional Extent Overlap Testing" (exthit), "Accurate Polygon Scan Conversion Using Half-Open Intervals" (accurate\_scan), and "Partitioning a 3-D Convex Polygon with an Arbitrary Plane" (partition3d).

There is C code for Badouel and Wurthrich's Gem II.9 (ndline.c) in the code distribution (but not in the text).

There is more explanatory C code for Woo's Gem VII.1 in the code distribution.

There is an improved rgbvary.c included in the code distribution for Microsoft Windows named rgbvaryW.c. On PC's this code takes up less space and is 8 times faster. It looks straightforward to regroove VaryDIB24() to other bitmap architectures.

Here is a little additional explanation to the "Use of Residency Masks..."



gem on p. 284-287 from Joe Cychosz (note errata correction above):

The process goes something like this:

```
ray_mask = 0

foreach (cell the ray passes through) {
    foreach (object in the cell) {
        if (ray_mask & object_mask == 0) {
            compute the ray-object intersection
        }
    }
    update ray_mask marking for cell
}
```

The important thing here is to mark the cell after it is processed.

The filter.c file is updated as the filter\_rcg.c by Ray Gardener. This new version of filter.c is faster, and also includes Targa file I/O for testing purposes.

```
# Makefile for Graphics Gems III source
#
# Eric Haines, 10/92
#
# This make file will build "gemslib3.a" and a number of executables.
# Gemslib3 is built solely for debugging purposes -- it is not intended
# to be used as a library.
#
# Some code uses ANSI headers, some doesn't, so you may have to mess with
# CFLAGS, depending.
#
# Note that some of the gems need additional macros, functions, tables
# driving routines, etc. before they will compile or run properly.
# These include:
#
# accurate_scan - the test program uses HP's Starbase graphics API
#
# bsp.c - need routines for FirstOfLinkList, NextOfLinkList, AddToLinkList,
#         and code for RayBoxIntersect, RayObjIntersect
#
# cyclic.c - note that this file is simply a set of macros, no code is compiled
#
# luminaire - need function "hit()"
#
# panorama.c - needs Rayshade include files.
#
# simplex - need function "bitCount()"
#
#
# C compiler flags
#
CFLAGS =
#
# Location of Graphics Gems library
#
LIBFILE = gemslib3.a
#
# Graphics Gems II Vector Library
#
VECLIB = GraphicsGems.o

MFLAGS = "LIBFILE = ../$(LIBFILE)" "GENCFLAGS = $(CFLAGS)"

SHELL = /bin/sh

OFILES = 3d.o PIR.o Polyintr.o accForm.o bitmap.o \
         bounding_volumes.o bsp.o bziprinter.o circlexc.o con2d.o contour.o \
         edgeCalc.o fastBitmap.o fastLinear.o fastSpan.o fillet.o filter.o \
         forfac.o hemis.o insectc.o intell.o intqdr.o motblur.o ndline.o \
         newell.o parelarc.o pl2plane.o planeSets.o pt2plane.o \
         quatspin.o rand_rotation.o rgbvary.o scallops8.o \
         sqfinal.o sqrt.o triangleCube.o urot.o zdepth.o

DIRS = accurate_scan alloc exttest luminaire partition3d simplex

ALL =  contour filter forfac scallops8 sqfinal $(LIBFILE)

all: $(ALL)
    @for d in $(DIRS) ; do \
        (cd $$d ; $(MAKE) $(MFLAGS)) ;\

```

done

```
$(LIBFILE): $(OFILES) $(VECLIB)
    ar rcs $(LIBFILE) $(OFILES) $(VECLIB)
```

```
contour: contour.o
    $(CC) $(CFLAGS) -o $@ contour.o
```

```
filter: filter.o
    $(CC) $(CFLAGS) -o $@ filter.o -lm
```

```
forfac: forfac.o
    $(CC) $(CFLAGS) -o $@ forfac.o -lm
```

```
scallops8: scallops8.o
    $(CC) $(CFLAGS) -o $@ scallops8.o
```

```
sqfinal: sqfinal.o
    $(CC) $(CFLAGS) -o $@ sqfinal.o -lm
```

```
clean:
    @for d in $(DIRS) ; do \
        (cd $$d ; $(MAKE) $(MFLAGS) clean) ;\
    done
    /bin/rm -f $(OFILES) $(VECLIB)
    /bin/rm -f 3d.o PIR.o Polyintr.o accForm.o bitmap.o \
        bounding_volumes.o bsp.o bzipinter.o circlexc.o con2d.o \
        contour.o edgeCalc.o fastBitmap.o fastLinear.o fastSpan.o \
        fillet.o filter.o forfac.o hemis.o insectc.o intell.o \
        intqdr.o motblur.o ndline.o newell.o parelarc.o \
        pl2plane.o planeSets.o pt2plane.o quatspin.o rand_rotation.o \
        rgbvary.o scallops8.o sqfinal.o sqrt.o \
        triangleCube.o urot.o zdepth.o \
        contour filter forfac scallops8 sqfinal \
        a.out core $(LIBFILE)
```

```
$(ALL): GraphicsGems.h
```

C Code From Graphics Gems III, Academic Press, Inc.

=====

This is a list of the files and directories containing the C code for Graphics Gems III.

They are listed in order of their appearance in the book. Beside each file name is the

Gem number, author's name, and Gem title to help you find what you're looking for.

Unfortunately, not all of the code listed in the appendix of the book is here (for various

reasons). The missing ones will be added as they turn up. Changes will be listed at the

bottom of this file.

File or Directory -----	Gem Number -----	Author and Title of Gem -----
-------------------------------	------------------------	----------------------------------

## I. Image Processing

fastBitmap.c comments: ANSI C	I.1	Tomas Moller, "Fast Bitmap Stretching" (doesn't compile)
----------------------------------	-----	--

filter.c	I.2	Dale Schumacher, "General Filtered Image Rescaling"
----------	-----	---

bitmap.c Operations"	I.3	Dale Schumacher, "Optimization of Bitmap Scaling
-------------------------	-----	--

rgbvary.c comments: ANSI C	I.4	Bragg, "A Simple Color Quantization Pre-processor"
-------------------------------	-----	--

contour.c	I.6	Tim Feldman, "Generating Iso-Contours from a Pixmap"
-----------	-----	--

scallops8.c Regions"	I.9	Eric Furman, "A Fast Boundary Generator for Composited
-------------------------	-----	--

## II. Numerical & Programming Techniques

sqrt.c	II.1	Steve Hill, "IEEE Fast Square Root"
--------	------	-------------------------------------

alloc/ alloc.c alloc.h	II.2	Steve Hill, "A Simple Fast Memory Allocator"
------------------------------	------	--

3d.c defs.h	II.3	Steven Hanson, "The Rolling Ball"
----------------	------	-----------------------------------

interval.C comments: C++	II.4	Jon Rokne, "Interval Arithmetic"
-----------------------------	------	----------------------------------

cyclic.c comments: macros only, no procedures	II.5	Alan Paeth, "Fast Generation of Cyclic Sequences"
--	------	---

ndline.c Generation in an N-dimensional Space"	II.9	Badouel / Wuethrich, "Face Connected Line Segment
---	------	---

## III. Modeling and Transformations

quatspin.c	III.1	Jack Morrison, "Quaternion Interpolation with Extra Spin"
rand_rotation.c	III.4	James Arvo, "Fast Random Rotation Matrices"
comments: ANSI C		
urot.c	III.6	Ken Shoemake, "Uniform Random Rotations"
comments: ANSI C		
bzinter.c	III.7	Gershon Elber, "Interpolation using Bezier Curves"
sqfinal.c	III.8	Alan Barr, "Physically Based Superquadrics"

#### IV. 2D Geometry and Algorithms

parelarc.c	IV.1	VanAken / Simar, "A Parametric Elliptical Arc Algorithm"
con2d.c	IV.2	Claudio Rosato, "A Simple Connection Algorithm for 2D Drawing"
circlexc.c	IV.3	Srinivasen, "A Fast Circle Clipping Algorithm"
comments: ANSI C		
Polyintr.c	IV.4	Shaffer, "Exact Computation of Cascaded 2D Intersections"
comments: ANSI C		
fillet.c	IV.5	Robert Miller, "Joining two Lines with a Circular Arc Fillet"
insectc.c	IV.6	Franklin Antonio, "Faster Line Segment Intersection"

#### V. 3D Geometry and Algorithms

partition3d	V.2	Chin, "Partitioning a 3D Convex Polygon with an Arbitrary Plane"
main.c		
partition.c		
partition.h		
pt2plane.c	V.3	Primos Georgiades, "Signed Distance from Point to Plane"
comments: ANSI C		
planeSets.c	V.4	David Salesin, Filippo Tampieri, "Grouping Nearly Coplanar Polygons into Coplanar Sets"
newell.c	V.5	Filippo Tampieri, "Newell's Method for Computing the Plane Equation of a Polygon"
pl2plane.c	V.6	Primos Georgiades, "Plane to Plane Intersection"
comments: ANSI C		
trianglecube.c	V.7	Douglas Voorhies, "Triangle-cube Intersection"
comments: ANSI C		
exttest/	V.8	Len Wanger, Mike Fusco, "Fast N-Dimensional Extent Overlap Testing"
exthit.C		
exthit.h		
ehetest1.C		
comments: C++		

simplex/ recur.C symm.C comments: C++	V.9	Moore, "Subdividing Simplices"
--	-----	--------------------------------

bezierTri.C Rectangular Patches" comments: C++	V.11	Dani Lischinski, "Converting Bezier Triangles into
--	------	--

## VI. Ray Tracing & Radiosity

bsp.c	VI.1	Sung, "Ray Tracing with the BSP Tree"
-------	------	---------------------------------------

intqdr.c Surface" intell.c	VI.2	Joseph Cychosz, "Intersecting a Ray with a Quadric
----------------------------------	------	--

panorama.c Tracing" (doesn't compile)	VI.4	Ken Musgrave, "A Panoramic Virtual Screen for Ray
--	------	---

bounding_volumes.c Primitives"	VI.5	Ben Trumbore, "Rectangular Bounding Volumes for Popular
-----------------------------------	------	---

luminaire/ Distribution Ray Tracing" geometry_object.C geometry_object.h sphere_luminaire.C sphere_luminaire.h triangle_luminaire.C triangle_luminaire.h utility.h special_instruction comments: C++	VI.7	Changyaw Wang, "Physically Correct Direct Lighting for
--	------	--

hemis.c	VI.8	Buming Bian, "Hemisphere Projection of a Triangle"
---------	------	--

forfac.c Cubic Tetrahedral Algorithm"	VI.10	Koehn / Pavicic, "Delta Form Factor Calculation for the
--	-------	---

accForm.c	VI.11	Filippo Tampieri, "Accurate Form Factor Computation"
-----------	-------	--

## VII. Rendering

zdepth.c	VII.1	Andrew Woo, "The Shadow Depth Map Revisited"
----------	-------	--

fastLinear.c	VII.2	Russell Cheng, "Fast Linear Color Rendering"
--------------	-------	--

edgeCalc.c Anti-aliasing"	VII.3	Russell Cheng, "Edge and Bitmask Calculations for
------------------------------	-------	---

fastSpan.c comments: ANSI C	VII.4	Thom Grace, "Fast Span Conversion: unrolling short loops"
--------------------------------	-------	---

PIR.c Sampling"	VII.5	Steve Hollasch, "Progressive Image Refinement via Gridded
--------------------	-------	---

accurate_scan/ "Accurate Polygon scan Conversion using half-open Intervals" makefile dblfixpoint.c	VII.6	Kurt Fleischer
---	-------	----------------

exhaust.c  
fixpoint.c  
fixpoint.h  
test.c  
tri.c

motblur.c                      VII.9      John Snyder, Ronen Barzel, "Motion Blur on Graphics  
Workstations"

## Changes

-----

5/26/93    Errata changes [by Eric Haines]  
5/26/93    Added rgbvaryW.c, integer math version of rgbvary.c  
8/25/93    Errata changes (SGN, ROUND, FLOOR, CEILING) [by Eric Haines]

```
#include <stdio.h>

/*
 * bzrinter.c:
 *   Bezier curve interpolation routines.
 *
 */

#define FloatType float

/*****
The following arrays were created by symbolically solving the Bezier
interpolant problem for orders 2 to 9. Each line holds the rational
weights for the given points to interpolate at node values (i/k, k is
the degree) as the i'th element divided by the zero element in line.
*****/
static long IB2[2][3] =
{
    {      1L,      1L,      0L },
    {      1L,      0L,      1L }
};
static long *InterpBlend2[2] =
{
    IB2[0], IB2[1]
};

static long IB3[3][4] =
{
    {      1L,      1L,      0L,      0L },
    {      2L,     -1L,      4L,     -1L },
    {      1L,      0L,      0L,      1L }
};
static long *InterpBlend3[3] =
{
    IB3[0], IB3[1], IB3[2]
};

static long IB4[4][5] =
{
    {      1L,      1L,      0L,      0L,      0L },
    {      6L,     -5L,     18L,     -9L,      2L },
    {      6L,      2L,     -9L,     18L,     -5L },
    {      1L,      0L,      0L,      0L,      1L }
};
static long *InterpBlend4[4] =
{
    IB4[0], IB4[1], IB4[2], IB4[3]
};

static long IB5[5][6] =
{
    {      1L,      1L,      0L,      0L,      0L,      0L },
    {     12L,    -13L,     48L,    -36L,     16L,    -3L },
    {     18L,     13L,    -64L,    120L,    -64L,     13L },
    {     12L,     -3L,     16L,    -36L,     48L,    -13L },
    {      1L,      0L,      0L,      0L,      0L,      1L }
};
static long *InterpBlend5[5] =
{
    IB5[0], IB5[1], IB5[2], IB5[3], IB5[4]
};
```



```
static long IB6[6][7] =
{
    {      1L,      1L,      0L,      0L,      0L,      0L,      0L },
    {     60L,    -77L,     300L,    -300L,     200L,    -75L,     12L },
    {    240L,    269L,   -1450L,    2950L,   -2300L,    925L,   -154L },
    {    240L,   -154L,     925L,   -2300L,    2950L,   -1450L,    269L },
    {     60L,     12L,    -75L,     200L,   -300L,    300L,    -77L },
    {      1L,      0L,      0L,      0L,      0L,      0L,      1L }
};
static long *InterpBlend6[6] =
{
    IB6[0], IB6[1], IB6[2], IB6[3], IB6[4], IB6[5]
};

static long IB7[7][8] =
{
    {      1L,      1L,      0L,      0L,      0L,      0L,      0L,
      0L },
    {     60L,    -87L,     360L,    -450L,     400L,    -225L,     72L,
    -10L },
    {    150L,     227L,   -1332L,    3015L,   -3080L,    1845L,   -612L,
     87L },
    {    200L,    -227L,    1512L,   -4185L,    6000L,   -4185L,    1512L,
    -227L },
    {    150L,      87L,    -612L,    1845L,   -3080L,    3015L,   -1332L,
    227L },
    {     60L,    -10L,     72L,    -225L,     400L,    -450L,     360L,
    -87L },
    {      1L,      0L,      0L,      0L,      0L,      0L,      0L,
      1L }
};
static long *InterpBlend7[7] =
{
    IB7[0], IB7[1], IB7[2], IB7[3], IB7[4], IB7[5], IB7[6]
};

static long IB8[8][9] =
{
    {      1L,      1L,      0L,      0L,      0L,      0L,      0L,
      0L,      0L },
    {     420L,    -669L,     2940L,    -4410L,     4900L,    -3675L,    1764L,
    -490L,     60L },
    {    7560L,   14318L,   -90846L,   228879L,   -288610L,   229320L, -113778L,
    32291L,   -4014L },
    {   25200L,  -42881L,   313012L, -960351L,  1568980L, -1409975L,  750876L,
    -223097L,   26836L },
    {   25200L,   28636L,  -223097L,   750876L, -1409975L,  1568980L, -960351L,
    313012L,  -42881L },
    {    7560L,   -4014L,   32291L, -113778L,   229320L, -288610L,  228879L,
    -90846L,   14318L },
    {     420L,      60L,    -490L,    1764L,    -3675L,     4900L,   -4410L,
    2940L,   -669L },
    {      1L,      0L,      0L,      0L,      0L,      0L,      0L,
      0L,      1L }
};
static long *InterpBlend8[8] =
{
    IB8[0], IB8[1], IB8[2], IB8[3], IB8[4], IB8[5], IB8[6], IB8[7]
};
```

```
static long IB9[9][10] =
{
    {      1L,      1L,      0L,      0L,      0L,      0L,      0L,
      {      840L,     -1443L,      6720L,     -11760L,      15680L,     -14700L,      9408L,
      {      8820L,      19939L,     -135936L,      379008L,     -568064L,      561960L,     -370944L,
      {     17640L,     -40953L,      324544L,     -1096144L,      2030784L,     -2229500L,      1563968L,
      {     22050L,      40953L,     -349184L,      1298304L,     -2731008L,      3503920L,     -2731008L,
      {     17640L,     -19939L,      176704L,     -691824L,      1563968L,     -2229500L,      2030784L,
      {     8820L,      4329L,     -39168L,      157696L,     -370944L,      561960L,     -568064L,
      {      840L,     -105L,      960L,     -3920L,      9408L,     -14700L,      15680L,
      {      1L,      0L,      0L,      0L,      0L,      0L,      0L,
      {      1L,      0L,      0L,      0L,      0L,      0L,      1L }
};
static long *InterpBlend9[9] =
{
    IB9[0], IB9[1], IB9[2], IB9[3], IB9[4], IB9[5], IB9[6], IB9[7], IB9[8]
};
static long **InterpBlend[] =
{
    NULL,
    NULL,
    InterpBlend2,
    InterpBlend3,
    InterpBlend4,
    InterpBlend5,
    InterpBlend6,
    InterpBlend7,
    InterpBlend8,
    InterpBlend9
};

/*****
Blends one Bezier control point using the Input points, the Bezier curve
should interpolate, and Interp (Inverse(M)) arrays.
*****/
static FloatType BzrCrvInterpOnePoint(Input, Interp, Size)
FloatType *Input;
long *Interp;
int Size;
{
    int i;
    long Denom = *Interp++;
    FloatType R = 0.0;

    for (i = 0; i < Size; i++)
        R += (*Input++ * *Interp++) / Denom;

    return R;
}

/*****/
Blends the Input points, the Bezier curve should interpolate, using the
```

```
Interp array (Inverse(M)) into the Result Bezier control points array.
Input and Result arrays are of size Size, which is the curve order from 2
(linear) to 9.
*****/
void BzrCrvInterp(Result, Input, Size)
FloatType *Result;
FloatType *Input;
int Size;
{
    int i;
    long **Interp = InterpBlend[Size];

    for (i = 0; i < Size; i++)
        *Result++ = BzrCrvInterpOnePoint(Input, *Interp++, Size);
}
```

Errata to `_Graphics Gems IV_`, first edition, edited by Paul Heckbert  
([Paul.Heckbert@HOSTESS.GRAPHICS.CS.CMU.EDU](mailto:Paul.Heckbert@HOSTESS.GRAPHICS.CS.CMU.EDU)), Academic Press 1994. Code  
available online at <http://www.acm.org/tog/GraphicsGems>

compiled by Eric Haines ([erich@acm.org](mailto:erich@acm.org)) from author and reader contributions

version 1.3

date: 11/7/97

-----

Errors in the text:

p. xvi, near the bottom of the page: change  
"in directory `pub/GraphicsGems/GemsIV`"  
to  
"in directory `pub/Graphics/GraphicsGems/GemsIV`".

p. 129, lines 4,7,10: changes:  
"if (`x < 0`)" to "if (`x >= 0`)",  
"if (`y < 0`)" to "if (`y >= 0`)",  
"if (`z < 0`)" to "if (`z >= 0`)".

p. 138, first equation: change "`(-ax,ay)`" to "`(-ay,ax)`" to match the text.

p. 349, for (Moore 1992): change "Rice University, Houston, TX" to "Cornell  
University, Ithaca, NY".

-----

The following are errors in the code listings (corrected in the online code at  
[princeton.edu:pub/Graphics/GraphicsGems/GemsIV](http://princeton.edu:pub/Graphics/GraphicsGems/GemsIV)). Note that many of the code  
listings online are different in minor and major ways from the code in the  
book.

Serious errors (ones your compiler cannot or may not catch):

[none so far]

-----

Syntax errors (ones that are not usually harmful, or are easily caught):

[none so far]

-----

The following are typographical errors in the comments:

[none so far]

-----

A new point in polygon test has been added to `ptpoly_haines/ptinpoly.c`, a  
variation on the Crossings test which is about 15% faster for triangles.

A potential divide by zero error has been removed from  
`ptpoly_haines/ptinpoly.c`; line 1286 had "`inv_y = tmax / ydiff;`", which is  
now deleted.

END

This directory contains the C and C++ code from the book  
"Graphics Gems IV", edited by Paul Heckbert, Academic Press, 1994.

Comments in the source file or subdirectory's README file tell you  
if the code is old C, ANSI C, or C++.

CONTENTS:

file or directory	book chapter	chapter title and author
-----		
	I	Polygons and Polyhedra
centroid.c	I.1	Centroid of a Polygon Gerard Bashein and Paul R. Detmer
convex_test/	I.2	Testing the Convexity of a Polygon Peter Schorn and Frederick Fisher
ptpoly_weiler/	I.3	An Incremental Angle Point in Polygon Test Kevin Weiler
ptpoly_haines/	I.4	Point in Polygon Strategies Eric Haines
delaunay/	I.5	Incremental Delaunay Triangulation Dani Lischinski
vert_norm/	I.6	Building Vertex Normals from an Unstructured Polygon List Andrew Glassner
collide.c	I.8	Fast Collision Detection of Moving Convex Polyhedra Rich Rabbitz
-----		
	II	Geometry
dist_fast.c	II.2	Fast Linear Approximations of Euclidean Distance in Higher Dimensions Yoshikazu Ohashi
outcode/	II.3	Direct Outcode Calculation for Faster Clip Testing Walt Donovan and Tim Van Hook
sph_poly.c	II.4	Computing the Area of a Spherical Polygon Robert D. Miller
-----		
	III	Transformations
arcball/	III.1	Arcball Rotation Control Ken Shoemake
inv_fast.c	III.3	Fast Inversion of Length- and Angle-Preserving Matrices Kevin Wu
polar_decomp/	III.4	Polar Matrix Decomposition Ken Shoemake

euler_angle/	III.5	Euler Angle Conversion Ken Shoemake
-----		
	IV	Curves and Surfaces
data_smooth/	IV.1	Smoothing and Interpolation with Finite Differences Paul H. C. Eilers
curve_isect/	IV.4	Intersecting Parametric Cubic Curves by Midpoint Subdivision R. Victor Klassen
patch_conv.C	IV.5	Converting Rectangular Patches into Bezier Triangles Dani Lischinski
nurb_polyg/	IV.6	Tessellation of NURB Surfaces John W. Peterson
implicit.c	IV.8	An Implicit Surface Polygonizer Jules Bloomenthal
-----		
	V	Ray Tracing
ray_cyl.c	V.2	Intersecting a Ray with a Cylinder Joseph M. Cychosz and Warren N. Waggenspack, Jr.
vox_traverse.c	V.3	Voxel Traversal along a 3D Line Daniel Cohen
multi_jitter/	V.4	Multi-Jittered Sampling Kenneth Chiu, Peter Shirley, and Changyaw Wang
minray/	V.5	A Minimal Ray Tracer Paul S. Heckbert
-----		
	VII	Frame Buffer Techniques
dyn_range/	VII.3	High Dynamic Range Pixels Christophe Schlick
-----		
	VIII	Image Processing
emboss.c	VIII.1	Fast Embossing Effects on Raster Image Data John Schlag
coons_warp.c	VIII.2	Bilinear Coons Patch Image Warping Paul S. Heckbert
convolve.c	VIII.3	Fast Convolution with Packed Lookup Tables George Wolberg and Henry Massalin
thin_image.c	VIII.4	Efficient Binary Image Thinning using Neighborhood Maps Joseph M. Cychosz
clahe.c	VIII.5	Contrast Limited Adaptive Histogram Equalization Karel Zuiderveld

mrsfoley.im	VIII.6	Ideal Tiles for Shading and Halftoning Alan Wm Paeth
-----		
	IX	Graphic Design
graph_layout/	IX.2	Dynamic Layout Algorithm to Display General Graphs Laszlo Szirmay-Kalos
-----		
	X	Utilities
trilerp.c	X.1	Tri-linear Interpolation Steve Hill
interp_fast.c	X.2	Faster Linear Interpolation Steven Eker
vec_mat/	X.3	C++ Vector and Matrix Algebra Routines Jean-Francois Doue (there's a ray caster in the subdirectory "ray")
GraphicsGems.c	X.4	C Header File and Vector Library
GraphicsGems.h		Andrew Glassner and Eric Haines
-----		

This code is available on the Internet by anonymous FTP from [princeton.edu](http://princeton.edu) (128.112.128.1) in the directory `pub/Graphics/GraphicsGems/GemsIV`. These FTP versions are updated periodically with bug fixes.

If you have corrections to this code, email to [ph@cs.cmu.edu](mailto:ph@cs.cmu.edu) .

Paul Heckbert, 3/15/94

-----

#### History:

5/16/94 - lint fixes made to many pieces of code; mostly cosmetic (Eric Haines)

8/18/94 - in README, fixed FTP path & added bug email addr. (Paul Heckbert)



```
@BOOK{Heckbert94gems,  
TITLE={Graphics Gems IV},  
EDITOR={Paul Heckbert},  
PUBLISHER={Academic Press},  
YEAR={1994},  
ADDRESS={Boston},  
KEYWORDS={computer graphics, image processing},  
NOTE={comes with either MAC or DOS floppy},  
FTP={code in princeton.edu pub/Graphics/GraphicsGems/GemsIV/GGemsIV.tar.Z},  
}
```

```
@INCOLLECTION{Bashein94,  
AUTHOR={Gerard Bashein and Paul R. Detmer},  
TITLE={Centroid of a Polygon},  
BOOKTITLE={Graphics Gems IV},  
EDITOR={Paul Heckbert},  
PAGES={3-6},  
PUBLISHER={Academic Press},  
YEAR={1994},  
ADDRESS={Boston},  
KEYWORDS={polygon area, polygon centroid},  
SUMMARY={  
Gives formulas and code to find the centroid (center of mass) of a  
polygon. This is useful when simulating Newtonian dynamics. Contains  
C code.  
},  
FTP={code in princeton.edu pub/Graphics/GraphicsGems/GemsIV/GGemsIV.tar.Z},  
}
```

```
@INCOLLECTION{Schorn94,  
AUTHOR={Peter Schorn and Frederick Fisher},  
TITLE={Testing the Convexity of a Polygon},  
BOOKTITLE={Graphics Gems IV},  
EDITOR={Paul Heckbert},  
PAGES={7-15},  
PUBLISHER={Academic Press},  
YEAR={1994},  
ADDRESS={Boston},  
KEYWORDS={computational geometry, concave polygon},  
SUMMARY={  
Gives an algorithm and code to determine if a polygon is convex,  
non-convex (concave but not convex), or non-simple  
(self-intersecting). For many polygon operations, faster algorithms  
can be used if the polygon is known to be convex. This is true when  
scan converting a polygon and when determining if a point is inside a  
polygon, for instance. Contains C code.  
},  
FTP={code in princeton.edu pub/Graphics/GraphicsGems/GemsIV/GGemsIV.tar.Z},  
}
```

```
@INCOLLECTION{Weiler94,  
AUTHOR={Kevin Weiler},  
TITLE={An Incremental Angle Point in Polygon Test},  
BOOKTITLE={Graphics Gems IV},  
EDITOR={Paul Heckbert},  
PAGES={16-23},  
PUBLISHER={Academic Press},  
YEAR={1994},  
ADDRESS={Boston},  
KEYWORDS={computational geometry, inclusion testing},  
SUMMARY={  
Provides an algorithm for testing if a point is inside a concave  
polygon, a task known as point inclusion testing in computational
```

geometry. Point-in-polygon testing is a basic task when ray tracing polygonal models, so these methods are useful for 3D as well as 2D graphics. Contains C code.

},

FTP={code in princeton.edu pub/Graphics/GraphicsGems/GemsIV/GGemsIV.tar.Z},

}

@INCOLLECTION{Haines94,

AUTHOR={Eric Haines},

TITLE={Point in Polygon Strategies},

BOOKTITLE={Graphics Gems IV},

EDITOR={Paul Heckbert},

PAGES={24-46},

PUBLISHER={Academic Press},

YEAR={1994},

ADDRESS={Boston},

KEYWORDS={computational geometry, inclusion testing, ray-polygon intersection testing},

SUMMARY={

Provides algorithms for testing if a point is inside a polygon, a task known as point inclusion testing in computational geometry. Point-in-polygon testing is a basic task when ray tracing polygonal models, so these methods are useful for 3D as well as 2D graphics. Haines surveys a number of algorithms for point inclusion testing in both convex and concave polygons, with empirical speed tests and practical optimizations. Contains C code.

},

FTP={code in princeton.edu pub/Graphics/GraphicsGems/GemsIV/GGemsIV.tar.Z},

}

@INCOLLECTION{Lischinski94triang,

AUTHOR={Dani Lischinski},

TITLE={Incremental {D}elaunay Triangulation},

BOOKTITLE={Graphics Gems IV},

EDITOR={Paul Heckbert},

PAGES={47-59},

PUBLISHER={Academic Press},

YEAR={1994},

ADDRESS={Boston},

KEYWORDS={mesh generation, polygonization, Voronoi diagram, Delaunay triangulation},

SUMMARY={

Gives some code to solve a very important problem: finding Delaunay triangulations and Voronoi diagrams in 2D. These two geometric constructions are useful for triangular mesh generation and for nearest neighbor finding, respectively. Triangular mesh generation comes up when doing interpolation of surfaces from scattered data points, and in finite element simulations of all kinds, such as radiosity. Voronoi diagrams are used in many computational geometry algorithms. Contains C++ code.

},

FTP={code in princeton.edu pub/Graphics/GraphicsGems/GemsIV/GGemsIV.tar.Z},

}

@INCOLLECTION{Glassner94normal,

AUTHOR={Andrew Glassner},

TITLE={Building Vertex Normals from an Unstructured Polygon List},

BOOKTITLE={Graphics Gems IV},

EDITOR={Paul Heckbert},

PAGES={60-73},

PUBLISHER={Academic Press},

YEAR={1994},

ADDRESS={Boston},

KEYWORDS={smooth shading, topology, polyhedron},

```
SUMMARY={
Solves a fairly common rendering problem:  if one is given a set of
polygons in raw form, with no topological (adjacency) information, and
asked to do smooth shading (Gouraud or Phong shading) of them, one must
infer topology and compute vertex normals.  Contains C code.
},
FTP={code in princeton.edu pub/Graphics/GraphicsGems/GemsIV/GGemsIV.tar.Z},
}
@INCOLLECTION{Greene94,
AUTHOR={Ned Greene},
TITLE={Detecting Intersection of a Rectangular Solid and a Convex Polyhedron},
BOOKTITLE={Graphics Gems IV},
EDITOR={Paul Heckbert},
PAGES={74-82},
PUBLISHER={Academic Press},
YEAR={1994},
ADDRESS={Boston},
KEYWORDS={collision detection, octree, computational geometry},
SUMMARY={
Presents an optimized technique to test for intersection between a
convex polyhedron and a box.  This is useful when comparing bounding
boxes against a viewing frustum in a rendering program, for instance.
Contains pseudocode.
},
}
@INCOLLECTION{Rabbitz94,
AUTHOR={Rich Rabbitz},
TITLE={Fast Collision Detection of Moving Convex Polyhedra},
BOOKTITLE={Graphics Gems IV},
EDITOR={Paul Heckbert},
PAGES={83-109},
PUBLISHER={Academic Press},
YEAR={1994},
ADDRESS={Boston},
KEYWORDS={animation, collision detection, computational geometry, polyhedron},
SUMMARY={
A turn-key piece of software that solves a difficult but basic problem
in physically based animation and interactive modeling.  Contains C
code.
},
FTP={code in princeton.edu pub/Graphics/GraphicsGems/GemsIV/GGemsIV.tar.Z},
}
@INCOLLECTION{Hart94,
AUTHOR={John C. Hart},
TITLE={Distance to an Ellipsoid},
BOOKTITLE={Graphics Gems IV},
EDITOR={Paul Heckbert},
PAGES={113-119},
PUBLISHER={Academic Press},
YEAR={1994},
ADDRESS={Boston},
KEYWORDS={ray tracing, ellipse},
SUMMARY={
Gives the formulas necessary to find the distance from a point to an
ellipsoid, or from a point to an ellipse.  These formulas can be useful
for geometric modeling or for ray tracing.
},
}
@INCOLLECTION{Ohashi94,
AUTHOR={Yoshikazu Ohashi},
TITLE={Fast Linear Approximations of {E}uclidean Distance in Higher Dimensions},
```

```
BOOKTITLE={Graphics Gems IV},
EDITOR={Paul Heckbert},
PAGES={120-124},
PUBLISHER={Academic Press},
YEAR={1994},
ADDRESS={Boston},
KEYWORDS={integer arithmetic, square root},
SUMMARY={
Provides optimized formulas for approximating Euclidean distance in two
or more dimensions without square roots.  Contains C code.
},
FTP={code in princeton.edu pub/Graphics/GraphicsGems/GemsIV/GGemsIV.tar.Z},
}
@INCOLLECTION{Donovan94,
AUTHOR={Walt Donovan and Tim Van Hook},
TITLE={Direct Outcode Calculation for Faster Clip Testing},
BOOKTITLE={Graphics Gems IV},
EDITOR={Paul Heckbert},
PAGES={125-131},
PUBLISHER={Academic Press},
YEAR={1994},
ADDRESS={Boston},
KEYWORDS={IEEE floating point arithmetic, line clipping},
SUMMARY={
A clever optimization of clip testing that exploits the properties
of IEEE floating point format.  Techniques are described to compute the
``outcodes'' needed for line clipping using only integer arithmetic.
Contains C code.
},
FTP={code in princeton.edu pub/Graphics/GraphicsGems/GemsIV/GGemsIV.tar.Z},
}
@INCOLLECTION{Miller94,
AUTHOR={Robert D. Miller},
TITLE={Computing the Area of a Spherical Polygon},
BOOKTITLE={Graphics Gems IV},
EDITOR={Paul Heckbert},
PAGES={132-137},
PUBLISHER={Academic Press},
YEAR={1994},
ADDRESS={Boston},
KEYWORDS={cartography, Earth},
SUMMARY={
Gives the formulas needed to find the area of a polygon on a sphere
that is bounded by great circle arcs.  This is useful in cartography.
Contains C code.
},
FTP={code in princeton.edu pub/Graphics/GraphicsGems/GemsIV/GGemsIV.tar.Z},
}
@INCOLLECTION{Hill94perp,
AUTHOR={F. S. {Hill Jr.}},
TITLE={The Pleasures of 'Perp Dot' Products},
BOOKTITLE={Graphics Gems IV},
EDITOR={Paul Heckbert},
PAGES={138-148},
PUBLISHER={Academic Press},
YEAR={1994},
ADDRESS={Boston},
KEYWORDS={geometry, fillet, perp-dot product, projection},
SUMMARY={
A tutorial on the ``perp dot product,'' which is the dot product, in
2D, of one vector and the vector perpendicular to another.
}
```

```
},
}
@INCOLLECTION{Hanson94,
AUTHOR={Andrew J. Hanson},
TITLE={Geometry for N-Dimensional Graphics},
BOOKTITLE={Graphics Gems IV},
EDITOR={Paul Heckbert},
PAGES={149-170},
PUBLISHER={Academic Press},
YEAR={1994},
ADDRESS={Boston},
KEYWORDS={geometry, hyperplane, determinant, Levi-Civita, volume},
SUMMARY={
A tutorial on n-dimensional geometry. Hanson generalizes a number of
familiar concepts, such as plane equations, clipping, volume, and
rotation, to n-D.
},
}
@INCOLLECTION{Shoemake94arcball,
AUTHOR={Ken Shoemake},
TITLE={Arcball Rotation Control},
BOOKTITLE={Graphics Gems IV},
EDITOR={Paul Heckbert},
PAGES={175-192},
PUBLISHER={Academic Press},
YEAR={1994},
ADDRESS={Boston},
KEYWORDS={interaction, quaternion},
SUMMARY={
Asks the question: how does one control the three degrees of freedom
of rotation in 3D, using a 2D input device such as a mouse? Shoemake's
answer: use a pair of points to designate a relative rotation, and use
quaternions to make the rotation axis specification intuitive and
consistent. Contains C code.
},
FTP={code in princeton.edu pub/Graphics/GraphicsGems/GemsIV/GGemsIV.tar.Z},
}
@INCOLLECTION{Cromwell94,
AUTHOR={Robert L. Cromwell},
TITLE={Efficient Eigenvalues for Visualization},
BOOKTITLE={Graphics Gems IV},
EDITOR={Paul Heckbert},
PAGES={193-198},
PUBLISHER={Academic Press},
YEAR={1994},
ADDRESS={Boston},
KEYWORDS={scatterplot, matrix of central moments},
SUMMARY={
Answers the question: if I have a set of points in 3D, from what
direction should I view them to get the best view (minimizing bunching
in the projection)? Solving this involves the eigenvalues of a 3x3
matrix. Optimized formulas are given for computing the eigenvalues.
},
}
@INCOLLECTION{Wu94,
AUTHOR={Kevin Wu},
TITLE={Fast Inversion of Length- and Angle-Preserving Matrices},
BOOKTITLE={Graphics Gems IV},
EDITOR={Paul Heckbert},
PAGES={199-206},
PUBLISHER={Academic Press},
```

```
YEAR={1994},
ADDRESS={Boston},
KEYWORDS={eigenvalue, 4x4 matrix},
SUMMARY={
Presents optimized formulas and code to compute the inverse of a 4x4
matrix that is known to be length- and angle-preserving (consisting of
only rotation, translation, and uniform scaling).  Contains C code.
},
FTP={code in princeton.edu pub/Graphics/GraphicsGems/GemsIV/GGemsIV.tar.Z},
}
@INCOLLECTION{Shoemake94decomp,
AUTHOR={Ken Shoemake},
TITLE={Polar Matrix Decomposition},
BOOKTITLE={Graphics Gems IV},
EDITOR={Paul Heckbert},
PAGES={207-221},
PUBLISHER={Academic Press},
YEAR={1994},
ADDRESS={Boston},
KEYWORDS={quaternion, spectral decomposition, singular value decomposition},
SUMMARY={
Describes a method for decomposing an affine 3D transformation into
translation, scaling, and rotation transformations in a physically
meaningful way.  This can be useful for keyframe animation.  Contains C
code.
},
FTP={code in princeton.edu pub/Graphics/GraphicsGems/GemsIV/GGemsIV.tar.Z},
}
@INCOLLECTION{Shoemake94euler,
AUTHOR={Ken Shoemake},
TITLE={Euler Angle Conversion},
BOOKTITLE={Graphics Gems IV},
EDITOR={Paul Heckbert},
PAGES={222-229},
PUBLISHER={Academic Press},
YEAR={1994},
ADDRESS={Boston},
KEYWORDS={quaternion, rotation},
SUMMARY={
Gives code to convert a rotation expressed by one triple of axes into
another triple.  Rotations in 3D are often described in terms of Euler
angles: rotations about x, y, and z in some order.  The order of
rotations is significant, but not standardized.  These routines are
useful for doing such conversions.  Contains C code.
},
FTP={code in princeton.edu pub/Graphics/GraphicsGems/GemsIV/GGemsIV.tar.Z},
}
@INCOLLECTION{Shoemake94twist,
AUTHOR={Ken Shoemake},
TITLE={Fiber Bundle Twist Reduction},
BOOKTITLE={Graphics Gems IV},
EDITOR={Paul Heckbert},
PAGES={230-236},
PUBLISHER={Academic Press},
YEAR={1994},
ADDRESS={Boston},
KEYWORDS={animation, camera control, quaternion, topology, rotation},
SUMMARY={
Applies some advanced concepts from topology to the problem of
minimizing twist (rotation about the z axis of screen space) in
animation.  Has color plate.
```

```
},
}
@INCOLLECTION{Eilers94,
AUTHOR={Paul H. C. Eilers},
TITLE={Smoothing and Interpolation with Finite Differences},
BOOKTITLE={Graphics Gems IV},
EDITOR={Paul Heckbert},
PAGES={241-250},
PUBLISHER={Academic Press},
YEAR={1994},
ADDRESS={Boston},
KEYWORDS={data smoothing, curve smoothing},
SUMMARY={
Gives simple techniques for smoothing a set of uniformly spaced
samples. This can be used to remove noise from input data. Contains C
code.
},
FTP={code in princeton.edu pub/Graphics/GraphicsGems/GemsIV/GGemsIV.tar.Z},
}
@INCOLLECTION{Barry94,
AUTHOR={Phillip Barry and Ron Goldman},
TITLE={Knot Insertion Using Forward Differences},
BOOKTITLE={Graphics Gems IV},
EDITOR={Paul Heckbert},
PAGES={251-255},
PUBLISHER={Academic Press},
YEAR={1994},
ADDRESS={Boston},
KEYWORDS={curve subdivision},
SUMMARY={
Describes a fast technique for subdivision of B-spline curves and
surfaces. Subdivision is a fundamental operation in modeling and
rendering. Contains pseudocode.
},
}
@INCOLLECTION{Bajaj94,
AUTHOR={Chandrajit Bajaj and Guoliang Xu},
TITLE={Converting a Rational Curve to a Standard Rational {B}ernstein-{B}\\'ezier
Representation},
BOOKTITLE={Graphics Gems IV},
EDITOR={Paul Heckbert},
PAGES={256-260},
PUBLISHER={Academic Press},
YEAR={1994},
ADDRESS={Boston},
KEYWORDS={Bezier curve, rational curve, NURB},
SUMMARY={
Discusses a low-level operation on curves: conversion of a rational
piecewise polynomial curve into rational Bernstein-Bezier form, a
commonly used type of non-uniform rational B-spline (NURB). Rational
splines are useful because they allow conic curves to be represented
exactly, whereas standard (polynomial) splines do not. Contains
pseudocode.
},
}
@INCOLLECTION{Klassen94,
AUTHOR={R. Victor Klassen},
TITLE={Intersecting Parametric Cubic Curves by Midpoint Subdivision},
BOOKTITLE={Graphics Gems IV},
EDITOR={Paul Heckbert},
PAGES={261-277},
```

```
PUBLISHER={Academic Press},
YEAR={1994},
ADDRESS={Boston},
KEYWORDS={Bezier curve, curve intersection, curve subdivision},
SUMMARY={
Provides code to find the intersection points of planar cubic curves.
Contains C++ code.
},
FTP={code in princeton.edu pub/Graphics/GraphicsGems/GemsIV/GGemsIV.tar.Z},
}
@INCOLLECTION{Lischinski94patch,
AUTHOR={Dani Lischinski},
TITLE={Converting Rectangular Patches into {B}\`ezier Triangles},
BOOKTITLE={Graphics Gems IV},
EDITOR={Paul Heckbert},
PAGES={278-285},
PUBLISHER={Academic Press},
YEAR={1994},
ADDRESS={Boston},
KEYWORDS={Bezier surface, biquadratic patch},
SUMMARY={
A parametric surface conversion routine.  Contains C++ code.
},
FTP={code in princeton.edu pub/Graphics/GraphicsGems/GemsIV/GGemsIV.tar.Z},
}
@INCOLLECTION{Peterson94,
AUTHOR={John W. Peterson},
TITLE={Tessellation of NURB Surfaces},
BOOKTITLE={Graphics Gems IV},
EDITOR={Paul Heckbert},
PAGES={286-320},
PUBLISHER={Academic Press},
YEAR={1994},
ADDRESS={Boston},
KEYWORDS={surface subdivision, NURB, Bezier surface, B-spline surface,
polygonization},
SUMMARY={
Gives code for polygonizing a very general class of parametric
surfaces: NURBs.  Polygonization of parametric surfaces is useful both
for rendering and for modeling.  Contains C code.
},
FTP={code in princeton.edu pub/Graphics/GraphicsGems/GemsIV/GGemsIV.tar.Z},
}
@INCOLLECTION{Shene94equation,
AUTHOR={Ching-Kuang Shene},
TITLE={Equations of Cylinders and Cones},
BOOKTITLE={Graphics Gems IV},
EDITOR={Paul Heckbert},
PAGES={321-323},
PUBLISHER={Academic Press},
YEAR={1994},
ADDRESS={Boston},
KEYWORDS={quadric surface},
SUMMARY={
Derives the implicit equations of cylinders and cones.
},
}
@INCOLLECTION{Bloomenthal94,
AUTHOR={Jules Bloomenthal},
TITLE={An Implicit Surface Polygonizer},
BOOKTITLE={Graphics Gems IV},
```



```
EDITOR={Paul Heckbert},
PAGES={324-349},
PUBLISHER={Academic Press},
YEAR={1994},
ADDRESS={Boston},
KEYWORDS={iso-surface, marching cubes, polygonization},
SUMMARY={
  Gives code to polygonize an arbitrary implicit surface. Polygonization
  is a common approach to implicit surface rendering and volume
  rendering. When combined with code for trilinear interpolation code, a
  program for polygonizing volume data can easily be constructed. The
  resulting polygonizations will be superior to those generated by the
  ``Marching Cubes'' algorithm, in many cases. Contains C code.
},
FTP={code in princeton.edu pub/Graphics/GraphicsGems/GemsIV/GGemsIV.tar.Z},
}
@INCOLLECTION{Shene94intersect,
AUTHOR={Ching-Kuang Shene},
TITLE={Computing the Intersection of a Line and a Cylinder},
BOOKTITLE={Graphics Gems IV},
EDITOR={Paul Heckbert},
PAGES={353-355},
PUBLISHER={Academic Press},
YEAR={1994},
ADDRESS={Boston},
KEYWORDS={quadric surface, ray tracing, cone},
SUMMARY={
  Geometric method for intersecting a ray with a cylinder.
},
}
@INCOLLECTION{Waggenpack94,
AUTHOR={Joseph M. Cychosz and Warren N. {Waggenpack Jr.}},
TITLE={Intersecting a Ray with a Cylinder},
BOOKTITLE={Graphics Gems IV},
EDITOR={Paul Heckbert},
PAGES={356-365},
PUBLISHER={Academic Press},
YEAR={1994},
ADDRESS={Boston},
KEYWORDS={quadric surface, ray tracing},
SUMMARY={
  Geometric method for intersecting a ray with a cylinder. Contains C
  code.
},
FTP={code in princeton.edu pub/Graphics/GraphicsGems/GemsIV/GGemsIV.tar.Z},
}
@INCOLLECTION{Cohen94,
AUTHOR={Daniel Cohen},
TITLE={Voxel Traversal along a 3{D} Line},
BOOKTITLE={Graphics Gems IV},
EDITOR={Paul Heckbert},
PAGES={366-369},
PUBLISHER={Academic Press},
YEAR={1994},
ADDRESS={Boston},
KEYWORDS={digital line drawing, grid traversal, ray tracing, scan conversion},
SUMMARY={
  Visits all the voxels along a 3D line segment with integer endpoints,
  something like a 3D Bresenham's algorithm. This code could be modified
  to take endpoints with fixed point coordinates. Then the algorithm
  could be very useful for ray tracing, when a uniform grid is being used
}
```

as a spatial data structure for optimization, or for volume rendering of grid data. Contains C code.

```
},  
FTP={code in princeton.edu pub/Graphics/GraphicsGems/GemsIV/GGemsIV.tar.Z},  
}
```

```
@INCOLLECTION{Chiu94,  
AUTHOR={Kenneth Chiu and Peter Shirley and Changyaw Wang},  
TITLE={Multi-Jittered Sampling},  
BOOKTITLE={Graphics Gems IV},  
EDITOR={Paul Heckbert},  
PAGES={370-374},  
PUBLISHER={Academic Press},  
YEAR={1994},  
ADDRESS={Boston},  
KEYWORDS={antialiasing, jitter, stochastic sampling},  
SUMMARY={
```

Presents a new technique for high quality stochastic sampling. This is useful for antialiasing. Contains C code.

```
},  
FTP={code in princeton.edu pub/Graphics/GraphicsGems/GemsIV/GGemsIV.tar.Z},  
}
```

```
@INCOLLECTION{Heckbert94minimal,  
AUTHOR={Paul S. Heckbert},  
TITLE={A Minimal Ray Tracer},  
BOOKTITLE={Graphics Gems IV},  
EDITOR={Paul Heckbert},  
PAGES={375-381},  
PUBLISHER={Academic Press},  
YEAR={1994},  
ADDRESS={Boston},  
KEYWORDS={ray tracing, obfuscation},  
SUMMARY={
```

Answers the question: how short can a ray tracer be? Contains C code.

```
},  
FTP={code in princeton.edu pub/Graphics/GraphicsGems/GemsIV/GGemsIV.tar.Z},  
}
```

```
@INCOLLECTION{Schlick94phong,  
AUTHOR={Christophe Schlick},  
TITLE={A Fast Alternative to {P}hong's Specular Model},  
BOOKTITLE={Graphics Gems IV},  
EDITOR={Paul Heckbert},  
PAGES={385-387},  
PUBLISHER={Academic Press},  
YEAR={1994},  
ADDRESS={Boston},  
KEYWORDS={Phong illumination, specular reflection, exponentiation},  
SUMMARY={
```

Describes a simple approximation to Phong's specular reflection formula that does not require exponentiation or table lookup.

```
},  
}  
@INCOLLECTION{Fisher94,  
AUTHOR={Frederick Fisher and Andrew Woo},  
TITLE={{R.E} versus {N.H} Specular Highlights},  
BOOKTITLE={Graphics Gems IV},  
EDITOR={Paul Heckbert},  
PAGES={388-400},  
PUBLISHER={Academic Press},  
YEAR={1994},  
ADDRESS={Boston},  
KEYWORDS={Phong illumination, specular reflection},
```

```
SUMMARY={
Compares two common variants of Phong's specular reflection formula and
derives a surprising relationship between them.
},
}
@INCOLLECTION{Schlick94perlin,
AUTHOR={Christophe Schlick},
TITLE={Fast Alternatives to {P}erlin's Bias and Gain Functions},
BOOKTITLE={Graphics Gems IV},
EDITOR={Paul Heckbert},
PAGES={401-403},
PUBLISHER={Academic Press},
YEAR={1994},
ADDRESS={Boston},
KEYWORDS={hypertexture, texture synthesis},
SUMMARY={
Gives a simple approximation to some formulas that are commonly used in
procedural texture synthesis and volume synthesis.
},
}
@INCOLLECTION{Behrens94,
AUTHOR={Uwe Behrens},
TITLE={Fence Shading},
BOOKTITLE={Graphics Gems IV},
EDITOR={Paul Heckbert},
PAGES={404-409},
PUBLISHER={Academic Press},
YEAR={1994},
ADDRESS={Boston},
KEYWORDS={Gouraud shading, Phong shading},
SUMMARY={
Proposes an approach halfway between Gouraud shading and Phong
shading: shade along the edges of the polygon, but interpolate across
the interior. Contains pseudocode.
},
}
@INCOLLECTION{Kopp94,
AUTHOR={Manfred Kopp and Michael Gervautz},
TITLE={{XOR}-Drawing with Guaranteed Contrast},
BOOKTITLE={Graphics Gems IV},
EDITOR={Paul Heckbert},
PAGES={413-414},
PUBLISHER={Academic Press},
YEAR={1994},
ADDRESS={Boston},
KEYWORDS={cursor},
SUMMARY={
Gives simple guidelines for the choice of write mask when drawing
cursors and other graphics with exclusive-OR on multi-bit frame
buffers.
},
}
@INCOLLECTION{Ward94,
AUTHOR={Greg Ward},
TITLE={A Contrast-Based Scalefactor for Luminance Display},
BOOKTITLE={Graphics Gems IV},
EDITOR={Paul Heckbert},
PAGES={415-421},
PUBLISHER={Academic Press},
YEAR={1994},
ADDRESS={Boston},
```

KEYWORDS={brightness, color, image display, perception, Radiance system},  
SUMMARY={  
Presents a simple technique for displaying pictures that have a dynamic  
range (ratio of brightest to dimmest pixel) beyond that of the  
display. Has color plates.  
},  
}  
@INCOLLECTION{Schlick94dynamic,  
AUTHOR={Christophe Schlick},  
TITLE={High Dynamic Range Pixels},  
BOOKTITLE={Graphics Gems IV},  
EDITOR={Paul Heckbert},  
PAGES={422-429},  
PUBLISHER={Academic Press},  
YEAR={1994},  
ADDRESS={Boston},  
KEYWORDS={quantization, dynamic range},  
SUMMARY={  
Proposes a pixel encoding technique that allows color images with high  
dynamic range to be stored using only 24 bits per pixel. Contains C  
code.  
},  
FTP={code in princeton.edu pub/Graphics/GraphicsGems/GemsIV/GGemsIV.tar.Z},  
}  
@INCOLLECTION{Schlag94,  
AUTHOR={John Schlag},  
TITLE={Fast Embossing Effects on Raster Image Data},  
BOOKTITLE={Graphics Gems IV},  
EDITOR={Paul Heckbert},  
PAGES={433-437},  
PUBLISHER={Academic Press},  
YEAR={1994},  
ADDRESS={Boston},  
KEYWORDS={bump mapping, texture, image filter, paint program, shading},  
SUMMARY={  
Presents a simple, fast technique that interprets an image as a height  
field or bump map, and then shades it, yielding an embossing effect.  
Has color plate. Contains C code.  
},  
FTP={code in princeton.edu pub/Graphics/GraphicsGems/GemsIV/GGemsIV.tar.Z},  
}  
@INCOLLECTION{Heckbert94coons,  
AUTHOR={Paul S. Heckbert},  
TITLE={Bilinear {C}oons Patch Image Warping},  
BOOKTITLE={Graphics Gems IV},  
EDITOR={Paul Heckbert},  
PAGES={438-446},  
PUBLISHER={Academic Press},  
YEAR={1994},  
ADDRESS={Boston},  
KEYWORDS={morph, resampling, bilinear Coons patch},  
SUMMARY={  
Presents a fast technique to warp an image according to four boundary  
curves. This can be used to correct distortions in images, or to  
introduce them, for special effects purposes. Has color plates.  
Contains C code.  
},  
FTP={code in princeton.edu pub/Graphics/GraphicsGems/GemsIV/GGemsIV.tar.Z},  
}  
@INCOLLECTION{Wolberg94,  
AUTHOR={George Wolberg and Henry Massalin},

```
TITLE={Fast Convolution with Packed Lookup Tables},
BOOKTITLE={Graphics Gems IV},
EDITOR={Paul Heckbert},
PAGES={447-464},
PUBLISHER={Academic Press},
YEAR={1994},
ADDRESS={Boston},
KEYWORDS={image filter, image magnification, image resampling},
SUMMARY={
Gives optimized code for convolution of discrete signals. This is
useful for image resampling and filtering. Contains C code.
},
FTP={code in princeton.edu pub/Graphics/GraphicsGems/GemsIV/GGemsIV.tar.Z},
}

@INCOLLECTION{Cychosz94,
AUTHOR={Joseph M. Cychosz},
TITLE={Efficient Binary Image Thinning Using Neighborhood Maps},
BOOKTITLE={Graphics Gems IV},
EDITOR={Paul Heckbert},
PAGES={465-473},
PUBLISHER={Academic Press},
YEAR={1994},
ADDRESS={Boston},
KEYWORDS={image skeleton, image filter},
SUMMARY={
Provides fast code to thin a bitmap image and find its ``skeleton.''
Image thinning is used for pattern recognition. Contains C code.
},
FTP={code in princeton.edu pub/Graphics/GraphicsGems/GemsIV/GGemsIV.tar.Z},
}

@INCOLLECTION{Zuiderveld94,
AUTHOR={Karel Zuiderveld},
TITLE={Contrast Limited Adaptive Histogram Equalization},
BOOKTITLE={Graphics Gems IV},
EDITOR={Paul Heckbert},
PAGES={474-485},
PUBLISHER={Academic Press},
YEAR={1994},
ADDRESS={Boston},
KEYWORDS={contrast enhancement},
SUMMARY={
Presents a contrast enhancement technique that overcomes some of the
flaws of simple histogram equalization, such as noise amplification and
suppression of local contrast. Contains C code.
},
FTP={code in princeton.edu pub/Graphics/GraphicsGems/GemsIV/GGemsIV.tar.Z},
}

@INCOLLECTION{Paeth94,
AUTHOR={Alan W. Paeth},
TITLE={Ideal Tiles for Shading and Halftoning},
BOOKTITLE={Graphics Gems IV},
EDITOR={Paul Heckbert},
PAGES={486-492},
PUBLISHER={Academic Press},
YEAR={1994},
ADDRESS={Boston},
KEYWORDS={bitblt, dither, stipple},
SUMMARY={
Discusses heuristics and design techniques for top-quality dither and
stipple patterns.
},
}
```

```
FTP={picture in princeton.edu pub/Graphics/GraphicsGems/GemsIV/GGemsIV.tar.Z},
}
@INCOLLECTION{Christensen94,
AUTHOR={Jon Christensen and Joe Marks and Stuart Shieber},
TITLE={Placing Text Labels on Maps and Diagrams},
BOOKTITLE={Graphics Gems IV},
EDITOR={Paul Heckbert},
PAGES={497-504},
PUBLISHER={Academic Press},
YEAR={1994},
ADDRESS={Boston},
KEYWORDS={cartography, label placement, layout, relaxation,
simulated annealing},
SUMMARY={
Presents an algorithm to arrange text labels in a way that avoids
overlap. This is useful in cartography.
},
}
@INCOLLECTION{Szirmay-Kalos94,
AUTHOR={L\'aszl\'o Szirmay-Kalos},
TITLE={Dynamic Layout Algorithm to Display General Graphs},
BOOKTITLE={Graphics Gems IV},
EDITOR={Paul Heckbert},
PAGES={505-517},
PUBLISHER={Academic Press},
YEAR={1994},
ADDRESS={Boston},
KEYWORDS={cartography, layout, graph visualization, physically-based methods},
SUMMARY={
Gives code that finds aesthetic arrangements for a graph. This could
be used to graphically display data structures. Contains C++ code.
},
FTP={code in princeton.edu pub/Graphics/GraphicsGems/GemsIV/GGemsIV.tar.Z},
}
@INCOLLECTION{Hill94trilinear,
AUTHOR={Steve Hill},
TITLE={Tri-linear Interpolation},
BOOKTITLE={Graphics Gems IV},
EDITOR={Paul Heckbert},
PAGES={521-525},
PUBLISHER={Academic Press},
YEAR={1994},
ADDRESS={Boston},
KEYWORDS={trilinear interpolation, bilinear interpolation, reconstruction},
SUMMARY={
Gives optimized code for performing linear interpolation in a 3D grid.
Trilinear interpolation is useful for volume rendering, and its 2D
variant, bilinear interpolation, is a very common operation in image
processing. Contains C code.
},
FTP={code in princeton.edu pub/Graphics/GraphicsGems/GemsIV/GGemsIV.tar.Z},
}
@INCOLLECTION{Eker94,
AUTHOR={Steven Eker},
TITLE={Faster Linear Interpolation},
BOOKTITLE={Graphics Gems IV},
EDITOR={Paul Heckbert},
PAGES={526-533},
PUBLISHER={Academic Press},
YEAR={1994},
ADDRESS={Boston},
```

```
KEYWORDS={line drawing, Gouraud shading, integer arithmetic, image resampling},
SUMMARY={
  Gives optimized code for generic linear interpolation.  This is most
  useful for assembler language programming of graphics operations such
  as Gouraud shading and image scaling.  Contains C code.
},
FTP={code in princeton.edu pub/Graphics/GraphicsGems/GemsIV/GGemsIV.tar.Z},
}
@INCOLLECTION{Doue94,
AUTHOR={Jean-Fran\c cois Dou\ 'e},
TITLE={C++ Vector and Matrix Algebra Routines},
BOOKTITLE={Graphics Gems IV},
EDITOR={Paul Heckbert},
PAGES={534-557},
PUBLISHER={Academic Press},
YEAR={1994},
ADDRESS={Boston},
KEYWORDS={vector algebra, transformation, ray tracing},
SUMMARY={
  A C++ subroutine library for 2D, 3D, and 4D vector and matrix
  operations.  Contains C++ code.  The Floppy disk that comes with the
  book and the FTP collection contain a simple ray tracer written using
  the library.
},
FTP={code in princeton.edu pub/Graphics/GraphicsGems/GemsIV/GGemsIV.tar.Z},
}
@INCOLLECTION{Glassner94library,
AUTHOR={Andrew Glassner and Eric Haines},
TITLE={C Header File and Vector Library},
BOOKTITLE={Graphics Gems IV},
EDITOR={Paul Heckbert},
PAGES={558-570},
PUBLISHER={Academic Press},
YEAR={1994},
ADDRESS={Boston},
KEYWORDS={vector algebra, root-finding},
SUMMARY={
  Revised version of the ``Graphics Gems'' subroutine library, used in
  several other articles in the book.  Contains C code.
},
FTP={code in princeton.edu pub/Graphics/GraphicsGems/GemsIV/GGemsIV.tar.Z},
}
```

2001  
Copyright 2001  
All rights reserved.  
This work is derived from the book "Graphics Gems IV" by Thomas D. Foley, van Dam, van Dam, and van Dam, published by Morgan Kaufmann Publishers, Inc. in 1997. This work is derived from the book "Graphics Gems IV" by Thomas D. Foley, van Dam, van Dam, and van Dam, published by Morgan Kaufmann Publishers, Inc. in 1997. This work is derived from the book "Graphics Gems IV" by Thomas D. Foley, van Dam, van Dam, and van Dam, published by Morgan Kaufmann Publishers, Inc. in 1997.











































































































































































```
#include "Camera.h"

/*****
 *
 * These methods set the field of view of the camera. They transform
 * the field of view between a user representation (2 angles specified
 * in degrees) an internal representation (the size of the film square).
 *
 *****/

void Camera::setFieldOfView(vec2& v)
{ film_size = (v * (M_PI / 360.0)).apply(&atan); }

vec2 Camera::fieldOfView()
{ return (360.0 * M_1_PI) * (vec2(film_size).apply(&atan)); }

/*****
 *
 * This method transforms a 2D point located on the film plane into a
 * normalized ray. The method only returns the direction of the ray.
 * The origin can be found by asking the camera its position.
 *
 *****/

vec3 Camera::pointToRay(vec2& p)
{ return orient * vec3(prod(film_size, p), -1.0).normalize(); }

/*****
 *
 * Input from stream.
 *
 *****/

istream& operator >> (istream& s, Camera& a)
{
    vec2    fov;

    s >> *((Object3D*) &a);
    s >> fov;
    a.setFieldOfView(fov);
    return s;
}
```

[illegible]

```
#ifndef Camera_h
#define Camera_h 1
#include "Object3D.h"

class Camera: public Object3D
{
protected:
    vec2    film_size;

public:

    void setFieldOfView(vec2& v);
    vec2 fieldOfView();
    vec3 pointToRay(vec2& p);

    // friends
    friend istream& operator >> (istream& s, Camera& a);
};

#endif
```

```
#include "Light.h"
```

```
istream& operator >> (istream& s, Light& a)
{
s >> *((Object3D*) &a);
s >> a.col;
return s;
}
```

```

/*****
 *
 * CLASS: Light
 * AUTHOR: Jean-Francois DOUE
 * LAST MODIFICATION: 12 Oct 1993
 *
 * This class implements a simple point light source.
 * It just contains the color of the light source.
 *
 *****/

#ifndef Light_h
#define Light_h 1
#include "Object3D.h"

class Light: public Object3D
{
protected:
    vec3    col;          // color of the light source

public:

    vec3 color() { return col; };
    friend istream& operator >> (istream& s, Light& a);
};

#endif
```

```
#
# NeXT Makefile for test (a program to test the algebra3 C++ routines)
# Requires libg++ and a c++ compiler. See README file
#

CC = cc++
CFLAGS = -g -I/usr/local/lib/g++-include -L/usr/local/lib
LIBS = -lg++
OBJS = algebra3.o Camera.o Light.o Object3D.o Polyhedron.o Primitive.o \
      Scene3D.o solver.o Sphere.o main.o

.c.o:
    $(CC) -c $@ $(CFLAGS) $*.c
ray: $(OBJS)
    $(CC) -o $@ $(CFLAGS) $(OBJS) $(LIBS)
```



```
#include "Object3D.h"
```

```

/*****
 *
 * Input from stream.
 *
 *****/
```

```
istream& operator >> (istream& s, Object3D& a)
{
s >> a.pos;
s >> a.orient;
return s;
}
```

```

/*****
 *
 * CLASS: Object3D
 * AUTHOR: Jean-Francois DOUE
 * LAST MODIFICATION: 12 Oct 1993
 *
 * This class is an abstract root class for all the 3D objects.
 * It stores the position an the orientation of the object
 *
 *****/

#ifndef Object3D_h
#define Object3D_h 1
#include "algebra3.h"

class Object3D
{
protected:
    vec3    pos;
    mat4    orient;

public:

    void setPosition(vec3& p) { pos = p; };
    vec3 position() { return pos; };
    void setOrientation(mat4& o) { orient = o; };
    mat4 orientation() { return orient; };

    // friends
    friend istream& operator >> (istream& s, Object3D& a);
};

#endif
```

```
#include "Polyhedron.h"

/*****
 *
 * Class destructor
 *
 *****/

Polyhedron::~Polyhedron()
{
    int i;

    // delete the vertices and the plane equations
    delete vList;
    delete pList;

    // delete the facets
    for(i = 0; i < facetN; i++)
        delete iList[i];
    delete iList;
}

/*****
 *
 * This method computes the intersection between a ray and the
 * polyhedron. It returns the distance between the origin of the ray
 * and the closest point of intersection (or 0.0 if not intersection
 * occurs).
 * The algorithm operates as follows:
 * For each facet of the polyhedron
 *   + determine the intersection between the ray and the plane
 *     supporting the facet.
 *   + if the intersection exists, project the facet on a 2D plane.
 *   + Test if the intersection point is inside the resulting polygon.
 *
 *****/

double Polyhedron::intersect(vec3& ray_org, vec3& ray_dir)
{
    int i, j,
        sNum = 0, // Number of intersections.
        axis, // Axis along which the facet will be projected.
        iNum; // Number of intersections between an imaginary half line
              // and the edges of a facet (used to determine inclusion).
    vec3 normal; // Normal to the current facet.
    vec2 p1, p2, // Two consecutive points on a facet.
        p, // A 2D projection of the point to test.
        n; // Normal to one of the edges of a facet.
    double s[facetN], // Space to store at most N intersections.
        vd,
        vn;

    for (i=0; i<facetN; i++) {
        // Test the case in which the ray is parallel to the facet (vd=0.0)
        vd = vec3(pList[i],PD) * ray_dir;
        if (vd == 0.0)
            continue;

        // Find a projection axis which maximizes the area of the facet
        (normal = vec3(pList[i], PD)).apply(&fabs);
        if (normal[VX] >= normal[VY])
```

```

    if(normal[VX] >= normal[VZ]) axis = VX;
    else axis = VZ;
else if(normal[VY] >= normal[VZ]) axis = VY;
    else axis = VZ;

// Compute the point of intersection between the ray and the facet plane.
// Project it along the desired axis.
vn = pList[i] * vec4(ray_org);
s[sNum] = -vn / vd;
p = vec2(ray_org + s[sNum] * ray_dir, axis);

// Test if the intersection point is inside the facet.
// We do this by checking for each edge of the facet if there is an
// intersection between this edge and a half-line, starting at p and
// going towards positive X. We count the number of intersections: if it
// is odd, the point is inside the facet, if not, it is outside.

iNum = 0;
p1 = vec2(vList[iList[i]][iList[i][0]], axis);
for (j=1; j<= iList[i][0]; j++) {
    p2 = vec2(vList[iList[i]][j], axis);

    // Is p in the band generated by sweeping the [p1p2] segment in the
    // positive X direction ?
    if ((p1[VY] - p[VY]) * (p2[VY] - p[VY]) <= 0.0) {

        // Does the half-line trivially intersect the edge ?
        if (p[VX] < min(p1[VX], p2[VX]))
            iNum++;

        // Does the half-line trivially miss the edge ?
        else if (p[VX] < max(p1[VX], p2[VX])) {

            // tough case: compute the normal to the edge pointing towards
            // positive X use the dot product between this normal and the
            // plp vector to see if the edge and the half line intersect.
            // (This is similar to the Cyrus-Beck line-clipping test)
            n[VX] = p1[VY] - p2[VY];
            n[VY] = p2[VX] - p1[VX];
            if (n[VX] < 0.0)
                n = -n;
            if (n * (p - p1) <= 0.0)
                iNum++;
        }
    }
    p1 = p2;
}
if (iNum % 2)
    sNum++;
}
return closest_intersection(s, sNum);
}

```

```

/*****
 *
 * This method computes the normal vector to the polyhedron at the point
 * of intersection.
 * It does so by finding out which plane the intersection point belongs
 * to. Once this plane is determined, it simply returns the normal to
 * this plane, which is already normalized.
 *
 *****/

```

\*\*\*\*\*/

```
vec3 Polyhedron::normalAt(vec3& p)
{
    int      i,
            index = 0;          // index of the facet being hit by the ray
    double   h,
            hmin = 1e16;        // initialize hmin to some very large number

    for (i=0; i<facetN; i++) {
        h = fabs(pList[i] * vec4(p));
        if (h < hmin) {
            index = i;
            hmin = h;
        }
    }
    return vec3(pList[index], PD);
}
```

```
/*
*
* Input from stream.
*
*****/
```

```
istream& operator >> (istream& s, Polyhedron& a)
{
    int      i,j,M;
    mat4     T;          // local coordinates to world coordinates transformation
    vec3     n;          // normal to the facet

    // call the implementation of the super-class
    s >> *((Primitive*) &a);

    // create the matrix to transform local coordinates to world coordinates
    T = translation3D(a.pos) * (a.orient.transpose());

    // read the vertices. Transform them on the fly to world coordinates
    s >> a.vertexN;
    a.vList = new vec3[a.vertexN];
    for (i = 0; i < a.vertexN; i++) {
        s >> a.vList[i];
        a.vList[i] = T * a.vList[i];
    }

    // read the facets.
    s >> a.facetN;
    a.iList = new int*[a.facetN];
    for (i = 0; i < a.facetN; i++) {
        s >> M;
        a.iList[i] = new int[M+1];
        a.iList[i][0] = M;
        for (j=1; j<=M; j++)
            s >> a.iList[i][j];
    }

    // compute the normal and plane equation of the facet using Newell method
    a.pList = new vec4[a.facetN];
    for (i = 0; i < a.facetN; i++) {
        n = vec3(0.0);
        for (j = 1; j <= a.iList[i][0]; j++)
```

```
        if (j != a.iList[i][0])
            n += a.vList[a.iList[i][j]] ^ a.vList[a.iList[i][j+1]];
        else
            n += a.vList[a.iList[i][j]] ^ a.vList[a.iList[i][1]];
    n.normalize();
    a.pList[i] = vec4(n, - a.vList[a.iList[i][1]] * n);
}
return s;
}
```

```

/*****
*
* CLASS: Polyhedron
* AUTHOR: Jean-Francois DOUE
* LAST MODIFICATION: 12 Oct 1993
*
* This class implements a 3D concave polyhedron.
* The polyhedron is defined by the following elements:
* + an array which contains the its vertices (vList).
* + an array of "facets" (iList). Each facet is itself defined as an
*   array of integer. The first one is the number of points composing
*   the facet. The other ones are the indices of these vertices in the
*   vList array.
* + an array of dimension 4 vectors, which stores the equations of the
*   planes on which the facets lie.
*
*****/

#ifndef Polyhedron_h
#define Polyhedron_h 1
#include "Primitive.h"

class Polyhedron : public Primitive
{
protected:
    vec3    *vList;        // vertices of the polyhedron
    vec4    *pList;        // equations for the planes supporting the facets
    int     vertexN,       // number of vertices of the polyhedron
           facetN,         // number of facets of the polyhedron
           **iList;        // lists of the vertex indices which define the facets

public:

    // Constructors and destructors
    ~Polyhedron();

    // virtual methods
    double intersect(vec3& ray_org, vec3& ray_dir);
    vec3 normalAt(vec3& p);

    // friends
    friend istream& operator >> (istream& s, Polyhedron& a);
};

#endif
```

```
#include "Primitive.h"

Primitive::~Primitive(){};

/*****
 *
 * This method receives as parameter a list of intersections and the
 * number of intersections in the list. It scans the list and returns
 * the closest positive intersection, or 0.0 if no such intersection
 * exists.
 *
 *****/

double Primitive::closest_intersection(double *x, int x_num)
{
    int i;
    double x_min = (x_num) ? x[0] : 0.0;

    for (i=1; i<x_num; i++)
        if (x[i] < x_min)
            x_min = x[i];
    return x_min;
}

/*****
 *
 * Input from stream.
 *
 *****/

istream& operator >> (istream& s, Primitive& a)
{
    s >> *((Object3D*) &a);
    s >> a.col;
    s >> a.ph;
    return s;
}
```



```

/*****
*
* CLASS: Primitive
* AUTHOR: Jean-Francois DOUE
* LAST MODIFICATION: 12 Oct 1993
*
* This class is an abstract class which defines the basic methods
* to which every ray-traceable object (primitive) will have to respond.
* It also stores the color and the phong coefficients of the primitive
*
*****/
```

```
#ifndef Primitive_h
#define Primitive_h 1
#include "Object3D.h"
```

```
class Primitive: public Object3D
{
protected:
    vec3    col;           // color of the primitive
    vec4    ph;            // phong coefficients of the object

public:

    vec3 color() { return col; };
    double phong(int i) { return ph[i]; };
    double closest_intersection(double *x, int x_num);

    // virtual methods
    virtual ~Primitive();
    virtual double intersect(vec3& ray_org, vec3& ray_dir) = 0;
    virtual vec3 normalAt(vec3& p) = 0;

    // friends
    friend istream& operator >> (istream& s, Primitive& a);
};
```

```
#endif
```



<pre> 0.0  0.0  1.0  0.0   0.0  0.0  0.0  1.0   60.0 60.0   </pre>	- field of view (in degrees);
--	-------------------------------

#### Sphere:

text file	comments
<pre> sphere 0.0 0.5 4.0   1.0  0.0  0.0  0.0   0.0  1.0  0.0  0.0   0.0  0.0  1.0  0.0   0.0  0.0  0.0  1.0   0.9 1.0 1.0   0.1 0.48 0.7 20.0   2.0 </pre>	<pre> - keyword; - position of the sphere; - orientation of the sphere;  - color of the sphere; - phong coefficients; - radius; </pre>

#### Polyhedron:

text file	comments
<pre> polyhedron 0.0 0.5 4.0   1.0  0.0  0.0  0.0   0.0  1.0  0.0  0.0   0.0  0.0  1.0  0.0   0.0  0.0  0.0  1.0   0.9 1.0 1.0   0.1 0.48 0.7 20.0   10 -1.0 -1.0 -1.0   1.0 -1.0 -1.0   ..... 0.0  1.8  1.0   7 5 0 3 8 2 1  5 4 5 6 9 7 ..... 4 0 4 7 3 </pre>	<pre> - keyword; - position of the polyhedron; - orientation of the polyhedron;  - color of the polyhedron; - phong coefficients; - number of vertices; - first vertex; - second vertex;  - last vertex; - number of facets; - first facet (number of vertices followed by the index of each vertex in the facet); - second facet;  - last facet; </pre>

#### Light:

text file	comments
<pre> light 0.0 0.5 4.0   1.0  0.0  0.0  0.0   0.0  1.0  0.0  0.0   </pre>	<pre> - keyword; - position of the light; - orientation of the light; </pre>

0.0	0.0	1.0	0.0		
0.0	0.0	0.0	1.0		
0.9	1.0	1.0		- color of the light;	

Comment lines (starting with the '%' sign) can be inserted in the text (between 3D objects only, not within the description of a 3D object !!!).

## APPENDIX

-----

Camera.c	// Implementation of the camera class
Camera.h	// Interface of the camera class
Light.c	// Implementation of the point light class
Light.h	// Interface of the point light class
Makefile	// Unix makefile
Object3D.c	// Implementation of the 3D object class
Object3D.h	// Interface of the 3D object class
Polyhedron.c	// Implementation of the concave polyhedron class
Polyhedron.h	// Interface of the concave polyhedron class
Primitive.c	// Implementation of the primitive class
Primitive.h	// Interface of the primitive class
README	// This file
Scene3D.c	// Implementation of the 3D scene class
Scene3D.h	// Interface of the 3D scene class
Sphere.c	// Implementation of the sphere class
Sphere.h	// Interface of the sphere class
algebra3.c	// The vector and matrix library
algebra3.h	// Include file of the vector and matrix library
main.c	// Main program
example.tiff	// A sample picture rendered with the program
example.data	// The script to render example.tiff
solver.c	// Math utilities by Jochen Schwarze (see Graphics Gems 1 for
solver.h	// more details)

```
#include "Scene3D.h"
#include "Sphere.h"
#include "Polyhedron.h"

#include <string.h>
#include <ctype.h>

#define FAR_AWAY 1e06

/*****
 *
 * Class constructor
 *
 *****/

Scene3D::Scene3D()
{
#define CAPACITY 128

lList = new Light*[CAPACITY];
pList = new Primitive*[CAPACITY];
lightN = primitiveN = 0;
}

/*****
 *
 * Class destructor
 *
 *****/

Scene3D::~Scene3D()
{
int i;

// delete the light sources
for (i = 0; i < lightN; i++)
    delete lList[i];
delete lList;

// delete the primitives
for (i = 0; i < primitiveN; i++)
    delete pList[i];
delete pList;

// delete the camera
delete camera;
}

/*****
 *
 * This method renders the scene using ray-tracing, at a resolution
 * specified by res. The method outputs an array of size
 * res[VX] * res[VY], each pixel being coded on 24 bits (8 bits per
 * color channel). The ray-tracing algorithm is simple and
 * non-recursive: it finds the closest intersection along each ray and
 * shades the corresponding point using Phong's reflection model.
 *
 *****/

char* Scene3D::rayTrace(vec2 res)
{
```

```
int      i, j,
         idx = 0;                // current image data
char     *picture;              // storage for the ray-traced image
vec3     ray_org(camera->position()), // origin of the ray
         ray_dir,                // direction of the ray
         p,                      // point of intersection
         n,                      // normal vector
         l,                      // light vector
         r,                      // reflection vector
         I,                      // light intensity
         p_color;               // color of the current pixel
Primitive *hit;                 // object being intersected
double    u,v,                  // coordinates on the film plane
         t, t_min,              // distance to the closest object
         d;
```

```
// allocate enough room for a 24 bits picture
picture = new char[(int)res[VX] * (int)res[VY] * 3];
```

```
// for each point in the picture
for (v = 1.; v > -1.; v -= 2. / (res[VY] - 1.))
    for (u = -1.; u < 1.; u += 2. / (res[VX] - 1.)) {
        // compute the direction of the ray through that point
        ray_dir = camera->pointToRay(vec2(u,v));
```

```
        // find the intersection with the closest primitive
        t_min = FAR_AWAY;
        for (i=0; i<primitiveN; i++)
            if ((t = pList[i]->intersect(ray_org, ray_dir)) && t < t_min) {
                t_min = t;
                hit = pList[i];
            }
```

```
        // shade the intersection point
        if (t_min < FAR_AWAY) {
            I = vec3(0.0);
            p = ray_org + t_min * ray_dir;
            n = hit->normalAt(p);
```

```
        // for all the light sources in the scene
        for (j=0; j<lightN; j++) {

            // compute the light vector and the reflection vector
            l = (lList[j]->position() - p).normalize();
            r = 2.0 * (n * l) * n - l;
```

```
            // add the diffuse component
            if ((d = n * l) > 0.0)
                I += hit->phong(KD) * d * lList[j]->color();
```

```
            // add the specular component
            if ((d = -ray_dir * r) > 0.0)
                I += hit->phong(KS) * pow(d, hit->phong(ES))
                    * lList[j]->color();
        }
```

```
        p_color = min(255.0 * prod(I, hit->color()), vec3(255.0));
    }
```

```
else
    p_color = vec3(0.0);
picture[idx++] = (int) p_color[RED];
picture[idx++] = (int) p_color[GREEN];
```

```
        picture[idx++] = (int) p_color[BLUE];
    }
return picture;
}

/*****
 *
 * This function converts vocabulary words specified in a vocabulary
 * table into integers equal to the index of the word in the table.
 * It automatically skips tabs, comment characters '%'.
 *
 *****/

int parseStream(istream& s, char** vocabulary)
{
    int    i;
    char    c,
            string[1024];

    // skip white space and comments
    while(s.get(c) && (isspace(c) || c == '%'))
        if (c == '%') // allow comment lines starting with the percent sign
            while(s.get(c) && c != '\n') ;
    s.putback(c);

    // read the text string and force it to uppercase
    strcpy(string, "");
    s >> string;
    for (i=0; i< strlen(string); i++)
        string[i] = toupper(string[i]);

    // If the string is in the dictionary, return its index
    i = 0;
    while(vocabulary[i]) {
        if (strcmp(vocabulary[i], string) == 0)
            break;
        i++;
    }
    return i+1;
}

/*****
 *
 * Input from stream.
 *
 *****/

istream& operator >> (istream& s, Scene3D& a)
{
    enum {CAMERA=1,
          LIGHT=2,
          SPHERE=3,
          POLYHEDRON=4,
          UNKNOWN=5};
    int    key;
    static char *key_words[] = {"CAMERA",
                                "LIGHT",
                                "SPHERE",
                                "POLYHEDRON",
                                0};
```

```
while ((key = parseStream(s, key_words)) && !s.eof())
    switch(key) {
        case CAMERA:
            a.camera = new Camera;
            s >> *a.camera;
            break;

        case LIGHT:
            Light *light = new Light;
            a.lList[a.lightN++] = light;
            s >> *light;
            break;

        case SPHERE:
            Sphere *sphere = new Sphere;
            a.pList[a.primitiveN++] = sphere;
            s >> *sphere;
            break;

        case POLYHEDRON:
            Polyhedron *polyhedron = new Polyhedron;
            a.pList[a.primitiveN++] = polyhedron;
            s >> *polyhedron;
            break;

        case UNKNOWN:
            cerr << "\nScene 3D: corrupted stream passed to operator >>";
            return s;
            break;
    }

return s;
}
```



```

/*****
 *
 * CLASS: Scene3D
 * AUTHOR: Jean-Francois DOUE
 * LAST MODIFICATION: 12 Oct 1993
 *
 * This class implements a 3D scene.
 * The scene is composed of a camera, a collection of light sources
 * and a collection of primitives to render.
 * The scene is capable of ray-tracing itself.
 * Scenes are typically parsed from a text file description. (see the
 * README file for a description of the format).
 *
 *****/

#include "Camera.h"
#include "Light.h"
#include "Primitive.h"

class Scene3D
{
protected:
    Camera      *camera;          // a camera
    Light       **lList;          // a list of point lights
    Primitive   **pList;          // a list of primitives
    int         lightN,           // number of light sources in the list
              primitiveN;        // number of primitives in the scene

public:
    Scene3D();
    ~Scene3D();
    char* rayTrace(vec2 res);

    // friends
    friend istream& operator >> (istream& s, Scene3D& a);
};
```

```
#include "Sphere.h"
#include "solver.h"

/*****
 *
 * This method computes the intersection between a ray and the sphere.
 * It returns the distance between the origin of the ray and the closest
 * point of intersection (or 0.0 if no intersection occurs).
 *
 *****/

double Sphere::intersect(vec3& ray_org, vec3& ray_dir)
{
    double    c[3],                // coefficients of the quadric equation.
              s[2];                // solutions of the quadric equation
    vec3      d = ray_org - pos;    // vector from the center of the sphere to
                                   // the origin of the ray.

    // compute the coefficients of the resolvent equation
    c[2] = 1.0;
    c[1] = 2.0 * ray_dir * d;
    c[0] = d * d - radius * radius;

    // return the closest intersection point
    return closest_intersection(s, solveQuadric(c, s));
}

/*****
 *
 * This method computes the normal vector to the sphere at the point of
 * intersection. (NB: normalization is "built-in" for the sphere).
 *
 *****/

vec3 Sphere::normalAt(vec3& p)
{ return (p - pos) / radius; }

/*****
 *
 * Input from stream.
 *
 *****/

istream& operator >> (istream& s, Sphere& a)
{
    s >> *((Primitive*) &a);
    s >> a.radius;
    return s;
}
```

```

/*****
*
* CLASS: Sphere
* AUTHOR: Jean-Francois DOUE
* LAST MODIFICATION: 12 Oct 1993
*
* This class implements a sphere.
*
*****/
```

```
#ifndef Sphere_h
#define Sphere_h 1
#include "Primitive.h"

class Sphere : public Primitive
{
protected:
    double radius;

public:

    // virtual methods
    double intersect(vec3& ray_org, vec3& ray_dir);
    vec3 normalAt(vec3& p);

    // friends
    friend istream& operator >> (istream& s, Sphere& a);
};

#endif
```

```
#include "algebra3.h"
#include <ctype.h>

/*****
*
*          vec2 Member functions
*
*****/

// CONSTRUCTORS

vec2::vec2() {}

vec2::vec2(const double x, const double y)
{ n[VX] = x; n[VY] = y; }

vec2::vec2(const double d)
{ n[VX] = n[VY] = d; }

vec2::vec2(const vec2& v)
{ n[VX] = v.n[VX]; n[VY] = v.n[VY]; }

vec2::vec2(const vec3& v) // it is up to caller to avoid divide-by-zero
{ n[VX] = v.n[VX]/v.n[VZ]; n[VY] = v.n[VY]/v.n[VZ]; };

vec2::vec2(const vec3& v, int dropAxis) {
    switch (dropAxis) {
        case VX: n[VX] = v.n[VY]; n[VY] = v.n[VZ]; break;
        case VY: n[VX] = v.n[VX]; n[VY] = v.n[VZ]; break;
        default: n[VX] = v.n[VX]; n[VY] = v.n[VY]; break;
    }
}

// ASSIGNMENT OPERATORS

vec2& vec2::operator = (const vec2& v)
{ n[VX] = v.n[VX]; n[VY] = v.n[VY]; return *this; }

vec2& vec2::operator += ( const vec2& v )
{ n[VX] += v.n[VX]; n[VY] += v.n[VY]; return *this; }

vec2& vec2::operator -= ( const vec2& v )
{ n[VX] -= v.n[VX]; n[VY] -= v.n[VY]; return *this; }

vec2& vec2::operator *= ( const double d )
{ n[VX] *= d; n[VY] *= d; return *this; }

vec2& vec2::operator /= ( const double d )
{ double d_inv = 1./d; n[VX] *= d_inv; n[VY] *= d_inv; return *this; }

double& vec2::operator [] ( int i) {
    if (i < VX || i > VY)
        V_ERROR("vec2 [] operator: illegal access; index = " << i << '\n')
    return n[i];
}

// SPECIAL FUNCTIONS

double vec2::length()
```

```
{ return sqrt(length2()); }

double vec2::length2()
{ return n[VX]*n[VX] + n[VY]*n[VY]; }

vec2& vec2::normalize() // it is up to caller to avoid divide-by-zero
{ *this /= length(); return *this; }

vec2& vec2::apply(V_FCT_PTR fct)
{ n[VX] = (*fct)(n[VX]); n[VY] = (*fct)(n[VY]); return *this; }

// FRIENDS

vec2 operator - (const vec2& a)
{ return vec2(-a.n[VX], -a.n[VY]); }

vec2 operator + (const vec2& a, const vec2& b)
{ return vec2(a.n[VX]+ b.n[VX], a.n[VY] + b.n[VY]); }

vec2 operator - (const vec2& a, const vec2& b)
{ return vec2(a.n[VX]-b.n[VX], a.n[VY]-b.n[VY]); }

vec2 operator * (const vec2& a, const double d)
{ return vec2(d*a.n[VX], d*a.n[VY]); }

vec2 operator * (const double d, const vec2& a)
{ return a*d; }

vec2 operator * (const mat3& a, const vec2& v) {
    vec3 av;

    av.n[VX] = a.v[0].n[VX]*v.n[VX] + a.v[0].n[VY]*v.n[VY] + a.v[0].n[VZ];
    av.n[VY] = a.v[1].n[VX]*v.n[VX] + a.v[1].n[VY]*v.n[VY] + a.v[1].n[VZ];
    av.n[VZ] = a.v[2].n[VX]*v.n[VX] + a.v[2].n[VY]*v.n[VY] + a.v[2].n[VZ];
    return av;
}

vec2 operator * (const vec2& v, mat3& a)
{ return a.transpose() * v; }

double operator * (const vec2& a, const vec2& b)
{ return (a.n[VX]*b.n[VX] + a.n[VY]*b.n[VY]); }

vec2 operator / (const vec2& a, const double d)
{ double d_inv = 1./d; return vec2(a.n[VX]*d_inv, a.n[VY]*d_inv); }

vec3 operator ^ (const vec2& a, const vec2& b)
{ return vec3(0.0, 0.0, a.n[VX] * b.n[VY] - b.n[VX] * a.n[VY]); }

int operator == (const vec2& a, const vec2& b)
{ return (a.n[VX] == b.n[VX]) && (a.n[VY] == b.n[VY]); }

int operator != (const vec2& a, const vec2& b)
{ return !(a == b); }

ostream& operator << (ostream& s, vec2& v)
{ return s << "|" << v.n[VX] << " " << v.n[VY] << "|"; }

istream& operator >> (istream& s, vec2& v) {
    vec2      v_tmp;
```

```
char      c = ' ';

while (isspace(c))
    s >> c;
// The vectors can be formatted either as x y or | x y |
if (c == '|') {
    s >> v_tmp[VX] >> v_tmp[VY];
    while (s >> c && isspace(c)) ;
    if (c != '|')
        s.set(_bad);
}
else {
    s.putback(c);
    s >> v_tmp[VX] >> v_tmp[VY];
}
if (s)
    v = v_tmp;
return s;
}

void swap(vec2& a, vec2& b)
{ vec2 tmp(a); a = b; b = tmp; }

vec2 min(const vec2& a, const vec2& b)
{ return vec2(MIN(a.n[VX], b.n[VX]), MIN(a.n[VY], b.n[VY])); }

vec2 max(const vec2& a, const vec2& b)
{ return vec2(MAX(a.n[VX], b.n[VX]), MAX(a.n[VY], b.n[VY])); }

vec2 prod(const vec2& a, const vec2& b)
{ return vec2(a.n[VX] * b.n[VX], a.n[VY] * b.n[VY]); }

/*****
*
*          vec3 Member functions
*
*****/

// CONSTRUCTORS

vec3::vec3() {}

vec3::vec3(const double x, const double y, const double z)
{ n[VX] = x; n[VY] = y; n[VZ] = z; }

vec3::vec3(const double d)
{ n[VX] = n[VY] = n[VZ] = d; }

vec3::vec3(const vec3& v)
{ n[VX] = v.n[VX]; n[VY] = v.n[VY]; n[VZ] = v.n[VZ]; }

vec3::vec3(const vec2& v)
{ n[VX] = v.n[VX]; n[VY] = v.n[VY]; n[VZ] = 1.0; }

vec3::vec3(const vec2& v, double d)
{ n[VX] = v.n[VX]; n[VY] = v.n[VY]; n[VZ] = d; }

vec3::vec3(const vec4& v) // it is up to caller to avoid divide-by-zero
{ n[VX] = v.n[VX] / v.n[VW]; n[VY] = v.n[VY] / v.n[VW];
  n[VZ] = v.n[VZ] / v.n[VW]; }
```

```
vec3::vec3(const vec4& v, int dropAxis) {
    switch (dropAxis) {
        case VX: n[VX] = v.n[VY]; n[VY] = v.n[VZ]; n[VZ] = v.n[VW]; break;
        case VY: n[VX] = v.n[VX]; n[VY] = v.n[VZ]; n[VZ] = v.n[VW]; break;
        case VZ: n[VX] = v.n[VX]; n[VY] = v.n[VY]; n[VZ] = v.n[VW]; break;
        default: n[VX] = v.n[VX]; n[VY] = v.n[VY]; n[VZ] = v.n[VZ]; break;
    }
}
```

// ASSIGNMENT OPERATORS

```
vec3& vec3::operator = (const vec3& v)
{ n[VX] = v.n[VX]; n[VY] = v.n[VY]; n[VZ] = v.n[VZ]; return *this; }

vec3& vec3::operator += ( const vec3& v )
{ n[VX] += v.n[VX]; n[VY] += v.n[VY]; n[VZ] += v.n[VZ]; return *this; }

vec3& vec3::operator -= ( const vec3& v )
{ n[VX] -= v.n[VX]; n[VY] -= v.n[VY]; n[VZ] -= v.n[VZ]; return *this; }

vec3& vec3::operator *= ( const double d )
{ n[VX] *= d; n[VY] *= d; n[VZ] *= d; return *this; }

vec3& vec3::operator /= ( const double d )
{ double d_inv = 1./d; n[VX] *= d_inv; n[VY] *= d_inv; n[VZ] *= d_inv;
  return *this; }

double& vec3::operator [] ( int i) {
    if (i < VX || i > VZ)
        V_ERROR("vec3 [] operator: illegal access; index = " << i << '\n')
    return n[i];
}
```

// SPECIAL FUNCTIONS

```
double vec3::length()
{ return sqrt(length2()); }

double vec3::length2()
{ return n[VX]*n[VX] + n[VY]*n[VY] + n[VZ]*n[VZ]; }

vec3& vec3::normalize() // it is up to caller to avoid divide-by-zero
{ *this /= length(); return *this; }

vec3& vec3::apply(V_FCT_PTR fct)
{ n[VX] = (*fct)(n[VX]); n[VY] = (*fct)(n[VY]); n[VZ] = (*fct)(n[VZ]);
  return *this; }
```

// FRIENDS

```
vec3 operator - (const vec3& a)
{ return vec3(-a.n[VX],-a.n[VY],-a.n[VZ]); }

vec3 operator + (const vec3& a, const vec3& b)
{ return vec3(a.n[VX]+ b.n[VX], a.n[VY] + b.n[VY], a.n[VZ] + b.n[VZ]); }

vec3 operator - (const vec3& a, const vec3& b)
{ return vec3(a.n[VX]-b.n[VX], a.n[VY]-b.n[VY], a.n[VZ]-b.n[VZ]); }
```

```
vec3 operator * (const vec3& a, const double d)
{ return vec3(d*a.n[VX], d*a.n[VY], d*a.n[VZ]); }

vec3 operator * (const double d, const vec3& a)
{ return a*d; }

vec3 operator * (const mat4& a, const vec3& v)
{ return a * vec4(v); }

vec3 operator * (const vec3& v, mat4& a)
{ return a.transpose() * v; }

double operator * (const vec3& a, const vec3& b)
{ return (a.n[VX]*b.n[VX] + a.n[VY]*b.n[VY] + a.n[VZ]*b.n[VZ]); }

vec3 operator / (const vec3& a, const double d)
{ double d_inv = 1./d; return vec3(a.n[VX]*d_inv, a.n[VY]*d_inv,
  a.n[VZ]*d_inv); }

vec3 operator ^ (const vec3& a, const vec3& b) {
    return vec3(a.n[VY]*b.n[VZ] - a.n[VZ]*b.n[VY],
                a.n[VZ]*b.n[VX] - a.n[VX]*b.n[VZ],
                a.n[VX]*b.n[VY] - a.n[VY]*b.n[VX]);
}

int operator == (const vec3& a, const vec3& b)
{ return (a.n[VX] == b.n[VX]) && (a.n[VY] == b.n[VY]) && (a.n[VZ] == b.n[VZ]); }

int operator != (const vec3& a, const vec3& b)
{ return !(a == b); }

ostream& operator << (ostream& s, vec3& v)
{ return s << "|" << v.n[VX] << " " << v.n[VY] << " " << v.n[VZ] << " |"; }

istream& operator >> (istream& s, vec3& v) {
    vec3      v_tmp;
    char      c = ' ';

    while (isspace(c))
        s >> c;
    // The vectors can be formatted either as x y z or | x y z |
    if (c == '|') {
        s >> v_tmp[VX] >> v_tmp[VY] >> v_tmp[VZ];
        while (s >> c && isspace(c)) ;
        if (c != '|')
            s.set(_bad);
    }
    else {
        s.putback(c);
        s >> v_tmp[VX] >> v_tmp[VY] >> v_tmp[VZ];
    }
    if (s)
        v = v_tmp;
    return s;
}

void swap(vec3& a, vec3& b)
{ vec3 tmp(a); a = b; b = tmp; }
```



```
vec3 min(const vec3& a, const vec3& b)
{ return vec3(MIN(a.n[VX], b.n[VX]), MIN(a.n[VY], b.n[VY]), MIN(a.n[VZ],
  b.n[VZ])); }

vec3 max(const vec3& a, const vec3& b)
{ return vec3(MAX(a.n[VX], b.n[VX]), MAX(a.n[VY], b.n[VY]), MAX(a.n[VZ],
  b.n[VZ])); }

vec3 prod(const vec3& a, const vec3& b)
{ return vec3(a.n[VX] * b.n[VX], a.n[VY] * b.n[VY], a.n[VZ] * b.n[VZ]); }

/*****
*
*                               *
*               vec4 Member functions               *
*
*                               *
*****/

// CONSTRUCTORS

vec4::vec4() {}

vec4::vec4(const double x, const double y, const double z, const double w)
{ n[VX] = x; n[VY] = y; n[VZ] = z; n[VW] = w; }

vec4::vec4(const double d)
{ n[VX] = n[VY] = n[VZ] = n[VW] = d; }

vec4::vec4(const vec4& v)
{ n[VX] = v.n[VX]; n[VY] = v.n[VY]; n[VZ] = v.n[VZ]; n[VW] = v.n[VW]; }

vec4::vec4(const vec3& v)
{ n[VX] = v.n[VX]; n[VY] = v.n[VY]; n[VZ] = v.n[VZ]; n[VW] = 1.0; }

vec4::vec4(const vec3& v, const double d)
{ n[VX] = v.n[VX]; n[VY] = v.n[VY]; n[VZ] = v.n[VZ]; n[VW] = d; }

// ASSIGNMENT OPERATORS

vec4& vec4::operator = (const vec4& v)
{ n[VX] = v.n[VX]; n[VY] = v.n[VY]; n[VZ] = v.n[VZ]; n[VW] = v.n[VW];
return *this; }

vec4& vec4::operator += ( const vec4& v )
{ n[VX] += v.n[VX]; n[VY] += v.n[VY]; n[VZ] += v.n[VZ]; n[VW] += v.n[VW];
return *this; }

vec4& vec4::operator -= ( const vec4& v )
{ n[VX] -= v.n[VX]; n[VY] -= v.n[VY]; n[VZ] -= v.n[VZ]; n[VW] -= v.n[VW];
return *this; }

vec4& vec4::operator *= ( const double d )
{ n[VX] *= d; n[VY] *= d; n[VZ] *= d; n[VW] *= d; return *this; }

vec4& vec4::operator /= ( const double d )
{ double d_inv = 1./d; n[VX] *= d_inv; n[VY] *= d_inv; n[VZ] *= d_inv;
  n[VW] *= d_inv; return *this; }

double& vec4::operator [] ( int i ) {
  if (i < VX || i > VW)
```

```
        V_ERROR("vec4 [] operator: illegal access; index = " << i << '\n')
    return n[i];
}

// SPECIAL FUNCTIONS

double vec4::length()
{ return sqrt(length2()); }

double vec4::length2()
{ return n[VX]*n[VX] + n[VY]*n[VY] + n[VZ]*n[VZ] + n[VW]*n[VW]; }

vec4& vec4::normalize() // it is up to caller to avoid divide-by-zero
{ *this /= length(); return *this; }

vec4& vec4::apply(V_FCT_PTR fct)
{ n[VX] = (*fct)(n[VX]); n[VY] = (*fct)(n[VY]); n[VZ] = (*fct)(n[VZ]);
  n[VW] = (*fct)(n[VW]); return *this; }

// FRIENDS

vec4 operator - (const vec4& a)
{ return vec4(-a.n[VX],-a.n[VY],-a.n[VZ],-a.n[VW]); }

vec4 operator + (const vec4& a, const vec4& b)
{ return vec4(a.n[VX] + b.n[VX], a.n[VY] + b.n[VY], a.n[VZ] + b.n[VZ],
  a.n[VW] + b.n[VW]); }

vec4 operator - (const vec4& a, const vec4& b)
{ return vec4(a.n[VX] - b.n[VX], a.n[VY] - b.n[VY], a.n[VZ] - b.n[VZ],
  a.n[VW] - b.n[VW]); }

vec4 operator * (const vec4& a, const double d)
{ return vec4(d*a.n[VX], d*a.n[VY], d*a.n[VZ], d*a.n[VW] ); }

vec4 operator * (const double d, const vec4& a)
{ return a*d; }

vec4 operator * (const mat4& a, const vec4& v) {
    #define ROWCOL(i) a.v[i].n[0]*v.n[VX] + a.v[i].n[1]*v.n[VY] \
    + a.v[i].n[2]*v.n[VZ] + a.v[i].n[3]*v.n[VW]
    return vec4(ROWCOL(0), ROWCOL(1), ROWCOL(2), ROWCOL(3));
    #undef ROWCOL
}

vec4 operator * (const vec4& v, mat4& a)
{ return a.transpose() * v; }

double operator * (const vec4& a, const vec4& b)
{ return (a.n[VX]*b.n[VX] + a.n[VY]*b.n[VY] + a.n[VZ]*b.n[VZ] +
  a.n[VW]*b.n[VW]); }

vec4 operator / (const vec4& a, const double d)
{ double d_inv = 1./d; return vec4(a.n[VX]*d_inv, a.n[VY]*d_inv, a.n[VZ]*d_inv,
  a.n[VW]*d_inv); }

int operator == (const vec4& a, const vec4& b)
{ return (a.n[VX] == b.n[VX]) && (a.n[VY] == b.n[VY]) && (a.n[VZ] == b.n[VZ])
  && (a.n[VW] == b.n[VW]); }
```

```
int operator != (const vec4& a, const vec4& b)
{ return !(a == b); }

ostream& operator << (ostream& s, vec4& v)
{ return s << "|" << v.n[VX] << " " << v.n[VY] << " " << v.n[VZ] << " "
  << v.n[VW] << " |"; }

istream& operator >> (istream& s, vec4& v) {
    vec4      v_tmp;
    char      c = ' ';

    while (isspace(c))
        s >> c;
    // The vectors can be formatted either as x y z w or | x y z w |
    if (c == '|') {
        s >> v_tmp[VX] >> v_tmp[VY] >> v_tmp[VZ] >> v_tmp[VW];
        while (s >> c && isspace(c)) ;
        if (c != '|')
            s.set(_bad);
    }
    else {
        s.putback(c);
        s >> v_tmp[VX] >> v_tmp[VY] >> v_tmp[VZ] >> v_tmp[VW];
    }
    if (s)
        v = v_tmp;
    return s;
}

void swap(vec4& a, vec4& b)
{ vec4 tmp(a); a = b; b = tmp; }

vec4 min(const vec4& a, const vec4& b)
{ return vec4(MIN(a.n[VX], b.n[VX]), MIN(a.n[VY], b.n[VY]), MIN(a.n[VZ],
  b.n[VZ]), MIN(a.n[VW], b.n[VW])); }

vec4 max(const vec4& a, const vec4& b)
{ return vec4(MAX(a.n[VX], b.n[VX]), MAX(a.n[VY], b.n[VY]), MAX(a.n[VZ],
  b.n[VZ]), MAX(a.n[VW], b.n[VW])); }

vec4 prod(const vec4& a, const vec4& b)
{ return vec4(a.n[VX] * b.n[VX], a.n[VY] * b.n[VY], a.n[VZ] * b.n[VZ],
  a.n[VW] * b.n[VW]); }

/*****
*
*                               *
*               mat3 member functions                               *
*                               *
*****/

// CONSTRUCTORS

mat3::mat3() {}

mat3::mat3(const vec3& v0, const vec3& v1, const vec3& v2)
{ v[0] = v0; v[1] = v1; v[2] = v2; }

mat3::mat3(const double d)
{ v[0] = v[1] = v[2] = vec3(d); }
```

```
mat3::mat3(const mat3& m)
{ v[0] = m.v[0]; v[1] = m.v[1]; v[2] = m.v[2]; }

// ASSIGNMENT OPERATORS

mat3& mat3::operator = ( const mat3& m )
{ v[0] = m.v[0]; v[1] = m.v[1]; v[2] = m.v[2]; return *this; }

mat3& mat3::operator += ( const mat3& m )
{ v[0] += m.v[0]; v[1] += m.v[1]; v[2] += m.v[2]; return *this; }

mat3& mat3::operator -= ( const mat3& m )
{ v[0] -= m.v[0]; v[1] -= m.v[1]; v[2] -= m.v[2]; return *this; }

mat3& mat3::operator *= ( const double d )
{ v[0] *= d; v[1] *= d; v[2] *= d; return *this; }

mat3& mat3::operator /= ( const double d )
{ v[0] /= d; v[1] /= d; v[2] /= d; return *this; }

vec3& mat3::operator [] ( int i ) {
    if (i < VX || i > VZ)
        V_ERROR("mat3 [] operator: illegal access; index = " << i << '\n')
    return v[i];
}

// SPECIAL FUNCTIONS

mat3 mat3::transpose() {
    return mat3(vec3(v[0][0], v[1][0], v[2][0]),
                vec3(v[0][1], v[1][1], v[2][1]),
                vec3(v[0][2], v[1][2], v[2][2]));
}

mat3 mat3::inverse()          // Gauss-Jordan elimination with partial pivoting
{
    mat3 a(*this),            // As a evolves from original mat into identity
        b(identity2D());      // b evolves from identity into inverse(a)
    int i, j, il;

    // Loop over cols of a from left to right, eliminating above and below diag
    for (j=0; j<3; j++) {     // Find largest pivot in column j among rows j..2
        il = j;                // Row with largest pivot candidate
        for (i=j+1; i<3; i++)
            if (fabs(a.v[i].n[j]) > fabs(a.v[il].n[j]))
                il = i;

        // Swap rows il and j in a and b to put pivot on diagonal
        swap(a.v[il], a.v[j]);
        swap(b.v[il], b.v[j]);

        // Scale row j to have a unit diagonal
        if (a.v[j].n[j]==0.)
            V_ERROR("mat3::inverse: singular matrix; can't invert\n")
        b.v[j] /= a.v[j].n[j];
        a.v[j] /= a.v[j].n[j];

        // Eliminate off-diagonal elems in col j of a, doing identical ops to b
```

```
    for (i=0; i<3; i++)
        if (i!=j) {
            b.v[i] -= a.v[i].n[j]*b.v[j];
            a.v[i] -= a.v[i].n[j]*a.v[j];
        }
    return b;
}

mat3& mat3::apply(V_FCT_PTR fct) {
    v[VX].apply(fct);
    v[VY].apply(fct);
    v[VZ].apply(fct);
    return *this;
}

// FRIENDS

mat3 operator - (const mat3& a)
{ return mat3(-a.v[0], -a.v[1], -a.v[2]); }

mat3 operator + (const mat3& a, const mat3& b)
{ return mat3(a.v[0] + b.v[0], a.v[1] + b.v[1], a.v[2] + b.v[2]); }

mat3 operator - (const mat3& a, const mat3& b)
{ return mat3(a.v[0] - b.v[0], a.v[1] - b.v[1], a.v[2] - b.v[2]); }

mat3 operator * (mat3& a, mat3& b) {
    #define ROWCOL(i, j) \
        a.v[i].n[0]*b.v[0][j] + a.v[i].n[1]*b.v[1][j] + a.v[i].n[2]*b.v[2][j]
    return mat3(vec3(ROWCOL(0,0), ROWCOL(0,1), ROWCOL(0,2)),
                vec3(ROWCOL(1,0), ROWCOL(1,1), ROWCOL(1,2)),
                vec3(ROWCOL(2,0), ROWCOL(2,1), ROWCOL(2,2)));
    #undef ROWCOL
}

mat3 operator * (const mat3& a, const double d)
{ return mat3(a.v[0] * d, a.v[1] * d, a.v[2] * d); }

mat3 operator * (const double d, const mat3& a)
{ return a*d; }

mat3 operator / (const mat3& a, const double d)
{ return mat3(a.v[0] / d, a.v[1] / d, a.v[2] / d); }

int operator == (const mat3& a, const mat3& b)
{ return (a.v[0] == b.v[0]) && (a.v[1] == b.v[1]) && (a.v[2] == b.v[2]); }

int operator != (const mat3& a, const mat3& b)
{ return !(a == b); }

ostream& operator << (ostream& s, mat3& m)
{ return s << m.v[VX] << '\n' << m.v[VY] << '\n' << m.v[VZ]; }

istream& operator >> (istream& s, mat3& m) {
    mat3    m_tmp;

    s >> m_tmp[VX] >> m_tmp[VY] >> m_tmp[VZ];
    if (s)
        m = m_tmp;
}
```

```
        return s;
    }

void swap(mat3& a, mat3& b)
{ mat3 tmp(a); a = b; b = tmp; }

/*****
*
*          mat4 member functions
*
*****/

// CONSTRUCTORS

mat4::mat4() {}

mat4::mat4(const vec4& v0, const vec4& v1, const vec4& v2, const vec4& v3)
{ v[0] = v0; v[1] = v1; v[2] = v2; v[3] = v3; }

mat4::mat4(const double d)
{ v[0] = v[1] = v[2] = v[3] = vec4(d); }

mat4::mat4(const mat4& m)
{ v[0] = m.v[0]; v[1] = m.v[1]; v[2] = m.v[2]; v[3] = m.v[3]; }

// ASSIGNMENT OPERATORS

mat4& mat4::operator = ( const mat4& m )
{ v[0] = m.v[0]; v[1] = m.v[1]; v[2] = m.v[2]; v[3] = m.v[3];
return *this; }

mat4& mat4::operator += ( const mat4& m )
{ v[0] += m.v[0]; v[1] += m.v[1]; v[2] += m.v[2]; v[3] += m.v[3];
return *this; }

mat4& mat4::operator -= ( const mat4& m )
{ v[0] -= m.v[0]; v[1] -= m.v[1]; v[2] -= m.v[2]; v[3] -= m.v[3];
return *this; }

mat4& mat4::operator *= ( const double d )
{ v[0] *= d; v[1] *= d; v[2] *= d; v[3] *= d; return *this; }

mat4& mat4::operator /= ( const double d )
{ v[0] /= d; v[1] /= d; v[2] /= d; v[3] /= d; return *this; }

vec4& mat4::operator [] ( int i ) {
    if (i < VX || i > VW)
        V_ERROR("mat4 [] operator: illegal access; index = " << i << '\n')
    return v[i];
}

// SPECIAL FUNCTIONS;

mat4 mat4::transpose() {
    return mat4(vec4(v[0][0], v[1][0], v[2][0], v[3][0]),
                vec4(v[0][1], v[1][1], v[2][1], v[3][1]),
                vec4(v[0][2], v[1][2], v[2][2], v[3][2]),
                vec4(v[0][3], v[1][3], v[2][3], v[3][3]));
}
```

```

mat4 mat4::inverse()          // Gauss-Jordan elimination with partial pivoting
{
    mat4 a(*this),            // As a evolves from original mat into identity
        b(identity3D());      // b evolves from identity into inverse(a)
    int i, j, il;

    // Loop over cols of a from left to right, eliminating above and below diag
    for (j=0; j<4; j++) {      // Find largest pivot in column j among rows j..3
        il = j;                // Row with largest pivot candidate
        for (i=j+1; i<4; i++)
            if (fabs(a.v[i].n[j]) > fabs(a.v[il].n[j]))
                il = i;

        // Swap rows il and j in a and b to put pivot on diagonal
        swap(a.v[il], a.v[j]);
        swap(b.v[il], b.v[j]);

        // Scale row j to have a unit diagonal
        if (a.v[j].n[j]==0.)
            V_ERROR("mat4::inverse: singular matrix; can't invert\n");
        b.v[j] /= a.v[j].n[j];
        a.v[j] /= a.v[j].n[j];

        // Eliminate off-diagonal elems in col j of a, doing identical ops to b
        for (i=0; i<4; i++)
            if (i!=j) {
                b.v[i] -= a.v[i].n[j]*b.v[j];
                a.v[i] -= a.v[i].n[j]*a.v[j];
            }
    }
    return b;
}

mat4& mat4::apply(V_FCT_PTR fct)
{ v[VX].apply(fct); v[VY].apply(fct); v[VZ].apply(fct); v[VW].apply(fct);
return *this; }

// FRIENDS

mat4 operator - (const mat4& a)
{ return mat4(-a.v[0], -a.v[1], -a.v[2], -a.v[3]); }

mat4 operator + (const mat4& a, const mat4& b)
{ return mat4(a.v[0] + b.v[0], a.v[1] + b.v[1], a.v[2] + b.v[2],
    a.v[3] + b.v[3]);
}

mat4 operator - (const mat4& a, const mat4& b)
{ return mat4(a.v[0] - b.v[0], a.v[1] - b.v[1], a.v[2] - b.v[2], a.v[3] - b.v[3]); }

mat4 operator * (mat4& a, mat4& b) {
    #define ROWCOL(i, j) a.v[i].n[0]*b.v[0][j] + a.v[i].n[1]*b.v[1][j] + \
        a.v[i].n[2]*b.v[2][j] + a.v[i].n[3]*b.v[3][j]
    return mat4(
        vec4(ROWCOL(0,0), ROWCOL(0,1), ROWCOL(0,2), ROWCOL(0,3)),
        vec4(ROWCOL(1,0), ROWCOL(1,1), ROWCOL(1,2), ROWCOL(1,3)),
        vec4(ROWCOL(2,0), ROWCOL(2,1), ROWCOL(2,2), ROWCOL(2,3)),
        vec4(ROWCOL(3,0), ROWCOL(3,1), ROWCOL(3,2), ROWCOL(3,3))
    );
}

```

```
}

mat4 operator * (const mat4& a, const double d)
{ return mat4(a.v[0] * d, a.v[1] * d, a.v[2] * d, a.v[3] * d); }

mat4 operator * (const double d, const mat4& a)
{ return a*d; }

mat4 operator / (const mat4& a, const double d)
{ return mat4(a.v[0] / d, a.v[1] / d, a.v[2] / d, a.v[3] / d); }

int operator == (const mat4& a, const mat4& b)
{ return ((a.v[0] == b.v[0]) && (a.v[1] == b.v[1]) && (a.v[2] == b.v[2]) &&
(a.v[3] == b.v[3])); }

int operator != (const mat4& a, const mat4& b)
{ return !(a == b); }

ostream& operator << (ostream& s, mat4& m)
{ return s << m.v[VX] << '\n' << m.v[VY] << '\n' << m.v[VZ] << '\n' << m.v[VW]; }

istream& operator >> (istream& s, mat4& m)
{
    mat4    m_tmp;

    s >> m_tmp[VX] >> m_tmp[VY] >> m_tmp[VZ] >> m_tmp[VW];
    if (s)
        m = m_tmp;
    return s;
}

void swap(mat4& a, mat4& b)
{ mat4 tmp(a); a = b; b = tmp; }

/*****
*
*          2D functions and 3D functions
*
*****/

mat3 identity2D()
{ return mat3(vec3(1.0, 0.0, 0.0),
               vec3(0.0, 1.0, 0.0),
               vec3(0.0, 0.0, 1.0)); }

mat3 translation2D(vec2& v)
{ return mat3(vec3(1.0, 0.0, v[VX]),
               vec3(0.0, 1.0, v[VY]),
               vec3(0.0, 0.0, 1.0)); }

mat3 rotation2D(vec2& Center, const double angleDeg) {
    double  angleRad = angleDeg * M_PI / 180.0,
           c = cos(angleRad),
           s = sin(angleRad);

    return mat3(vec3(c, -s, Center[VX] * (1.0-c) + Center[VY] * s),
                 vec3(s, c, Center[VY] * (1.0-c) - Center[VX] * s),
                 vec3(0.0, 0.0, 1.0));
}
```



```
mat3 scaling2D(vec2& scaleVector)
{
    return mat3(vec3(scaleVector[VX], 0.0, 0.0),
                vec3(0.0, scaleVector[VY], 0.0),
                vec3(0.0, 0.0, 1.0)); }

mat4 identity3D()
{
    return mat4(vec4(1.0, 0.0, 0.0, 0.0),
                vec4(0.0, 1.0, 0.0, 0.0),
                vec4(0.0, 0.0, 1.0, 0.0),
                vec4(0.0, 0.0, 0.0, 1.0)); }

mat4 translation3D(vec3& v)
{
    return mat4(vec4(1.0, 0.0, 0.0, v[VX]),
                vec4(0.0, 1.0, 0.0, v[VY]),
                vec4(0.0, 0.0, 1.0, v[VZ]),
                vec4(0.0, 0.0, 0.0, 1.0)); }

mat4 rotation3D(vec3& Axis, const double angleDeg) {
    double angleRad = angleDeg * M_PI / 180.0,
           c = cos(angleRad),
           s = sin(angleRad),
           t = 1.0 - c;

    Axis.normalize();
    return mat4(vec4(t * Axis[VX] * Axis[VX] + c,
                    t * Axis[VX] * Axis[VY] - s * Axis[VZ],
                    t * Axis[VX] * Axis[VZ] + s * Axis[VY],
                    0.0),
                vec4(t * Axis[VX] * Axis[VY] + s * Axis[VZ],
                    t * Axis[VY] * Axis[VY] + c,
                    t * Axis[VY] * Axis[VZ] - s * Axis[VX],
                    0.0),
                vec4(t * Axis[VX] * Axis[VZ] - s * Axis[VY],
                    t * Axis[VY] * Axis[VZ] + s * Axis[VX],
                    t * Axis[VZ] * Axis[VZ] + c,
                    0.0),
                vec4(0.0, 0.0, 0.0, 1.0));
}

mat4 scaling3D(vec3& scaleVector)
{
    return mat4(vec4(scaleVector[VX], 0.0, 0.0, 0.0),
                vec4(0.0, scaleVector[VY], 0.0, 0.0),
                vec4(0.0, 0.0, scaleVector[VZ], 0.0),
                vec4(0.0, 0.0, 0.0, 1.0)); }

mat4 perspective3D(const double d)
{
    return mat4(vec4(1.0, 0.0, 0.0, 0.0),
                vec4(0.0, 1.0, 0.0, 0.0),
                vec4(0.0, 0.0, 1.0, 0.0),
                vec4(0.0, 0.0, 1.0/d, 0.0)); }
```

```
%  
% Camera placement  
%
```

Camera

```
| 2.146539 2.827176 3.467622 |  
  
| 0.851829 -0.294472 0.433212 0.000000 |  
| 0.000000 0.827026 0.562164 0.000000 |  
| -0.523819 -0.478868 0.704485 0.000000 |  
| 0.000000 0.000000 0.000000 1.000000 |  
  
| 57.509806 57.509806 |
```

```
%  
% Light sources  
%
```

Light

```
| 0.0 0.5 4.0 |  
| 1.0 0.0 0.0 0.0 |  
| 0.0 1.0 0.0 0.0 |  
| 0.0 0.0 1.0 0.0 |  
| 0.0 0.0 0.0 1.0 |  
| 0.5 0.5 0.5 |
```

Light

```
| 3.08 0.0 0.0 |  
| 1.0 0.0 0.0 0.0 |  
| 0.0 1.0 0.0 0.0 |  
| 0.0 0.0 1.0 0.0 |  
| 0.0 0.0 0.0 1.0 |  
| 0.5 0.5 0.5 |
```

Light

```
| 3.08 3.35 5.55 |  
| 1.0 0.0 0.0 0.0 |  
| 0.0 1.0 0.0 0.0 |  
| 0.0 0.0 1.0 0.0 |  
| 0.0 0.0 0.0 1.0 |  
| 1.0 1.0 1.0 |
```

```
%  
% Description of the objects  
%
```

Polyhedron

```
| 0.0 0.0 0.0 |  
| 1.0 0.0 0.0 0.0 |  
| 0.0 1.0 0.0 0.0 |  
| 0.0 0.0 1.0 0.0 |  
| 0.0 0.0 0.0 1.0 |  
| 1.0 1.0 1.0 |  
| 0.1 0.48 0.7 20.0 |
```

10

```
| -1.0 -1.0 -1.0 |  
| 1.0 -1.0 -1.0 |  
| 1.0 1.0 -1.0 |  
| -1.0 1.0 -1.0 |
```

	-1.0	-1.0	1.0	
	1.0	-1.0	1.0	
	1.0	1.0	1.0	
	-1.0	1.0	1.0	
	0.0	1.8	-1.0	
	0.0	1.8	1.0	

7

5 0 3 8 2 1

5 4 5 6 9 7

4 0 1 5 4

4 1 2 6 5

4 2 8 9 6

4 3 7 9 8

4 0 4 7 3

Sphere

	1.0	-0.2	1.0		
	1.0	0.0	0.0	0.0	
	0.0	1.0	0.0	0.0	
	0.0	0.0	1.0	0.0	
	0.0	0.0	0.0	1.0	
	1.0	1.0	1.0		
	0.1	0.48	0.7	20.0	

0.8

```
#include <stream.h>
#include "Scene3D.h"

int main(int, char**)
{
    char    script[1024],
           *image;
    Scene3D scene;
    vec2    res;

    cout << "\nSimple Ray-Tracer v1.0\nby Jean-Francois Doue\n\nenter a script name:";
    cin >> script;
    istream s(script, io_readonly, a_useonly);
    s >> scene;
    cout << "\nResolution of the rendering (x y):";
    cin >> res;
    image = scene.rayTrace(res);

    // put some code here to save the image in your preferred format

    delete image;
    return 1;
}
```

```
#include "solver.h"

/*****
 *
 * This function determines if a double is small enough
 * to be zero. The purpose of the subroutine is to try
 * to overcome precision problems in math routines.
 *
 *****/

static int isZero(double x)
{
    return x > -EQN_EPS && x < EQN_EPS;
}

int solveLinear(double c[2], double s[1])
{
    if (isZero(c[1]))
        return 0;
    s[0] = - c[0] / c[1];
    return 1;
}

/*****
 *
 * This function determines the roots of a quadric
 * equation.
 * It takes as parameters a pointer to the three
 * coefficient of the quadric equation (the c[2] is the
 * coefficient of x2 and so on) and a pointer to the
 * two element array in which the roots are to be
 * placed.
 * It outputs the number of roots found.
 *
 *****/

int solveQuadric(double c[3], double s[2])
{
    double p, q, D;

    // make sure we have a d2 equation

    if (isZero(c[2]))
        return solveLinear(c, s);

    // normal for: x^2 + px + q
    p = c[1] / (2.0 * c[2]);
    q = c[0] / c[2];
    D = p * p - q;

    if (isZero(D))
    {
        // one double root
        s[0] = s[1] = -p;
        return 1;
    }
}
```

```
if (D < 0.0)
    // no real root
    return 0;

else
    {
        // two real roots
        double sqrt_D = sqrt(D);
        s[0] = sqrt_D - p;
        s[1] = -sqrt_D - p;
        return 2;
    }
}

/*****
 *
 * This function determines the roots of a cubic
 * equation.
 * It takes as parameters a pointer to the four
 * coefficient of the cubic equation (the c[3] is the
 * coefficient of x3 and so on) and a pointer to the
 * three element array in which the roots are to be
 * placed.
 * It outputs the number of roots found
 *
 *****/

int solveCubic(double c[4], double s[3])
{
    int    i, num;
    double sub,
           A, B, C,
           sq_A, p, q,
           cb_p, D;

    // normalize the equation:  $x^3 + Ax^2 + Bx + C = 0$ 
    A = c[2] / c[3];
    B = c[1] / c[3];
    C = c[0] / c[3];

    // substitute  $x = y - A / 3$  to eliminate the quadric term:  $x^3 + px + q = 0$ 

    sq_A = A * A;
    p = 1.0/3.0 * (-1.0/3.0 * sq_A + B);
    q = 1.0/2.0 * (2.0/27.0 * A * sq_A - 1.0/3.0 * A * B + C);

    // use Cardano's formula

    cb_p = p * p * p;
    D = q * q + cb_p;

    if (isZero(D))
    {
        if (isZero(q))
        {
            // one triple solution
            s[0] = 0.0;
            num = 1;
        }
    }
}
```

```

    }
else
{
    // one single and one double solution
    double u = cbrt(-q);
    s[0] = 2.0 * u;
    s[1] = - u;
    num = 2;
}
}
else
    if (D < 0.0)
    {
        // casus irreductibilis: three real solutions
        double phi = 1.0/3.0 * acos(-q / sqrt(-cb_p));
        double t = 2.0 * sqrt(-p);
        s[0] = t * cos(phi);
        s[1] = -t * cos(phi + M_PI / 3.0);
        s[2] = -t * cos(phi - M_PI / 3.0);
        num = 3;
    }
else
{
    // one real solution
    double sqrt_D = sqrt(D);
    double u = cbrt(sqrt_D + fabs(q));
    if (q > 0.0)
        s[0] = - u + p / u ;
    else
        s[0] = u - p / u;
    num = 1;
}

// resubstitute
sub = 1.0 / 3.0 * A;
for (i = 0; i < num; i++)
    s[i] -= sub;
return num;
}

```

```

/*****
*
* This function determines the roots of a quartic
* equation.
* It takes as parameters a pointer to the five
* coefficient of the quartic equation (the c[4] is the
* coefficient of x4 and so on) and a pointer to the
* four element array in which the roots are to be
* placed. It outputs the number of roots found.
*
*****/

```

```

int
solveQuartic(double c[5], double s[4])
{
    double
        coeffs[4],
        z, u, v, sub,
        A, B, C, D,

```

```
sq_A, p, q, r;
int i, num;

// normalize the equation:  $x^4 + Ax^3 + Bx^2 + Cx + D = 0$ 

A = c[3] / c[4];
B = c[2] / c[4];
C = c[1] / c[4];
D = c[0] / c[4];

// substitute  $x = y - A / 4$  to eliminate the cubic term:  $x^4 + px^2 + qx + r = 0$ 

sq_A = A * A;
p = -3.0 / 8.0 * sq_A + B;
q = 1.0 / 8.0 * sq_A * A - 1.0 / 2.0 * A * B + C;
r = -3.0 / 256.0 * sq_A * sq_A + 1.0 / 16.0 * sq_A * B - 1.0 / 4.0 * A * C + D;

if (isZero(r))
{
    // no absolute term:  $y(y^3 + py + q) = 0$ 
    coeffs[0] = q;
    coeffs[1] = p;
    coeffs[2] = 0.0;
    coeffs[3] = 1.0;

    num = solveCubic(coeffs, s);
    s[num++] = 0;
}
else
{
    // solve the resolvent cubic...
    coeffs[0] = 1.0 / 2.0 * r * p - 1.0 / 8.0 * q * q;
    coeffs[1] = -r;
    coeffs[2] = -1.0 / 2.0 * p;
    coeffs[3] = 1.0;
    (void) solveCubic(coeffs, s);

    // ...and take the one real solution...
    z = s[0];

    // ...to build two quadratic equations
    u = z * z - r;
    v = 2.0 * z - p;

    if (isZero(u))
        u = 0.0;
    else if (u > 0.0)
        u = sqrt(u);
    else
        return 0;

    if (isZero(v))
        v = 0;
    else if (v > 0.0)
        v = sqrt(v);
    else
        return 0;

    coeffs[0] = z - u;
    coeffs[1] = q < 0 ? -v : v;
}
```



```
coeffs[2] = 1.0;

num = solveQuadric(coeffs, s);

coeffs[0] = z + u;
coeffs[1] = q < 0 ? v : -v;
coeffs[2] = 1.0;

num += solveQuadric(coeffs, s + num);
}
```

```
// resubstitute
sub = 1.0 / 4 * A;
for (i = 0; i < num; i++)
    s[i] -= sub;
```

```
return num;
```

```
}
```

```

/*****
 *
 * Polynomial root finder (polynoms up to degree 4)
 * AUTHOR: Jochen SCHARZE (See 'Cubic & Quartic Roots' in
 *                        'Graphics Gems 1', AP)
 *
 *****/
```

```
#include <math.h>
#define EQN_EPS 1e-9
```

```
int isZero(double x);
int solveQuadric(double c[3], double s[2]);
int solveCubic(double c[4], double s[3]);
int solveQuartic(double c[5], double s[4]);
```

Errata to `_Graphics Gems_`, first edition, edited by Andrew Glassner  
([andrew\\_glassner@yahoo.com](mailto:andrew_glassner@yahoo.com)), Academic Press 1990. Code available online at  
<http://www.graphicsgems.org/>

compiled by Eric Haines ([erich@acm.org](mailto:erich@acm.org)) from author and reader contributions

version 1.26

date: 1/3/2001

-----

Errors in the text:

- p. 3, bottom: The equation " $N \cdot P + c = 0$ " is better expressed as  
" $N \cdot P - c = 0$ " in order to match Figure 1a. [also see p. 9 errata]
- p. 5, V2 Perpendicular: change " $N \leftarrow (-V_x, V_y)$ " to " $N \leftarrow (-V_y, V_x)$ "
- p. 5, V2 Reflect: change " $N \leftarrow (-V_y, -V_x)$ " to " $N \leftarrow (-V_x, -V_y)$ "
- p. 6: change P1 and P2 lines to  
 $P1 \leftarrow ((-b + \sqrt{d}) / 2a) * l_v + l_u$   
 $P2 \leftarrow ((-b - \sqrt{d}) / 2a) * l_v + l_u$   
i.e. the value computed is multiplied by the direction vector and the  
line's origin is added to this new vector to get the intersection point.
- p. 9-10, starting at bottom: If the equation on p. 3 is expressed as  
" $N \cdot P - c = 0$ ", then change all "+ c" references to "- c" and "l-sub-c"  
to "- l-sub-c".
- p. 10, for "if not l-normalized", the operation in the next line should divide  
q by " $(l\text{-sub-}n \text{ dot } l\text{-sub-}n)$ ", not " $\text{Length}(l\text{-sub-}n)$ ".
- p. 11, bottom: change the lower bound of the sum ( $\sigma$ ) from  $i=1$  to  $i=0$ .
- p. 16, Product Relations. In the right hand parts of the equations the cosine  
and sine functions should be applied only to the numerator, then the  
division by 2 is done. Specifically:
- $$\begin{aligned}\sin(a) * \sin(b) &= \cos(a-b)/2 - \cos(a+b)/2 \\ \cos(a) * \cos(b) &= \cos(a-b)/2 + \cos(a+b)/2 \\ \sin(a) * \cos(b) &= \sin(a+b)/2 + \sin(a-b)/2\end{aligned}$$
- p. 105, last sentence of first paragraph: "ajacent" to "adjacent".
- p. 216, caption for figure 2: "54" should read "45" to be consistent with  
the figure (error in two places in caption).
- p. 224, the Bit Width 23 mask is listed as 0x00400000, it should be  
0x00420000.
- p. 282, 5 and 7 lines from bottom: should read  
"then PUSH(dadRx + 1, rx, pushlx, pushrx, y-dir, -dir )"  
and  
"then PUSH(lx, dadLx-1, pushlx, pushrx, y-dir, -dir )"  
i.e. the final "dir" should be "-dir".
- p. 283, 3 lines from bottom: add an "end" above the "else begin".
- p. 284, 12 lines down: move " $x \leftarrow x + 1$ ;" to after the next "end" statement  
(move it down only one line).

- p. 299, bottom: P1 and P2 as shown are actually the distances along the line from the line's origin (l.sub.U). Change the P1 and P2 lines to
- ```
t1 = (-b + sqrt(d)) / 2a
t2 = (-b - sqrt(d)) / 2a

P1 <- l.sub.U + t1 * l.sub.V
P2 <- l.sub.U + t2 * l.sub.V
```
- p. 365, last line: "Kajia" to "Kajiya".
- p. 375, "revlect v" to "reflect v".
- p. 395, first paragraph: change "discussed by Haines (1989)" to "discussed by Haines in Glassner (1989)".
- p. 406, last equation: if q is negative, the same signs should be used for the square root terms, i.e.
- $$y^2 \pm y \sqrt{2z - p} + z \pm \sqrt{z^2 - r} = 0$$
- p. 448, last sentence of second paragraph: change "and now nearly as simple" to "and not nearly as simple".
- p. 463, second to last line: change "then alpha <- alpha + pi/2" to "then alpha <- pi - alpha".
- p. 479, Table 1: the number of Multiplies for rotation should be 16, not 12.
- p. 495, equation 5: this should have an equal sign (=) before the plus-or-minus (+/-).
- p. 499, middle of page: change "and i,j,K" to "and i,j,k".
- p. 503, last sentence: change "Let P' = Rot\_(alpha, N) ..." to "Let P' = Rot\_(theta, N) ...".
- p. 516, last paragraph: a reader notes an additional reference which predates Berger and Salmon & Slater, namely "The Viewing Transformation," Technical Memo. no. 84, Alvy Ray Smith, Computer Graphics Project, Lucasfilm, June 24, 1983 (rev. May 4, 1984).
- p. 602, second paragraph: the matrix Tij should be:
- ```
[ 1  0  0  0 ]
[ 0 1/2 0  0 ]
[ 0  0 1/4 0 ]
[ 0  0  0 1/8 ]
```
- p. 610: the binomial " $\binom{n-i}{j}$ " should be " $\binom{n-1}{j}$ ". This error appears on the fifth line of the long derivation and within the Zi,j definition.
- p. 809: the author of "Approximation of Sweep Surfaces by Tensor Product B-Splines" is M. (not J.) Bloomenthal. The author is correctly attributed in the text (page 569).
- p. 814: 5th line from bottom. "Knuth 1981" should read "Knuth 1981b" and "Vol. 2" should read "Vol. 1". The reference above this should be "Knuth 1981a".

p. 820, 9th line from bottom: "D.P. Greenburg" should be "D.P. Greenberg".

-----

The following are errors in the code listings (corrected in the online code at <http://www.graphicsgems.org/>).

Serious errors (ones your compiler cannot or may not catch):

p. 630: Delete FLOOR and CEILING macros (they're more like truncations).  
Change ROUND macro to (i.e. add parentheses around "a"):  
#define ROUND(a) ((a)>0 ? (int)((a)+0.5) : -(int)(0.5-(a)))  
Change SGN macro to (i.e. change positive condition result to "1"):  
#define SGN(a) (((a)<0) ? -1 : 1)

p. 632: procedure declarations for routines in the "2D and 2D Vector C Library" (next pages) are missing from "GraphicsGems.h", e.g.

```
double V2SquaredLength() ;  
double V2Length() ;  
Vector2 *V2Negate() ;  
...
```

p. 638, third line from bottom: in V3Combine change last "result->y" to "result->z"

p. 640: V3MulPointByMatrix() does not work. A separate local Point3 (e.g. "Point3 q ;") should be used in place of "p" for assignment and then passed back.

p. 649, top: add "#include <math.h>"

p. 655, top: add the code:  
if ((px == qx) && (py == qy))  
 if ((tx == px) && (ty == py)) return 2;  
 else return 0;  
in order to test for the special case where the line endpoints are the same.

p. 656, line 23: change "return ((R->min.x \* R->min.x) < Rad2);" to "return ((R->min.x \* R->min.x + R->max.y \* R->max.y) < Rad2);"

p. 662, line 10: add "#include <math.h>"

p. 665, line 1: change "FLOOR(...)" to "(floor((double)(...)))".

p. 663, line 45: change first "+" to "="; should read  
"VnextLeft = (Vleft=VnextLeft) + 1;"

p. 667, line 6: change "POLY\_NMAX 8" to "POLY\_NMAX 10" (for triangles and quadrilaterals). Six clipping planes used on convex polygons gives +6 potential extra vertices generated.

p. 670-673, throughout: change "int mask" declarations to "unsigned long mask" declarations. This avoids an infinite loop occurring when the highest bit is set.

p. 676, line 11: add the line "up = (double \*)u;"

p. 671, line 1: add at top of page the following test (or make sure the Poly\_vert structure has <= 32 doubles at compilation time):

```
if (sizeof(Poly_vert)/sizeof(double) > 32) {
fprintf(stderr, "Poly_vert structure too big; must be <=32 doubles\n");
exit(1);
}
```

- p. 687, line 8: Identical points cause two points to be drawn. Between the first two plot() commands, add the line:

```
if ( pixels_left < 0 ) return ;
```

- p. 714, line 20: the last "1" in "if (i + 1 < 1 \* 1)" should be an "l"

- p. 716, line 27: missing "/" at end of comment (if not fixed, code compiles but is wrong)

- p. 720, lines 3 and 6 from bottom: change "m+1>>1" to "(m+1)>>1" to establish correct evaluation order.

- p. 737, lines 19-24, from "if ((quadrant..." to "}", should read (and note corrected indentations on "else" statement):

```
                if (coord[i] < minB[i] || coord[i] > maxB[i])
                    return (FALSE);
            } else {
                coord[i] = candidatePlane[i];
            }
```

- p. 742, lines 18-24 should read:

```
coeffs[ 0 ] = z - u;
coeffs[ 1 ] = q < 0 ? -v : v;
coeffs[ 2 ] = 1;

num = SolveQuadric(coeffs, s);

coeffs[ 0 ] = z + u;
coeffs[ 1 ] = q < 0 ? v : -v;
coeffs[ 2 ] = 1;
```

The sign of "q" affects the sign of coeffs[1].

- p. 745, throughout: delete references to "lx", which is set but not used

- p. 748, line 22: change "negetive" to "negative"

- p. 753, line 35: change "int n1, n2," to "int n1=0, n2=0," so that first fprintf() error message has defined values

- p. 756, line 15: "unsigned int \*fi=&f;" to "unsigned int \*fi = (unsigned int \*) &f;" for type consistency, and some compilers think "=&" means "&="

- p. 766, top: add '#include "GraphicsGems.h"' and '#include <math.h>'
line 25: change "det = det4x4( out );" to "det = det4x4( in );"
throughout: change "matrix4" to "Matrix4"

- p. 774, line 5: change "theta," to "theta = 0,"

- p. 799, line 32: add between first "DrawBezierCurve(...);" call and "return;": free((void \*)bezCurve);

- p. 799, bottom and p. 800, line 11: add this pair of lines between "DrawBezierCurve(...);" and "return;":
- ```
free((void *)u);
free((void *)bezCurve);
```
- Also, see errata note at end of this file.
- p. 800, line 19: add this pair of lines before "tHatCenter = ...":
- ```
free((void *)u);
free((void *)bezCurve);
```
- p. 802, line 2: Change " < 0.0" to " < 1.0e-6" in the alpha tests. Otherwise you get coincident control points that lead to divide by zero in any subsequent NewtonRaphsonRootFind() call.

Syntax errors (ones your compiler or lint will catch):

- p. 633-642, throughout: replace "};" with "}" to make lint happy
- p. 640, gcd(): the variable "k" is set but not used - remove it
- p. 649, line 31: change "LengthVector3" to "V3Length"
- p. 650, line 1: bad end-of comment; delete "/"
- p. 651, throughout: Can't use "const" as a variable name, as it is a reserved word in ANSI C. Use "liconst" instead.
- p. 657, 659: make "nicenum" declarations match, i.e. use either "double nicenum()" or "static double nicenum()" for both occurrences
- p. 659: change "exp" to "expv", since "exp()" is a math library function.
- p. 660, line 11: header missing end of comment "\*/"
- p. 662, line 13: change "SYBYRES" to "SUBYRES"
- line 16: bad space after "MODRES"
- line 42: change "XRmax" to "xRmax"
- p. 665, line 15: missing semicolon after "int area"
- line 27: change "0" to "0" in "if (partialArea>0)"
- p. 666, line 13: change "0" to "0" in "rightMask = 0;"
- p. 670, top: add to make lint happy
- ```
static scanline();
static incrementalize_y();
static incrementalize_x();
static increment();
```
- p. 671, line 35: missing "{" at end of "while (y<ly && y<ry)"
- p. 671-2, throughout: add "(double \*)" castings to all "incrementalize\*" calls to make lint happy
- p. 676, line 8: change "if (tu<=0. ^ tv<=0.) {" to "if ( (tu<=0.) ^ (tv<=0.) ) {" to avoid precedence confusion
- end: change "void pixelproc();" to "static void pixelproc();"
- p. 681, throughout: "y0" and "y1" are Bessel functions in the math library, so lint complains; ignore complaint or rename
- p. 684: change "delete" to "cdelete" and "insert" to "cinsert", since these routine names are already used by dbm database manager
- p. 687, near end: change "+ =" to "+="
- p. 696, line 8: add '#include "GraphicsGems.h"'
- p. 700, line 5: add '#include "GraphicsGems.h"'

- p. 702-706: change "max" to "MAX", "min" to "MIN"
- p. 705, line 18: delete "\*sp", as it is not used
- p. 706, end: add '#include "GraphicsGems.h"'
- p. 707, in clip: delete "\*sp", as it is not used
- p. 709, line 23: missing semicolon at end of "up = (up) ? FALSE : TRUE"
- p. 710, top: change "max" to "MAX", "min" to "MIN"
  
- p. 713, line 26: change ":" to ";" in "char \*\*argv:"
  
- p. 715, top: pseudo-code at head of file (to advance from one element to the next) should be commented out.
  - line 14: missing declaration "int randmasks[32];"
  - throughout: three calls to "bitwidth()" need to cast argument to "(unsigned int)" to make lint happy
  
- p. 719: add "#include <string.h>" (or strings.h)
  
- p. 727, line 11: remove ")" in "static double bigC,..." line
- p. 728, lines 21-23: change "cal\_q\_msq" to "calc\_q\_msq"
  - lines 23,42: change "NULL" to "(double \*)NULL" to make lint happy
  - line 26: change "con\_const" to "cone\_const" in  
"bigC = mlsq + con\_const \* q1"
  - last line: add a "}" to end albers\_project procedure
  
- p. 730: missing inclusion of GraphicsGems.h.
  
- p. 734, line 4: change "exit();" to "exit(1);"
  
- p. 736, line 20: change "O" to "0" in "for (i=0;"
  
- p. 739, line 12: change "else if (D > 0)" to "else", since at this point D must be greater than 0; makes lint happy
  
- p. 757, line 4: change "{" to "[" in "sqrttab{"
  - line 14: change "= &n" to "= (unsigned int \*)&n"
  - line 21: change "\*num &= 0x7ffffff:" to "\*num &= 0x7ffffff;" to fit ANSI C, and to fix error of ":" at end of line.
  - line 22: change "| =" to "|="
  - line 27: change ":" to ";"
  
- p. 765, line 20,22: change "Matrix" to "Matrix4"
  
- p. 766, line 29: change "exit();" to "exit(1);"
  
- p. 774, line 2: missing ";" at end of "long \*argx, \*argy"
  
- p. 775: P, Q, and M need to be declared as externs
  
- p. 780, line 18: bad start of comment for "/ re-parameterization"
  
- p. 785, line 1: bad start-of-comment
  
- p. 789, lines 7,25: change both "NULL" to "(Point2 \*)NULL" to make lint happy in ConvertToBezierForm: "t" is not used, can be deleted
- p. 791, line 24: remove "break;" after "return 0;"; unnecessary
- p. 793, ControlPolygonFlatEnough: "t" is not used, can be deleted
- p. 795, ComputeXIntercept: "T" and "Y" do not have to be computed, since the result "Y" is not returned. Note that there are many operations in this routine that are unnecessary (e.g. "0.0 - 0.0").
  
- p. 797-807, throughout: change "V2ScaleII" to "V2ScaleIII" and "Bezier" to



"BezierII" in order to avoid name collisions with the code on pages 787-796 (i.e. the same subroutine names are used in both, but with different argument types, etc). Important only if you link in both of these subroutine libraries.

-----

The following are typographical errors in the comments:

p. 687, line 3: "plottted" to "plotted"

p. 701, line 26: change "interscetion" to "intersection"

p. 723, line 1: "tight" is not used in the strict mathematical sense that the sphere generated is optimal (i.e. no smaller sphere could be found), but rather meaning "near optimal".

p. 728, line 10: "latitute" to "latitude"

p. 729, line 8: "degress" to "degrees"

p. 724, line 38: "seperated" to "separated"

p. 725, lines 7-9: "componant" to "component"

p. 730, line 17: "minium" to "minimum"

p. 752, line 2: "positve" to "positive"

p. 760, line 5: "interger" to "integer"

p. 761, line 4: "preceed" to "precede"

p. 766, throughout (5 times): "determinent" to "determinant"

p. 781, lines 7,23: "demoninator" to "denominator"

-----

Addenda: There is additional code for Kelvin Thompson's "Rendering Anti-Aliased Lines" gem in the online distribution of the code.

-----

Concerning page 799, Philip Schneider's Bezier code:

If you are operating in a dimensional system such that the desired error in the fitting process is a fraction (e.g., 0.01 inches) rather than a whole number (e.g., 2.0 pixels), then the line on page 799 reading

```
iterationError = error * error;
```

should be changed to

```
iterationError = ERRFACTOR * error;
```

where ERRFACTOR is #defined to some implementation-dependent value such as 4.0. If this is not done, then reparameterization will never occur. The result is not an erroneous curve, but a suboptimal one; the algorithm will always subdivide when the initial fit is too "loose."

(from Earl Boebert, boebert@SCTC.COM)

| File Name            | Archive # | Description                                     |
|----------------------|-----------|-------------------------------------------------|
| -----                |           |                                                 |
| 2DClip               | 1         | 2D Clipping: A Vector-Based Approach            |
| 2DClip/Makefile      | 1         | Makefile                                        |
| 2DClip/bio.c         | 2         | Basic operations                                |
| 2DClip/box.h         | 1         | BOX definition                                  |
| 2DClip/clip.c        | 2         | Clipping routines                               |
| 2DClip/cross.c       | 4         | Intersection calculation                        |
| 2DClip/line.h        | 1         | Major definitions                               |
| AALines              | 1         | Rendering Anti-Aliased Lines                    |
| AALines/00README     | 1         | Information about AALines Gem                   |
| AALines/AALines.c    | 4         | Code to render an anti-aliased line             |
| AALines/AALines.h    | 1         | Symbols & globals                               |
| AALines/AAMain.c     | 1         | Calling routine                                 |
| AALines/AATables.c   | 4         | Initialization of tables and frame buffer       |
| AALines/FastMatMul.c | 5         | Fast routines to multiply 4x4 matrices          |
| AALines/LongConst.h  | 1         | High-precision constants                        |
| AALines/Makefile     | 1         | Makefile                                        |
| AALines/utah.c       | 3         | Interface to Utah Raster Toolkit                |
| AALines/utah.h       | 1         | Declarations for URT interface                  |
| AAPolyScan.c         | 4         | Fast Anti_aliasing Polygon Scan Conversion      |
| Albers.c             | 3         | Albers Equal-Area Conic Map Projection          |
| BinRec.c             | 1         | Recording Animation in Binary Order             |
| BoundSphere.c        | 3         | An Efficient Bounding Sphere                    |
| BoxSphere.c          | 2         | Box-Sphere Intersection Checking                |
| CircleRect.c         | 1         | Fast Circle-Rectangle Intersection Checking     |
| ConcaveScan.c        | 4         | Concave Polygon Scan Conversion                 |
| DigitalLine.c        | 1         | Digital Line Drawing                            |
| Dissolve.c           | 3         | A Digital "Dissolve" Effect                     |
| DoubleLine.c         | 3         | Symmetric Double Step Line Algorithm            |
| Errata.GraphicsGems  | *         | Known errata for _Graphics Gems_ book           |
| FastJitter.c         | 1         | Efficient Generation of Sampling Jitter         |
| FitCurves.c          | 5         | Automatically Fit Digitized Curves              |
| FixedTrig.c          | 1         | Fixed-Point Trig with CORDIC Iterations         |
| Forms.c              | 4         | Forms, Vectors, and Transformations             |
| Gems.TOC             |           | Table of Contents of all books in Gems series   |
| GGVecLib.c           | 5         | 2D and 3D Vector C Library                      |
| GraphicsGems.h       | 3         | Graphics Gems header file                       |
| HSLtoRGB.c           | 1         | A Fast HSL-to-RGB Transform                     |
| Hash3D.c             | 1         | 3D Grid Hashing Function                        |
| HypotApprox.c        | 1         | A Fast Approximation to the Hypotenuse          |
| Interleave.c         | 4         | Bit Interleaving for Quad- or Octrees           |
| Label.c              | 2         | Nice Numbers for Graph Labels                   |
| LineEdge.c           | 3         | Fast Line-Edge Intersections on a Uniform Grid  |
| MANIFEST             | 1         | This shipping list                              |
| Makefile             | 2         | Makefile for the whole shebang                  |
| MatrixInvert.c       | 3         | Matrix Inversion                                |
| MatrixOrtho.c        | 1         | Matrix Orthogonalization                        |
| MatrixPost.c         | 2         | Efficient Post-Concatenation of Trans. Matrices |
| Median.c             | 2         | Median Finding on a 3x3 Grid                    |
| NearestPoint.c       | 5         | Nearest-Point-On-Curve and Bezier Root-Finder   |
| OrderDither.c        | 2         | Ordered Dithering                               |
| PixelInteger.c       | 1         | Proper Treatment of Pixels As Integers          |
| PntOnLine.c          | 2         | A Fast 2D Point-On-Line Test                    |
| PolyScan             | 1         | Convex Polygon Scan Conversion & Clipping       |
| PolyScan/Makefile    | 1         | Makefile                                        |
| PolyScan/fancytest.c | 2         | Phong-shading a Texture mapping test            |
| PolyScan/poly.c      | 2         | Simple utilities for polygon data structure     |
| PolyScan/poly.h      | 2         | Definitions for polygon package                 |
| PolyScan/poly_clip.c | 3         | Homogeneous 3D polygon clipper                  |
| PolyScan/poly_scan.c | 3         | Convex polygon point-sampled scan conversion    |

|                     |   |                                                 |
|---------------------|---|-------------------------------------------------|
| PolyScan/scantest.c | 2 | Gouraud shading and Z-buffer demo               |
| Quaternions.c       | 2 | Using Quaternions for Coding 3D Transformations |
| README              | 1 | General information                             |
| RGBTo4Bits.c        | 1 | Mapping RGB Triples Onto Four Bits              |
| RayBox.c            | 1 | Fast Ray-Box Intersection                       |
| RayPolygon.c        | 1 | An Efficient Ray-Polygon Intersection           |
| Roots3And4.c        | 3 | Cubic and Quartic Roots                         |
| SeedFill.c          | 2 | A Seed Fill Algorithm                           |
| SquareRoot.c        | 2 | A High-Speed, Low-Precision Square Root         |
| Sturm               | 1 | Using Sturm Sequences to Bracket Real Roots     |
| Sturm/Makefile      | 1 | Makefile                                        |
| Sturm/main.c        | 2 | Sample driver program                           |
| Sturm/solve.h       | 1 | Useful constants and types                      |
| Sturm/sturm.c       | 4 | Functions to build and evaluate Sturm sequence  |
| Sturm/util.c        | 1 | Root polishing and evaluating utilities         |
| TransBox.c          | 1 | Transforming Axis-Aligned Bounding Boxes        |
| TriPoints.c         | 2 | Generating Random Points in Triangles           |
| ViewTrans.c         | 1 | 3D Viewing and Rotation using Orthonormal Bases |

```
#
# Makefile for Graphics Gems source
#
# Craig Kolb, 8/90
#
# This make file will build "gemslib.a" and a number of executables.
# Gemslib is built solely for debugging purposes -- it is not intended
# to be used as a library.
#
# Note that some of the gems need additional macros, functions, tables
# driving routines, etc. before they will compile or run properly.
# These include:
#
# AALines
#     Needs variables set in the Makefile.
# Dissolve
#     Needs a table filled.
# FitCurves
#     Needs a functioning DrawBezierCurve().
# MatrixOrtho
#     Needs a number of matrix routines.
# PolyScan
#     Needs driving routines and other additional code.
# RayPolygon
#     Needs surrounding function structure, declaration of
#     variable types, etc.

#
# C compiler flags
#
CFLAGS = -g
#
# Location of Graphics Gems library
#
LIBFILE = gemslib.a

#
# Graphics Gems Vector Library
#
VECLIB = GGVecLib.o

MFLAGS = "LIBFILE = ../$(LIBFILE)" "GENCFLAGS = $(CFLAGS)"

FTPDIR = /usr/spool/uucppublic/ftp/pub/GraphicsGems/src
ZOOFILE = Gems.zoo

SHELL = /bin/sh

OFILES =      PntOnLine.o ViewTrans.o AAPolyScan.o Albers.o \
              Interleave.o BoundSphere.o BoxSphere.o CircleRect.o \
              ConcaveScan.o Roots3And4.o Dissolve.o DigitalLine.o \
              FastJitter.o FixedTrig.o HSLtoRGB.o HypotApprox.o LineEdge.o \
              MatrixInvert.o MatrixPost.o Median.o PixelInteger.o \
              TriPoints.o Quaternions.o RGBTo4Bits.o RayBox.o \
              SeedFill.o SquareRoot.o DoubleLine.o TransBox.o

DIRS = 2DClip PolyScan Sturm AALines

ALL =      Hash3D FitCurves Forms NearestPoint Label \
          OrderDither BinRec $(LIBFILE)
```

```
all: $(ALL)
    @for d in $(DIRS) ; do \
        (cd $$d ; $(MAKE) $(MFLAGS)) ;\
    done

$(LIBFILE): $(OFILES) $(VECLIB)
    ar rcs $(LIBFILE) $(OFILES) $(VECLIB)

Hash3D: Hash3D.o
    $(CC) $(CFLAGS) -o $@ Hash3D.o

FitCurves: FitCurves.o $(VECLIB)
    $(CC) $(CFLAGS) -o $@ FitCurves.o $(VECLIB) -lm

Forms: Forms.o
    $(CC) $(CFLAGS) -o $@ Forms.o

NearestPoint: NearestPoint.o $(VECLIB)
    $(CC) $(CFLAGS) -o $@ NearestPoint.o $(VECLIB) -lm

Label: Label.o
    $(CC) $(CFLAGS) -o $@ Label.o -lm

OrderDither: OrderDither.o
    $(CC) $(CFLAGS) -o $@ OrderDither.o

BinRec: BinRec.o
    $(CC) $(CFLAGS) -o $@ BinRec.o

ftpreadme:
    /bin/rm -f $(FTPDIR)/README
    sed -e "s/___DATE___/`date`/g" < README.ftp > $(FTPDIR)/README

readme:
    /bin/rm -rf README
    sed -e "s/___DATE___/`date`/g" < README.dist > README

clean:
    @for d in $(DIRS) ; do \
        (cd $$d ; $(MAKE) $(MFLAGS) clean) ;\
    done
    /bin/rm -f $(OFILES) $(VECLIB)
    /bin/rm -f Hash3D.o FitCurves.o \
        Forms.o NearestPoint.o Label.o OrderDither.o BinRec.o \
        Hash3D FitCurves Forms NearestPoint Label OrderDither BinRec \
        bugs a.out core Part??.Z Part?? $(ZOOFILE) gemslib.a

kit: readme
    /bin/rm -f Part??.Z Part??
    (makekit -iPACKING_LIST -oMANIFEST)

zoo: readme
    /bin/rm -f Gems.zoo
    cut -d" " -f2 PACKING_LIST | zoo aI $(ZOOFILE)

ftp: kit zoo ftpreadme
    compress Part??
    /bin/cp Part??.Z $(ZOOFILE) $(FTPDIR)

$(ALL): GraphicsGems.h
```

This directory contains source code related to the book "Graphics Gems" (editor, Andrew S. Glassner, published by Academic Press, Cambridge, MA, 1990, ISBN 0-12-286165-5, 833 pgs.).

The authors and the publisher hold no copyright restrictions on any of these files; this source code is public domain, and is freely available to the entire computer graphics community for study, use, and modification. We do request that the comment at the top of each file, identifying the original author and its original publication in the book Graphics Gems, be retained in all programs that use these files.

The original source files are stored in a plain text format; you may download them without any special techniques (i.e. you need not enable binary transfer, or have a special formatter to read and use the code). The updated source files in the "src" directory should be downloaded using binary transfer mode.

Additional submissions (bug fixes, skeleton programs, auxiliary routines, etc.) may be directed to the site administrator, Craig Kolb ([cek@cs.princeton.edu](mailto:cek@cs.princeton.edu)). He will determine on a case-by-case basis if a particular submission should be included in this archive. If accepted, these routines will be made available in a companion directory.

|           |                                                                                |
|-----------|--------------------------------------------------------------------------------|
| src/      | The most up-to-date version of each gem.<br>See src/README for details.        |
| original/ | Source code as printed in "Graphics Gems".<br>See original/README for details. |
| patches/  | Patches to the original source code.                                           |

CC = gcc

PROG=sx11

GCCFLAGS = -fpcc-struct-return

CFLAGS = -O2

INCLUDE = -I/usr/X11R5/include -I../tga

LDLIBS = -lX11 -ltga -lm

LDFLAGS = -L/usr/X11R5/lib -L../tga

\$(PROG):

\$(CC) \$(GCCFLAGS) \$(INCLUDE) \$(CFLAGS) \$(PROG).c \  
\$(LDFLAGS) \$(LDLIBS) -o \$(PROG)

```
/*
 * I don't wanna write the header so only ...
 *
 * (c) 1992, Raul Rivero
 */

#include <lug.h>
#include <lugfnts.h>

/* extern int LUGverbose; */
char *MY_NAME;

main(argc, argv)
int argc;
char **argv;
{
    register int i;

    bitmap_hdr in;
    ifunptr read_file;

    MY_NAME = argv[0];

    /*
     * Get options ( some day I'll build a procedure ).
     */
    if ( argc > 1 ) {
        /* else core on SGI */
        while ( argv[1][0] == '-' ) {
            for ( i = 1; argv[1][i]; i++ ) {
                switch ( argv[1][i] ) {
                    case 'v':
                        /* LUGverbose++; */
                        break;
                    case '!':
                        print_copyright();
                        break;
                    default :
                        usage();
                        break;
                }
            }
            argv++;
            argc--;
        }
    }

    if ( argc < 2 ) {
        usage();
    }

    if ( argc > 1 ) {
        /*
         * We have arguments, so read the files ...
         */
        for ( i = 1; i < argc; i++ ) {
            /* Read the file */
            read_file = get_readlug_function( argv[i] );
            read_file( argv[i], &in );
            /* Release the child and show the image */
        }
    }
}
```



```
    show_bitmap_x11( argv[i], &in );
    /* Ok, the child is displaying the image, so the parent continues */
    freebitmap( &in );
}
}
exit( 0 );
}

usage()
{
    char *msg = "\n\
%s: Usage: %s [-v!] <input_file> [<input_file>]\n\n\
Flags:\n\
\t-v: verbose\n\
\t-!: hey!, what about this program ?!\n\n\
The file type is got using its suffix:\n\n\
\t* .gif\t\t\t* .hf\t\t\t* .pbm/.pgm/.ppm\n\
\t* .pcx\t\t\t* .raw\t\t\t* .rgb\n\
\t* .rla\t\t\t* .rle\t\t\t* .sgi\n\
\t* .sun\t\t\t* .tga\t\t\t* .tif/.tiff\n\
\t* .ps\t\t\t* .jpeg/.jpg\t\t* .pix (** default **)\n\n\
The Alias 'pix' format will be used by default.\n\n\
If required, the quantization method is the default process to reduce\n\
the number of colors.\n\n";

    fprintf( stderr, msg, MY_NAME, MY_NAME );
    exit( 1 );
}

print_copyright()
{
    char *msg = "\n\
sx11 ( v.1.0 ) - show several image file formats\n\n\
This program - (c) 1992, Raul Rivero\n\
LUG library - (c) 1992, Raul Rivero && Math Dept. ( U. of Oviedo )\n\n\
This software is free and you can get a full copy of original LUG library\n\
via E-mail to rivero@pinon.ccu.uniovi.es or via anonymous ftp to \n\
telva.ccu.uniovi.es ( 156.35.31.31, /uniovi/mathdept/src ).\n\n\
The LUG library includes support for several file formats, viewers on\n\
different architectures and digital image processing.\n\n\
Supported input formats:\n\n\
\t* Pix ( Alias ) *** default ***\n\
\t* TIFF ( needs Sam Leffler's TIFF library )\n\
\t* RLE ( needs Utah Raster Toolkit library )\n\
\t* RLA ( Wavefront )\n\
\t* SGI ( internal Silicon Graphics file format )\n\
\t* Targa ( Truevision )\n\
\t* GIF ( Compuserve )\n\
\t* PCX ( ZSoft )\n\
\t* PBM/PGM/PPM\n\
\t* Postscript\n\
\t* JPEG ( needs Thomas G. Lane's JPEG library )\n";

    fputs( msg, stderr );
    exit( 1 );
}
```

```
# An ANSI C compiler: gnu's "gcc"
CC = gcc

# The UNIX flag must be defined for UNIX implementations
#CPPFLAGS = -DUNIX -DX11 -I.

# Very important! The first flag (struct-return) makes it safe to link
# gcc objects with cc objects.
GCCFLAGS = -fpcc-struct-return

# Object files will go in this subdirectory
OBJ_DIR = .

# User-defined; to pass options in to the compilation, say things
# like 'make CFLAGS=-g'. Put flags here to make them permanent.

CFLAGS = -O2

#####

TARGET = libtga.a

#HFILES = alias.h lug.h lugconf.h lugfnts.h rla.h targa.h tga.h

OBJECTS = \
bitmap.o \
tga.o \
in_out.o \
memory.o \
error.o \
general.o \
cnv.o \
hsl.o \
x11.o \
dither.o \
gif.o \
encodgif.o \
tobw.o \
#####

PATH_OBJECTS = $(OBJECTS:%=$(OBJ_DIR)/%)

$(OBJ_DIR)/%.o: %.c
    $(CC) $(GCCFLAGS) $(CFLAGS) -c -o $(OBJ_DIR)/$*.o $*.c

$(TARGET): $(PATH_OBJECTS)
    echo $(PATH_OBJECTS)
    ar rv $(TARGET) $(PATH_OBJECTS)
    ranlib $(TARGET)

remove:
    /bin/rm -f $(PATH_OBJECTS)
    /bin/rm -f *~

clean: remove $(TARGET)

#$(PATH_OBJECTS): $(HFILES)
```

This code is obtained from the LUG library due to Raul Rivero.

```
/*
 * This software is copyrighted as noted below.  It may be freely copied,
 * modified, and redistributed, provided that the copyright notice is
 * preserved on all copies.
 *
 * There is no warranty or other guarantee of fitness for this software,
 * it is provided solely "as is".  Bug reports or fixes may be sent
 * to the author, who may or may not act on them as he desires.
 *
 * You may not include this software in a program or other software product
 * without supplying the source, or without informing the end-user that the
 * source is available for no extra charge.
 *
 * If you modify this software, you should include a notice giving the
 * name of the person performing the modification, the date of modification,
 * and the reason for such modification.
 */
/*
 * rla.h - type define to Alias "pix" format.
 *
 * Author:      Raul Rivero
 *              Mathematics Dept.
 *              University of Oviedo
 * Date:        Wed Jan 22 1992
 * Copyright (c) 1992, Raul Rivero
 *
 */

#ifndef MY_ALIAS
#define MY_ALIAS

typedef struct {
    short xsize, ysize;
    short xinit, yinit;
    short depth;
} alias_hdr;

#endif      /* MY_ALIAS */
```

```
/*
 * This software is copyrighted as noted below.  It may be freely copied,
 * modified, and redistributed, provided that the copyright notice is
 * preserved on all copies.
 *
 * There is no warranty or other guarantee of fitness for this software,
 * it is provided solely "as is".  Bug reports or fixes may be sent
 * to the author, who may or may not act on them as he desires.
 *
 * You may not include this software in a program or other software product
 * without supplying the source, or without informing the end-user that the
 * source is available for no extra charge.
 *
 * If you modify this software, you should include a notice giving the
 * name of the person performing the modification, the date of modification,
 * and the reason for such modification.
 */
/*
 * bitmap.c - manage bitmap memory 'problems'.
 *
 * Author:      Raul Rivero
 *              Mathematics Dept.
 *              University of Oviedo
 * Date:        Sat Jan 4 1992
 * Copyright (c) 1992, Raul Rivero
 */
```

```
#include "lug.h"
```

```
#include "lugfnts.h"
```

```
allocatebitmap(image, xsize, ysize, depth, colors)
```

```
bitmap_hdr *image;
```

```
int xsize, ysize, depth, colors;
```

```
{
```

```
    int totalsize;
```

```
    if ( image->magic == LUGUSED )
        freebitmap( image );
```

```
    if ( (totalsize = xsize * ysize) <= 0 )
        error( 12 );          /* an error, a negative size ? */
```

```
    /* Fill sizes */
    image->xsize = xsize;
    image->ysize = ysize;
    image->magic = LUGUSED;
```

```
    /*
     * If colors are equal to 0 then the real information
     * are stored on detph;
     */
```

```
    if ( colors == 0 ) {
        image->depth = depth;
        image->colors = ( 1 << image->depth );
    } else {
        image->colors = colors;
        image->depth = no_bits( colors ) + 1;
    }
}
```

```
image->r = (byte *) Malloc( totalsize );
switch ( image->depth ) {
    case 8: image->cmap = (byte *) Malloc( 3 * image->colors );
        break;
    case 24: image->g = (byte *) Malloc( totalsize );
        image->b = (byte *) Malloc( totalsize );
        break;
    default: return 1;          /* an error, unkown depth */
}

return 0;
}

freebitmap(image)
bitmap_hdr *image;
{
    if ( image->magic != LUGUSED )
        return 0;

    image->magic = -LUGUSED;
    switch ( image->depth ) {
        case 32: /* for the future */
        case 24:
            free( image->r );
            free( image->g );
            free( image->b );
            break;
        default:
            if ( image->depth <= 8 ) {
                free( image->r );
                free( image->cmap );
            } else {
                /* Unknown depth */
                error( 10 );
            }
            break;
    }
    return 0;
}

copy_bitmap( inbitmap, outbitmap)
bitmap_hdr *inbitmap, *outbitmap;
{
    int totalsize = inbitmap->xsize * inbitmap->ysize;

    if ( outbitmap->magic == LUGUSED )
        freebitmap( outbitmap );

    if ( inbitmap->magic != LUGUSED )
        error( 19 );

    /*
     * Fill header.
     */
    outbitmap->magic = LUGUSED;
    outbitmap->xsize = inbitmap->xsize;
    outbitmap->ysize = inbitmap->ysize;
    outbitmap->depth = inbitmap->depth;
    outbitmap->colors = inbitmap->colors;

    /*
```

```
    * Copy raster information.
    */
    outbitmap->r = (byte *) Malloc( totalsize );
    bcopy( inbitmap->r, outbitmap->r, totalsize );
    if ( outbitmap->depth > 8 ) {
        outbitmap->g = (byte *) Malloc( totalsize );
        bcopy( inbitmap->g, outbitmap->g, totalsize );
        outbitmap->b = (byte *) Malloc( totalsize );
        bcopy( inbitmap->b, outbitmap->b, totalsize );
    }else {
        outbitmap->cmap = (byte *) Malloc( 3 * outbitmap->colors );
        bcopy( inbitmap->cmap, outbitmap->cmap, 3*outbitmap->colors );
    }
}
```

```
/*
 * This software is copyrighted as noted below.  It may be freely copied,
 * modified, and redistributed, provided that the copyright notice is
 * preserved on all copies.
 *
 * There is no warranty or other guarantee of fitness for this software,
 * it is provided solely "as is".  Bug reports or fixes may be sent
 * to the author, who may or may not act on them as he desires.
 *
 * You may not include this software in a program or other software product
 * without supplying the source, or without informing the end-user that the
 * source is available for no extra charge.
 *
 * If you modify this software, you should include a notice giving the
 * name of the person performing the modification, the date of modification,
 * and the reason for such modification.
 */
/*
 * cnv.c - select the read/write function using the file name ( !!! ).
 *
 * Author:      Raul Rivero
 *              Mathematics Dept.
 *              University of Oviedo
 * Date:        Sat Feb 6 1993
 * Copyright (c) 1993, Raul Rivero
 *
 */
/*
 * The algorithm is not too acurate, but this is to me a
 * curiosity not a vital part of LUG.
 *
 * If u wanna fix the problems, please do it ( and thank u ).
 */
```

```
#include "lug.h"
#include "lugfnts.h"
```

```
/*
 * Ok, we recognize these extensions ...
 */
```

```
static char *lug_ff_ext[] = {
    ".tga"
};
```

```
/*
 * Read file functions.
 */
```

```
static int (*read_lug_fnts[])() = {
    read_tga_file
};
```

```
read_lug_file( name, bitmap )
char *name;
bitmap_hdr *bitmap;
{
    ifunptr read_file;
```

```
/*
 * Get the correct funtion to read that file ( please, remember
 * tha we use the file extension and this is not perfect ).
 */
```



```
    read_file = get_readlug_function( name );

    /* Ok, do it. */
    read_file( name, bitmap );
}

ifunptr get_readlug_function( str )
char *str;
{
    ifunptr ptr;
    int value;

    /* Get the correct function for reading */
    ptr = read_lug_fnts[ value = get_index_function(str) ];
    return ptr;
}

get_index_function( str )
char *str;
{
    char dup[132];
    char *ptr;
    int len = strlen( str );
    int indexx;

    /* Duplicates the string */
    strcpy( dup, str );

    /* Check if exists the '.Z' extension ( and discard it ) */
    if ( dup[len-2] == '.' && dup[len-1] == 'Z' )
        dup[len-2] = 0;

    while ( ptr = strrchr( dup, '.' ) ) {
        /*
         * If the extension exists then we return the index
         * to the read/write function else continue ( and
         * discard the extension ).
         */
        if ( (indexx = get_real_index_function( ptr )) != -1 )
            return indexx;
        else ptr[0] = 0;
    }

    return DEFAULT_FORMAT;
}

get_real_index_function( str )
char *str;
{
    register int i = 0;

    while ( lug_ff_ext[i] ) {
        if ( strstr( str, lug_ff_ext[i] ) )
            return i;
        i++;
    }

    return -1;
}
```

```
/*
 * This software is copyrighted as noted below.  It may be freely copied,
 * modified, and redistributed, provided that the copyright notice is
 * preserved on all copies.
 *
 * There is no warranty or other guarantee of fitness for this software,
 * it is provided solely "as is".  Bug reports or fixes may be sent
 * to the author, who may or may not act on them as he desires.
 *
 * You may not include this software in a program or other software product
 * without supplying the source, or without informing the end-user that the
 * source is available for no extra charge.
 *
 * If you modify this software, you should include a notice giving the
 * name of the person performing the modification, the date of modification,
 * and the reason for such modification.
 */
/*
 * dither.c - dithering on images.
 *
 * Author:      Raul Rivero
 *              Mathematics Dept.
 *              University of Oviedo
 * Date:        Tue Feb 11 1992
 * Copyright (c) 1992, Raul Rivero
 */
```

```
#include "lug.h"
#include "lugfnts.h"
```

```
#define DMAP(v,x,y)          (modN[v] > dithermatrix[x][y] ? \
                             divN[v] + 1 : divN[v])
```

```
void create_dithermap();
void create_square();
```

```
dither_image(inbitmap, outbitmap, levels, gamma)
```

```
bitmap_hdr *inbitmap;
```

```
bitmap_hdr *outbitmap;
```

```
int levels;
```

```
double gamma;
```

```
{
    register int i, j;
    int total_size;
    byte *r, *g, *b;
    byte *rout;
    int divN[256], modN[256];
    int dithermatrix[16][16];
    int col, row;
```

```
    if ( inbitmap->magic != LUGUSED )
        error( 19 );
```

```
    if ( levels < 2 || levels > 6 )
        error( 11 );
```

```
    if ( inbitmap->depth <= 8 )
        error( 10 );
```

```
    /* Fill new header */
```

```
outbitmap->magic = LUGUSED;
outbitmap->xsize = inbitmap->xsize;
outbitmap->ysize = inbitmap->ysize;
total_size = outbitmap->xsize * outbitmap->ysize;
outbitmap->depth = no_bits( levels*levels*levels ) + 1;
outbitmap->colors = ( 1 << outbitmap->depth );
outbitmap->cmap = (byte *) Malloc( 3 * outbitmap->colors );
rout = outbitmap->r = (byte *) Malloc( total_size );

/* Pointers to in image */
r = inbitmap->r;
g = inbitmap->g;
b = inbitmap->b;

/*
 * Init the cmap of the dithered image ( and other
 * parameters ).
 */
create_dithermap( levels, gamma, (color_map *) outbitmap->cmap, divN, modN,
dithermatrix);

/*
 * Dither each pixels across the dither matrix.
 */
for ( i = 0; i < outbitmap->ysize; i++ ) {
    row = i % 16;
    for ( j = 0; j < outbitmap->xsize; j++ ) {
        col = j % 16;
        *rout++ = DMAP(*r, col, row) + DMAP(*g, col, row) * levels +
                DMAP(*b, col, row) * levels*levels;
        r++, g++, b++;
    }
}

/*
 * Functions for RGB color dithering.
 *
 * Author:      Spencer W. Thomas
 *              Computer Science Dept.
 *              University of Utah
 * Date:        Mon Feb  2 1987
 * Copyright (c) 1987, University of Utah
 *
 * From Graphics Gems II and Utah Raster Toolkit.
 *
 * The functions name have been changed ( dithermap --> create_... and
 * make_square --> create_square ) due to problems with the RLE header
 * definitions.
 */

static int magic4x4[4][4] = {
    0, 14,  3, 13,
    11,  5,  8,  6,
    12,  2, 15,  1,
    7,  9,  4, 10
};

void
create_dithermap( levels, gamma, rgbmap, divN, modN, magic )
```

```
int levels;
double gamma;
byte rgbmap[][3];
int divN[256];
int modN[256];
int magic[16][16];
{
    double N;
    register int i;
    int levelsq, levelsc;
    byte gammamap[256];

    for ( i = 0; i < 256; i++ )
        gammamap[i] = (byte) (.5 + 255. * pow( i / 255., 1./gamma ));

    levelsq = levels*levels;      /* squared */
    levelsc = levels*levelsq;    /* and cubed */

    N = 255. / (levels - 1.);    /* Get size of each step */

    /*
     * Set up the color map entries.
     */
    for(i = 0; i < levelsc; i++) {
        rgbmap[i][0] = gammamap[(int)(.5 + (i%levels) * N)];
        rgbmap[i][1] = gammamap[(int)(.5 + ((i/levels)%levels) * N)];
        rgbmap[i][2] = gammamap[(int)(.5 + ((i/levelsq)%levels) * N)];
    }

    create_square( N, divN, modN, magic );
}

void
create_square( N, divN, modN, magic )
double N;
int divN[256];
int modN[256];
int magic[16][16] ;
{
    register int i, j, k, l;
    double magicfact;

    for ( i = 0; i < 256; i++ ) {
        divN[i] = (int)(i / N);
        modN[i] = i - (int)(N * divN[i]);
    }
    modN[255] = 0;                /* always */

    /*
     * Expand 4x4 dither pattern to 16x16. 4x4 leaves obvious patterning,
     * and doesn't give us full intensity range (only 17 sublevels).
     *
     * magicfact is (N - 1)/16 so that we get numbers in the matrix from 0 to
     * N - 1: mod N gives numbers in 0 to N - 1, don't ever want all
     * pixels incremented to the next level (this is reserved for the
     * pixel value with mod N == 0 at the next level).
     */
    magicfact = (N - 1) / 16.;
    for ( i = 0; i < 4; i++ )
        for ( j = 0; j < 4; j++ )
            for ( k = 0; k < 4; k++ )
```

```
for ( l = 0; l < 4; l++ )  
    magic[4*k+i][4*l+j] =  
        (int)(0.5 + magic4x4[i][j] * magicfact +  
            (magic4x4[k][l] / 16.) * magicfact);  
}
```

```

/*****
 *
 *   GIFENCOD.C           -   GIF Image compression routines
 *
 *   Lempel-Ziv compression based on 'compress'.  GIF modifications by
 *   David Rowley (mgardi@watdcsu.waterloo.edu)
 *
 *****/

/*
 * General DEFINES
 */
#define GIFBITS 12
#define MSDOS 1

#define HSIZE 5003          /* 80% occupancy */

/*
 * a code_int must be able to hold 2**GIFBITS values of type int, and also -1
 */
typedef int          code_int;

#ifdef SIGNED_COMPARE_SLOW
typedef unsigned long int count_int;
typedef unsigned short int count_short;
#else
typedef long int          count_int;
#endif

#ifdef NO_UCHAR
typedef char  char_type;
#else
typedef      unsigned char  char_type;
#endif /* UCHAR */

/*
 *
 *   GIF Image compression - modified 'compress'
 *
 *   Based on: compress.c - File compression ala IEEE Computer, June 1984.
 *
 *   By Authors:  Spencer W. Thomas      (decvax!harpo!utah-cs!utah-gr!thomas)
 *                Jim McKie             (decvax!mcvax!jim)
 *                Steve Davies          (decvax!vax135!petsd!peora!srd)
 *                Ken Turkowski        (decvax!decwrl!turtlevax!ken)
 *                James A. Woods       (decvax!ihnp4!ames!jaw)
 *                Joe Orost            (decvax!vax135!petsd!joe)
 *
 */
#include "lug.h"
#include "lugfnts.h"
#include <ctype.h>

static output();
static cl_block();
static cl_hash();
static writeerr();
static char_init();
static char_out();
static flush_char();

```

```
static int n_bits; /* number of bits/code */
static int maxbits = GIFBITS; /* user settable max # bits/code */
static code_int maxcode; /* maximum code, given n_bits */
static code_int maxmaxcode = (code_int)1 << GIFBITS; /* should NEVER generate this code */
/*
#ifdef COMPATIBLE /* But wrong! */
# define MAXCODE(n_bits) ((code_int) 1 << (n_bits) - 1)
#else
# define MAXCODE(n_bits) (((code_int) 1 << (n_bits)) - 1)
#endif /* COMPATIBLE */

static count_int htab [HSIZE];
static unsigned short codetab [HSIZE];
#define HashTabOf(i) htab[i]
#define CodeTabOf(i) codetab[i]

static code_int hsize = HSIZE; /* for dynamic table sizing */
static count_int fsize;

/*
 * To save much memory, we overlay the table used by compress() with those
 * used by decompress(). The tab_prefix table is the same size and type
 * as the codetab. The tab_suffix table needs 2**GIFBITS characters. We
 * get this from the beginning of htab. The output stack uses the rest
 * of htab, and contains characters. There is plenty of room for any
 * possible stack (stack used to be 8000 characters).
 */

#define tab_prefixof(i) CodeTabOf(i)
#define tab_suffixof(i) ((char_type *) (htab))[i]
#define de_stack ((char_type *)&tab_suffixof((code_int)1<<GIFBITS))

static code_int free_ent = 0; /* first unused entry */
static int exit_stat = 0;

/*
 * block compression parameters -- after all codes are used up,
 * and compression rate changes, start over.
 */
static int clear_flg = 0;

static int offset;
static long int in_count = 1; /* length of input */
static long int out_count = 0; /* # of codes output (for debugging) */

/*
 * compress stdin to stdout
 *
 * Algorithm: use open addressing double hashing (no chaining) on the
 * prefix code / next character combination. We do a variant of Knuth's
 * algorithm D (vol. 3, sec. 6.4) along with G. Knott's relatively-prime
 * secondary probe. Here, the modular division first probe is gives way
 * to a faster exclusive-or manipulation. Also do block compression with
 * an adaptive reset, whereby the code table is cleared when the compression
 * ratio decreases, but after the table fills. The variable-length output
 * codes are re-sized at this point, and a special CLEAR code is generated
 * for the decompressor. Late addition: construct the table according to
 * file size for noticeable speed improvement on small files. Please direct
 * questions about this implementation to ames!jaw.
 */
```

```
static int g_init_bits;
static FILE *g_outfile;

static int ClearCode;
static int EOFCode;

compress( init_bits, outfile, ReadValue )
int init_bits;
FILE *outfile;
ifunptr ReadValue;
{
    register long fcode;
    register code_int i = 0;
    register int c;
    register code_int ent;
    register code_int disp;
    register code_int hsize_reg;
    register int hshift;

    /*
     * Set up the globals:  g_init_bits - initial number of bits
     *                      g_outfile   - pointer to output file
     */
    g_init_bits = init_bits;
    g_outfile = outfile;

    /*
     * Set up the necessary values
     */
    offset = 0;
    out_count = 0;
    clear_flg = 0;
    in_count = 1;
    maxcode = MAXCODE(n_bits = g_init_bits);

    ClearCode = (1 << (init_bits - 1));
    EOFCode = ClearCode + 1;
    free_ent = ClearCode + 2;

    char_init();

    ent = ReadValue();

    hshift = 0;
    for ( fcode = (long) hsize;  fcode < 65536L; fcode *= 2L )
        hshift++;
    hshift = 8 - hshift;                /* set hash code range bound */

    hsize_reg = hsize;
    cl_hash( (count_int) hsize_reg);    /* clear hash table */

    output( (code_int)ClearCode );

#ifdef SIGNED_COMPARE_SLOW
    while ( (c = ReadValue()) != (unsigned) EOF ) {
#else
    while ( (c = ReadValue()) != EOF ) {
#endif
        in_count++;
    }
}
```



```
fcode = (long) (((long) c << maxbits) + ent);
i = (((code_int)c << hshift) ^ ent);    /* xor hashing */

if ( HashTabOf (i) == fcode ) {
    ent = CodeTabOf (i);
    continue;
} else if ( (long)HashTabOf (i) < 0 )      /* empty slot */
    goto nomatch;
disp = hsize_reg - i;                    /* secondary hash (after G. Knott) */
if ( i == 0 )
    disp = 1;
```

probe:

```
if ( (i -= disp) < 0 )
    i += hsize_reg;

if ( HashTabOf (i) == fcode ) {
    ent = CodeTabOf (i);
    continue;
}
if ( (long)HashTabOf (i) > 0 )
    goto probe;
```

nomatch:

```
output ( (code_int) ent );
out_count++;
ent = c;
#ifdef SIGNED_COMPARE_SLOW
    if ( (unsigned) free_ent < (unsigned) maxmaxcode) {
#else
    if ( free_ent < maxmaxcode ) {
#endif
        CodeTabOf (i) = free_ent++; /* code -> hashtable */
        HashTabOf (i) = fcode;
    } else
        cl_block();
}
```

```
/*
 * Put out the final code.
 */
output( (code_int)ent );
out_count++;
output( (code_int) EOFCode );

return;
}
```

```
/******
 * TAG( output )
 *
 * Output the given code.
 * Inputs:
 *     code:   A n_bits-bit integer.  If == -1, then EOF.  This assumes
 *             that n_bits =< (long)wordsize - 1.
 * Outputs:
 *     Outputs code to the file.
 * Assumptions:
 *     Chars are 8 bits long.
 * Algorithm:
 *     Maintain a GIFBITS character long buffer (so that 8 codes will
 * fit in it exactly).  Use the VAX insv instruction to insert each
 * code in turn.  When the buffer fills up empty it and start over.
 */
```

```
static unsigned long cur_accum = 0;
static int cur_bits = 0;

static
unsigned long masks[] = { 0x0000, 0x0001, 0x0003, 0x0007, 0x000F,
                          0x001F, 0x003F, 0x007F, 0x00FF,
                          0x01FF, 0x03FF, 0x07FF, 0x0FFF,
                          0x1FFF, 0x3FFF, 0x7FFF, 0xFFFF };

static
output( code )
code_int code;
{
    cur_accum &= masks[ cur_bits ];

    if( cur_bits > 0 )
        cur_accum |= ((long)code << cur_bits);
    else
        cur_accum = code;

    cur_bits += n_bits;

    while( cur_bits >= 8 ) {
        char_out( (unsigned int)(cur_accum & 0xff) );
        cur_accum >>= 8;
        cur_bits -= 8;
    }

    /*
     * If the next entry is going to be too big for the code size,
     * then increase it, if possible.
     */
    if ( free_ent > maxcode || clear_flg ) {

        if( clear_flg ) {

            maxcode = MAXCODE (n_bits = g_init_bits);
            clear_flg = 0;

        } else {

            n_bits++;
            if ( n_bits == maxbits )
                maxcode = maxmaxcode;
            else
                maxcode = MAXCODE(n_bits);
        }
    }

    if( code == EOFCode ) {
        /*
         * At EOF, write the rest of the buffer.
         */
        while( cur_bits > 0 ) {
            char_out( (unsigned int)(cur_accum & 0xff) );
            cur_accum >>= 8;
            cur_bits -= 8;
        }

        flush_char();
    }
}
```

```
fflush( g_outfile );

if( ferror( g_outfile ) )
    writeerr();
}

/*
 * Clear out the hash table
 */
static
cl_block ( )          /* table clear for block compress */
{

    cl_hash ( (count_int) hsize );
    free_ent = ClearCode + 2;
    clear_flg = 1;

    output( (code_int)ClearCode );
}

static
cl_hash(hsize)        /* reset code table */
register count_int hsize;
{

    register count_int *htab_p = htab+hsize;

    register long i;
    register long m1 = -1;

    i = hsize - 16;
    do {
        /* might use Sys V memset(3) here */
        *(htab_p-16) = m1;
        *(htab_p-15) = m1;
        *(htab_p-14) = m1;
        *(htab_p-13) = m1;
        *(htab_p-12) = m1;
        *(htab_p-11) = m1;
        *(htab_p-10) = m1;
        *(htab_p-9) = m1;
        *(htab_p-8) = m1;
        *(htab_p-7) = m1;
        *(htab_p-6) = m1;
        *(htab_p-5) = m1;
        *(htab_p-4) = m1;
        *(htab_p-3) = m1;
        *(htab_p-2) = m1;
        *(htab_p-1) = m1;
        htab_p -= 16;
    } while ((i -= 16) >= 0);

    for ( i += 16; i > 0; i-- )
        *--htab_p = m1;
}

static
writeerr()
{
    printf( "error writing output file\n" );
}
```

```
        exit(1);
    }

/*****
 *
 * GIF Specific routines
 *
 *****/

/*
 * Number of characters so far in this 'packet'
 */
static int a_count;

/*
 * Set up the 'byte output' routine
 */
static
char_init()
{
    a_count = 0;
}

/*
 * Define the storage for the packet accumulator
 */
static char accum[ 256 ];

/*
 * Add a character to the end of the current packet, and if it is 254
 * characters, flush the packet to disk.
 */
static
char_out( c )
int c;
{
    accum[ a_count++ ] = c;
    if( a_count >= 254 )
        flush_char();
}

/*
 * Flush the packet to disk, and reset the accumulator
 */
static
flush_char()
{
    if( a_count > 0 ) {
        fputc( a_count, g_outfile );
        fwrite( accum, 1, a_count, g_outfile );
        a_count = 0;
    }
}

/* The End */
```

```
/*
 * This software is copyrighted as noted below.  It may be freely copied,
 * modified, and redistributed, provided that the copyright notice is
 * preserved on all copies.
 *
 * There is no warranty or other guarantee of fitness for this software,
 * it is provided solely "as is".  Bug reports or fixes may be sent
 * to the author, who may or may not act on them as he desires.
 *
 * You may not include this software in a program or other software product
 * without supplying the source, or without informing the end-user that the
 * source is available for no extra charge.
 *
 * If you modify this software, you should include a notice giving the
 * name of the person performing the modification, the date of modification,
 * and the reason for such modification.
 */
/*
 * error.c - error function.
 *
 * Author:      Raul Rivero
 *              Mathematics Dept.
 *              University of Oviedo
 * Date:        Sat Jan 4 1992
 * Copyright (c) 1992, Raul Rivero
 */

#include "lug.h"

char *MY_NAME = "liblug";

int lugerrno = 0;
int stop_on_error = 1;

char *lug_errlist[] = {
    /* 0 */    "Bad format of usage\n",
    /* 1 */    "Cannot open file\n",
    /* 2 */    "Out of memory\n",
    /* 3 */    "Error while reading input file\n",
    /* 4 */    "Error while writing output file\n",
    /* 5 */    "Unkown input file type\n",
    /* 6 */    "File corrupt ( uncompress too bytes )\n",
    /* 7 */    "File is not a RGB image or gray scaled\n",
    /* 8 */    "Uncompress failed or compressed file don't exist\n",
    /* 9 */    "Unkown encoding type\n",
    /* 10 */   "Incorrect number of planes\n",
    /* 11 */   "Incorrect number of levels for dither\n",
    /* 12 */   "Incorrect size of images\n",
    /* 13 */   "Mapped image without color map ( ?! )\n",
    /* 14 */   "Incorrect image bits ( only 16, 24 or 32 )\n",
    /* 15 */   "File is not a mapped image\n",
    /* 16 */   "Cannot open graphics display\n",
    /* 17 */   "Interlazed image\n",
    /* 18 */   "File contains an uncomplete image\n",
    /* 19 */   "Using an image with MAGIC unset\n",
    /* 20 */   "Standard input not avaible with this format\n",
    /* 21 */   "Cannot get a Tag from the TIFF file\n",
    /* 22 */   "Incorrect resampling values\n",
    /* 23 */   "Image is not RGB/Gray scaled image\n",
    NULL
}
```

```
};

char *lugerrmsg( code )
{
    return lug_errlist[ code ];
}

Error(code)
int code;
{
    int static last = 0;

    lugerrno = code;

    if ( !last ) {
        /*
         * Search, the first time, the last
         * error message.
         */
        for ( ; lug_errlist[last]; last++);
    }

    if ( stop_on_error ) {
        /*
         * Write the error message and
         * stop the program.
         */
        fprintf(stderr, "%s: ", MY_NAME);
        switch ( code ) {
            case 99:
                fprintf(stderr, "Not ready\n");
                break;
            default:
                if ( code >= 0 && code <= last ) {
                    fputs( lug_errlist[code], stderr );
                } else {
                    fprintf( stderr, "Unknown error code (%d)\n", code);
                }
                break;
        }
        exit( 1 );
    }
}
```

```
/*
 * This software is copyrighted as noted below.  It may be freely copied,
 * modified, and redistributed, provided that the copyright notice is
 * preserved on all copies.
 *
 * There is no warranty or other guarantee of fitness for this software,
 * it is provided solely "as is".  Bug reports or fixes may be sent
 * to the author, who may or may not act on them as he desires.
 *
 * You may not include this software in a program or other software product
 * without supplying the source, or without informing the end-user that the
 * source is available for no extra charge.
 *
 * If you modify this software, you should include a notice giving the
 * name of the person performing the modification, the date of modification,
 * and the reason for such modification.
 */
/*
 * general.c - general ( non clasified ) functions.
 *
 * Author:      Raul Rivero
 *              Mathematics Dept.
 *              University of Oviedo
 * Date:        Sat Jan 4 1992
 * Copyright (c) 1992, Raul Rivero
 *
 */
```

```
#include "lug.h"
#include "lugfnts.h"
```

```
int LUGverbose = 0;
```

```
/*
 * Return the number of bits ( -1 ) to represent a given
 * number of colors ( ex: 256 colors => 7 ).
 */
```

```
no_bits( colors )
int colors;
{
    register int bits= 0;

    colors--;
    while ( colors >> bits )
        bits++;

    return (bits-1);
}
```

```
/*
 * Like basename command.
 */
char *basename(string, suffix)
char *string;
char *suffix;
{
    char *st_ptr= string;
    char *su_ptr= suffix;
```

```
/*
 * Pointers to the end of strings.
```

```
    */
    while (*st_ptr)
        st_ptr++;
    while (*su_ptr)
        su_ptr++;

    /*
     * Ahora los vamos retrasando hasta que sean distintas
     * ( y en el string metemos un cero ).
     */
    while ( *st_ptr == *su_ptr )
        st_ptr--, su_ptr--;

    /* The end of the string */
    *(++st_ptr) = 0;

    return( string );
}

short *bytetoshort( in, out, size )
byte *in;
short *out;
int size;
{
    short *ptr;

    /*
     * If the output buffer is NULL then we need allocate
     * memory, so ...
     */
    if ( out == NULL ) {
        ptr = out = (short *) Malloc( size * sizeof(short) );
    } else ptr = out;

    while ( size-- ) {
        *ptr++ = *in++;
    }

    return out;
}

byte *shorttobyte( in, out, size )
short *in;
byte *out;
int size;
{
    byte *ptr;

    /*
     * If the output buffer is NULL then we need allocate
     * memory, so ...
     */
    if ( out == NULL ) {
        ptr = out = (byte *) Malloc( size );
    } else ptr = out;

    while ( size-- ) {
        *ptr++ = *in++;
    }

    return out;
}
```



```
}

byte *floattobyte( in, out, size )
float *in;
byte *out;
int size;
{
    byte *ptr;

    /*
     * If the output buffer is NULL then we need allocate
     * memory, so ...
     */
    if ( out == NULL ) {
        ptr = out = (byte *) Malloc( size );
    } else ptr = out;

    while ( size-- ) {
        *ptr++ = (byte) (255. * *in++);
    }

    return out;
}

float *bytetofloat( in, out, size )
float *out;
byte *in;
int size;
{
    float *ptr;

    /*
     * If the output buffer is NULL then we need allocate
     * memory, so ...
     */
    if ( out == NULL ) {
        ptr = out = (float *) Malloc( size * sizeof(float));
    }

    while ( size-- ) {
        *ptr++ = ((float)*in++) / 255.;
    }

    return out;
}

Atoi( string )
char *string;
{
    int aux = -1234;

    sscanf( string, "%d", &aux );
    if ( aux == -1234 )
        error( 0 );

    return aux;
}

double
Atod( string )
char *string;
```

```
{
    double aux = -1234.;

    sscanf( string, "%lf", &aux );
    if ( aux == -1234. )
        error( 0 );

    return aux;
}

isnumber( c )
char *c;
{
    while ( *c )
        if ( !isdigit(*c++) )
            return 0;

    return 1;
}

/*
 * Compress and uncompress subroutines.
 */

Uncompress(name, aux_file)
char *name, *aux_file;
{
#ifdef MSDOS
    int pid, handle;

    if ( pid = fork() ) {
        /*
         * Parent wait until the REAL 'end of days'
         * of the child.
         */
        do{
        }while( pid != wait(NULL) );
    }else {
        handle = creat( aux_file, 0644 );
        close( 1 );
        dup( handle );
#ifdef USE_GNU_GZIP
        execlp( "gzip", "gzip", "-dc", name, 0 );
#else
        execlp( "compress", "compress", "-dc", name, 0 );
#endif /* USE_GNU_GZIP */
    }
#else /* MSDOS */
    int handle;
    char cmd[132];

    handle = creat( aux_file, 0644 );
    close( 1 );
#ifdef USE_GNU_GZIP
    sprintf( cmd, "gzip -dc %s", aux_file );
#else
    sprintf( cmd, "compress -dc %s", aux_file );
#endif /* USE_GNU_GZIP */
    dup( handle );
    system( cmd );
}
```

```
#endif /* MSDOS */
}

Compress(name)
char *name;
{
#ifdef MSDOS
    int pid;

    if ( pid = fork() ) {
        /*
         * Parent wait until the REAL 'end of days'
         * of the child.
         */
        do{
        }while( pid != wait(NULL) );
    }else {
        /*
         * Lets go baby ...
         */
#ifdef USE_GNU_GZIP
        execlp( "gzip", "gzip", "-9", name, 0 );
#else
        execlp( "compress", "compress", name, 0 );
#endif /* USE_GNU_GZIP */
    }
#else /* MSDOS */
    char cmd[132];

#ifdef USE_GNU_GZIP
    sprintf( cmd, "gzip -9 %s", name );
#else
    sprintf( cmd, "compress %s", name );
#endif /* USE_GNU_GZIP */
    system( cmd );
#endif /* MSDOS */
}

compute_levels( no_colors )
int no_colors;
{
    int n = 7;

    do {
        n--;
    }while ( n*n*n > no_colors );

    return n;
}
```

```
/*
 * This software is copyrighted as noted below.  It may be freely copied,
 * modified, and redistributed, provided that the copyright notice is
 * preserved on all copies.
 *
 * There is no warranty or other guarantee of fitness for this software,
 * it is provided solely "as is".  Bug reports or fixes may be sent
 * to the author, who may or may not act on them as he desires.
 *
 * You may not include this software in a program or other software product
 * without supplying the source, or without informing the end-user that the
 * source is available for no extra charge.
 *
 * If you modify this software, you should include a notice giving the
 * name of the person performing the modification, the date of modification,
 * and the reason for such modification.
 */
/*
 * gif.c - interface with Compuserve's GIF format.
 *
 * Authors:      Raul Rivero
 *               Mathematics Dept.
 *               University of Oviedo
 * Date:         Wed Jan 8 1992
 * Copyright (c) 1992, Raul Rivero
 */
```

```
#include "lug.h"
#include "lugfnts.h"
```

```
#define GIFHEADER          "GIF87a"
#define NEWGIFHEADER       "GIF89a"
#define GIFIMGSEPAR        ' ,'
#define ENDGIF              ' ; '
#define GIFEXISTCOLOR      (1 << 7)
#define GIFINTERLAZE       (1 << 6)
```

```
byte *read_gif_screen_hdr();
byte *read_gif_image_hdr();
byte *unblock();
```

```
/*int oldread;*/
byte *ptr_image;
int image_size;
int lug_read_code_position;
```

```
extern int LUGverbose;
```

```
read_gif_file( name, bitmap )
char *name;
bitmap_hdr *bitmap;
{
    FILE *handle;

    /* Open the file descriptor */
    if ( name != NULL )
        handle = Fopen( name, "rb" );
    else error( 20 );

    lug_read_code_position = 0;
```

```
/* Read the bitmap */
read_gif( handle, bitmap );
rm_compress();

/* Close the file */
Fclose( handle );
}

read_gif( handle, image )
FILE *handle;
bitmap_hdr *image;
{
    int xsize, ysize;
    byte *globalcmap, *localcmap;
    int mask;
    int codesize;
    int totalsize;
    int globalcolors, localcolors;
    int localmask, globalmask;

    /*
     * Read GIF headers ( we skip many parameters ).
     */
    read_gif_hdr( handle );
    globalcmap = read_gif_screen_hdr( handle, &globalcolors, &globalmask );
    localcmap = read_gif_image_hdr( handle, &localcolors, &localmask,
                                   &xsize, &ysize );

    /*
     * We only use one color map.
     */
    if ( localcolors ) {
        mask = localmask;
        image->cmap = localcmap;
        if ( globalcolors )
            free( globalcmap );
    }else
        if ( globalcolors ) {
            mask = globalmask;
            image->cmap = globalcmap;
        }else error( 13 );

    /*
     * Original ( root ) codesize.
     */
    codesize= fgetc( handle );

    /*
     * Fill our header.
     */
    totalsize = xsize * ysize;
    image->magic = LUGUSED;
    image->xsize = xsize;
    image->ysize = ysize;
    image->colors = ( localcolors ? localcolors : globalcolors );
    image->depth = no_bits( image->colors ) + 1;
    image->r = (byte *) Malloc( totalsize );
    uncode_gif( handle, codesize, mask, image );
}
```

```
#define READCODE()      { code = read_code( ptrblocks, datamask, \
                                &offset, codesize ); \
                                ptrblocks += offset; }
```

```
uncode_gif( handle, codesize, mask, image )
FILE *handle;
int codesize, mask;
bitmap_hdr *image;
{
    int endblock;
    int clearcode;
    int offset;
    int freecode;
    int code;
    int current, old;
    int datamask;
    int codetop;
    int orig_codesize;
    short *sufix, *prefix;
    byte *blocks, *ptrblocks;
    int count = 0;
    byte *stack;
    int k;
    int totalsize;
    int position;

    /*
     * Allocate memory for internal buffers.
     */
    stack = (byte *) Malloc( 4096 );
    sufix = (short *) Malloc( 4096 * sizeof(short) );
    prefix = (short *) Malloc( 4096 * sizeof(short) );

    /*
     * Compute predefined values for uncompress.
     */
    clearcode = (1 << codesize);
    endblock = clearcode + 1;
    freecode = clearcode + 2;
    orig_codesize = ++codesize;
    codetop = 1 << codesize;
    datamask = codetop - 1;

    /*
     * Unblock the raster information.
     */
    ptrblocks = blocks = (byte *) unblock( handle );

    totalsize = image->xsize * image->ysize;

    /*
     * and now, ... UNPACK !!!
     */
    VPRINTF( stderr, "Uncompressing GIF raster information\n" );
    READCODE(); /* read first code */
    while ( code != endblock ) {
        if ( code == clearcode ) {
            /*
             * Current code is clear so we reinitialize our tables.
             */
            codesize = orig_codesize;
        }
    }
}
```

```
freecode = clearcode + 2;
codetop = 1 << codesize;
datamask = codetop - 1;
READCODE();
old = current = code;
/* Next code to 'clear' is a root code */
k = code & mask;
position = push_gif( image->r, k );
}else {
/*
 * We have a normal code.
 */
current = code;
/*
 * If it's a new code then we need add the last translation.
 */
if( !(code < freecode) ) {
    current = old;
    stack[count++] = (byte) k;
}

/*
 * mask equals to 'last_root_code' so while current
 * code greater [than mask] we haven't a root code
 * and continue adding values into ioutput.
 */
while ( current > mask ) {
    stack[count++] = suffix[current];
    current = prefix[current];
}
k = current & mask;
stack[count] = k;
for ( ; count >= 0; count-- )
    position = push_gif( image->r, (int) stack[count] );
count = 0;
prefix[freecode ] = old;
suffix [freecode++] = k;
old = code;
/*
 * Check if we use all posibles values ( for this
 * codesize ).
 */
if ( freecode >= codetop && codesize != 12 ) {
    codesize++;
    codetop = 1 << codesize;
    datamask = codetop - 1;
}
}
READCODE();
if ( position > totalsize )
    /*error(6)*/ break;
}

/*
 * Free original GIF raster information and
 * internal buffers.
 */
free( blocks );
free( prefix );
free( suffix );
free( stack );
```

```
}

read_code( buffer, mask, offset, codesize )
byte *buffer;
int mask;
int *offset;
int codesize;
{
    int aux = 0;
    int moveptr;

    aux = *buffer++;
    aux |= ( *buffer++ << 8 );
    if ( codesize > 7 )
        aux |= ( *buffer++ << 16 );
    aux >>= lug_read_code_position;
    lug_read_code_position += codesize;
    moveptr = lug_read_code_position / 8;
    if ( moveptr ) {
        *offset = moveptr;
        lug_read_code_position %= 8;
    }else *offset = 0;

    return( aux & mask );
}

push_gif(buffer, indexx)
byte *buffer;
int indexx;
{
    static int offset;
    static byte *ptr;
    static byte *baseptr;

    if ( baseptr != buffer ) {
        ptr = baseptr = buffer;
        offset = 0;
    }

    *ptr++ = (byte) indexx;
    return ++offset;
}

read_gif_hdr( handle )
FILE *handle;
{
    char buffer[6];

    Fread( buffer, 6, 1, handle);
    if ( strncmp( buffer, GIFHEADER, 6 ) &&
        strncmp( buffer, NEWGIFHEADER, 6 ) )
        error( 5 );
}

byte *read_gif_screen_hdr( handle, colors, mask )
FILE *handle;
int *colors;
int *mask;
{
    /* int swidth, sheight; */
    byte buffer[7];
```



```
byte *cmap;
int cmapflag;
int bitsperpixel;
/* int background; */

VPRINTF( stderr, "Reading GIF header\n" );

Fread( buffer, 7, 1, handle);

/* Screen size */
/* swidth    = (buffer[1] << 8) | buffer[0]; */
/* sheight   = (buffer[3] << 8) | buffer[2]; */

cmapflag = buffer[4] & GIFEXISTCOLOR;
bitsperpixel = (buffer[4] & 7) + 1;
*colors = 1 << bitsperpixel;
*mask = *colors - 1;
/* background= buffer[5]; */

if ( buffer[6] )
    errornull( 6 );

if ( cmapflag ) {
    cmap = (byte *) Malloc( 3 * *colors );
    Fread( cmap, *colors, 3, handle );
}

return cmap;
}

byte *read_gif_image_hdr( handle, colors, mask, width, height )
FILE *handle;
int *colors;
int *mask;
int *width, *height;
{
    /* int left, right; */
    byte buffer[10];
    byte *cmap;
    int cmapflag;
    int interlace;
    int bitsperpixel;

    VPRINTF( stderr, "Reading GIF image header\n" );

    Fread( buffer, 10, 1, handle );

    if ( buffer[0] != GIFIMGSEPAR ) {
        if (buffer[0] == '!') {
            int c;

            while ((c=getc(handle)) > 0) {
            }
            if (c == EOF)
                errornull( 5 );
            Fread( buffer, 10, 1, handle );
            if ( buffer[0] != GIFIMGSEPAR )
                errornull( 5 );
        }
    }
}
```

```
/* left      = (buffer[2] << 8) | buffer[1]; */
/* right     = (buffer[4] << 8) | buffer[3]; */
*width      = (buffer[6] << 8) | buffer[5];
*height     = (buffer[8] << 8) | buffer[7];
cmapflag    = buffer[9] & GIFEXISTCOLOR;
interlace   = buffer[9] & GIFINTERLAZE;

if ( interlace )
    errornull( 17 );

if ( cmapflag ) {
    bitsperpixel = ( buffer[9] & 7 ) + 1;
    *colors = 1 << bitsperpixel;
    *mask = *colors - 1;
    cmap = (byte *) Malloc( 3 * *colors );
    Fread( cmap, *colors, 3, handle );
}else *colors = 0;

return cmap;
}

byte *unblock( handle )
FILE *handle;
{
    long position;
    int totalsize;
    byte *out, *ptr;
    int size;
    int count = 0;

    VPRINTF( stderr, "Unblocking GIF file\n" );
    /*
     * We need read the raster information and strip the block
     * size marks. GIF files can store more than one image but
     * this is not usually ( so we move handle to the end and
     * suppose that size ).
     */
    position = ftell( handle );
    fseek( handle, 0L, 2 );
    totalsize = ftell( handle ) - position;
    fseek( handle, position, 0 );

    ptr = out = (byte *) Malloc( totalsize );

    /*
     * Format is:  <size><...block...><size><...block...>[...]<0>
     */
    size = fgetc( handle );
    while ( size ) {
        /* Check if no problems with space */
        count += size;
        if ( count > totalsize )
            errornull( 7 );
        /* No problems => read the block */
        Fread( ptr, size, 1, handle );
        ptr += size;
        size = fgetc( handle );
    }

    return out;
}
```

```
write_gif_file( name, image )
char *name;
bitmap_hdr *image;
{
    FILE *handle;

    /* Open the file descriptor */
    if ( name != NULL )
        handle = Fopen( name, "wb" );
    else handle = stdout;

    /* Write the bitmap */
    write_gif( handle, image );

    /* Close the file */
    Fclose( handle );
}

write_gif(handle, image)
FILE *handle;
bitmap_hdr *image;
{
    int codesize;
    int read_pixel();

    if ( image->magic != LUGUSED )
        error( 19 );

    /* GIF only support mapped images */
    if ( image->depth > 8 )
        error( 15 );

    VPRINTF(stderr, "Writing GIF headers\n");
    /*
     * Write GIF headers.
     */
    write_gif_hdr( handle );
    write_gif_screen_hdr( handle, image );
    write_gif_cmap( handle, image );
    write_gif_image_hdr( handle, image );

    /*
     * Now store the original code size.
     */
    codesize = image->depth;
    fputc( codesize, handle );

    /*
     * Image size.
     */
    image_size = image->xsize * image->ysize;
    /* el primer codigo libre sera de +1 bits */
    codesize++;
    /*
     * Compress the image.
     */
    VPRINTF(stdout, "Compressing raster information\n");
    ptr_image = image->r;
    compress( codesize, handle, read_pixel );
```

```
/*
 * Block with a size of 0 bytes.
 */
fputc( 0, handle );

/*
 * End of gif file.
 */
fputc( ENDGIF, handle );      /* fin del fichero GIF */
}

write_gif_hdr(handle)
FILE *handle;
{
    Fwrite( GIFHEADER, 6, 1, handle );
}

write_gif_screen_hdr(handle, image)
FILE *handle;
bitmap_hdr *image;
{
    byte buffer[7];
    byte *ptr;

    ptr = buffer;
    /* Width ( of screen ) */
    *ptr++ = LSB(image->xsize);
    *ptr++ = MSB(image->xsize);
    /* Height ( of screen ) */
    *ptr++ = LSB(image->ysize);
    *ptr++ = MSB(image->ysize);
    /* Colors */
    *ptr++ = (1 << 7) | (no_bits(image->colors) << 4) | (no_bits(image->colors));
    /* Background color */
    *ptr++ = 0;
    /* End of block */
    *ptr++ = 0;

    Fwrite( buffer, 7, 1, handle);
}

write_gif_cmap(handle, image)
FILE *handle;
bitmap_hdr *image;
{
    Fwrite(image->cmap, 3, image->colors, handle);
}

write_gif_image_hdr(handle, image)
FILE *handle;
bitmap_hdr *image;
{
    byte buffer[10];
    byte *ptr;

    ptr = buffer;
    /* Separator */
    *ptr++ = GIFIMGSEPAR;
    /* Left adjust */
    *ptr++ = 0;
```

```
*ptr++ = 0;
/* Right adjust */
*ptr++ = 0;
*ptr++ = 0;
/* Width ( of image ) */
*ptr++ = LSB(image->xsize);
*ptr++ = MSB(image->xsize);
/* Height ( of image ) */
*ptr++ = LSB(image->ysize);
*ptr++ = MSB(image->ysize);
/* Word of definition ( without colormap, ... ) */
*ptr++ = 7;

Fwrite( buffer, 10, 1, handle);
}

read_pixel()
{
    static int size;
    static byte *base, *ptr;

    if ( base != ptr_image ) {
        /*
         * It's a new pointer, so we need stored.
         */
        base = ptr = ptr_image;
        size = image_size;
    }

    if ( size ) {
        /*
         * We have pixels ...
         */
        size--;
        return( (int)(*ptr++) );
    }else {
        return EOF; /* no more pixels, this is the end */
    }
}
```

```
/*
 * This software is copyrighted as noted below.  It may be freely copied,
 * modified, and redistributed, provided that the copyright notice is
 * preserved on all copies.
 *
 * There is no warranty or other guarantee of fitness for this software,
 * it is provided solely "as is".  Bug reports or fixes may be sent
 * to the author, who may or may not act on them as he desires.
 *
 * You may not include this software in a program or other software product
 * without supplying the source, or without informing the end-user that the
 * source is available for no extra charge.
 *
 * If you modify this software, you should include a notice giving the
 * name of the person performing the modification, the date of modification,
 * and the reason for such modification.
 */
/*
 * hsl.c - convert rgb images to hsl images.
 *
 * Author:      Raul Rivero
 *              Mathematics Dept.
 *              University of Oviedo
 * Date:        Sat Feb 1 1992
 * Copyright (c) 1992, Raul Rivero
 *
 */
/*
 * RGB <--> HSL routines extracted from Graphics Gems I.
 */
```

```
#include "lug.h"
#include "lugfnts.h"
```

```
hsl_to_rgb_buffer( h, s, l, r, g, b, size )
double *h, *s, *l;
byte *r, *g, *b;
int size;
{
    while ( size-- ) {
        HSL_to_RGB( *h, *s, *l, r, g, b );
        h++, s++, l++;
        r++, g++, b++;
    }
}
```

```
rgb_to_hsl_buffer( r, g, b, h, s, l, size )
double *h, *s, *l;
byte *r, *g, *b;
int size;
{
    while ( size-- ) {
        RGB_to_HSL(*r, *g, *b, h, s, l );
        h++, s++, l++;
        r++, g++, b++;
    }
}
```

```
RGB_to_HSL( r1, g1, b1, h, s, l)
byte r1, g1, b1;
double *h, *s, *l;
```

```
{
    double v;
    double m;
    double vm;
    double r2, g2, b2;
    double r, g, b;

    r = ((double)r1) / 255.;
    g = ((double)g1) / 255.;
    b = ((double)b1) / 255.;

    v = LUGMAX( r, LUGMAX( g, b ) );
    m = LUGMIN( r, LUGMIN( g, b ) );

    if ((*l = (m + v) / 2.0) <= 0.0)
        return;
    if ((*s = vm = v - m) > 0.0) {
        *s /= (*l <= 0.5) ? (v + m) : (2.0 - v - m);
    } else return;

    r2 = (v - r) / vm;
    g2 = (v - g) / vm;
    b2 = (v - b) / vm;

    if (r == v)
        *h = (g == m ? 5.0 + b2 : 1.0 - g2);
    else if (g == v)
        *h = (b == m ? 1.0 + r2 : 3.0 - b2);
    else
        *h = (r == m ? 3.0 + g2 : 5.0 - r2);

    *h /= 6;
}

HSL_to_RGB( h, sl, l, r, g, b)
double h, sl, l;
byte *r, *g, *b;
{
    double v;

    v = (l <= 0.5) ? (l * (1.0 + sl)) : (l + sl - l * sl);
    if (v <= 0) {
        *r = *g = *b = 0;
    } else {
        double m;
        double sv;
        int sextant;
        double fract, vsf, mid1, mid2;

        m = l + l - v;
        sv = (v - m) / v;
        h *= 6.0;
        sextant = h;
        fract = h - sextant;
        vsf = v * sv * fract;
        mid1 = 255 * (m + vsf);
        mid2 = 255 * (v - vsf);
        v *= 255; m *= 255;
        switch (sextant) {
            case 0:
                *r = CORRECT(v);
```

```
        *g = CORRECT(mid1);
        *b = CORRECT(m);
        break;
    case 1:
        *r = CORRECT(mid2);
        *g = CORRECT(v);
        *b = CORRECT(m);
        break;
    case 2:
        *r = CORRECT(m);
        *g = CORRECT(v);
        *b = CORRECT(mid1);
        break;
    case 3:
        *r = CORRECT(m);
        *g = CORRECT(mid2);
        *b = CORRECT(v);
        break;
    case 4:
        *r = CORRECT(mid1);
        *g = CORRECT(m);
        *b = CORRECT(v);
        break;
    case 5:
        *r = CORRECT(v);
        *g = CORRECT(m);
        *b = CORRECT(mid2);
        break;
    }
}
```



```
/*
 * This software is copyrighted as noted below.  It may be freely copied,
 * modified, and redistributed, provided that the copyright notice is
 * preserved on all copies.
 *
 * There is no warranty or other guarantee of fitness for this software,
 * it is provided solely "as is".  Bug reports or fixes may be sent
 * to the author, who may or may not act on them as he desires.
 *
 * You may not include this software in a program or other software product
 * without supplying the source, or without informing the end-user that the
 * source is available for no extra charge.
 *
 * If you modify this software, you should include a notice giving the
 * name of the person performing the modification, the date of modification,
 * and the reason for such modification.
 */
/*
 * in_out.c - input and output subroutines.
 *
 * Author:      Raul Rivero
 *              Mathematics Dept.
 *              University of Oviedo
 * Date:        Sat Jan 4 1992
 * Copyright (c) 1992, Raul Rivero
 */

/*
 * Subrutinas para entrada y salida de ficheros.
 */

#include "lug.h"
#include "lugfnts.h"

extern int getpid();
extern char *getenv();

static char aux_file[80];      /* temporal file for uncompressing files */

/*
 * Fread and Fwrite functions are defined in lug.h.
 */

FILE *Fopen(name, mode)
char *name;
char *mode;
{
    static byte first_fopen = 0;
    FILE *handle;
    int flag;
    struct stat statbuf;

    if ( mode[0] == 'r' ) {
        if ( !(flag = exist_file(name)) )
            error( 1 ); /* the file doesn't exist */

        if ( flag == FILE_EXIST ) {
            /* The file exist with the <name> name :-), so open it !*/
            if ( (handle = fopen( name, mode )) == NULL )
                error( 1 );
        }
    }
}
```

```
    return handle;
}else {
    /*
     * The file exist, but compressed!. So we uncompressed over
     * a tmp file and return a handle for that file.
     */
    if ( !first_fopen ) {        /* is the first time ? */
        first_fopen++;
        sprintf( aux_file, "%s/lug%d", TMPDIR, getpid() );
    }
    /* Uncompress it */
    Uncompress( name, aux_file );
    /* Open it */
    if ( ( handle = fopen(aux_file, mode) ) == NULL) {
        fputs( aux_file, stderr );
        error(1);
    }else {
        stat( aux_file, &statbuf );
        if ( statbuf.st_size < 1 )
            error( 1 );
    }
    return handle;
}
}
}else {
    /*
     * Ask for writing ( we don't need check for
     * compressed files ).
     */
    if ( (handle = fopen(name, mode)) == NULL)
        error(1);
}

return handle;
}

exist_file( name )
char *name;
{
    int len = strlen(name) - 1;
    char aux[132];

    /* No characters ?! */
    if ( len < 0 )
        return FILE_NO_EXIST;

    /* Exists this file ? */
    if ( !access( name, 0 ) ) {
        /*
         * Ok, exist, but is a compressed file ?.
         */
        if ( !strcmp(&(name[len-2]), ".gz") )
            return FILE_IS_GZIP;    /* was a .gz file ==> a GZIP file */
        else if ( !strcmp(&(name[len-1]), ".Z") )
            return FILE_IS_COMPRESS;    /* was a .Z file ==> a COMPRESS file */

        return FILE_EXIST;    /* well, a normal file :- ) */
    }

    /*
     * The file doesn't exist, so check compressed versions.
     */
}
```

```
sprintf( aux, "%s.gz", name );
if ( !access( aux, 0 ) )
    return FILE_WITH_GZIP;    /* we have a <name>.gz file */
sprintf( aux, "%s.Z", name );
if ( !access( aux, 0 ) )
    return FILE_WITH_COMPRESS; /* we have a <name>.Z file */

/* Oopppsssss!, no file */
return FILE_NO_EXIST;
}

Fclose( handle )
FILE *handle;
{
    if ( handle != stdout && handle != stdin )
        return fclose( handle );
    else return 0;
}

rm_compress()
{
    if ( access( aux_file, 0 ) == 0 )
        unlink( aux_file );
}

char *read_file(handle, bytes)
FILE *handle;
int *bytes;
{
    char *buffer;

    /* Get size of file and allocate memory */
    *bytes = (int) filelen(handle);
    if ( *bytes > 0 ) {                /* Ok, it's a regular file. */
        buffer = (char *) Malloc(*bytes);

        /* Read it */
        Fread(buffer, (int) (*bytes), 1, handle);
    } else {                            /* Oops! It's a pipe. */
        int n = 0, bufsize = 0;

        /* Read in chunks of BUFSIZ. */
        buffer = (char *) Malloc( BUFSIZ );
        while ( (n = fread( buffer + bufsize, 1, BUFSIZ, handle )) == BUFSIZ ) {
            bufsize += BUFSIZ;
            buffer = (char *) realloc( buffer, bufsize + BUFSIZ );
        }
        if ( n >= 0 )
            n += bufsize;
        else
            n = bufsize;
        *bytes = n;
    }

    /* Return the buffer */
    return buffer;
}

long filelen(handle)
FILE *handle;
{

```

```
    long current_pos;
    long len;

    /* Save current position */
    current_pos= ftell(handle);

    /* Get len of file */
    fseek(handle, 0, 2);
    len= ftell(handle);

    /* Restore position */
    fseek(handle, current_pos, 0);

    return len;
}

putshortMSBF( value, handle )
int value;
FILE *handle;
{
    putc( MSB(value), handle );
    putc( LSB(value), handle );
}

putshortLSBF( value, handle )
int value;
FILE *handle;
{
    putc( LSB(value), handle );
    putc( MSB(value), handle );
}

getshortMSBF( handle )
FILE *handle;
{
    int aux;

    aux = getc( handle ) << 8;
    aux |= getc( handle );

    return aux;
}

getshortLSBF( handle )
FILE *handle;
{
    int aux;

    aux = getc( handle );
    aux |= (getc( handle ) << 8);

    return aux;
}
```

```
/*
 * This software is copyrighted as noted below.  It may be freely copied,
 * modified, and redistributed, provided that the copyright notice is
 * preserved on all copies.
 *
 * There is no warranty or other guarantee of fitness for this software,
 * it is provided solely "as is".  Bug reports or fixes may be sent
 * to the author, who may or may not act on them as he desires.
 *
 * You may not include this software in a program or other software product
 * without supplying the source, or without informing the end-user that the
 * source is available for no extra charge.
 *
 * If you modify this software, you should include a notice giving the
 * name of the person performing the modification, the date of modification,
 * and the reason for such modification.
 */
/*
 * lug.h - main LUG include.
 *
 * Author:      Raul Rivero
 *              Mathematics Dept.
 *              University of Oviedo
 * Date:        Fri Dec 27 1991
 * Copyright (c) 1991, Raul Rivero
 */
```

```
#ifndef MY_LUG
```

```
#define MY_LUG
```

```
/*
 * Load the LUG config header ( CHECK THAT FILE !!! ).
 */
```

```
#include "lugconf.h"
```

```
/*
 * System includes.
 */
```

```
#include <stdio.h>
#include <math.h>
#include <signal.h>
#include <string.h>
#include <sys/types.h>
#ifndef MSDOS
#    include <sys/wait.h>
#endif
#include <sys/stat.h>
```

```
#ifdef USE_STDLIB_H
#    include <stdlib.h>
#else
#    ifdef VOID_STAR
        extern void *malloc();
#    else
        extern char *malloc();
#    endif /* VOID_STAR */
    extern void free();
#endif /* USE_STDLIB_H */
```

```
#ifndef USE_STRINGS
#    include <strings.h>
#else
#    include <string.h>
#endif /* USE_STRINGS */

#ifndef BSD
#    define bzero(s, n)                memset((s), 0, (n))
#    define bcopy(f, t, c)             memcpy((t), (f), (c))
#    define index(s, c)                strchr((s), (c))
#endif /* BSD */

/*
 * New data types.
 */
#define byte                unsigned char
#define ushort              unsigned short
#define word                unsigned int
#define ulong               unsigned long

/*
 * Flags.
 */
/*#define FALSE              0
#define TRUE                 1 */
#define CHROMABORDER        20
#define CHROMAMASK          20
#define LUGUSED             12345

/*
 * Supported file formats.
 */
#define GIF_FORMAT          00
#define HF_FORMAT           01
#define SUN_FORMAT          02
#define PBM_FORMAT          03
#define PGM_FORMAT          04
#define PPM_FORMAT          05
#define PCX_FORMAT          06
#define PIX_FORMAT          07
#define RAW_FORMAT          08
#define RGB_FORMAT          09
#define RLA_FORMAT          10
#define RLB_FORMAT          11
#define RLE_FORMAT          12
#define SGI_FORMAT          13
#define TGA_FORMAT          14
#define TIFF_FORMAT         15

/*
 * And the default file format is ...
 */
#define DEFAULT_FORMAT      PIX_FORMAT

/*
 * Conversion support defines.
 */
#define ONLY_RAW_PLANES     -8
#define ONLY_FLOAT_PLANES  -1
#define ALL_PLANES          0
#define ONLY_BINARY_PLANES  1
#define ONLY_8_PLANES       8
```

```
#define ONLY_24_PLANES                24

/*
 * Returned by exist_file.
 *
 * ...
 * FILE_IS_GZIP/COMPRESS --> the name exist but is compressed
 * FILE_WITH_GZIP/COMPRESS -> the file doesn't exist, but if you
 *                               add a .gz/.Z suffix then ... voila!.
 */
#define FILE_NO_EXIST                0
#define FILE_EXIST                    1
#define FILE_IS_GZIP                  2
#define FILE_IS_COMPRESS              3
#define FILE_WITH_GZIP                4
#define FILE_WITH_COMPRESS            5

/*
 * Temporary directory.
 */
#ifndef MSDOS
#   define TMPDIR                    "/usr/tmp"
#else
#   define TMPDIR                    getenv("TEMP")
#endif

/*
 * Some define functions.
 */
#define MSB(a)                        ((byte) (((short)(a)) >> 8))
#define LSB(a)                        ((byte) (((short)(a)) & 0x00FF))
#define LUGABS(a)                     ((a) < 0 ? -(a) : (a))
#define FLOOR(a)                      ((a) > 0 ? (int)(a) : -(int)(-a))
#define CEILING(a)                    ((a) == (int)(a) ? (a) : (a) > 0 ? \
    1 + (int)(a) : -(1 + (int)(-a)))
#define ROUND(a)                      ((a) > 0 ? (int)((a)+0.5) : -(int)(0.5-(a)))
#define SQR(a)                        ((a) * (a))
#define LUGMAX(a,b)                   ((a) > (b) ? (a) : (b))
#define LUGMIN(a,b)                   ((a) > (b) ? (b) : (a))
/* #define SWAP(a,b)                  { a ^= b; b ^= a; a ^= b; } */
#define CORRECT(a)                    ((a) < 0 ? 0 : (a) > 255 ? 255 : (a) )
#define CLAMP(v,l,h)                  ((v) < (l) ? (l) : (v) > (h) ? (h) : v)
#define RGB_to_BW(r,g,b)              ((int) (30*(r) + 59*(g) + 11*(b)) / 100)

/*
 * I/O define functions.
 */
#define Fread(p, s, n, f)              if ( fread(p, s, n, f) != n ) error(3)
#define Fwrite(p, s, n, f)            if ( fwrite(p, s, n, f) != n ) error(4)
#define VPRINTF                        if (LUGverbose) fprintf
#define VFLUSH                         if (LUGverbose) fflush
#define setverbose(f)                  LUGverbose = (f)
#define getverbose()                   LUGverbose
#define setcmdname(n)                  MY_NAME = (n)
#define getcmdname()                   MY_NAME

/*
 * If you define ERROR_CALLS then the library functions
 * seems system calls ( and return -1 or NULL ), but this
 * needs that you check this.
 */
```

```
#ifndef ERROR_CALLS
#    define error(code)            Error(code)
#    define errornull(code)        Error(code)
#else
#    define error(code)            { Error(code); return -1; }
#    define errornull(code)        { Error(code); return NULL; }
#endif /* ERROR_CALLS */

/*
 * Pointer to functions data type.
 */
typedef int (* ifunptr)();

/*
 * We'll use it when need a pointer to cmap.
 */
typedef byte color_map[3];

/*
 * I use a intermediate format ( a simple bitmap ) with
 * this format ...
 */
typedef struct {
    int xsize, ysize;        /* sizes */
    int depth;               /* # of colors */
    int colors;              /* # of colors */
    byte *r, *g, *b;        /* components or bitmap (planes < 8) */
    byte *cmap;              /* cmap if planes < 8 */
    int magic;               /* used ? */
} bitmap_hdr;

#endif /* MY_LUG */
```



```
/*
 * This software is copyrighted as noted below.  It may be freely copied,
 * modified, and redistributed, provided that the copyright notice is
 * preserved on all copies.
 *
 * There is no warranty or other guarantee of fitness for this software,
 * it is provided solely "as is".  Bug reports or fixes may be sent
 * to the author, who may or may not act on them as he desires.
 *
 * You may not include this software in a program or other software product
 * without supplying the source, or without informing the end-user that the
 * source is available for no extra charge.
 *
 * If you modify this software, you should include a notice giving the
 * name of the person performing the modification, the date of modification,
 * and the reason for such modification.
 */
/*
 * lugconf.h - machine depend configuration.
 *
 * Author:      Raul Rivero
 *              Mathematics Dept.
 *              University of Oviedo
 * Date:        Wed 15 Jul 1992
 * Copyright (c) 1992, Raul Rivero
 */
```

```
#ifndef MY_CONFIGURE_LUG
```

```
#define MY_CONFIGURE_LUG
```

```
/*
 * If your machine is not bigendian then
 * you'll need define the NO_BIGENDIAN macro.
 *
 * The DEC and PC machines needs this.
 *
 * If your compiler defines the MSDOS macro ( ex: the
 * djgcc ), the definition of BIG_ENDIAN is done
 * automaticly.
 */
```

```
#ifdef MSDOS
#    define NO_BIGENDIAN
#else
#    undef NO_BIGENDIAN
#endif
```

```
/*
 * If you has an ANSI compiler ( ex: gcc ), define
 * the USE_PROTOTYPES macro.
 */
```

```
#undef USE_PROTOTYPES
```

```
/*
 * Define the USE_STDLIB_H macro if your machine has
 * the stdlib.h header.
 */
```

```
#define USE_STDLIB_H

/*
 * Define the VOID_STAR macro if your malloc definition
 * needs void* instead of char*.
 *
 * ( Only used if USE_STDLIB_H hasn't been defined. )
 */
#ifndef USE_STDLIB_H
#    define USE_STDLIB_H
#endif

/*
 * Define USE_STRINGS if you have the strings.h
 * header ( and not the standard string.h ).
 */
#undef USE_STRINGS

/*
 * If your library has bcopy, bzero, index, ...
 * then you'll need define BSD ( else LUG will use
 * memcpy, memset, ... ).
 */
#undef BSD

/*
 * If you have installed the GNU's GZIP and you wanna
 * use it instead of COMPRESS then define the USE_GNU_GZIP
 * macro.
 */
#define USE_GNU_GZIP

#endif /* MY_CONFIGURE_LUG */
```

```
/*
 * This software is copyrighted as noted below.  It may be freely copied,
 * modified, and redistributed, provided that the copyright notice is
 * preserved on all copies.
 *
 * There is no warranty or other guarantee of fitness for this software,
 * it is provided solely "as is".  Bug reports or fixes may be sent
 * to the author, who may or may not act on them as he desires.
 *
 * You may not include this software in a program or other software product
 * without supplying the source, or without informing the end-user that the
 * source is available for no extra charge.
 *
 * If you modify this software, you should include a notice giving the
 * name of the person performing the modification, the date of modification,
 * and the reason for such modification.
 */
/*
 * lugfnts.c - define all prototypes of LUG subroutines.
 *
 * Author:      Raul Rivero
 *              Mathematics Dept.
 *              University of Oviedo
 * Date:        Fri 9 Jul 1993
 * Copyright (c) 1993, Raul Rivero
 */

#ifndef MY_DEFINE_PROTOTYPES

#define MY_DEFINE_PROTOTYPES

/*
 * We need some of the lug header definitions.
 */
#include "lug.h"

/*
 * Include all what functions needs.
 */
#include "targa.h"
#include "alias.h"
#include "rla.h"

/*****
 *
 * Interface (read/write) definitions.
 *
 */

/* cnv.c */

extern int
read_lug_file(
#ifdef USE_PROTOTYPES
    char *,
    bitmap_hdr *
#endif
);

extern int
```

```
write_lug_file(
#ifdef USE_PROTOTYPES
    char *,
    bitmap_hdr *
#endif
);

extern ifunptr
get_readlug_function(
#ifdef USE_PROTOTYPES
    char *
#endif
);

extern ifunptr
get_writelug_function(
#ifdef USE_PROTOTYPES
    char *
#endif
);

extern int
get_depth_writelug_function(
#ifdef USE_PROTOTYPES
    char *
#endif
);

extern int
get_index_function(
#ifdef USE_PROTOTYPES
    char *
#endif
);

extern int
get_real_index_function(
#ifdef USE_PROTOTYPES
    char *
#endif
);

/* gif.c */

extern int
read_gif_file(
#ifdef USE_PROTOTYPES
    char *,
    bitmap_hdr *
#endif
);

extern int
read_gif(
#ifdef USE_PROTOTYPES
    FILE *,
    bitmap_hdr *
#endif
);
```

```
extern int
uncode_gif(
#ifdef USE_PROTOTYPES
    FILE *,
    int,
    int,
    bitmap_hdr *
#endif
);

extern int
read_code(
#ifdef USE_PROTOTYPES
    byte *,
    int,
    int *,
    int
#endif
);

extern int
push_gif(
#ifdef USE_PROTOTYPES
    byte *,
    int
#endif
);

extern int
read_gif_hdr(
#ifdef USE_PROTOTYPES
    FILE *
#endif
);

extern byte
*read_gif_screen_hdr(
#ifdef USE_PROTOTYPES
    FILE *,
    int *,
    int *
#endif
);

extern byte
*read_gif_image_hdr(
#ifdef USE_PROTOTYPES
    FILE *,
    int *,
    int *,
    int *,
    int *
#endif
);

extern byte
*unblock(
#ifdef USE_PROTOTYPES
    FILE *
#endif
);
```

```
extern int
write_gif_file(
#ifdef USE_PROTOTYPES
    char *,
    bitmap_hdr *
#endif
);

extern int
write_gif(
#ifdef USE_PROTOTYPES
    FILE *,
    bitmap_hdr *
#endif
);

extern int
write_gif_hdr(
#ifdef USE_PROTOTYPES
    FILE *
#endif
);

extern int
write_gif_screen_hdr(
#ifdef USE_PROTOTYPES
    FILE *,
    bitmap_hdr *
#endif
);

extern int
write_gif_cmap(
#ifdef USE_PROTOTYPES
    FILE *,
    bitmap_hdr *
#endif
);

extern int
write_gif_image_hdr(
#ifdef USE_PROTOTYPES
    FILE *,
    bitmap_hdr *
#endif
);

/* heightfield.c */

extern int
write_hf_file(
#ifdef USE_PROTOTYPES
    char *,
    bitmap_hdr *
#endif
);

extern int
write_hf(
```

```
#ifdef USE_PROTOTYPES
    FILE *,
    bitmap_hdr *,
    double
#endif
);
```

```
/* jpeg.c */
```

```
extern int
write_jpeg_file(
#ifdef USE_PROTOTYPES
    char *,
    bitmap_hdr *
#endif
);
```

```
extern int
write_jpeg(
#ifdef USE_PROTOTYPES
    FILE *,
    bitmap_hdr *
#endif
);
```

```
extern int
write_jpeg_opt(
#ifdef USE_PROTOTYPES
    FILE *,
    bitmap_hdr *,
    int
#endif
);
```

```
extern int
read_jpeg_file(
#ifdef USE_PROTOTYPES
    char *,
    bitmap_hdr *
#endif
);
```

```
extern int
read_jpeg(
#ifdef USE_PROTOTYPES
    FILE *,
    bitmap_hdr *
#endif
);
```

```
/* pbm.c */
```

```
extern int
read_pbm_file(
#ifdef USE_PROTOTYPES
    char *,
    bitmap_hdr *
#endif
);
```

```
extern int
read_pbm(
#ifdef USE_PROTOTYPES
    FILE *,
    bitmap_hdr *
#endif
);
```

```
extern int
write_pbm_file(
#ifdef USE_PROTOTYPES
    char *,
    bitmap_hdr *
#endif
);
```

```
extern int
write_pbm(
#ifdef USE_PROTOTYPES
    FILE *,
    bitmap_hdr *
#endif
);
```

```
extern int
skip_pbm(
#ifdef USE_PROTOTYPES
    FILE *
#endif
);
```

```
/* pcx.c */
```

```
extern int
read_pcx_file(
#ifdef USE_PROTOTYPES
    char *,
    bitmap_hdr *
#endif
);
```

```
extern int
read_pcx(
#ifdef USE_PROTOTYPES
    FILE *,
    bitmap_hdr *
#endif
);
```

```
extern int
decodePCX(
#ifdef USE_PROTOTYPES
    FILE *,
    byte *,
    int
#endif
);
```

```
extern int
```



```
write_pcx_file(
#ifdef USE_PROTOTYPES
    char *,
    bitmap_hdr *
#endif
);

extern int
write_pcx(
#ifdef USE_PROTOTYPES
    FILE *,
    bitmap_hdr *
#endif
);

extern int
write_pcx_header(
#ifdef USE_PROTOTYPES
    FILE *,
    int,
    int
#endif
);

extern int
write_pcx_cmap(
#ifdef USE_PROTOTYPES
    FILE *,
    int ,
    byte *
#endif
);

extern int
encodePCX(
#ifdef USE_PROTOTYPES
    FILE *,
    byte *,
    int
#endif
);

/* pix.c */

extern int
read_alias_file(
#ifdef USE_PROTOTYPES
    char *,
    bitmap_hdr *
#endif
);

extern int
read_alias(
#ifdef USE_PROTOTYPES
    FILE *,
    bitmap_hdr *
#endif
);
```

```
extern int
read_alias_header(
#ifdef USE_PROTOTYPES
    FILE *,
    alias_hdr *
#endif
);

extern int
read_line_alias24(
#ifdef USE_PROTOTYPES
    FILE *,
    byte *,
    byte *,
    byte *,
    register int
#endif
);

extern int
read_line_alias(
#ifdef USE_PROTOTYPES
    FILE *,
    byte *,
    register int
#endif
);

extern int
create_alias_cmap(
#ifdef USE_PROTOTYPES
    bitmap_hdr *
#endif
);

extern int
unicode_alias24(
#ifdef USE_PROTOTYPES
    byte *,
    byte *,
    byte *,
    byte *,
    byte *
#endif
);

extern int
unicode_alias(
#ifdef USE_PROTOTYPES
    byte *,
    byte *,
    byte *
#endif
);

extern int
write_alias_file(
#ifdef USE_PROTOTYPES
    char *,
    bitmap_hdr *
#endif
);
```

```
);
```

```
extern int
write_alias(
#ifdef USE_PROTOTYPES
    FILE *,
    bitmap_hdr *
#endif
);
```

```
extern int
code_alias24(
#ifdef USE_PROTOTYPES
    byte *,
    byte *,
    byte *,
    int ,
    FILE *
#endif
);
```

```
/* ps.c */
```

```
extern int
write_ps_file(
#ifdef USE_PROTOTYPES
    char *,
    bitmap_hdr *
#endif
);
```

```
extern int
write_ps_file_dimensions(
#ifdef USE_PROTOTYPES
    char *,
    bitmap_hdr *,
    double,
    double
#endif
);
```

```
extern int
ImagenGrisesPs(
#ifdef USE_PROTOTYPES
    int ,
    int ,
    byte *,
    double ,
    double ,
    char *
#endif
);
```

```
/* raw.c */
```

```
extern int
read24bitmap(
#ifdef USE_PROTOTYPES
    int ,
```

```
        int ,
        FILE *,
        FILE *,
        FILE *,
        bitmap_hdr *
#endif
);

extern int
read_raw_file(
#ifdef USE_PROTOTYPES
    char *,
    bitmap_hdr *
#endif
);

extern int
read_8bitmap_file(
#ifdef USE_PROTOTYPES
    char *,
    bitmap_hdr *,
    int ,
    int
#endif
);

extern int
read8bitmap(
#ifdef USE_PROTOTYPES
    FILE *,
    bitmap_hdr *,
    int,
    int
#endif
);

extern int
write24bitmap(
#ifdef USE_PROTOTYPES
    FILE *,
    FILE *,
    FILE *,
    bitmap_hdr *
#endif
);

extern int
write_raw_file(
#ifdef USE_PROTOTYPES
    char *,
    bitmap_hdr *
#endif
);

extern int
write_8bitmap_file(
#ifdef USE_PROTOTYPES
    char *,
    bitmap_hdr *
#endif
);
```

```
extern int
write8bitmap(
#ifdef USE_PROTOTYPES
    FILE *,
    bitmap_hdr *
#endif
);
```

/\* rgb.c \*/

```
extern int
read_rgb_i_file(
#ifdef USE_PROTOTYPES
    char *,
    bitmap_hdr *
#endif
);
```

```
extern int
read_rgb_file(
#ifdef USE_PROTOTYPES
    char *,
    bitmap_hdr *,
    int ,
    int
#endif
);
```

```
extern int
read_rgb(
#ifdef USE_PROTOTYPES
    FILE *,
    bitmap_hdr *,
    int,
    int
#endif
);
```

```
extern int
write_rgb_file(
#ifdef USE_PROTOTYPES
    char *,
    bitmap_hdr *
#endif
);
```

```
extern int
write_rgb(
#ifdef USE_PROTOTYPES
    FILE *,
    bitmap_hdr *
#endif
);
```

/\* rla.c \*/

```
extern int
read_rla_file(
```

```
#ifdef USE_PROTOTYPES
    char *,
    bitmap_hdr *
#endif
);

extern int
read_rla(
#ifdef USE_PROTOTYPES
    FILE *,
    bitmap_hdr *
#endif
);

extern int
decodeRLA(
#ifdef USE_PROTOTYPES
    byte *,
    byte *,
    int
#endif
);

extern int
write_rla_file(
#ifdef USE_PROTOTYPES
    char *,
    bitmap_hdr *
#endif
);

extern int
write_rla(
#ifdef USE_PROTOTYPES
    FILE *,
    bitmap_hdr *
#endif
);

extern int
encodeRLA(
#ifdef USE_PROTOTYPES
    byte *,
    byte *,
    int
#endif
);

/* rle.c */

extern int
read_rle_file(
#ifdef USE_PROTOTYPES
    char *,
    bitmap_hdr *
#endif
);

extern int
read_rle(
```

```
#ifdef USE_PROTOTYPES
    FILE *,
    bitmap_hdr *
#endif
);

extern int
write_rle_file(
#ifdef USE_PROTOTYPES
    char *,
    bitmap_hdr *
#endif
);

extern int
write_rle(
#ifdef USE_PROTOTYPES
    FILE *,
    bitmap_hdr *
#endif
);

/* sgi.c */

extern int
read_sgi_file(
#ifdef USE_PROTOTYPES
    char *,
    bitmap_hdr *
#endif
);

extern int
write_sgi_file(
#ifdef USE_PROTOTYPES
    char *,
    bitmap_hdr *
#endif
);

/* tga.c */

extern int
read_tga_file(
#ifdef USE_PROTOTYPES
    char *,
    bitmap_hdr *
#endif
);

extern int
read_tga(
#ifdef USE_PROTOTYPES
    FILE *,
    bitmap_hdr *
#endif
);

extern int
```

```
read_tga24(  
#ifdef USE_PROTOTYPES  
    FILE *,  
    byte *,  
    byte *,  
    byte *,  
    tga_hdr *  
#endif  
);  
  
extern int  
read_tga_to24(  
#ifdef USE_PROTOTYPES  
    FILE *,  
    byte *,  
    byte *,  
    byte *,  
    tga_hdr *,  
    byte *  
#endif  
);  
  
extern int  
read_tga8(  
#ifdef USE_PROTOTYPES  
    FILE *,  
    byte *,  
    tga_hdr *  
#endif  
);  
  
extern int  
read_tga_header(  
#ifdef USE_PROTOTYPES  
    FILE *,  
    tga_hdr *  
#endif  
);  
  
extern int  
read_tga_data(  
#ifdef USE_PROTOTYPES  
    byte *,  
    int ,  
    FILE *  
#endif  
);  
  
extern int  
write_tga_file(  
#ifdef USE_PROTOTYPES  
    char *,  
    bitmap_hdr *  
#endif  
);  
  
extern int  
write_rle_tga_file(  
#ifdef USE_PROTOTYPES  
    char *,  
    bitmap_hdr *
```



```
#endif
);

extern int
write_tga(
#ifdef USE_PROTOTYPES
    FILE *,
    bitmap_hdr *,
    int
#endif
);

extern int
write_tga_header(
#ifdef USE_PROTOTYPES
    FILE *,
    bitmap_hdr *,
    int
#endif
);

extern int
write_tga_line24(
#ifdef USE_PROTOTYPES
    FILE *,
    byte *,
    byte *,
    byte *,
    int
#endif
);

extern int
write_tga_rle_line24(
#ifdef USE_PROTOTYPES
    FILE *,
    byte *,
    byte *,
    byte *,
    int
#endif
);

/* tiff.c */

extern int
read_tiff_file(
#ifdef USE_PROTOTYPES
    char *,
    bitmap_hdr *
#endif
);

extern int
write_tiff_file(
#ifdef USE_PROTOTYPES
    char *,
    bitmap_hdr *
#endif
);
```

```

/*****
 *
 * Internal library functions.
 *
 */

/* bitmap.c */

extern int
allocatebitmap(
#ifdef USE_PROTOTYPES
    bitmap_hdr *,
    int,
    int,
    int,
    int
#endif
);

extern int
freebitmap(
#ifdef USE_PROTOTYPES
    bitmap_hdr *
#endif
);

extern int
copy_bitmap(
#ifdef USE_PROTOTYPES
    bitmap_hdr *,
    bitmap_hdr *
#endif
);

/* error.c */

extern char *
lugerrmsg(
#ifdef USE_PROTOTYPES
    int
#endif
);

extern int
Error(
#ifdef USE_PROTOTYPES
    int
#endif
);

/* general.c */

extern int
no_bits(
#ifdef USE_PROTOTYPES
    int
#endif
);
```

```
);

extern char *
basename(
#ifdef USE_PROTOTYPES
    char *,
    char *
#endif
);

extern short *
bytetoshort(
#ifdef USE_PROTOTYPES
    byte *,
    short *,
    int
#endif
);

extern byte *
shorttobyte(
#ifdef USE_PROTOTYPES
    short *,
    byte *,
    int
#endif
);

extern byte *
floattobyte(
#ifdef USE_PROTOTYPES
    float *,
    byte *,
    int
#endif
);

extern float *
bytetofloat(
#ifdef USE_PROTOTYPES
    byte *,
    float *,
    int
#endif
);

extern int
Atoi(
#ifdef USE_PROTOTYPES
    char *
#endif
);

extern double
Atod(
#ifdef USE_PROTOTYPES
    char *
#endif
);

extern int
```

```
isnumber(  
#ifdef USE_PROTOTYPES  
    char *  
#endif  
);  
  
extern int  
Uncompress(  
#ifdef USE_PROTOTYPES  
    char *,  
    char *  
#endif  
);  
  
extern int  
Compress(  
#ifdef USE_PROTOTYPES  
    char *  
#endif  
);  
  
extern int  
compute_levels(  
#ifdef USE_PROTOTYPES  
    int  
#endif  
);  
  
/* in_out.c */  
  
extern FILE *  
Fopen(  
#ifdef USE_PROTOTYPES  
    char *,  
    char *  
#endif  
);  
  
extern int  
exist_file(  
#ifdef USE_PROTOTYPES  
    char *  
#endif  
);  
  
extern int  
Fclose(  
#ifdef USE_PROTOTYPES  
    FILE *  
#endif  
);  
  
extern char *  
read_file(  
#ifdef USE_PROTOTYPES  
    FILE *,  
    int *  
#endif  
);
```

```
long
filelen(
#ifdef USE_PROTOTYPES
    FILE *
#endif
);

extern int
getshortMSBF(
#ifdef USE_PROTOTYPES
    FILE *
#endif
);

extern int
getshortLSBF(
#ifdef USE_PROTOTYPES
    FILE *
#endif
);

extern int
putshortMSBF(
    int,
    FILE *
#endif
);

extern int
putshortLSBF(
    int,
    FILE *
#endif
);

/* memory.c */

extern char *
Malloc(
#ifdef USE_PROTOTYPES
    int
#endif
);

extern int
Free(
#ifdef USE_PROTOTYPES
    void *
#endif
);

/*****
 *
 * Functions to show images.
 *
 */
```

```
/* viewers */

extern int
show_bitmap(
#ifdef USE_PROTOTYPES
    char *,
    bitmap_hdr *,
    int
#endif
);

extern int
write_vfr(
#ifdef USE_PROTOTYPES
    bitmap_hdr *,
    int,
    char *
#endif
);

extern int
show_bitmap_x11(
#ifdef USE_PROTOTYPES
    char *,
    bitmap_hdr *
#endif
);

/*****
 *
 * Digital images editing functions.
 *
 */

/* blur.c */

extern int
blur_bitmap(
#ifdef USE_PROTOTYPES
    bitmap_hdr *,
    bitmap_hdr *
#endif
);

byte *
blur(
#ifdef USE_PROTOTYPES
    byte *,
    int,
    int
#endif
);

/* convolve.c */

extern int
convolve_bitmap(
#ifdef USE_PROTOTYPES
```

```
        bitmap_hdr *,
        bitmap_hdr *,
        double *
#endif
);

byte *
convolve(
#ifdef USE_PROTOTYPES
    byte *,
    int,
    int,
    double *
#endif
);

/* cut.c */

extern int
cut_bitmap(
#ifdef USE_PROTOTYPES
    bitmap_hdr *,
    bitmap_hdr *,
    int,
    int,
    int,
    int
#endif
);

/* change.c */

extern int
change_color(
#ifdef USE_PROTOTYPES
    bitmap_hdr *,
    bitmap_hdr *,
    int,
    int,
    int,
    int,
    int,
    int,
    int
#endif
);

/* chroma.c */

extern int
chroma_bitmaps(
#ifdef USE_PROTOTYPES
    bitmap_hdr *,
    bitmap_hdr *,
    bitmap_hdr *
#endif
);

extern int
chroma(
```

```
#ifndef USE_PROTOTYPES
    bitmap_hdr *,
    bitmap_hdr *
#endif
);
```

```
/* dither.c */
```

```
extern int
dither_image(
#ifdef USE_PROTOTYPES
    bitmap_hdr *,
    bitmap_hdr *,
    int,
    double
#endif
);
```

```
void
create_dithermap(
#ifdef USE_PROTOTYPES
    int,
    double,
    byte [][3],
    int [256],
    int [256],
    int [16][16]
#endif
);
```

```
void
create_square(
#ifdef USE_PROTOTYPES
    double,
    int [256],
    int [256],
    int [16][16]
#endif
);
```

```
/* flip.c */
```

```
extern int
flip_image(
#ifdef USE_PROTOTYPES
    bitmap_hdr *,
    bitmap_hdr *
#endif
);
```

```
byte *
flip(
#ifdef USE_PROTOTYPES
    byte *,
    int,
    int
#endif
);
```



```
/* gamma.c */
```

```
extern int
gamma_correction(
#ifdef USE_PROTOTYPES
    bitmap_hdr *,
    bitmap_hdr *,
    double
#endif
);
```

```
/* histoequalization.c */
```

```
extern int
histogram_equalization(
#ifdef USE_PROTOTYPES
    bitmap_hdr *,
    bitmap_hdr *
#endif
);
```

```
/* hsl.c */
```

```
extern int
hsl_to_rgb_buffer(
#ifdef USE_PROTOTYPES
    double *,
    double *,
    double *,
    byte *,
    byte *,
    byte *,
    int
#endif
);
```

```
extern int
rgb_to_hsl_buffer(
#ifdef USE_PROTOTYPES
    byte *,
    byte *,
    byte *,
    double *,
    double *,
    double *,
    int
#endif
);
```

```
extern int
RGB_to_HSL(
#ifdef USE_PROTOTYPES
    byte,
    byte,
    byte,
    double *,
    double *,
    double *
#endif
);
```

```
#endif
);

extern int
HSL_to_RGB(
#ifdef USE_PROTOTYPES
    double,
    double,
    double,
    byte *,
    byte *,
    byte *
#endif
);
```

/\* mask.c \*/

```
extern int
chroma_mask(
#ifdef USE_PROTOTYPES
    bitmap_hdr *,
    bitmap_hdr *,
    bitmap_hdr *,
    bitmap_hdr *
#endif
);
```

```
extern int
chroma_shadow_mask(
#ifdef USE_PROTOTYPES
    bitmap_hdr *,
    bitmap_hdr *,
    bitmap_hdr *,
    bitmap_hdr *
#endif
);
```

```
extern int
mask_change_color(
#ifdef USE_PROTOTYPES
    bitmap_hdr *,
    bitmap_hdr *,
    bitmap_hdr *,
    int,
    int,
    int
#endif
);
```

```
extern int
mask_change_to_bw(
#ifdef USE_PROTOTYPES
    bitmap_hdr *,
    bitmap_hdr *,
    bitmap_hdr *
#endif
);
```

```
extern int
mask_darken_color(
```

```
#ifndef USE_PROTOTYPES
    bitmap_hdr *,
    bitmap_hdr *,
    bitmap_hdr *
#endif
);
```

```
extern int
fade_mask(
#ifdef USE_PROTOTYPES
    bitmap_hdr *,
    bitmap_hdr *,
    bitmap_hdr *,
    bitmap_hdr *
#endif
);
```

/\* fade.c \*/

```
extern int
medianfilter(
#ifdef USE_PROTOTYPES
    bitmap_hdr *,
    bitmap_hdr *
#endif
);
```

```
byte *
filternoise(
#ifdef USE_PROTOTYPES
    byte *,
    int,
    int
#endif
);
```

```
extern int
med3x3(
#ifdef USE_PROTOTYPES
    byte *,
    byte *,
    byte *
#endif
);
```

/\* mirror.c \*/

```
extern int
mirror_image(
#ifdef USE_PROTOTYPES
    bitmap_hdr *,
    bitmap_hdr *
#endif
);
```

```
byte *
mirror(
#ifdef USE_PROTOTYPES
    byte *,
```

```
        int,
        int
#endif
);

/* paste.c */

extern int
paste(
#ifdef USE_PROTOTYPES
    bitmap_hdr *,
    bitmap_hdr *,
    int,
    int
#endif
);

extern int
center_image(
#ifdef USE_PROTOTYPES
    bitmap_hdr *,
    bitmap_hdr *
#endif
);

/* quantize.c */

extern int
quantize(
#ifdef USE_PROTOTYPES
    bitmap_hdr *,
    bitmap_hdr *,
    int
#endif
);

/* rotate.c */

extern int
rotate_bitmap(
#ifdef USE_PROTOTYPES
    bitmap_hdr *,
    bitmap_hdr *,
    double
#endif
);

extern int
rotate90_image(
#ifdef USE_PROTOTYPES
    bitmap_hdr *,
    bitmap_hdr *
#endif
);

byte *
rotate90(
#ifdef USE_PROTOTYPES
```

```
        byte *,
        int,
        int
#endif
);

/* sharpen.c */

extern int
sharpen_bitmap(
#ifdef USE_PROTOTYPES
    bitmap_hdr *,
    bitmap_hdr *
#endif
);

/* slowzoom.c */

extern int
slow_adjust_bitmap(
#ifdef USE_PROTOTYPES
    bitmap_hdr *,
    bitmap_hdr *,
    int,
    int
#endif
);

/* solid.c */

extern int
create_solid_image(
#ifdef USE_PROTOTYPES
    bitmap_hdr *,
    int,
    int,
    byte,
    byte,
    byte
#endif
);

/* to24.c */

extern int
to24(
#ifdef USE_PROTOTYPES
    bitmap_hdr *,
    bitmap_hdr *
#endif
);

/* tobw.c */

extern int
tobw(
```

```
#ifndef USE_PROTOTYPES
    bitmap_hdr *,
    bitmap_hdr *
#endif
);

extern int
to_raw_bw(
#ifdef USE_PROTOTYPES
    bitmap_hdr *,
    bitmap_hdr *
#endif
);

byte *
create_bw_pallette();

byte *
cmap_to_bw(
#ifdef USE_PROTOTYPES
    bitmap_hdr *image
#endif
);

extern int
isagrayscaled(
#ifdef USE_PROTOTYPES
    bitmap_hdr *
#endif
);

extern int
rgbtobw(
#ifdef USE_PROTOTYPES
    byte *,
    byte *,
    byte *,
    byte *,
    int
#endif
);

/* tohalftone.c */

extern int
tohalftone(
#ifdef USE_PROTOTYPES
    bitmap_hdr *,
    bitmap_hdr *
#endif
);

extern int
bw_to_halftone(
#ifdef USE_PROTOTYPES
    int,
    int,
    byte
#endif
);
```

```
byte *
create_halftone_pallette();

/* toinverse.c */

extern int
toinverse(
#ifdef USE_PROTOTYPES
    bitmap_hdr *,
    bitmap_hdr *
#elseif
);

byte *
cmap_to_inverse(
#ifdef USE_PROTOTYPES
    bitmap_hdr *
#elseif
);

extern int
inverse(
#ifdef USE_PROTOTYPES
    byte *,
    int
#elseif
);

/* zoom.c */

extern int
adjust_bitmap(
#ifdef USE_PROTOTYPES
    bitmap_hdr *,
    bitmap_hdr *,
    int,
    int,
    int
#elseif
);

byte *
zoom(
#ifdef USE_PROTOTYPES
    int,
    int,
    int,
    int,
    register byte *,
    int
#elseif
);

#endif /* MY_DEFINE_PROTOTYPES */
```

```
/*
 * This software is copyrighted as noted below.  It may be freely copied,
 * modified, and redistributed, provided that the copyright notice is
 * preserved on all copies.
 *
 * There is no warranty or other guarantee of fitness for this software,
 * it is provided solely "as is".  Bug reports or fixes may be sent
 * to the author, who may or may not act on them as he desires.
 *
 * You may not include this software in a program or other software product
 * without supplying the source, or without informing the end-user that the
 * source is available for no extra charge.
 *
 * If you modify this software, you should include a notice giving the
 * name of the person performing the modification, the date of modification,
 * and the reason for such modification.
 */
/*
 * memory.c - allocate mamory.
 *
 * Author:      Raul Rivero
 *              Mathematics Dept.
 *              University of Oviedo
 * Date:        Sat Jan 4 1992
 * Copyright (c) 1992, Raul Rivero
 */
```

```
#include "lug.h"
```

```
extern int stop_on_error;
```

```
char *Malloc(size)
int size;
{
    char *ptr;

    if ((ptr = (char *) malloc(size)) == NULL) {
        stop_on_error = 1;
        Error( 2 );
    }

    /*
     * Usually compilers fill buffers with zeros,
     * but ...
     */
    bzero( ptr, size );
    return ptr;
}
```

```
Free( ptr )
void *ptr;
{
    if ( ptr != NULL )
        free( ptr );
}
```



```
/*
 * This software is copyrighted as noted below.  It may be freely copied,
 * modified, and redistributed, provided that the copyright notice is
 * preserved on all copies.
 *
 * There is no warranty or other guarantee of fitness for this software,
 * it is provided solely "as is".  Bug reports or fixes may be sent
 * to the author, who may or may not act on them as he desires.
 *
 * You may not include this software in a program or other software product
 * without supplying the source, or without informing the end-user that the
 * source is available for no extra charge.
 *
 * If you modify this software, you should include a notice giving the
 * name of the person performing the modification, the date of modification,
 * and the reason for such modification.
 */
/*
 * rla.h - type define to Wavefront's RLA format.
 *
 * Author:      Raul Rivero
 *              Mathematics Dept.
 *              University of Oviedo
 * Date:        Sat Jan 4 1992
 * Copyright (c) 1992, Raul Rivero
 *
 */
```

```
#ifndef MY_RLA
```

```
#define MY_RLA
```

```
typedef struct {
    short left, right, bottom, top;
} WINDOW_S;
```

```
typedef struct {
    WINDOW_S      window;
    WINDOW_S      act_window;
    short         frame;
    short         storage_type;
    short         num_chan;
    short         num_matte;
    short         num_aux;
    short         aux_mask;
    char          gamma[16];
    char          red_pri[24];
    char          green_pri[24];
    char          blue_pri[24];
    char          white_pt[24];
    long          job_num;
    char          name[128];
    char          desc[128];
    char          program[64];
    char          machine[32];
    char          user[32];
    char          date[20];
    char          aspect[32];
    char          chan[32];
    char          space[128];
} RLA_HEADER;
```

```
#ifdef MSDOS
#define cuserid(a)          "nouser"
#define gethostname(a, b)  strcpy( a, "my_PC_with_MSDOS" )
#endif

#endif          /* MY_RLA */
```

```
/*
 * This software is copyrighted as noted below.  It may be freely copied,
 * modified, and redistributed, provided that the copyright notice is
 * preserved on all copies.
 *
 * There is no warranty or other guarantee of fitness for this software,
 * it is provided solely "as is".  Bug reports or fixes may be sent
 * to the author, who may or may not act on them as he desires.
 *
 * You may not include this software in a program or other software product
 * without supplying the source, or without informing the end-user that the
 * source is available for no extra charge.
 *
 * If you modify this software, you should include a notice giving the
 * name of the person performing the modification, the date of modification,
 * and the reason for such modification.
 */
/*
 * targa.h - type define to Targa format.
 *
 * Author:      Raul Rivero
 *              Mathematics Dept.
 *              University of Oviedo
 * Date:        Wed Jan 22 1992
 * Copyright (c) 1992, Raul Rivero
 */

#ifndef MY_TGA

#define MY_TGA

#define TGA_INTERLACED(a)      ( (a) & 0xc0 )
#define TGA_FLIP(a)            ( ((a) & 0x20) ? 0 : 1 )
#define TGA_MAPPED             1
#define TGA_RGB                2
#define TGA_RLE_MAPPED         9
#define TGA_RLE_RGB            10

/*
 * Targa header.
 */
typedef struct {
    byte    num_id;
    byte    cmap_type;
    byte    image_type;
    ushort  cmap_orign;
    ushort  cmap_length;
    byte    cmap_entry_size;
    ushort  xorig, yorig;
    ushort  xsize, ysize;
    byte    pixel_size;
    byte    image_descriptor;
} tga_hdr;

#endif      /* MY_TGA */
```

```
/*
 * This software is copyrighted as noted below.  It may be freely copied,
 * modified, and redistributed, provided that the copyright notice is
 * preserved on all copies.
 *
 * There is no warranty or other guarantee of fitness for this software,
 * it is provided solely "as is".  Bug reports or fixes may be sent
 * to the author, who may or may not act on them as he desires.
 *
 * You may not include this software in a program or other software product
 * without supplying the source, or without informing the end-user that the
 * source is available for no extra charge.
 *
 * If you modify this software, you should include a notice giving the
 * name of the person performing the modification, the date of modification,
 * and the reason for such modification.
 */
/*
 * tga.c - interface with Targa format.
 *
 * Author:      Raul Rivero
 *              Mathematics Dept.
 *              University of Oviedo
 * Date:        Wed Jan 22 1992
 * Copyright (c) 1992, Raul Rivero
 */
```

```
#include "lug.h"
#include "lugfnts.h"
```

```
read_tga_file( name, bitmap )
char *name;
bitmap_hdr *bitmap;
{
    FILE *handle;

    /* Open the file descriptor */
    if ( name != NULL )
        handle = Fopen( name, "rb" );
    else handle = stdin;

    /* Read the bitmap */
    read_tga( handle, bitmap );
    rm_compress();

    /* Close the file */
    Fclose( handle );
}
```

```
read_tga(handle, image)
FILE *handle;
bitmap_hdr *image;
{
    register int i;
    tga_hdr tga;
    int allplanes = 0;
    int lessplanes = 0;
    int forceallplanes = 0;
    byte aux4[4];
    int totalsize;
```

```
byte *ptr;
byte *r, *g, *b;
byte *aux;

/* Read the Targa header */
read_tga_header( handle, &tga );

/* Fill our header */
image->magic = LUGUSED;
image->xsize = tga.xsize;
image->ysize = tga.ysize;
totalsize = image->xsize * image->ysize;
/*
 * Targa format can be mapped but with moer than 256 colors,
 * our mapped images only support 8 bits, so we need convert
 * these images to 24 bits.
 */
switch ( tga.image_type ) {
    case TGA_RGB:
    case TGA_RLE_RGB:
        allplanes = 1;
        break;
    case TGA_MAPPED:
    case TGA_RLE_MAPPED:
        lessplanes = 1;
        if ( tga.cmap_type != 1 )
            error( 10 );
        if ( tga.cmap_length > 256 )
            forceallplanes = 1;
        break;
}

/*
 * Read the cmap info ( if exists ).
 */
if ( tga.cmap_type ) {
/*    VPRINTF(stderr, "Reading cmap info\n"); */
    if ( allplanes ) {
        /* We only need skip the cmap block */
        for ( i = 0; i < tga.cmap_length; i++ )
            read_tga_data( aux4, tga.cmap_entry_size, handle );
    } else {
        ptr = image->cmap = (byte *) Malloc( 3 * tga.cmap_length );
        for ( i = 0; i < tga.cmap_length; i++ ) {
            read_tga_data( ptr, tga.cmap_entry_size, handle );
            ptr += 3;
        }
    }
}

/*
 * Allocate memory for the bitmap and fill our header.
 */
image->xsize = tga.xsize;
image->ysize = tga.ysize;
totalsize = image->xsize * image->ysize;
if ( allplanes || forceallplanes ) {
    r = image->r = (byte *) Malloc( totalsize );
    g = image->g = (byte *) Malloc( totalsize );
    b = image->b = (byte *) Malloc( totalsize );
    image->depth = 24;
}
```

```
}else {
    r = image->r = (byte *) Malloc( totalsize );
    image->depth = no_bits( tga.cmap_length ) + 1;
}
image->colors = ( 1 << image->depth );

/*
 * Read the raster information.
 */
if ( allplanes ) {
    read_tga24( handle, r, g, b, &tga );

}else if ( forceallplanes )
    read_tga_to24( handle, r, g, b, &tga, image->cmap );
    else read_tga8( handle, r, &tga );

/* if ( TGA_FLIP( tga.image_descriptor ) ) {
    VPRINTF(stderr, "Flipping the image\n");
    aux = (byte *) flip( r, image->xsize, image->ysize );
    free( r );
    image->r = aux;
    if ( allplanes ) {

        aux = (byte *) flip( g, image->xsize, image->ysize );
        free( g );
        image->g = aux;
        aux = (byte *) flip( b, image->xsize, image->ysize );
        free( b );
        image->b = aux;
    }
} */
}

read_tga24( handle, r, g, b, tga )
FILE *handle;
byte *r, *g, *b;
tga_hdr *tga;
{
    register int i, j;
    int aux;
    byte *ptr;
    byte *buffer;
    int totalsize;
    int count = 0;
    byte repeat;
    byte rbyte, gbyte, bbyte;
    int code;

    if ( tga->image_type < 9 ) {
        /* No RLE ! */
        /* VPRINTF(stderr, "Reading true color Targa image\n"); */
        buffer = (byte *) Malloc( tga->pixel_size * tga->xsize );
        for ( j = 0; j < tga->ysize; j++ ) {
#ifdef DEBUG
            /* VPRINTF(stderr, "Reading line %d\r", j); VFLUSH( stderr ); */
#endif
            fread( buffer, tga->xsize, tga->pixel_size, handle);
            ptr = buffer;
            if ( tga->pixel_size < 3 ) {
                for ( i = 0 ; i < tga->xsize; i++ ) {
                    aux = ( ptr[1] << 8 ) | ptr[0];
```

```
        *r++ = (aux & 0x7c00) >> 7;
        *g++ = (aux & 0x03e0) >> 2;
        *b++ = (aux & 0x001f) << 3;
        ptr += 2;
    }
} else {
    for ( i = 0 ; i < tga->xsize; i++ ) {
        *b++ = *ptr++;
        *g++ = *ptr++;
        *r++ = *ptr++;
        if ( tga->pixel_size == 4 )
            ptr++;
    }
}
}
#endif
#ifdef DEBUG
/*      VPRINTF( stderr, "\n" ); */
#endif
    free( buffer );
} else {
/*      VPRINTF(stderr, "Reading true color Targa RLE image\n"); */
/* RLE */
    buffer = (byte *) Malloc( 256 * tga->pixel_size );
    totalsize = tga->xsize * tga->ysize;
    while ( count < totalsize ) {
        Fread( &repeat, 1, 1, handle );
        code = (repeat & 127) + 1;
        count += code;
        if ( (repeat & 128) ) {
            Fread( buffer, tga->pixel_size, 1, handle );
            if ( tga->pixel_size < 3 ) {
                aux = ( buffer[1] << 8 ) | buffer[0];
                rbyte = (aux & 0x7c00) >> 7;
                gbyte = (aux & 0x03e0) >> 2;
                bbyte = (aux & 0x001f) << 3;
            } else {
                rbyte = buffer[2];
                gbyte = buffer[1];
                bbyte = buffer[0];
            }
            memset( r, rbyte, code ); r += code;
            memset( g, gbyte, code ); g += code;
            memset( b, bbyte, code ); b += code;
        } else {
            Fread( buffer, tga->pixel_size, code, handle );
            ptr = buffer;
            while ( code-- ) {
                if ( tga->pixel_size < 3 ) {
                    aux = ( ptr[1] << 8 ) | ptr[0];
                    *r++ = (aux & 0x7c00) >> 7;
                    *g++ = (aux & 0x03e0) >> 2;
                    *b++ = (aux & 0x001f) << 3;
                    ptr += 2;
                } else {
                    *b++ = *ptr++;
                    *g++ = *ptr++;
                    *r++ = *ptr++;
                    if ( tga->pixel_size == 4 )
                        ptr++;
                }
            }
        }
    }
}
```

```
    } /* if ( (repeat ... ) ) */
  } /* while ( count ... ) */
  free( buffer );
}

read_tga_to24( handle, r, g, b, tga, cmap )
FILE *handle;
byte *r, *g, *b;
tga_hdr *tga;
byte *cmap;
{
  fprintf(stderr, "Converting to 24 planes\n");
  error( 99 );
}

read_tga8( handle, r, tga )
FILE *handle;
byte *r;
tga_hdr *tga;
{
  if ( tga->image_type == 1 ) {
/*    VPRINTF( stderr, "Reading mapped Targa image\n" ); */
    Fread( r, tga->xsize, tga->ysize, handle );
  }else {
    byte buffer, repeat;
    int totalsize = tga->xsize * tga->ysize;
    int count = 0;
    int code;

/*    VPRINTF( stderr, "Reading mapped RLE Targa image\n" ); */
/* RLE */
    while ( count < totalsize ) {
      Fread( &repeat, 1, 1, handle );
      code = (repeat & 127) + 1;
      count += code;
      if ( (repeat & 128) ) {
        Fread( &buffer, 1, 1, handle );
        memset( r, buffer, code ); r += code;
      }else {
        Fread( r, code, 1, handle );
        r += code;
      }
    } /* while ( count ... ) */
  }
}

read_tga_header(handle, tga)
FILE *handle;
tga_hdr *tga;
{
  byte buffer[18];

  Fread( buffer, 18, 1, handle );

  /*
   * Bytes are in reverse order so ...
   */
  tga->num_id = buffer[0];
  tga->cmap_type = buffer[1];
  tga->image_type = buffer[2];
}
```



```
tga->cmap_orign = ( buffer[4] << 8 ) | buffer[3];
tga->cmap_length = ( buffer[6] << 8 ) | buffer[5];
tga->cmap_entry_size = buffer[7] / 8;
tga->xorig = ( buffer[9] << 8 ) | buffer[8];
tga->yorig = ( buffer[11] << 8 ) | buffer[10];
tga->xsize = ( buffer[13] << 8 ) | buffer[12];
tga->>ysize = ( buffer[15] << 8 ) | buffer[14];
tga->pixel_size = buffer[16] / 8;      /* we'll use it directly */
tga->image_descriptor = buffer[17];

/*  VPRINTF(stderr, "Image type %d\n", tga->image_type);
    VPRINTF(stderr, "%s color map\n", (tga->cmap_type ? "With" : "Without"));
    VPRINTF(stderr, "Cmap origin: %d\n", tga->cmap_orign );
    VPRINTF(stderr, "Cmap length: %d\n", tga->cmap_length );
    VPRINTF(stderr, "Cmap entry size: %d\n", tga->cmap_entry_size );
    VPRINTF(stderr, "Image size: %dx%d\n", tga->xsize, tga->>ysize );
    VPRINTF(stderr, "Pixel size: %d\n", tga->pixel_size ); */
/*  if ( TGA_INTERLACED(tga->image_descriptor) )
    VPRINTF(stderr, "Interlace image\n");
    if ( TGA_FLIP(tga->image_descriptor) )
    VPRINTF(stderr, "Flipped image\n"); */

/*  Interlaced ? */
if ( TGA_INTERLACED(tga->image_descriptor) ) {
    fprintf( stderr, "Interlaced image\n");
    error( 99 );
}

/*
 * Skip the identification.
 */
if ( tga->num_id ) {
/*    VPRINTF( stderr, "Image identification: "); */
    while ( tga->num_id ) {
        (void) getc( handle );
        tga->num_id--;
    }
/*    VPRINTF( stderr, "\n" ); */
}

/*
 * Check the image type.
 */
switch ( tga->image_type ) {
    case TGA_RGB:
    case TGA_RLE_RGB:
    case TGA_MAPPED:
    case TGA_RLE_MAPPED:
        break;

    default:
        fprintf(stderr, "Image type %d\n", tga->image_type);
        error( 99 );
        break;
}
}

read_tga_data( buffer, no_bytes, handle )
byte *buffer;
int no_bytes;
FILE *handle;
{
```

```
int aux1;
byte aux[4];

Fread( aux, no_bytes, 1, handle );

if ( no_bytes < 3 ) {
    aux1 = ( aux[1] << 8 ) | aux[0];
    *buffer++ = (aux1 & 0x7c00) >> 7;
    *buffer++ = (aux1 & 0x03e0) >> 2;
    *buffer++ = (aux1 & 0x001f) << 3;
}else {
    *buffer++ = aux[2];
    *buffer++ = aux[1];
    *buffer++ = aux[0];
}
}
```

```
write_tga_file( name, image )
char *name;
bitmap_hdr *image;
{
    FILE *handle;

    /* Open the file descriptor */
    if ( name != NULL )
        handle = Fopen( name, "wb" );
    else handle = stdout;

    /* Write the bitmap */
    write_tga( handle, image, 0 );

    /* Close the file */
    Fclose( handle );
}
```

```
write_rle_tga_file( name, image )
char *name;
bitmap_hdr *image;
{
    FILE *handle;

    /* Open the file descriptor */
    if ( name != NULL )
        handle = Fopen( name, "wb" );
    else handle = stdout;

    /* Write the bitmap */
    write_tga( handle, image, 1 );

    /* Close the file */
    Fclose( handle );
}
```

```
write_tga(handle, image, rle)
FILE *handle;
bitmap_hdr *image;
int rle;
{
    register int i;
    byte *r, *g, *b;
```

```
if ( image->magic != LUGUSED )
    error( 19 );

if ( image->depth <= 8 )
    error( 7 );

/* Write the header */
write_tga_header( handle, image, rle );

/* Set the pointers */
r = image->r;
g = image->g;
b = image->b;

/* Write each line */
/* VPRINTF(stderr, "Writing Targa %s raster data\n", (rle ? "RLE" : "")); */
for ( i = 0; i < image->ysize; i++ ) {
    if ( !rle )
        write_tga_line24( handle, r, g, b, image->xsize );
    else write_tga_rle_line24( handle, r, g, b, image->xsize );
    r += image->xsize;
    g += image->xsize;
    b += image->xsize;
}
}

write_tga_header(handle, image, rle)
FILE *handle;
bitmap_hdr *image;
int rle;
{
    byte buffer[ sizeof(tga_hdr) ];

/* VPRINTF(stderr, "Writing Targa header\n"); */
/* First we fill with zero, then skip too asigments */
bzero( buffer, sizeof(tga_hdr) );

/* image_type */
if ( image->depth > 8 ) {
    if ( rle )
        buffer[2] = TGA_RLE_RGB;
    else buffer[2] = TGA_RGB;
}else {
    if ( rle )
        buffer[2] = TGA_RLE_MAPPED;
    else buffer[2] = TGA_MAPPED;
}

buffer[12] = LSB( image->xsize );
buffer[13] = MSB( image->xsize );      /* the weight */
buffer[14] = LSB( image->ysize );
buffer[15] = MSB( image->ysize );      /* the height */
buffer[16] = 24;                      /* Targa 24 => RGB */
buffer[17] = 32;                      /* flipped */

/* Write the header */
Fwrite( buffer, 18, 1, handle );
}

write_tga_line24(handle, r, g, b, xsize)
```

```
FILE *handle;
byte *r, *g, *b;
int xsize;
{
    static byte buffer[3*2560];
    byte *ptr;
    byte *end;

    end = r + xsize;
    ptr = buffer;

    /* Do a single buffer */
    while ( r < end ) {
        *ptr++ = *b++;
        *ptr++ = *g++;
        *ptr++ = *r++;
    }

    /* Write it ! */
    Fwrite( buffer, xsize, 3, handle );
}

write_tga_rle_line24(handle, r, g, b, xsize)
FILE *handle;
byte *r, *g, *b;
int xsize;
{
    byte *end;
    byte cr, cb, cg;
    short repeat= 1;
    short diferent= 0;
    static byte salida[4*2560];
    byte bufaux[4*256];
    byte *ptr, *aux;

    ptr = salida;
    end = r + xsize + 1;
    cr = *r++;
    cg = *g++;
    cb = *b++;
    while ( r < end ) {
        while ( cr == *r  && cg == *g  && cb == *b  &&
                repeat < 127 && r < end) {
            r++, g++, b++;
            repeat++;
        }
        if (repeat > 1 ) {
            /* Put this information on our buffer */
            *ptr++ = 128 | (repeat - 1);
            *ptr++ = cb;
            *ptr++ = cg;
            *ptr++ = cr;
            repeat = 1;
            cr = *r++;
            cg = *g++;
            cb = *b++;
        }
        *ptr++ = *r++;
        *ptr++ = *g++;
        *ptr++ = *b++;
    }
    aux= bufaux;
    /*
     * If there are diferents bytes, then we need
     * know how many.
    */
}
```

```
    */
    while ( (cr != *r || cg != *g || cb != *b)  &&
            diferent< 127 && r < end) {
        *aux++ = cb;
        *aux++ = cg;
        *aux++ = cr;
        cr = *r++;
        cg = *g++;
        cb = *b++;
        diferent++;
    }
    if (diferent) {
        *ptr++ = diferent - 1;
        bcopy( bufaux, ptr, 3*diferent );
        ptr += 3*diferent;
        diferent = 0;
    }
}

Fwrite(salida, ptr-salida, 1, handle);
}
```

```
/*
 * This software is copyrighted as noted below.  It may be freely copied,
 * modified, and redistributed, provided that the copyright notice is
 * preserved on all copies.
 *
 * There is no warranty or other guarantee of fitness for this software,
 * it is provided solely "as is".  Bug reports or fixes may be sent
 * to the author, who may or may not act on them as he desires.
 *
 * You may not include this software in a program or other software product
 * without supplying the source, or without informing the end-user that the
 * source is available for no extra charge.
 *
 * If you modify this software, you should include a notice giving the
 * name of the person performing the modification, the date of modification,
 * and the reason for such modification.
 */
/*
 * tobw.c - conversions from a color image to a bw image.
 *
 * Author:      Raul Rivero
 *              Mathematics Dept.
 *              University of Oviedo
 * Date:        Sat Jan 11 1992
 * Copyright (c) 1992, Raul Rivero
 */
```

```
#include "lug.h"
#include "lugfnts.h"
```

```
extern int LUGverbose;
```

```
tobw(inbitmap, outbitmap)
bitmap_hdr *inbitmap;
bitmap_hdr *outbitmap;
{
    register int i;
    byte *rptr, *gptra, *bptr;
    byte *ptr;
    int totalsize;
    int directcolor;

    if ( inbitmap->magic != LUGUSED )
        error( 19 );

    VPRINTF(stderr, "Converting to b&w\n");
    directcolor = ( inbitmap->depth < 24 ? 0 : 1 );

    /*
     * Fill new header ...
     */
    outbitmap->magic = LUGUSED;
    outbitmap->xsize = inbitmap->xsize;
    outbitmap->ysize = inbitmap->ysize;
    totalsize = outbitmap->xsize * outbitmap->ysize;
    /* One component => r */
    ptr = outbitmap->r = (byte *) Malloc( totalsize );

    if ( !directcolor ) {
        /*
```

```
* If the image has less than 24 planes then we only
* need convert its palette.
*/
outbitmap->depth = inbitmap->depth;
outbitmap->colors = ( 1 << outbitmap->depth );
outbitmap->cmap = (byte *) cmap_to_bw( inbitmap );
/* Copy the raster information */
bcopy( inbitmap->r, outbitmap->r, totalsize );
}else {
/*
* We have a real color image.
*/
outbitmap->depth = 8;
outbitmap->colors = ( 1 << outbitmap->depth );
outbitmap->cmap = (byte *) create_bw_pallete();

rptra = inbitmap->r;
gptra = inbitmap->g;
bptra = inbitmap->b;

for ( i = 0; i < inbitmap->ysize; i++ ) {
/* Convert each line */
rgbtobw( rptra, gptra, bptra, ptr, inbitmap->xsize );
rptra += inbitmap->xsize;
gptra += inbitmap->xsize;
bptra += inbitmap->xsize;
ptr  += inbitmap->xsize;
}
}
}

to_raw_bw(inbitmap, outbitmap)
bitmap_hdr *inbitmap, *outbitmap;
{
    bitmap_hdr bwbitmap;
    int totalsize = inbitmap->xsize * inbitmap->ysize;
    byte *iptra, *optra;
    color_map *level;

    if ( inbitmap->magic != LUGUSED )
        error( 19 );

    if ( isgrayscaled( inbitmap ) ) {
        VPRINTF( stderr, "Preserving the raw b&w\n");
        copy_bitmap( inbitmap, outbitmap );
        return 0;
    }

    VPRINTF( stderr, "Converting to raw b&w\n" );
    /* Convert inbitmap to bw */
    tobw( inbitmap, &bwbitmap );

    /* Fill new header */
    outbitmap->magic = LUGUSED;
    outbitmap->xsize = bwbitmap.xsize;
    outbitmap->ysize = bwbitmap.ysize;
    outbitmap->depth = bwbitmap.depth;
    outbitmap->colors = bwbitmap.colors;

    /* Get memory */
    optra = outbitmap->r = (byte *) Malloc( totalsize );
```

```
outbitmap->cmap = (byte *) create_bw_pallette();
level = (color_map *) bwbitmap.cmap;

/*
 * What we wanna is a raster information which could
 * be used to represent the level of intensity.
 */
for ( iptr = bwbitmap.r; totalsize; totalsize-- ) {
    *optr++ = level[*iptr++][0];
}

/* Ok. Free memory */
freebitmap( &bwbitmap );

return 0;
}
```

```
byte *create_bw_pallette()
{
    register int i;
    byte *buffer = (byte *) Malloc( 3 * 256 );
    byte *ptr;

    /* I'll use a pointer */
    ptr = buffer;
    for ( i = 0; i < 256; i++ ) {
        *ptr++ = (byte) i;    /* R */
        *ptr++ = (byte) i;    /* G */
        *ptr++ = (byte) i;    /* B */
    }

    return buffer;
}
```

```
byte *cmap_to_bw( image )
bitmap_hdr *image;
{
    register int i;
    byte r[256];
    byte g[256];
    byte b[256];
    byte *buffer, *ptrbuf;
    byte *ptr;
    byte *position;

    if ( image->depth > 8 )
        error( 11 );

    ptrbuf = buffer = (byte *) Malloc ( 3 * image->colors );
    position = (byte *) Malloc( image->colors );

    /* Split the three components */
    ptr = image->cmap;
    for ( i = 0; i < image->colors; i++ ) {
        r[i] = *ptr++;
        g[i] = *ptr++;
        b[i] = *ptr++;
    }

    /* Convert these buffers to bw ( to position ) */
    rgbtobw( r, g, b, position, image->colors );
}
```



```
/*
 * Now convert these position to a real cmap.
 */
ptr = position;
for (i = 0; i < image->colors; i++, position++) {
    *ptrbuf++ = *position;      /* R */
    *ptrbuf++ = *position;      /* G */
    *ptrbuf++ = *position;      /* B */
}

free( position );
return buffer;
}

isagrayscaled( bitmap )
bitmap_hdr *bitmap;
{
    if ( bitmap->depth > 8 ) {
        /* A true color bitmap is NOT! a gray scaled image */
        return 0;
    } else {
        register i;
        color_map *ptr;

        /*
         * Check if the cmap defines a gray map and its
         * correct position.
         */
        for ( i = 0, ptr = (color_map *) bitmap->cmap; i < bitmap->colors; i++ ) {
            if ( i != ptr[i][0] || i != ptr[i][1] || i != ptr[i][2] )
                return 0;      /* ooops!, no gray mapped */
        }
        /* Yeah, we have a gray map image. */
        return 1;
    }
}

rgbtobw( r, g, b, position, size )
byte *r, *g, *b;
byte *position;
int size;
{
    /*
     * It's better use integers ( and need a division ) than
     * uses doubles ( and don't need it ).
     */
    while ( size-- )
        *position++ = (byte) ( (30*(*r++) + 59*(*g++) + 11*(*b++)) / 100);
}
```

```
/*
 * This software is copyrighted as noted below.  It may be freely copied,
 * modified, and redistributed, provided that the copyright notice is
 * preserved on all copies.
 *
 * There is no warranty or other guarantee of fitness for this software,
 * it is provided solely "as is".  Bug reports or fixes may be sent
 * to the author, who may or may not act on them as he desires.
 *
 * You may not include this software in a program or other software product
 * without supplying the source, or without informing the end-user that the
 * source is available for no extra charge.
 *
 * If you modify this software, you should include a notice giving the
 * name of the person performing the modification, the date of modification,
 * and the reason for such modification.
 */
/*
 * x11.c - Show images on X11 Window Systems.
 *
 * Author:      Raul Rivero
 *              Mathematics Dept.
 *              University of Oviedo
 * Date:        Sat Mar 14 1992
 * Copyright (c) 1992, Raul Rivero
 */
```

```
#include "lug.h"
#include "lugfnts.h"
```

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/cursorfont.h>
```

```
#define BORDER_WIDTH    0
```

```
Display      *dpy;
Window        wnd;
GC            gc;
Colormap      cmap;
int           screen;
```

```
show_bitmap_x11( name, inbitmap )
char *name;
bitmap_hdr *inbitmap;
{
    int colors, levels, depth;
    bitmap_hdr outbitmap;
    bitmap_hdr *bitmap;
    int cmapref[256], cmap_flag[256];
    XSetWindowAttributes wattr;
    long wmask;
    XSizeHints whints;
    XEvent event;
    XExposeEvent *exp;
    XComposeStatus statusio;
    int totalsize = inbitmap->xsize * inbitmap->ysize;
    byte *ptr, *iptr, *end;
    XImage *Ximagebuffer;
```

```
byte *Xbitmap;
int loop;
double gamma = 1.0;
int used_colors;
Cursor normal_pointer, wait_pointer;

/* Connect with X11 */
if ( (dpy = (Display *) XOpenDisplay(getenv("DISPLAY"))) == NULL )
    error( 15 );

/* Get some information */
screen = DefaultScreen( dpy );
gc = DefaultGC( dpy, screen );
depth = DisplayPlanes( dpy, screen );
Xcmap = DefaultColormap( dpy, screen );
XSetBackground( dpy, gc, BlackPixel( dpy, screen ) );
XSetForeground( dpy, gc, WhitePixel( dpy, screen ) );

/*
 * We know # of colors of X11 display so we need adjust
 * our # of colors to this.
 */
if ( inbitmap->depth > depth ) {
    colors = 1 << depth;
    levels = compute_levels( colors );
    used_colors = Xdither_image( inbitmap, &outbitmap,
                                levels, gamma, cmap_flag );
    bitmap = &outbitmap;
}else {
    bitmap = inbitmap;
    used_colors = set_cmap_flags( inbitmap, cmap_flag );
}

/*
 * We have a color map and we wanna use it but preserving the
 * current X11 cmap.
 */
setXcmap( bitmap, cmapref, cmap_flag );

/*
 * We wanna predefine ...
 */
whints.flags = PSize | PMinSize | PMaxSize;
whints.x = whints.y = 1;
whints.width = whints.min_width = whints.max_width = bitmap->xsize;
whints.height = whints.min_height = whints.max_height = bitmap->ysize;
wattr.event_mask = ExposureMask | ButtonPressMask;
wattr.border_pixel = WhitePixel( dpy, screen );
wattr.background_pixel = BlackPixel( dpy, screen );
wmask = CWBackPixel | CWBorderPixel | CWEventMask;
normal_pointer = XCreateFontCursor( dpy, XC_crosshair );
wait_pointer = XCreateFontCursor( dpy, XC_coffee_mug );

/*
 * We have all what we wanna, so ... DO IT !!!
 */
wnd = XCreateWindow( dpy, RootWindow( dpy, screen ),
                    0, 0, bitmap->xsize, bitmap->ysize,
                    BORDER_WIDTH, depth,
                    InputOutput,
                    CopyFromParent,
```

```
        wmask,
        &wattr );
XStoreName( dpy, wnd, name );
XSetNormalHints( dpy, wnd, &whints );
XDefineCursor( dpy, wnd, wait_pointer );
/* XSetInput( dpy, wnd, ExposureMask | ButtonPressMask ); */

/*
 * Allocate memory for the image and copy it with new
 * references ( to X11 colormap ).
 */
ptr = Xbitmap = (byte *) Malloc( totalsize );
iptr = bitmap->r;
end = iptr + totalsize;
while ( iptr < end ) {
    *ptr++ = cmapref[ *iptr++ ];
}
Ximagebuffer = XCreateImage( dpy, DefaultVisual( dpy, screen ), depth,
                             ZPixmap, 0, Xbitmap,
                             bitmap->xsize, bitmap->ysize,
                             8, 0 );

XMapWindow( dpy, wnd );

/*
 * We left a process controlling the window.
 */
XDefineCursor( dpy, wnd, normal_pointer );
loop = 1;
do {
    XNextEvent( dpy, &event );
    switch( event.type ) {
        case Expose : XDefineCursor( dpy, wnd, wait_pointer );
                      exp = (XExposeEvent *) &event;
                      XPutImage( dpy, wnd, gc,
                                  Ximagebuffer,
                                  exp->x, exp->y,
                                  exp->x, exp->y,
                                  exp->width, exp->height );
                      XDefineCursor( dpy, wnd, normal_pointer );
                      break;
        case ButtonPress: loop = ( event.xbutton.button != Button3 );
                          break;
    }
}while( loop );

/* End of this window's life ( ooopps ! ) */
XUndefineCursor( dpy, wnd );
XUnmapWindow( dpy, wnd );
XDestroyImage( Ximagebuffer );
XDestroyWindow( dpy, wnd );
XCloseDisplay( dpy );

}
```

```
setXcmap( image, cmapref, cmap_flag )
bitmap_hdr *image;
int *cmapref, *cmap_flag;
{
    register int i, j;
    static XColor Xnewcmap[256];
    int Error = 0;
```

```
color_map *cmap;
int r, g, b;

cmap = (color_map *) image->cmap;
bzero( cmapref, image->colors * sizeof(int) );

/*
 * Lets go to get our colors!.
 */
for ( i = 0, j = 0; i < image->colors; i++ ) {
    if ( cmap_flag[i] ) {
        /* This color is used */
        Xnewcmap[j].red    = cmap[i][0] << 8;
        Xnewcmap[j].green = cmap[i][1] << 8;
        Xnewcmap[j].blue  = cmap[i][2] << 8;
        Xnewcmap[j].flags = DoRed | DoGreen | DoBlue;
        if ( XAllocColor( dpy, Xcmap, &Xnewcmap[j] ) ) {
            /* Alloc accepted */
            cmapref[i] = Xnewcmap[j].pixel;
            j++;
        }else Error = 1;
    }
}

if ( Error ) {
    /*
     * We got errors allocating colors, then we get the current
     * X11 cmap and aproximate those colors to nearest.
     */
    XColor *Xoldcmap;
    int num_colors;
    double distance, cd;
    int r, g, b;
    double *hue, *saturation, *level;
    double h, s, l;

    /* Get the server's number of colors */
    num_colors = DisplayCells( dpy, screen );
    /* Now get the current cmap of X11 */
    Xoldcmap = (XColor *) Malloc( sizeof(XColor) * num_colors );
    for ( i = 0; i < num_colors; i++ ) {
        Xoldcmap[i].pixel = i;
    }
    XQueryColors( dpy, Xcmap, Xoldcmap, num_colors );
    /*
     * Ok, now we wanna the Hue value of this colors.
     */
    hue = (double *) Malloc( sizeof(double) * num_colors );
    saturation = (double *) Malloc( sizeof(double) * num_colors );
    level = (double *) Malloc( sizeof(double) * num_colors );
    for ( i = 0; i < num_colors; i++ ) {
        /* We need values scaled [0..255] */
        r = Xoldcmap[i].red    >> 8;
        g = Xoldcmap[i].green >> 8;
        b = Xoldcmap[i].blue  >> 8;
        RGB_to_HSL( r, g, b, &hue[i], &saturation[i], &level[i] );
    }
    /*
     * All do it!. Lets go choose new colors.
     */
    for ( i = 0; i < image->colors; i++ ) {
```

```
    if ( cmap_flag[i] && !cmapref[i] ) {
        /*
         * We got an error with this color, it's used by image
         * but we cannot allocated.
         */
        /* Reset the distance value ... */
        distance = 30.;
        /* ... and convert the current color to HSL */
        RGB_to_HSL( cmap[i][0], cmap[i][1], cmap[i][2], &h, &s, &l );
        for ( j = 0; j < num_colors; j++ ) {
            cd = LUGABS( h - hue[j] ) +
                LUGABS( s - saturation[j] ) +
                LUGABS( l - level[j] );
            if ( cd < distance ) {
                distance = cd;
                cmapref[i] = j;
            }
        }
    }
}
free( Xoldcmap );
free( hue );
free( saturation );
free( level );
}
```

```
set_cmap_flags( bitmap, cmap_flag )
bitmap_hdr *bitmap;
int *cmap_flag;
{
    register int used_colors = 0;
    register int totalcolors = bitmap->colors;
    byte *end, *ptr;

    if ( bitmap->depth > 8 )
        error( 10 );

    bzero( cmap_flag, totalcolors * sizeof(int) );

    /* Set pointers */
    ptr = bitmap->r;
    end = ptr + bitmap->xsize * bitmap->ysize;

    /*
     * Loop while not the end and we don't use
     * all possible colors.
     */
    while ( ptr < end && used_colors != totalcolors ) {
        if ( !cmap_flag[ *ptr ] ) {
            cmap_flag[*ptr]++;
            used_colors++;
        }
        ptr++;
    }

    return used_colors;
}
```

```
#define DMAP(v,x,y)          (modN[v] > dithermatrix[x][y] ? \
                             divN[v] + 1 : divN[v])
```

```
Xdither_image(inbitmap, outbitmap, levels, gamma, cmap_flag)
bitmap_hdr *inbitmap;
bitmap_hdr *outbitmap;
int levels;
double gamma;
int *cmap_flag;
{
    register int i, j;
    int total_size;
    byte *r, *g, *b;
    byte *rout;
    int divN[256], modN[256];
    int dithermatrix[16][16];
    int col, row;
    int used_colors = 0;

    if ( levels < 2 || levels > 6 )
        error( 11 );

    if ( inbitmap->depth <= 8 )
        error( 10 );

    /* Fill new header */
    outbitmap->xsize = inbitmap->xsize;
    outbitmap->ysize = inbitmap->ysize;
    total_size = outbitmap->xsize * outbitmap->ysize;
    outbitmap->depth = no_bits( levels*levels*levels ) + 1;
    outbitmap->colors = ( 1 << outbitmap->depth );
    outbitmap->cmap = (byte *) Malloc( 3 * outbitmap->colors );
    rout = outbitmap->r = (byte *) Malloc( total_size );

    /* Pointers to in image */
    r = inbitmap->r;
    g = inbitmap->g;
    b = inbitmap->b;

    bzero( cmap_flag, 256 * sizeof(int) );
    /*
     * Init the cmap of the dithered image ( and other
     * parameters ).
     */
    create_dithermap( levels, gamma, outbitmap->cmap, divN, modN, dithermatrix);

    /*
     * Dither each pixels through the dither matrix.
     */
    for ( i = 0; i < outbitmap->ysize; i++ ) {
        row = i % 16;
        for ( j = 0; j < outbitmap->xsize; j++ ) {
            col = j % 16;
            *rout = DMAP(*r, col, row) + DMAP(*g, col, row) * levels +
                    DMAP(*b, col, row) * levels*levels;
            if ( !cmap_flag[ *rout++ ] )
                usesizelors++;
            r++, g++, b++;
        }
    }

    return used_colors;
}
```





```
/*
 * GGems.h (GraphicsGems.h)
 * Version 1.0 - Andrew Glassner
 * from "Graphics Gems", Academic Press, 1990
 */

#ifndef GG_H

#define GG_H 1

/*****
 * 2d geometry types */
*****/

typedef struct Point2Struct {    /* 2d point */
    double x, y;
} Point2;
typedef Point2 Vector2;

typedef struct IntPoint2Struct {    /* 2d integer point */
    int x, y;
} IntPoint2;

typedef struct Matrix3Struct {    /* 3-by-3 matrix */
    double element[3][3];
} Matrix3;

typedef struct Box2dStruct {    /* 2d box */
    Point2 min, max;
} Box2;

/*****
 * 3d geometry types */
*****/

typedef struct Point3Struct {    /* 3d point */
    double x, y, z;
} Point3;
typedef Point3 Vector3;

typedef struct IntPoint3Struct {    /* 3d integer point */
    int x, y, z;
} IntPoint3;

typedef struct Matrix4Struct {    /* 4-by-4 matrix */
    double element[4][4];
} Matrix4;

typedef struct Box3dStruct {    /* 3d box */
    Point3 min, max;
} Box3;

/*****
 * one-argument macros */
*****/

/* absolute value of a */
```

```
#define ABS(a)          (((a)<0) ? -(a) : (a))

/* round a to nearest integer towards 0 */
#define FLOOR(a)        ((a)>0 ? (int)(a) : -(int)(-a))

/* round a to nearest integer away from 0 */
#define CEILING(a) \
((a)==(int)(a) ? (a) : (a)>0 ? 1+(int)(a) : -(1+(int)(-a)))

/* round a to nearest int */
#define ROUND(a)        ((a)>0 ? (int)(a+0.5) : -(int)(0.5-a))

/* take sign of a, either -1, 0, or 1 */
#define ZSGN(a)         (((a)<0) ? -1 : (a)>0 ? 1 : 0)

/* take binary sign of a, either -1, or 1 if >= 0 */
#define SGN(a)          (((a)<0) ? -1 : 0)

/* shout if something that should be true isn't */
#define ASSERT(x) \
if (!(x)) fprintf(stderr," Assert failed: x\n");

/* square a */
#define SQR(a)          ((a)*(a))

/*****/
/* two-argument macros */
/*****/

/* find minimum of a and b */
#define MIN(a,b)        (((a)<(b))?(a):(b))

/* find maximum of a and b */
#define MAX(a,b)        (((a)>(b))?(a):(b))

/* swap a and b (see Gem by Wyvill) */
#define SWAP(a,b)       { a^=b; b^=a; a^=b; }

/* linear interpolation from l (when a=0) to h (when a=1)*/
/* (equal to (a*h)+((1-a)*l) */
#define LERP(a,l,h)      ((l)+(((h)-(l))*(a)))

/* clamp the input to the specified range */
#define CLAMP(v,l,h)     ((v)<(l) ? (l) : (v) > (h) ? (h) : v)

/*****/
/* memory allocation macros */
/*****/

/* create a new instance of a structure (see Gem by Hultquist) */
#define NEWSTRUCT(x)     (struct x *) (malloc((unsigned)sizeof(struct x)))

/* create a new instance of a type */
#define NEWTYPE(x)       (x *) (malloc((unsigned)sizeof(x)))

/*****/
/* useful constants */
/*****/
```

```
#define PI                3.141592        /* the venerable pi */
#define PITIMES2          6.283185        /* 2 * pi */
#define PIOVER2           1.570796        /* pi / 2 */
#define E                 2.718282        /* the venerable e */
#define Sqrt2             1.414214        /* sqrt(2) */
#define Sqrt3             1.732051        /* sqrt(3) */
#define GOLDEN            1.618034        /* the golden ratio */
#define DTOR              0.017453        /* convert degrees to radians */
#define RTOD              57.29578        /* convert radians to degrees */

/*****/
/* booleans */
/*****/

#define TRUE              1
#define FALSE             0
#define ON                1
#define OFF               0
typedef int boolean;      /* boolean data type */
typedef boolean flag;     /* flag data type */

extern double V2SquaredLength(), V2Length();
extern double V2Dot(), V2DistanceBetween2Points();
extern Vector2 *V2Negate(), *V2Normalize(), *V2Scale(), *V2Add(), *V2Sub();
extern Vector2 *V2Lerp(), *V2Combine(), *V2Mul(), *V2MakePerpendicular();
extern Vector2 *V2New(), *V2Duplicate();
extern Point2 *V2MulPointByProjMatrix();
extern Matrix3 *V2MatMul(), *TransposeMatrix3();

extern double V3SquaredLength(), V3Length();
extern double V3Dot(), V3DistanceBetween2Points();
extern Vector3 *V3Normalize(), *V3Scale(), *V3Add(), *V3Sub();
extern Vector3 *V3Lerp(), *V3Combine(), *V3Mul(), *V3Cross();
extern Vector3 *V3New(), *V3Duplicate();
extern Point3 *V3MulPointByMatrix(), *V3MulPointByProjMatrix();
extern Matrix4 *V3MatMul();

extern double RegulaFalsi(), NewtonRaphson(), findroot();

#endif
```

```
#ifndef GG4D_H
#define GG4D_H 1

/*****/
/* 4d geometry types */
/*****/

/* Point4, Vector4
 *
 * 4-space point and vector types
 */

typedef struct Point4Struct
{
    double  x, y, z, w;
}
Point4;

typedef Point4 Vector4;

/* IntPoint4
 *
 * Integer 4-space point.
 */

typedef struct IntPoint4Struct
{
    int      x, y, z, w;
}
IntPoint4;

/* Matrix5
 *
 * 5x5 matrix for general linear 4-space transformations
 */

typedef struct Matrix5Struct
{
    double  element[5][5];
}
Matrix5;

/* Box4
 *
 * A bounding box for four dimensions.
 */

typedef struct Box4dStruct
{
    Point4  min, max;
}
Box4;

/* See C file for function elaboration.
 */

extern double  V4SquaredLength(), V4Length();
extern double  V4Dot(), V4DistanceBetween2Points();
extern Vector4 *V4Negate(), *V4Normalize(), *V4Scale(), *V4Add(), *V4Sub();
extern Vector4 *V4Lerp(), *V4Combine(), *V4Mul(), *V4Cross();
extern Vector4 *V4New(), *V4Duplicate();
```

```
extern Point4    *V4MulPointByMatrix(), *V4MulPointByProjMatrix();
extern Matrix5   *V4MatMul();

extern void      V4MatPrint();
extern Matrix5   *V4MatCopy(), *V4MatMul(), *V4MatMul2();
extern Matrix5   *V4MatZero(), *V4MatIdentity();
extern Matrix5   *V4MatRotationXY(), *V4MatRotationXW(), *V4MatRotationYZ();
extern Matrix5   *V4MatRotationYW(), *V4MatRotationZX(), *V4MatRotationZW();
extern Matrix5   *V4MatTranslation(), *V4MatScaling();
extern Matrix5   *V4MatRotateXY(), *V4MatRotateXW(), *V4MatRotateYZ();
extern Matrix5   *V4MatRotateYW(), *V4MatRotateZX(), *V4MatRotateZW();
extern Matrix5   *V4MatTranslate(), *V4MatScale();

#endif
```

```
CC = gcc
```

```
example:      example.o VecLib.o VecLib4d.o
              $(CC) -o example example.o VecLib.o VecLib4d.o -lm
```

```
/*
2d and 3d Vector C Library
by Andrew Glassner
from "Graphics Gems", Academic Press, 1990
*/

#include <math.h>
#include "GGems.h"

/*****
/*    2d Library    */
*****/

/* returns squared length of input vector */
double V2SquaredLength(a)
Vector2 *a;
{
    return((a->x * a->x)+(a->y * a->y));
}

/* returns length of input vector */
double V2Length(a)
Vector2 *a;
{
    return(sqrt(V2SquaredLength(a)));
}

/* negates the input vector and returns it */
Vector2 *V2Negate(v)
Vector2 *v;
{
    v->x = -v->x;  v->y = -v->y;
    return(v);
}

/* normalizes the input vector and returns it */
Vector2 *V2Normalize(v)
Vector2 *v;
{
    double len = V2Length(v);
    if (len != 0.0) { v->x /= len;  v->y /= len; }
    return(v);
}

/* scales the input vector to the new length and returns it */
Vector2 *V2Scale(v, newlen)
Vector2 *v;
double newlen;
{
    double len = V2Length(v);
    if (len != 0.0) { v->x *= newlen/len;  v->y *= newlen/len; }
    return(v);
}

/* return vector sum c = a+b */
Vector2 *V2Add(a, b, c)
Vector2 *a, *b, *c;
{
    c->x = a->x+b->x;  c->y = a->y+b->y;
    return(c);
}
```

```
/* return vector difference c = a-b */
Vector2 *V2Sub(a, b, c)
Vector2 *a, *b, *c;
{
    c->x = a->x-b->x;  c->y = a->y-b->y;
    return(c);
}

/* return the dot product of vectors a and b */
double V2Dot(a, b)
Vector2 *a, *b;
{
    return((a->x*b->x)+(a->y*b->y));
}

/* linearly interpolate between vectors by an amount alpha */
/* and return the resulting vector. */
/* When alpha=0, result=lo.  When alpha=1, result=hi. */
Vector2 *V2Lerp(lo, hi, alpha, result)
Vector2 *lo, *hi, *result;
double alpha;
{
    result->x = LERP(alpha, lo->x, hi->x);
    result->y = LERP(alpha, lo->y, hi->y);
    return(result);
}

/* make a linear combination of two vectors and return the result. */
/* result = (a * ascl) + (b * bscl) */
Vector2 *V2Combine (a, b, result, ascl, bscl)
Vector2 *a, *b, *result;
double ascl, bscl;
{
    result->x = (ascl * a->x) + (bscl * b->x);
    result->y = (ascl * a->y) + (bscl * b->y);
    return(result);
}

/* multiply two vectors together component-wise */
Vector2 *V2Mul (a, b, result)
Vector2 *a, *b, *result;
{
    result->x = a->x * b->x;
    result->y = a->y * b->y;
    return(result);
}

/* return the distance between two points */
double V2DistanceBetween2Points(a, b)
Point2 *a, *b;
{
    double dx = a->x - b->x;
    double dy = a->y - b->y;
    return(sqrt((dx*dx)+(dy*dy)));
}

/* return the vector perpendicular to the input vector a */
Vector2 *V2MakePerpendicular(a, ap)
Vector2 *a, *ap;
```



```
{
    ap->x = -a->y;
    ap->y = a->x;
    return(ap);
}

/* create, initialize, and return a new vector */
Vector2 *V2New(x, y)
double x, y;
{
    Vector2 *v = NEWTYPE(Vector2);
    v->x = x;  v->y = y;
    return(v);
}

/* create, initialize, and return a duplicate vector */
Vector2 *V2Duplicate(a)
Vector2 *a;
{
    Vector2 *v = NEWTYPE(Vector2);
    v->x = a->x;  v->y = a->y;
    return(v);
}

/* multiply a point by a projective matrix and return the transformed point */
Point2 *V2MulPointByProjMatrix(pin, m, pout)
Point2 *pin, *pout;
Matrix3 *m;
{
    double w;
    pout->x = (pin->x * m->element[0][0]) +
        (pin->y * m->element[1][0]) + m->element[2][0];
    pout->y = (pin->x * m->element[0][1]) +
        (pin->y * m->element[1][1]) + m->element[2][1];
    w = (pin->x * m->element[0][2]) +
        (pin->y * m->element[1][2]) + m->element[2][2];
    if (w != 0.0) { pout->x /= w;  pout->y /= w; }
    return(pout);
}

/* multiply together matrices c = ab */
/* note that c must not point to either of the input matrices */
Matrix3 *V2MatMul(a, b, c)
Matrix3 *a, *b, *c;
{
    int i, j, k;
    for (i=0; i<3; i++) {
        for (j=0; j<3; j++) {
            c->element[i][j] = 0;
            for (k=0; k<3; k++) c->element[i][j] +=
                a->element[i][k] * b->element[k][j];
        }
    }
    return(c);
}

/* transpose matrix a, return b */
Matrix3 *TransposeMatrix3(a, b)
Matrix3 *a, *b;
{

```

```
int i, j;
    for (i=0; i<3; i++) {
        for (j=0; j<3; j++)
            b->element[i][j] = a->element[j][i];
    }
    return(b);
}

/*****
/*    3d Library    */
*****/

/* returns squared length of input vector */
double V3SquaredLength(a)
Vector3 *a;
{
    return((a->x * a->x)+(a->y * a->y)+(a->z * a->z));
}

/* returns length of input vector */
double V3Length(a)
Vector3 *a;
{
    return(sqrt(V3SquaredLength(a)));
}

/* negates the input vector and returns it */
Vector3 *V3Negate(v)
Vector3 *v;
{
    v->x = -v->x;  v->y = -v->y;  v->z = -v->z;
    return(v);
}

/* normalizes the input vector and returns it */
Vector3 *V3Normalize(v)
Vector3 *v;
{
    double len = V3Length(v);
    if (len != 0.0) { v->x /= len;  v->y /= len; v->z /= len; }
    return(v);
}

/* scales the input vector to the new length and returns it */
Vector3 *V3Scale(v, newlen)
Vector3 *v;
double newlen;
{
    double len = V3Length(v);
    if (len != 0.0) {
        v->x *= newlen/len;  v->y *= newlen/len;  v->z *= newlen/len;
    }
    return(v);
}

/* return vector sum c = a+b */
Vector3 *V3Add(a, b, c)
```

```
Vector3 *a, *b, *c;
{
    c->x = a->x+b->x;  c->y = a->y+b->y;  c->z = a->z+b->z;
    return(c);
}

/* return vector difference c = a-b */
Vector3 *V3Sub(a, b, c)
Vector3 *a, *b, *c;
{
    c->x = a->x-b->x;  c->y = a->y-b->y;  c->z = a->z-b->z;
    return(c);
}

/* return the dot product of vectors a and b */
double V3Dot(a, b)
Vector3 *a, *b;
{
    return((a->x*b->x)+(a->y*b->y)+(a->z*b->z));
}

/* linearly interpolate between vectors by an amount alpha */
/* and return the resulting vector. */
/* When alpha=0, result=lo.  When alpha=1, result=hi. */
Vector3 *V3Lerp(lo, hi, alpha, result)
Vector3 *lo, *hi, *result;
double alpha;
{
    result->x = LERP(alpha, lo->x, hi->x);
    result->y = LERP(alpha, lo->y, hi->y);
    result->z = LERP(alpha, lo->z, hi->z);
    return(result);
}

/* make a linear combination of two vectors and return the result. */
/* result = (a * ascl) + (b * bscl) */
Vector3 *V3Combine (a, b, result, ascl, bscl)
Vector3 *a, *b, *result;
double ascl, bscl;
{
    result->x = (ascl * a->x) + (bscl * b->x);
    result->y = (ascl * a->y) + (bscl * b->y);
    result->z = (ascl * a->z) + (bscl * b->z);
    return(result);
}

/* multiply two vectors together component-wise and return the result */
Vector3 *V3Mul (a, b, result)
Vector3 *a, *b, *result;
{
    result->x = a->x * b->x;
    result->y = a->y * b->y;
    result->z = a->z * b->z;
    return(result);
}

/* return the distance between two points */
double V3DistanceBetween2Points(a, b)
Point3 *a, *b;
{

```

```
double dx = a->x - b->x;
double dy = a->y - b->y;
double dz = a->z - b->z;
    return(sqrt((dx*dx)+(dy*dy)+(dz*dz)));
}

/* return the cross product c = a cross b */
Vector3 *V3Cross(a, b, c)
Vector3 *a, *b, *c;
{
    c->x = (a->y*b->z) - (a->z*b->y);
    c->y = (a->z*b->x) - (a->x*b->z);
    c->z = (a->x*b->y) - (a->y*b->x);
    return(c);
}

/* create, initialize, and return a new vector */
Vector3 *V3New(x, y, z)
double x, y, z;
{
Vector3 *v = NEWTYPE(Vector3);
    v->x = x;  v->y = y;  v->z = z;
    return(v);
}

/* create, initialize, and return a duplicate vector */
Vector3 *V3Duplicate(a)
Vector3 *a;
{
Vector3 *v = NEWTYPE(Vector3);
    v->x = a->x;  v->y = a->y;  v->z = a->z;
    return(v);
}

/* multiply a point by a matrix and return the transformed point */
Point3 *V3MulPointByMatrix(pin, m, pout)
Point3 *pin, *pout;
Matrix3 *m;
{
    pout->x = (pin->x * m->element[0][0]) + (pin->y * m->element[1][0]) +
              (pin->z * m->element[2][0]);
    pout->y = (pin->x * m->element[0][1]) + (pin->y * m->element[1][1]) +
              (pin->z * m->element[2][1]);
    pout->z = (pin->x * m->element[0][2]) + (pin->y * m->element[1][2]) +
              (pin->z * m->element[2][2]);
    return(pout);
}

/* multiply a point by a projective matrix and return the transformed point */
Point3 *V3MulPointByProjMatrix(pin, m, pout)
Point3 *pin, *pout;
Matrix4 *m;
{
double w;
    pout->x = (pin->x * m->element[0][0]) + (pin->y * m->element[1][0]) +
              (pin->z * m->element[2][0]) + m->element[3][0];
    pout->y = (pin->x * m->element[0][1]) + (pin->y * m->element[1][1]) +
              (pin->z * m->element[2][1]) + m->element[3][1];
    pout->z = (pin->x * m->element[0][2]) + (pin->y * m->element[1][2]) +
              (pin->z * m->element[2][2]) + m->element[3][2];
}
```

```

w = (pin->x * m->element[0][3]) + (pin->y * m->element[1][3]) +
    (pin->z * m->element[2][3]) + m->element[3][3];
if (w != 0.0) { pout->x /= w; pout->y /= w; pout->z /= w; }
return(pout);
}

```

```

/* multiply together matrices c = ab */
/* note that c must not point to either of the input matrices */
Matrix4 *V3MatMul(a, b, c)
Matrix4 *a, *b, *c;
{
int i, j, k;
for (i=0; i<4; i++) {
for (j=0; j<4; j++) {
c->element[i][j] = 0;
for (k=0; k<4; k++) c->element[i][j] +=
a->element[i][k] * b->element[k][j];
}
}
return(c);
}

```

```

/* binary greatest common divisor by Silver and Terzian. See Knuth */
/* both inputs must be >= 0 */
gcd(u, v)
int u, v;
{
int t, f;
if ((u<0) || (v<0)) return(1); /* error if u<0 or v<0 */
f = 1;
while ((0 == (u%2)) && (0 == (v%2))) {
u>>=1; v>>=1, f*=2;
}
if (u&01) { t = -v; goto B4; } else { t = u; }
B3: if (t > 0) { t >>= 1; } else { t = -((-t) >> 1); }
B4: if (0 == (t%2)) goto B3;

if (t > 0) u = t; else v = -t;
if (0 != (t = u - v)) goto B3;
return(u*f);
}

```

```

/*****
/* Useful Routines */
*****/

```

```

/* return roots of ax^2+bx+c */
/* stable algebra derived from Numerical Recipes by Press et al.*/
int quadraticRoots(a, b, c, roots)
double a, b, c, *roots;
{
double d, q;
int count = 0;
d = (b*b)-(4*a*c);
if (d < 0.0) { *roots = *(roots+1) = 0.0; return(0); }
q = -0.5 * (b + (SGN(b)*sqrt(d)));
if (a != 0.0) { *roots++ = q/a; count++; }
if (q != 0.0) { *roots++ = c/q; count++; }
return(count);
}

```

```
/* generic 1d regula-falsi step. f is function to evaluate */
/* interval known to contain root is given in left, right */
/* returns new estimate */
double RegulaFalsi(f, left, right)
double (*f)(), left, right;
{
double d = (*f)(right) - (*f)(left);
    if (d != 0.0) return (right - (*f)(right)*(right-left)/d);
    return((left+right)/2.0);
}

/* generic 1d Newton-Raphson step. f is function, df is derivative */
/* x is current best guess for root location. Returns new estimate */
double NewtonRaphson(f, df, x)
double (*f)(), (*df)(), x;
{
double d = (*df)(x);
    if (d != 0.0) return (x-((*f)(x)/d));
    return(x-1.0);
}

/* hybrid 1d Newton-Raphson/Regula Falsi root finder. */
/* input function f and its derivative df, an interval */
/* left, right known to contain the root, and an error tolerance */
/* Based on Blinn */
double findroot(left, right, tolerance, f, df)
double left, right, tolerance;
double (*f)(), (*df)();
{
double newx = left;
    while (ABS((*f)(newx)) > tolerance) {
        newx = NewtonRaphson(f, df, newx);
        if (newx < left || newx > right)
            newx = RegulaFalsi(f, left, right);
        if ((*f)(newx) * (*f)(left) <= 0.0) right = newx;
        else left = newx;
    }
    return(newx);
}
```

```
#include <stdio.h>
#include <stdlib.h>      /* for malloc() definition */
#include <math.h>
#include "GGems.h"
#include "GGems4d.h"

/* 4D VECTOR LIBRARY.
 *
 * Steve Hill, Computing Laboratory, University of Kent, UK
 *
 * Email: S.A.Hill@ukc.ac.uk
 *
 * Included with Graphics Gems V, Academic Press ed. Alan Paeth.
 *
 * Extends the standard Graphics Gems library to four dimensions
 * and provides functions to create and manipulate homogeneous
 * transformations for 4-vectors (including projections).
 */

/* V4SquaredLength()
 *
 * Compute the square of the magnitude of a 4-vector
 */

double
V4SquaredLength(v)
Vector4 *v;
{
    return v->x * v->x +
           v->y * v->y +
           v->z * v->z +
           v->w * v->w;
}

/* V4Length()
 *
 * Compute the magnitude of a 4-vector
 * If square of magnitude is required use V4SquaredLength().
 */

double
V4Length(v)
Vector4 *v;
{
    return sqrt(V4SquaredLength(v));
}

/* V4Negate()
 *
 * Negate vector elementwise.
 */

Vector4 *
V4Negate(v)
Vector4 *v;
{
    v->x = -v->x;
    v->y = -v->y;
    v->z = -v->z;
    v->w = -v->w;
}
```

```
        return v;
    }

/* V4Normalize()
 *
 * Convert vector to unit magnitude.
 * A zero-length vector is unchanged.
 */

Vector4 *
V4Normalize(v)
Vector4 *v;
{
    double len = V4Length(v);

    if (len != 0.0)
    {
        v->x /= len;
        v->y /= len;
        v->z /= len;
        v->w /= len;
    }

    return v;
}

/* V4Scale()
 *
 * Scale vector to requested magnitude.
 * Zero length vectors unchanged.
 */

Vector4 *
V4Scale(v, newlen)
Vector4 *v;
double newlen;
{
    double len = V4Length(v);

    if (len != 0.0)
    {
        v->x *= newlen/len;
        v->y *= newlen/len;
        v->z *= newlen/len;
        v->w *= newlen/len;
    }

    return v;
}

/* V4Add()
 *
 * Add (elementwise) two 4-vectors
 */

Vector4 *
V4Add(a, b, result)
Vector4 *a, *b, *result;
{
    result->x = a->x + b->x;
    result->y = a->y + b->y;
```



```
    result->z = a->z + b->z;  
    result->w = a->w + b->w;
```

```
    return result;  
}
```

```
/* V4Sub()  
 *  
 * Subtract (elementwise) vector b from vector a.  
 */
```

```
Vector4 *  
V4Sub(a, b, result)  
Vector4 *a, *b, *result;  
{  
    result->x = a->x - b->x;  
    result->y = a->y - b->y;  
    result->z = a->z - b->z;  
    result->w = a->w - b->w;  
  
    return result;  
}
```

```
/* V4Dot()  
 *  
 * Returns the 4-space dot-product of two vectors.  
 */
```

```
double  
V4Dot(a, b)  
Vector4 *a, *b;  
{  
    return  a->x * b->x +  
           a->y * b->y +  
           a->z * b->z +  
           a->w * b->w;  
}
```

```
/* V4Lerp()  
 *  
 * Calculates vector that is a linear interpolation between  
 * lo and hi, according to alpha.  
 * See also GraphicsGems.h (GGems.h)  
 */
```

```
Vector4 *  
V4Lerp(lo, hi, alpha, result)  
Vector4 *lo, *hi, *result;  
double alpha;  
{  
    result->x = LERP(alpha, lo->x, hi->x);  
    result->y = LERP(alpha, lo->y, hi->y);  
    result->z = LERP(alpha, lo->z, hi->z);  
    result->w = LERP(alpha, lo->w, hi->w);  
  
    return result;  
}
```

```
/* V4Combine()  
 *  
 * Calculate a linear combination of two vectors.
```

\*/

```
Vector4 *
V4Combine(a, b, result, ascl, bscl)
Vector4 *a, *b, *result;
double ascl, bscl;
{
    result->x = ascl * a->x + bscl * b->x;
    result->y = ascl * a->y + bscl * b->y;
    result->z = ascl * a->z + bscl * b->z;
    result->w = ascl * a->w + bscl * b->w;

    return result;
}
```

```
/* V4Mul()
 *
 * Multiply (elementwise) two vectors.
 */
```

```
Vector4 *
V4Mul(a, b, result)
Vector4 *a, *b, *result;
{
    result->x = a->x * b->x;
    result->y = a->y * b->y;
    result->z = a->z * b->z;
    result->w = a->w * b->w;

    return result;
}
```

```
/* V4DistanceBetween2Points()
 *
 * Calculates the distance between two 4-space points.
 */
```

```
double
V4DistanceBetween2Points(a, b)
Point4 *a, *b;
{
    double dx = a->x - b->x,
           dy = a->y - b->y,
           dz = a->z - b->z,
           dw = a->w - b->w;

    return sqrt(dx*dx + dy*dy + dz*dz + dw*dw);
}
```

```
/* V4Cross()
 *
 * Calculates the cross product of three 4-space vectors.
 */
```

```
Vector4 *
V4Cross(a, b, c, result)
Vector4 *a, *b, *c, *result;
{
    double d1, d2, d3, d4, d5, d6;

    d1 = (b->z * c->w) - (b->w * c->z);
```

```
    d2 = (b->y * c->w) - (b->w * c->y);
    d3 = (b->y * c->z) - (b->z * c->y);
    d4 = (b->x * c->w) - (b->w * c->x);
    d5 = (b->x * c->z) - (b->z * c->x);
    d6 = (b->x * c->y) - (b->y * c->x);

    result->x = - a->y * d1 + a->z * d2 - a->w * d3;
    result->y =  a->x * d1 - a->z * d4 + a->w * d5;
    result->z = - a->x * d2 + a->y * d4 - a->w * d6;
    result->w =  a->x * d3 - a->y * d5 + a->z * d6;

    return result;
}
```

```
/* V4New()
 *
 * Allocate and initialise a new 4-vector
 */
```

```
Vector4 *
V4New(x, y, z, w)
double x, y, z, w;
{
    Vector4 *v = NEWTYPE(Vector4);

    v->x = x;
    v->y = y;
    v->z = z;
    v->w = w;

    return v;
}
```

```
/* V4Duplicate()
 *
 * Create a copy of a 4-vector.
 */
```

```
Vector4 *
V4Duplicate(a)
Vector4 *a;
{
    Vector4 *v = NEWTYPE(Vector4);

    v->x = a->x;
    v->y = a->y;
    v->z = a->z;
    v->w = a->w;

    return v;
}
```

```
/* V4MatPrint()
 *
 * Diagnostic function to print out a 5x5 matrix
 *
 */
```

```
void
V4MatPrint(file, mat)
FILE *file;
```

```
Matrix5 *mat;
{
    int    i, j;

    for (i = 0; i < 5; i += 1)
    {
        for (j = 0; j < 5; j += 1)
            fprintf(file, "%lf ", mat->element[i][j]);
        putc('\n', file);
    }
}

/* V4MatCopy()
 *
 * Copy a 5x5 matrix.
 */

Matrix5 *
V4MatCopy(from, to)
Matrix5 *from, *to;
{
    int    i, j;

    for (i = 0; i < 5; i += 1)
        for (j = 0; j < 5; j += 1)
            to->element[i][j] = from->element[i][j];

    return to;
}

/* V4MulPointByMatrix()
 *
 * Apply 4x4 transformation matrix to 4-space point.
 *
 * In common with the standard Graphics Gems library, points/vectors
 * are considered to be row vectors, hence multiplication is
 * post-multiplication. The transformation T o S o R represents
 * a translation followed by a scale followed by a rotation.
 * Where o is matrix multiplication (see V4MatMul() and V4MatMul2() below).
 */

Point4 *
V4MulPointByMatrix(pin, m, pout)
Point4 *pin, *pout;
Matrix4 *m;
{
    pout->x = pin->x * m->element[0][0] +
              pin->y * m->element[1][0] +
              pin->z * m->element[2][0] +
              pin->w * m->element[3][0];
    pout->y = pin->x * m->element[0][1] +
              pin->y * m->element[1][1] +
              pin->z * m->element[2][1] +
              pin->w * m->element[3][1];
    pout->z = pin->x * m->element[0][2] +
              pin->y * m->element[1][2] +
              pin->z * m->element[2][2] +
              pin->w * m->element[3][2];
    pout->w = pin->x * m->element[0][3] +
              pin->y * m->element[1][3] +
              pin->z * m->element[2][3] +
              pin->w * m->element[3][3];
}
```

```
        pin->w * m->element[3][3];

    return pout;
}

/* V4MulPointByProjMatrix()
 *
 * Apply 5x5 (projection) matrix to 4-space point
 *
 * For example, the projection matrix:
 *
 * [ 1  0  0  0  0 ]
 * [ 0  1  0  0  0 ]
 * [ 0  0  0  0  1/dz ]
 * [ 0  0  0  0  1/dw ]
 * [ 0  0  0  0  1 ]
 *
 * represents a combined perspective/parallel projection along
 * the Z and W axes. As dz or dw tend to infinity the projections
 * along those axes becomes parallel.
 *
 * 5x5 matrices may also be used to perform translations.
 *
 * [ 1  0  0  0  0 ]
 * [ 0  1  0  0  0 ]
 * [ 0  0  1  0  0 ]
 * [ 0  0  0  1  0 ]
 * [ x  y  z  w  1 ]
 */
```

```
Point4 *
V4MulPointByProjMatrix(pin, m, pout)
Point4 *pin, *pout;
Matrix5 *m;
{
    double v;

    pout->x = pin->x * m->element[0][0] +
              pin->y * m->element[1][0] +
              pin->z * m->element[2][0] +
              pin->w * m->element[3][0] +
              m->element[4][0];
    pout->y = pin->x * m->element[0][1] +
              pin->y * m->element[1][1] +
              pin->z * m->element[2][1] +
              pin->w * m->element[3][1] +
              m->element[4][1];
    pout->z = pin->x * m->element[0][2] +
              pin->y * m->element[1][2] +
              pin->z * m->element[2][2] +
              pin->w * m->element[3][2] +
              m->element[4][2];
    pout->w = pin->x * m->element[0][3] +
              pin->y * m->element[1][3] +
              pin->z * m->element[2][3] +
              pin->w * m->element[3][3] +
              m->element[4][3];

    v = pin->x * m->element[0][4] +
        pin->y * m->element[1][4] +
        pin->z * m->element[2][4] +
```

```
    pin->w * m->element[3][4] +  
    m->element[4][4];
```

```
if (v != 0.0)  
{  
    pout->x /= v;  
    pout->y /= v;  
    pout->z /= v;  
    pout->w /= v;  
}
```

```
return pout;
```

```
}
```

```
/* V4MatMul()
```

```
*
```

```
* Multiply two 5x5 matrices.  Result matrix must be distinct  
* from argument matrices.
```

```
*/
```

```
Matrix5 *
```

```
V4MatMul(a, b, result)
```

```
Matrix5 *a, *b, *result;
```

```
{
```

```
    int    i, j, k;
```

```
    for (i = 0; i < 5; i++)
```

```
    for (j = 0; j < 5; j++)
```

```
    {
```

```
        double t = 0.0;
```

```
        for (k = 0; k < 5; k++)
```

```
            t += a->element[i][k] * b->element[k][j];
```

```
        result->element[i][j] = t;
```

```
    }
```

```
    return result;
```

```
}
```

```
/* V4MatMul2
```

```
*
```

```
* Variation of V4MatMul() where result matrix may be the same  
* as matrices a or b.
```

```
*/
```

```
Matrix5 *
```

```
V4MatMul2(a, b, result)
```

```
Matrix5 *a, *b, *result;
```

```
{
```

```
    Matrix5 tmp;
```

```
    V4MatMul(a, b, &tmp);
```

```
    V4MatCopy(&tmp, result);
```

```
    return result;
```

```
}
```

```
/* 4D TRANSFORMATIONS
```

```
*
```

```
* The following functions provide various methods  
* to apply and manipulate transformations.
```

```
*/  
  
/* V4MatZero()  
*  
* Clear a 5x5 matrix.  
*/  
  
Matrix5 *  
V4MatZero(mat)  
Matrix5 *mat;  
{  
    int    i, j;  
  
    for (i = 0; i < 5; i += 1)  
        for (j = 0; j < 5; j += 1)  
            mat->element[i][j] = 0.0;  
}  
  
/* STANDARD TRANSFORMATIONS  
*  
* The following functions initialise the standard affine transformations  
* ie. Identity, Rotation (6 kinds), Translation and Scaling.  
*/  
  
/* V4MatIdentity()  
*  
* Initialises to identity matrix.  
*/  
  
Matrix5 *  
V4MatIdentity(mat)  
Matrix5 *mat;  
{  
    int    i, j;  
  
    for (i = 0; i < 5; i += 1)  
        for (j = 0; j < 5; j += 1)  
            mat->element[i][j] = i == j ? 1.0 : 0.0;  
}  
  
/* V4MatRotationXY() etc.  
*  
* Initialises a matrix to a rotation by theta about the indicated hyperplane.  
*/  
  
Matrix5 *  
V4MatRotationXY(mat, theta)  
Matrix5 *mat;  
double  theta;  
{  
    double  s, c;  
  
    s = sin(theta);  
    c = cos(theta);  
  
    V4MatIdentity(mat);  
    mat->element[0][0] = c;  
    mat->element[0][1] = s;  
    mat->element[1][0] = -s;  
    mat->element[1][1] = c;  
}
```

```
        return mat;
    }

Matrix5 *
V4MatRotationXW(mat, theta)
Matrix5 *mat;
double theta;
{
    double s, c;

    s = sin(theta);
    c = cos(theta);

    V4MatIdentity(mat);
    mat->element[0][0] = c;
    mat->element[0][3] = s;
    mat->element[3][0] = -s;
    mat->element[3][3] = c;

    return mat;
}

Matrix5 *
V4MatRotationYZ(mat, theta)
Matrix5 *mat;
double theta;
{
    double s, c;

    s = sin(theta);
    c = cos(theta);

    V4MatIdentity(mat);
    mat->element[1][1] = c;
    mat->element[1][2] = s;
    mat->element[2][1] = -s;
    mat->element[2][2] = c;

    return mat;
}

Matrix5 *
V4MatRotationYW(mat, theta)
Matrix5 *mat;
double theta;
{
    double s, c;

    s = sin(theta);
    c = cos(theta);

    V4MatIdentity(mat);
    mat->element[1][1] = c;
    mat->element[1][3] = -s;
    mat->element[3][1] = s;
    mat->element[3][3] = c;

    return mat;
}

Matrix5 *
```



```
V4MatRotationZX(mat, theta)
Matrix5 *mat;
double theta;
{
    double s, c;

    s = sin(theta);
    c = cos(theta);

    V4MatIdentity(mat);
    mat->element[0][0] = c;
    mat->element[0][2] = -s;
    mat->element[2][0] = s;
    mat->element[2][2] = c;

    return mat;
}

Matrix5 *
V4MatRotationZW(mat, theta)
Matrix5 *mat;
double theta;
{
    double s, c;

    s = sin(theta);
    c = cos(theta);

    V4MatIdentity(mat);
    mat->element[2][2] = c;
    mat->element[2][3] = -s;
    mat->element[3][2] = s;
    mat->element[3][3] = c;

    return mat;
}

/* V4MatTranslation()
 *
 * Initialise a matrix to the translation v
 */

Matrix5 *
V4MatTranslation(mat, v)
Matrix5 *mat;
Vector4 *v;
{
    V4MatIdentity(mat);
    mat->element[4][0] = v->x;
    mat->element[4][1] = v->y;
    mat->element[4][2] = v->z;
    mat->element[4][3] = v->w;

    return mat;
}

/* V4MatScaling()
 *
 * Initialise a matrix to the scaling v
 */
```

```
Matrix5 *
V4MatScaling(mat, v)
Matrix5 *mat;
Vector4 *v;
{
    V4MatIdentity(mat);
    mat->element[0][0] = v->x;
    mat->element[1][1] = v->y;
    mat->element[2][2] = v->z;
    mat->element[3][3] = v->w;

    return mat;
}

/* MATRIX OPERATIONS
 *
 * The following functions are provided for a fast compact method
 * to create compound transformations. Matrix multiplication has
 * been folded into the functions thus eliminating the need for
 * many redundant calculations. For sparse matrices containing
 * mainly just ones and zeros this represents a considerable saving.
 *
 * Each function takes the form F(M, args) and has the effect
 * M := M o F1(args) where F1 is the transformation corresponding
 * to F.
 */
```

```
Matrix5 *
V4MatRotateXY(mat, theta)
Matrix5 *mat;
double theta;
{
    double a, b, f, g, k, l, p, q, u, v;
    double st, ct;

    st = sin(theta);
    ct = cos(theta);

    a = mat->element[0][0]; b = mat->element[0][1];
    f = mat->element[1][0]; g = mat->element[1][1];
    k = mat->element[2][0]; l = mat->element[2][1];
    p = mat->element[3][0]; q = mat->element[3][1];
    u = mat->element[4][0]; v = mat->element[4][1];

    mat->element[0][0] = a * ct - b * st;
    mat->element[0][1] = a * st + b * ct;
    mat->element[1][0] = f * ct - g * st;
    mat->element[1][1] = f * st + g * ct;
    mat->element[2][0] = k * ct - l * st;
    mat->element[2][1] = k * st + l * ct;
    mat->element[3][0] = p * ct - q * st;
    mat->element[3][1] = p * st + q * ct;
    mat->element[4][0] = u * ct - v * st;
    mat->element[4][1] = u * st + v * ct;

    return mat;
}
```

```
Matrix5 *
V4MatRotateXW(mat, theta)
Matrix5 *mat;
```

```
double theta;
{
    double a, d, f, i, k, n, p, s, u, x;
    double st, ct;

    st = sin(theta);
    ct = cos(theta);

    a = mat->element[0][0]; d = mat->element[0][3];
    f = mat->element[1][0]; i = mat->element[1][3];
    k = mat->element[2][0]; n = mat->element[2][3];
    p = mat->element[3][0]; s = mat->element[3][3];
    u = mat->element[4][0]; x = mat->element[4][3];

    mat->element[0][0] = a * ct - d * st;
    mat->element[0][3] = a * st + d * ct;
    mat->element[1][0] = f * ct - i * st;
    mat->element[1][3] = f * st + i * ct;
    mat->element[2][0] = k * ct - n * st;
    mat->element[2][3] = k * st + n * ct;
    mat->element[3][0] = p * ct - s * st;
    mat->element[3][3] = p * st + s * ct;
    mat->element[4][0] = u * ct - x * st;
    mat->element[4][3] = u * st + x * ct;

    return mat;
}
```

```
Matrix5 *
V4MatRotateYZ(mat, theta)
Matrix5 *mat;
double theta;
{
    double b, c, g, h, l, m, q, r, v, w;
    double st, ct;

    st = sin(theta);
    ct = cos(theta);

    b = mat->element[0][1]; c = mat->element[0][2];
    g = mat->element[1][1]; h = mat->element[1][2];
    l = mat->element[2][1]; m = mat->element[2][2];
    q = mat->element[3][1]; r = mat->element[3][2];
    v = mat->element[4][1]; w = mat->element[4][2];

    mat->element[0][1] = b * ct - c * st;
    mat->element[0][2] = b * st + c * ct;
    mat->element[1][1] = g * ct - h * st;
    mat->element[1][2] = g * st + h * ct;
    mat->element[2][1] = l * ct - m * st;
    mat->element[2][2] = l * st + m * ct;
    mat->element[3][1] = q * ct - r * st;
    mat->element[3][2] = q * st + r * ct;
    mat->element[4][1] = v * ct - w * st;
    mat->element[4][2] = v * st + w * ct;

    return mat;
}
```

```
Matrix5 *
V4MatRotateYW(mat, theta)
```

```
Matrix5 *mat;
double theta;
{
    double b, d, g, i, l, n, q, s, v, x;
    double st, ct;

    st = sin(theta);
    ct = cos(theta);

    b = mat->element[0][1]; d = mat->element[0][3];
    g = mat->element[1][1]; i = mat->element[1][3];
    l = mat->element[2][1]; n = mat->element[2][3];
    q = mat->element[3][1]; s = mat->element[3][3];
    v = mat->element[4][1]; x = mat->element[4][3];

    mat->element[0][1] = b * ct + d * st;
    mat->element[0][3] = - b * st + d * ct;
    mat->element[1][1] = g * ct + i * st;
    mat->element[1][3] = - g * st + i * ct;
    mat->element[2][1] = l * ct + n * st;
    mat->element[2][3] = - l * st + n * ct;
    mat->element[3][1] = q * ct + s * st;
    mat->element[3][3] = - q * st + s * ct;
    mat->element[4][1] = v * ct + x * st;
    mat->element[4][3] = - v * st + x * ct;

    return mat;
}
```

```
Matrix5 *
V4MatRotateZX(mat, theta)
Matrix5 *mat;
double theta;
{
    double a, c, f, h, k, m, p, r, u, w;
    double st, ct;

    st = sin(theta);
    ct = cos(theta);

    a = mat->element[0][0]; c = mat->element[0][2];
    f = mat->element[1][0]; h = mat->element[1][2];
    k = mat->element[2][0]; m = mat->element[2][2];
    p = mat->element[3][0]; r = mat->element[3][2];
    u = mat->element[4][0]; w = mat->element[4][2];

    mat->element[0][0] = a * ct + c * st;
    mat->element[0][2] = - a * st + c * ct;
    mat->element[1][0] = f * ct + h * st;
    mat->element[1][2] = - f * st + h * ct;
    mat->element[2][0] = k * ct + m * st;
    mat->element[2][2] = - k * st + m * ct;
    mat->element[3][0] = p * ct + r * st;
    mat->element[3][2] = - p * st + r * ct;
    mat->element[4][0] = u * ct + w * st;
    mat->element[4][2] = - u * st + w * ct;

    return mat;
}
```

```
Matrix5 *
V4MatRotateZW(mat, theta)
Matrix5 *mat;
double theta;
{
    double c, d, h, i, m, n, r, s, w, x;
    double st, ct;

    st = sin(theta);
    ct = cos(theta);

    c = mat->element[0][2]; d = mat->element[0][3];
    h = mat->element[1][2]; i = mat->element[1][3];
    m = mat->element[2][2]; n = mat->element[2][3];
    r = mat->element[3][2]; s = mat->element[3][3];
    w = mat->element[4][2]; x = mat->element[4][3];

    mat->element[0][2] = c * ct + d * st;
    mat->element[0][3] = - c * st + d * ct;
    mat->element[1][2] = h * ct + i * st;
    mat->element[1][3] = - h * st + i * ct;
    mat->element[2][2] = m * ct + n * st;
    mat->element[2][3] = - m * st + n * ct;
    mat->element[3][2] = r * ct + s * st;
    mat->element[3][3] = - r * st + s * ct;
    mat->element[4][2] = w * ct + x * st;
    mat->element[4][3] = - w * st + x * ct;

    return mat;
}
```

```
Matrix5 *
V4MatTranslate(mat, v)
Matrix5 *mat;
Vector4 *v;
{
    double e, j, o, t, y;

    e = mat->element[0][4];
    j = mat->element[1][4];
    o = mat->element[2][4];
    t = mat->element[3][4];
    y = mat->element[4][4];

    if (e != 0.0)
    {
        mat->element[0][0] += e * v->x;
        mat->element[0][1] += e * v->y;
        mat->element[0][2] += e * v->z;
        mat->element[0][3] += e * v->w;
    }
    if (j != 0.0)
    {
        mat->element[1][0] += j * v->x;
        mat->element[1][1] += j * v->y;
        mat->element[1][2] += j * v->z;
        mat->element[1][3] += j * v->w;
    }
    if (o != 0.0)
    {
        mat->element[2][0] += o * v->x;
```

```
        mat->element[2][1] += o * v->y;
        mat->element[2][2] += o * v->z;
        mat->element[2][3] += o * v->w;
    }
    if (t != 0.0)
    {
        mat->element[3][0] += t * v->x;
        mat->element[3][1] += t * v->y;
        mat->element[3][2] += t * v->z;
        mat->element[3][3] += t * v->w;
    }
    if (y != 0.0)
    {
        mat->element[4][0] += y * v->x;
        mat->element[4][1] += y * v->y;
        mat->element[4][2] += y * v->z;
        mat->element[4][3] += y * v->w;
    }

    return mat;
}
```

```
Matrix5 *
V4MatScale(mat, v)
Matrix5 *mat;
Vector4 *v;
{
    if (v->x != 1.0)
    {
        mat->element[0][0] *= v->x;
        mat->element[1][0] *= v->x;
        mat->element[2][0] *= v->x;
        mat->element[3][0] *= v->x;
        mat->element[4][0] *= v->x;
    }
    if (v->y != 1.0)
    {
        mat->element[0][1] *= v->y;
        mat->element[1][1] *= v->y;
        mat->element[2][1] *= v->y;
        mat->element[3][1] *= v->y;
        mat->element[4][1] *= v->y;
    }
    if (v->z != 1.0)
    {
        mat->element[0][2] *= v->z;
        mat->element[1][2] *= v->z;
        mat->element[2][2] *= v->z;
        mat->element[3][2] *= v->z;
        mat->element[4][2] *= v->z;
    }
    if (v->w != 1.0)
    {
        mat->element[0][3] *= v->w;
        mat->element[1][3] *= v->w;
        mat->element[2][3] *= v->w;
        mat->element[3][3] *= v->w;
        mat->element[4][3] *= v->w;
    }

    return mat;
}
```

```
}

/* Postscript
 *
 * The elements of the 5x5 matrix are labelled in the above functions
 * as follows:
 *
 * a = mat->element[0][0];
 * b = mat->element[0][1];
 * c = mat->element[0][2];
 * d = mat->element[0][3];
 * e = mat->element[0][4];
 * f = mat->element[1][0];
 * g = mat->element[1][1];
 * h = mat->element[1][2];
 * i = mat->element[1][3];
 * j = mat->element[1][4];
 * k = mat->element[2][0];
 * l = mat->element[2][1];
 * m = mat->element[2][2];
 * n = mat->element[2][3];
 * o = mat->element[2][4];
 * p = mat->element[3][0];
 * q = mat->element[3][1];
 * r = mat->element[3][2];
 * s = mat->element[3][3];
 * t = mat->element[3][4];
 * u = mat->element[4][0];
 * v = mat->element[4][1];
 * w = mat->element[4][2];
 * x = mat->element[4][3];
 * y = mat->element[4][4];
 *
 * ie.
 * [ a b c d e ]
 * [ f g h i j ]
 * [ k l m n o ]
 * [ p q r s t ]
 * [ u v w x y ]
 *
 */
```

```
/* example.c
 *
 * Steve Hill, Computing Laboratory, University of Kent, UK.
 *
 * Email: S.A.Hill@ukc.ac.uk
 *
 * This file contains a short example of using the 4D vector
 * library. It concentrates on the matrix functions since these
 * are the main extension not present in the original Graphics Gems
 * libraries.
 */

#include <stdio.h>
#include <math.h>
#include "GGems.h"
#include "GGems4d.h"

#ifndef M_PI
#define M_PI      3.14159265358979323846
#endif

void
Example1()
{
    Vector4 *a, *b, *c, r;
    Vector3 *a1, *b1, r1;

    a = V4New(1.0, 0.0, 0.0, 0.0);
    b = V4New(0.0, 1.0, 0.0, 0.0);
    c = V4New(0.0, 0.0, 1.0, 0.0);

    a1 = V3New(1.0, 0.0, 0.0);
    b1 = V3New(0.0, 1.0, 0.0);

    V3Cross(a1, b1, &r1);
    V4Cross(a, b, c, &r);

    printf("(%lf, %lf, %lf)\n", r1.x, r1.y, r1.z);
    printf("(%lf, %lf, %lf, %lf)\n", r.x, r.y, r.z, r.w);
}

/* Example2
 *
 * Constructing a compound transformation using matrix product.
 * T(-1,-2,-3,-4) o Rxw(PI) o T(1, 2, 3, 4)
 */

void
Example2()
{
    Vector4 v = {-1.0, -2.0, -3.0, -4.0};
    Matrix5 m1, m2;

    V4MatTranslation(&m1, &v);
    V4MatRotationXW(&m2, M_PI);
    V4MatMul2(&m1, &m2, &m1);
    V4Negate(&v);
    V4MatTranslation(&m2, &v);
    V4MatMul2(&m1, &m2, &m1);
}
```



```
        V4MatPrint(stdout, &m1);
    }

/* Example3
 *
 * Constructing a compound transformation using affine operations
 * T(-1,-2,-3,-4) o Rxw(PI) o T(1, 2, 3, 4)
 */

void
Example3()
{
    Vector4 v = {-1.0, -2.0, -3.0, -4.0};
    Matrix5 m;

    V4MatIdentity(&m);
    V4MatTranslate(&m, v);
    V4MatRotateXW(&m, M_PI);
    V4Negate(&v);
    V4MatTranslate(&m, v);

    V4MatPrint(stdout, &m);
}

/* Example4
 *
 * All transformation functions return a pointer to their
 * result, it is possible therefore to use the result directly in
 * subsequent function call.
 * Here is Example3 recoded in this style.
 * NB
 * Some care has to be exercised since the order of argument evaluation
 * is not defined.
 */

void
Example4()
{
    Matrix5 m;
    Vector4 v = {-1.0, -2.0, -3.0, -4.0};

    V4MatRotateXW(V4MatTranslate(V4MatIdentity(&m), &v), M_PI);
    V4MatTranslate(&m, V4Negate(&v));

    V4MatPrint(stdout, &m);
}

void
main()
{
    Example1();
    putchar('\n');
    Example2();
    putchar('\n');
    Example3();
    putchar('\n');
    Example4();
}
```

```
CC                = CC
CCOPTIONS         =
CCINCLUDES       = $(STD_INCLUDES) -I/usr/include/CC -I.
EXTRA_DEFINES    =
EXTRA_LIBRARIES  =

.SUFFIXES:       .C $(SUFFIXES)

LIBNAME          = gm++

SRCS             = gmMatrix3.C gmMatrix4.C
OBJS            = $(SRCS:.C=.o)

.C.o:
    $(CC) $(CDEBUGFLAGS) -c $(CCOPTIONS) $(EXTRA
b INES) $< $(CCINCLUDES)

NormalLibraryTarget($(LIBNAME), $(OBJS))
```

```
// bool.h - boolean type
//
// libgm++: Graphics Math Library
// Ferdi Scheepers and Stephen F May
// 15 June 1994
```

```
#ifndef BOOL_H
#define BOOL_H
```

```
typedef int bool;
enum { false, true };
```

```
#endif
```

```
// gm.h - library header
//
// libgm++: Graphics Math Library
// Ferdi Scheepers and Stephen F May
// 15 June 1994
```

```
#ifndef GM_H
#define GM_H
```

```
#include <bool.h>
#include <gmConst.h>
#include <gmUtils.h>
#include <gmVec2.h>
#include <gmVec3.h>
#include <gmVec4.h>
#include <gmMat3.h>
#include <gmMat4.h>
```

```
#endif
```

```
// gmConst.h - graphics math constants
//
// libgm++: Graphics Math Library
// Ferdi Scheepers and Stephen F May
// 15 June 1994

#ifndef GMCONST_H
#define GMCONST_H

// USEFUL CONSTANTS

double const gm2PI = 6.28318530717958623200;
double const gmDEGTORAD = 0.01745329251994329547;
double const gmE = 2.71828182845904553488;
double const gmEEXPPPI = 23.14069263277927390732;
double const gmGOLDEN = 1.61803398874989490253;
double const gmINVPI = 0.31830988618379069122;
double const gmLN10 = 2.30258509299404590109;
double const gmLN2 = 0.69314718055994528623;
double const gmLOG10E = 0.43429448190325187218;
double const gmLOG2E = 1.44269504088896338700;
double const gmPI = 3.14159265358979323846;
double const gmPIDIV2 = 1.57079632679489655800;
double const gmPIDIV4 = 0.78539816339744827900;
double const gmRADTODEG = 57.29577951308232286465;
double const gmSQRT2 = 1.41421356237309514547;
double const gmSQRT2PI = 2.50662827463100024161;
double const gmSQRT3 = 1.73205080756887719318;
double const gmSQRT10 = 3.16227766016837952279;
double const gmSQRTE = 1.64872127070012819416;
double const gmSQRTHALF = 0.70710678118654757274;
double const gmSQRTL2 = 0.83255461115769768821;
double const gmSQRTPI = 1.77245385090551588192;
double const gmEPSILON = 1.0e-10;
double const gmGOOGOL = 1.0e50;

#endif
```

```
// gmMatrix3.cc - 3x3 element matrix class
//
// libgm++: Graphics Math Library
// Ferdi Scheepers and Stephen F May
// 15 June 1994

#include "gmMat3.h"
#include "gmVec2.h"
#include "gmVec3.h"

// private function: RCD
// - dot product of row i of matrix A and row j of matrix B

inline double RCD(const gmMatrix3& A, const gmMatrix3& B, int i, int j)
{
    return A[i][0] * B[0][j] + A[i][1] * B[1][j] + A[i][2] * B[2][j];
}

// private function: MINOR
// - MINOR of M[r][c]; r in {0,1,2}-{r0,r1}; c in {0,1,2}-{c0,c1}

inline double MINOR(const gmMatrix3& M, int r0, int r1, int c0, int c1)
{
    return M[r0][c0] * M[r1][c1] - M[r1][c0] * M[r0][c1];
}

// CONSTRUCTORS

gmMatrix3::gmMatrix3()
{
    assign(0,0,0, 0,0,0, 0,0,0);
}

gmMatrix3::gmMatrix3(const gmMatrix3& M)
{
    assign(M[0][0], M[0][1], M[0][2],
          M[1][0], M[1][1], M[1][2],
          M[2][0], M[2][1], M[2][2]);
}

gmMatrix3::gmMatrix3(double a00, double a01, double a02,
                     double a10, double a11, double a12,
                     double a20, double a21, double a22)
{
    assign(a00, a01, a02,
          a10, a11, a12,
          a20, a21, a22);
}

// ASSIGNMENT

gmMatrix3& gmMatrix3::assign(double a00, double a01, double a02,
                             double a10, double a11, double a12,
                             double a20, double a21, double a22)
{
    {
        m_[0][0] = a00; m_[0][1] = a01; m_[0][2] = a02;
        m_[1][0] = a10; m_[1][1] = a11; m_[1][2] = a12;
        m_[2][0] = a20; m_[2][1] = a21; m_[2][2] = a22;
        return *this;
    }
}
```

```
gmMatrix3& gmMatrix3::operator =(const gmMatrix3& M)
{
    assign(M[0][0], M[0][1], M[0][2],
           M[1][0], M[1][1], M[1][2],
           M[2][0], M[2][1], M[2][2]);
    return *this;
}

// OPERATORS

gmMatrix3& gmMatrix3::operator +=(const gmMatrix3& M)
{
    m_[0][0] += M[0][0]; m_[0][1] += M[0][1]; m_[0][2] += M[0][2];
    m_[1][0] += M[1][0]; m_[1][1] += M[1][1]; m_[1][2] += M[1][2];
    m_[2][0] += M[2][0]; m_[2][1] += M[2][1]; m_[2][2] += M[2][2];
    return *this;
}

gmMatrix3& gmMatrix3::operator -=(const gmMatrix3& M)
{
    m_[0][0] -= M[0][0]; m_[0][1] -= M[0][1]; m_[0][2] -= M[0][2];
    m_[1][0] -= M[1][0]; m_[1][1] -= M[1][1]; m_[1][2] -= M[1][2];
    m_[2][0] -= M[2][0]; m_[2][1] -= M[2][1]; m_[2][2] -= M[2][2];
    return *this;
}

gmMatrix3& gmMatrix3::operator *=(const gmMatrix3& M)
{
    assign(RCD(*this, M, 0, 0), RCD(*this, M, 0, 1), RCD(*this, M, 0, 2),
           RCD(*this, M, 1, 0), RCD(*this, M, 1, 1), RCD(*this, M, 1, 2),
           RCD(*this, M, 2, 0), RCD(*this, M, 2, 1), RCD(*this, M, 2, 2));
    return *this;
}

gmMatrix3& gmMatrix3::operator *=(double d)
{
    m_[0][0] *= d; m_[0][1] *= d; m_[0][2] *= d;
    m_[1][0] *= d; m_[1][1] *= d; m_[1][2] *= d;
    m_[2][0] *= d; m_[2][1] *= d; m_[2][2] *= d;
    return *this;
}

gmMatrix3& gmMatrix3::operator /=(double d)
{
    double di = 1 / d;
    m_[0][0] *= di; m_[0][1] *= di; m_[0][2] *= di;
    m_[1][0] *= di; m_[1][1] *= di; m_[1][2] *= di;
    m_[2][0] *= di; m_[2][1] *= di; m_[2][2] *= di;
    return *this;
}

gmMatrix3 gmMatrix3::operator +(const gmMatrix3& M) const
{
    return gmMatrix3(m_[0][0] + M[0][0], m_[0][1] + M[0][1], m_[0][2] + M[0][2],
                     m_[1][0] + M[1][0], m_[1][1] + M[1][1], m_[1][2] + M[1][2],
                     m_[2][0] + M[2][0], m_[2][1] + M[2][1], m_[2][2] + M[2][2]);
}

gmMatrix3 gmMatrix3::operator -(const gmMatrix3& M) const
{
    return gmMatrix3(m_[0][0] - M[0][0], m_[0][1] - M[0][1], m_[0][2] - M[0][2],
                     m_[1][0] - M[1][0], m_[1][1] - M[1][1], m_[1][2] - M[1][2],
                     m_[2][0] - M[2][0], m_[2][1] - M[2][1], m_[2][2] - M[2][2]);
}
```

```
        m_[1][0] - M[1][0], m_[1][1] - M[1][1], m_[1][2] - M[1][2],
        m_[2][0] - M[2][0], m_[2][1] - M[2][1], m_[2][2] - M[2][2]);
}

gmMatrix3 gmMatrix3::operator -() const
{
    return gmMatrix3(-m_[0][0], -m_[0][1], -m_[0][2],
                     -m_[1][0], -m_[1][1], -m_[1][2],
                     -m_[2][0], -m_[2][1], -m_[2][2]);
}

gmMatrix3 gmMatrix3::operator *(const gmMatrix3& M) const
{
    return
        gmMatrix3(RCD(*this, M, 0, 0), RCD(*this, M, 0, 1), RCD(*this, M, 0, 2),
                  RCD(*this, M, 1, 0), RCD(*this, M, 1, 1), RCD(*this, M, 1, 2),
                  RCD(*this, M, 2, 0), RCD(*this, M, 2, 1), RCD(*this, M, 2, 2));
}

gmMatrix3 gmMatrix3::operator *(double d) const
{
    return gmMatrix3(m_[0][0] * d, m_[0][1] * d, m_[0][2] * d,
                     m_[1][0] * d, m_[1][1] * d, m_[1][2] * d,
                     m_[2][0] * d, m_[2][1] * d, m_[2][2] * d);
}

gmMatrix3 gmMatrix3::operator /(double d) const
{
    assert(!gmIsZero(d));
    double di = 1 / d;
    return gmMatrix3(m_[0][0] * di, m_[0][1] * di, m_[0][2] * di,
                     m_[1][0] * di, m_[1][1] * di, m_[1][2] * di,
                     m_[2][0] * di, m_[2][1] * di, m_[2][2] * di);
}

gmMatrix3 operator *(double d, const gmMatrix3& M)
{
    return gmMatrix3(M[0][0] * d, M[0][1] * d, M[0][2] * d,
                     M[1][0] * d, M[1][1] * d, M[1][2] * d,
                     M[2][0] * d, M[2][1] * d, M[2][2] * d);
}

bool gmMatrix3::operator ==(const gmMatrix3& M) const
{
    return(gmFuzEQ(m_[0][0], M[0][0]) &&
           gmFuzEQ(m_[0][1], M[0][1]) &&
           gmFuzEQ(m_[0][2], M[0][2]) &&

           gmFuzEQ(m_[1][0], M[1][0]) &&
           gmFuzEQ(m_[1][1], M[1][1]) &&
           gmFuzEQ(m_[1][2], M[1][2]) &&

           gmFuzEQ(m_[2][0], M[2][0]) &&
           gmFuzEQ(m_[2][1], M[2][1]) &&
           gmFuzEQ(m_[2][2], M[2][2]));
}

bool gmMatrix3::operator !=(const gmMatrix3& M) const
{
    return (!(*this == M));
}
```



```
gmVector3 gmMatrix3::operator *(const gmVector3& v) const
{
    return gmVector3(m_[0][0] * v[0] + m_[0][1] * v[1] + m_[0][2] * v[2],
                     m_[1][0] * v[0] + m_[1][1] * v[1] + m_[1][2] * v[2],
                     m_[2][0] * v[0] + m_[2][1] * v[1] + m_[2][2] * v[2]);
}

gmVector3 operator *(const gmVector3& v, const gmMatrix3& M)
{
    return gmVector3(v[0] * M[0][0] + v[1] * M[1][0] + v[2] * M[2][0],
                     v[0] * M[0][1] + v[1] * M[1][1] + v[2] * M[2][1],
                     v[0] * M[0][2] + v[1] * M[1][2] + v[2] * M[2][2]);
}

// OPERATIONS

gmMatrix3 gmMatrix3::transpose() const
{
    return gmMatrix3(m_[0][0], m_[1][0], m_[2][0],
                     m_[0][1], m_[1][1], m_[2][1],
                     m_[0][2], m_[1][2], m_[2][2]);
}

gmMatrix3 gmMatrix3::inverse() const
{
    assert(!isSingular());
    return adjoint() * gmInv(determinant());
}

gmMatrix3 gmMatrix3::adjoint() const
{
    gmMatrix3 A;

    A[0][0] = MINOR(*this, 1, 2, 1, 2);
    A[0][1] = -MINOR(*this, 0, 2, 1, 2);
    A[0][2] = MINOR(*this, 0, 1, 1, 2);
    A[1][0] = -MINOR(*this, 1, 2, 0, 2);
    A[1][1] = MINOR(*this, 0, 2, 0, 2);
    A[1][2] = -MINOR(*this, 0, 1, 0, 2);
    A[2][0] = MINOR(*this, 1, 2, 0, 1);
    A[2][1] = -MINOR(*this, 0, 2, 0, 1);
    A[2][2] = MINOR(*this, 0, 1, 0, 1);

    return A;
}

double gmMatrix3::determinant() const
{
    return m_[0][0] * MINOR(*this, 1, 2, 1, 2) -
           m_[0][1] * MINOR(*this, 1, 2, 0, 2) +
           m_[0][2] * MINOR(*this, 1, 2, 0, 1);
}

bool gmMatrix3::isSingular() const
{
    return gmIsZero(determinant());
}

gmVector2 gmMatrix3::transform(const gmVector2& v) const
{

```

```
    return gmVector2(v[0] * m_[0][0] + v[1] * m_[1][0] + m_[2][0],
                     v[0] * m_[0][1] + v[1] * m_[1][1] + m_[2][1]);
}

// TRANSFORMATION MATRICES

gmMatrix3 gmMatrix3::identity()
{
    return gmMatrix3(1, 0, 0,
                     0, 1, 0,
                     0, 0, 1);
}

gmMatrix3 gmMatrix3::rotate(double angle)
{
    double sine = sin(gmRadians(angle));
    double cosine = cos(gmRadians(angle));

    return gmMatrix3(cosine, sine, 0,
                     -sine, cosine, 0,
                     0, 0, 1);
}

gmMatrix3 gmMatrix3::scale(double x, double y)
{
    return gmMatrix3(x, 0, 0,
                     0, y, 0,
                     0, 0, 1);
}

gmMatrix3 gmMatrix3::translate(double x, double y)
{
    return gmMatrix3(1, 0, 0,
                     0, 1, 0,
                     x, y, 1);
}
```

```
// gmMatrix3.h - 3x3 element matrix class
//
// libgm++: Graphics Math Library
// Ferdi Scheepers and Stephen F May
// 15 June 1994

#ifndef GMMATRIX3_H
#define GMMATRIX3_H

#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <gmUtils.h>

class gmVector2;
class gmVector3;

class gmMatrix3 {

protected:
    double m_[3][3];

public:
    gmMatrix3();
    gmMatrix3(const gmMatrix3&);
    gmMatrix3(double, double, double,
               double, double, double,
               double, double, double);

    // array access

    double* operator [] (int);
    const double* operator [] (int) const;

    // assignment

    gmMatrix3& assign(double, double, double,
                     double, double, double,
                     double, double, double);
    gmMatrix3& operator =(const gmMatrix3&);

    // operators

    gmMatrix3& operator +=(const gmMatrix3&);
    gmMatrix3& operator -=(const gmMatrix3&);
    gmMatrix3& operator *=(const gmMatrix3&);
    gmMatrix3& operator *=(double);
    gmMatrix3& operator /=(double);

    gmMatrix3 operator +(const gmMatrix3&) const;
    gmMatrix3 operator -(const gmMatrix3&) const;
    gmMatrix3 operator -() const;
    gmMatrix3 operator *(const gmMatrix3&) const;
    gmMatrix3 operator *(double) const;
    gmMatrix3 operator /(double) const;

friend gmMatrix3 operator *(double, const gmMatrix3&);

    bool operator ==(const gmMatrix3&) const;
    bool operator !=(const gmMatrix3&) const;
```

```
    gmVector3 operator *(const gmVector3&) const;

friend gmVector3 operator *(const gmVector3&, const gmMatrix3&);

// operations

gmMatrix3 inverse() const;
gmMatrix3 transpose() const;
gmMatrix3 adjoint() const;

double determinant() const;
bool isSingular() const;

gmVector2 transform(const gmVector2&) const;

void copyTo(float [3][3]) const;
void copyTo(double [3][3]) const;

// transformation matrices

static gmMatrix3 identity();
static gmMatrix3 rotate(double);
static gmMatrix3 scale(double, double);
static gmMatrix3 translate(double, double);
};

// ARRAY ACCESS

inline double* gmMatrix3::operator [](int i)
{
    assert(i == 0 || i == 1 || i == 2 || i == 3);
    return &m_[i][0];
}

inline const double* gmMatrix3::operator [](int i) const
{
    assert(i == 0 || i == 1 || i == 2 || i == 3);
    return &m_[i][0];
}

inline void gmMatrix3::copyTo(float f[3][3]) const
{
    f[0][0] = m_[0][0]; f[0][1] = m_[0][1]; f[0][2] = m_[0][2];
    f[1][0] = m_[1][0]; f[1][1] = m_[1][1]; f[1][2] = m_[1][2];
    f[2][0] = m_[2][0]; f[2][1] = m_[2][1]; f[2][2] = m_[2][2];
}

inline void gmMatrix3::copyTo(double f[3][3]) const
{
    f[0][0] = m_[0][0]; f[0][1] = m_[0][1]; f[0][2] = m_[0][2];
    f[1][0] = m_[1][0]; f[1][1] = m_[1][1]; f[1][2] = m_[1][2];
    f[2][0] = m_[2][0]; f[2][1] = m_[2][1]; f[2][2] = m_[2][2];
}

#endif
```

```
// gmMatrix4.cc - 4x4 element matrix class
//
// libgm++: Graphics Math Library
// Ferdi Scheepers and Stephen F May
// 15 June 1994

#include "gmMat4.h"
#include "gmVec3.h"
#include "gmVec4.h"

// private function: RCD
// - dot product of row i of matrix A and row j of matrix B

inline double RCD(const gmMatrix4& A, const gmMatrix4& B, int i, int j)
{
    return A[i][0] * B[0][j] + A[i][1] * B[1][j] + A[i][2] * B[2][j] +
           A[i][3] * B[3][j];
}

// private function: MINOR
// - MINOR of M[r][c]; r in {0,1,2,3}-{r0,r1,r2}; c in {0,1,2,3}-{c0,c1,c2}

inline double MINOR(const gmMatrix4& M,
                    int r0, int r1, int r2, int c0, int c1, int c2)
{
    return M[r0][c0] * (M[r1][c1] * M[r2][c2] - M[r2][c1] * M[r1][c2]) -
           M[r0][c1] * (M[r1][c0] * M[r2][c2] - M[r2][c0] * M[r1][c2]) +
           M[r0][c2] * (M[r1][c0] * M[r2][c1] - M[r2][c0] * M[r1][c1]);
}

// CONSTRUCTORS

gmMatrix4::gmMatrix4()
{
    assign(0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0);
}

gmMatrix4::gmMatrix4(const gmMatrix4& M)
{
    assign(M[0][0], M[0][1], M[0][2], M[0][3],
           M[1][0], M[1][1], M[1][2], M[1][3],
           M[2][0], M[2][1], M[2][2], M[2][3],
           M[3][0], M[3][1], M[3][2], M[3][3]);
}

gmMatrix4::gmMatrix4(double a00, double a01, double a02, double a03,
                    double a10, double a11, double a12, double a13,
                    double a20, double a21, double a22, double a23,
                    double a30, double a31, double a32, double a33)
{
    assign(a00, a01, a02, a03,
           a10, a11, a12, a13,
           a20, a21, a22, a23,
           a30, a31, a32, a33);
}

// ASSIGNMENT

gmMatrix4& gmMatrix4::assign(double a00, double a01, double a02, double a03,
                            double a10, double a11, double a12, double a13,
                            double a20, double a21, double a22, double a23,
```

```
double a30, double a31, double a32, double a33)
{
    m_[0][0] = a00; m_[0][1] = a01; m_[0][2] = a02; m_[0][3] = a03;
    m_[1][0] = a10; m_[1][1] = a11; m_[1][2] = a12; m_[1][3] = a13;
    m_[2][0] = a20; m_[2][1] = a21; m_[2][2] = a22; m_[2][3] = a23;
    m_[3][0] = a30; m_[3][1] = a31; m_[3][2] = a32; m_[3][3] = a33;
    return *this;
}

gmMatrix4& gmMatrix4::operator =(const gmMatrix4& M)
{
    assign(M[0][0], M[0][1], M[0][2], M[0][3],
           M[1][0], M[1][1], M[1][2], M[1][3],
           M[2][0], M[2][1], M[2][2], M[2][3],
           M[3][0], M[3][1], M[3][2], M[3][3]);
    return *this;
}

//////////////////////////////////////
// OPERATORS

gmMatrix4& gmMatrix4::operator +=(const gmMatrix4& M)
{
    m_[0][0] += M[0][0]; m_[0][1] += M[0][1];
    m_[0][2] += M[0][2]; m_[0][3] += M[0][3];

    m_[1][0] += M[1][0]; m_[1][1] += M[1][1];
    m_[1][2] += M[1][2]; m_[1][3] += M[1][3];

    m_[2][0] += M[2][0]; m_[2][1] += M[2][1];
    m_[2][2] += M[2][2]; m_[2][3] += M[2][3];

    m_[3][0] += M[3][0]; m_[3][1] += M[3][1];
    m_[3][2] += M[3][2]; m_[3][3] += M[3][3];
    return *this;
}

gmMatrix4& gmMatrix4::operator -=(const gmMatrix4& M)
{
    m_[0][0] -= M[0][0]; m_[0][1] -= M[0][1];
    m_[0][2] -= M[0][2]; m_[0][3] -= M[0][3];

    m_[1][0] -= M[1][0]; m_[1][1] -= M[1][1];
    m_[1][2] -= M[1][2]; m_[1][3] -= M[1][3];

    m_[2][0] -= M[2][0]; m_[2][1] -= M[2][1];
    m_[2][2] -= M[2][2]; m_[2][3] -= M[2][3];

    m_[3][0] -= M[3][0]; m_[3][1] -= M[3][1];
    m_[3][2] -= M[3][2]; m_[3][3] -= M[3][3];
    return *this;
}

gmMatrix4& gmMatrix4::operator *=(const gmMatrix4& M)
{
    assign(RCD(*this, M, 0, 0), RCD(*this, M, 0, 1),
           RCD(*this, M, 0, 2), RCD(*this, M, 0, 3),
           RCD(*this, M, 1, 0), RCD(*this, M, 1, 1),
           RCD(*this, M, 1, 2), RCD(*this, M, 1, 3),
           RCD(*this, M, 2, 0), RCD(*this, M, 2, 1),
           RCD(*this, M, 2, 2), RCD(*this, M, 2, 3),
```

```
        RCD(*this, M, 3, 0), RCD(*this, M, 3, 1),  
        RCD(*this, M, 3, 2), RCD(*this, M, 3, 3));
```

```
    return *this;
```

```
}
```

```
gmMatrix4& gmMatrix4::operator *=(double d)
```

```
{
```

```
    m_[0][0] *= d; m_[0][1] *= d; m_[0][2] *= d; m_[0][3] *= d;  
    m_[1][0] *= d; m_[1][1] *= d; m_[1][2] *= d; m_[1][3] *= d;  
    m_[2][0] *= d; m_[2][1] *= d; m_[2][2] *= d; m_[2][3] *= d;  
    m_[3][0] *= d; m_[3][1] *= d; m_[3][2] *= d; m_[3][3] *= d;  
    return *this;
```

```
}
```

```
gmMatrix4& gmMatrix4::operator /=(double d)
```

```
{
```

```
    double di = 1 / d;  
    m_[0][0] *= di; m_[0][1] *= di; m_[0][2] *= di; m_[0][3] *= di;  
    m_[1][0] *= di; m_[1][1] *= di; m_[1][2] *= di; m_[1][3] *= di;  
    m_[2][0] *= di; m_[2][1] *= di; m_[2][2] *= di; m_[2][3] *= di;  
    m_[3][0] *= di; m_[3][1] *= di; m_[3][2] *= di; m_[3][3] *= di;  
    return *this;
```

```
}
```

```
gmMatrix4 gmMatrix4::operator +(const gmMatrix4& M) const
```

```
{
```

```
    return gmMatrix4(m_[0][0] + M[0][0], m_[0][1] + M[0][1],  
                     m_[0][2] + M[0][2], m_[0][3] + M[0][3],  
                     m_[1][0] + M[1][0], m_[1][1] + M[1][1],  
                     m_[1][2] + M[1][2], m_[1][3] + M[1][3],  
                     m_[2][0] + M[2][0], m_[2][1] + M[2][1],  
                     m_[2][2] + M[2][2], m_[2][3] + M[2][3],  
                     m_[3][0] + M[3][0], m_[3][1] + M[3][1],  
                     m_[3][2] + M[3][2], m_[3][3] + M[3][3]);
```

```
}
```

```
gmMatrix4 gmMatrix4::operator -(const gmMatrix4& M) const
```

```
{
```

```
    return gmMatrix4(m_[0][0] - M[0][0], m_[0][1] - M[0][1],  
                     m_[0][2] - M[0][2], m_[0][3] - M[0][3],  
                     m_[1][0] - M[1][0], m_[1][1] - M[1][1],  
                     m_[1][2] - M[1][2], m_[1][3] - M[1][3],  
                     m_[2][0] - M[2][0], m_[2][1] - M[2][1],  
                     m_[2][2] - M[2][2], m_[2][3] - M[2][3],  
                     m_[3][0] - M[3][0], m_[3][1] - M[3][1],  
                     m_[3][2] - M[3][2], m_[3][3] - M[3][3]);
```

```
}
```

```
gmMatrix4 gmMatrix4::operator -() const
```

```
{
```

```
    return gmMatrix4(-m_[0][0], -m_[0][1], -m_[0][2], -m_[0][3],  
                     -m_[1][0], -m_[1][1], -m_[1][2], -m_[1][3],  
                     -m_[2][0], -m_[2][1], -m_[2][2], -m_[2][3],  
                     -m_[3][0], -m_[3][1], -m_[3][2], -m_[3][3]);
```

```
}
```

```
gmMatrix4 gmMatrix4::operator *(const gmMatrix4& M) const
```

```
{
```

```
    return gmMatrix4(RCD(*this, M, 0, 0), RCD(*this, M, 0, 1),  
                     RCD(*this, M, 0, 2), RCD(*this, M, 0, 3),  
                     RCD(*this, M, 1, 0), RCD(*this, M, 1, 1),
```

```
        RCD(*this, M, 1, 2), RCD(*this, M, 1, 3),
        RCD(*this, M, 2, 0), RCD(*this, M, 2, 1),
        RCD(*this, M, 2, 2), RCD(*this, M, 2, 3),
        RCD(*this, M, 3, 0), RCD(*this, M, 3, 1),
        RCD(*this, M, 3, 2), RCD(*this, M, 3, 3));
}

gmMatrix4 gmMatrix4::operator *(double d) const
{
    return gmMatrix4(m_[0][0] * d, m_[0][1] * d, m_[0][2] * d, m_[0][3] * d,
        m_[1][0] * d, m_[1][1] * d, m_[1][2] * d, m_[1][3] * d,
        m_[2][0] * d, m_[2][1] * d, m_[2][2] * d, m_[2][3] * d,
        m_[3][0] * d, m_[3][1] * d, m_[3][2] * d, m_[3][3] * d);
}

gmMatrix4 gmMatrix4::operator /(double d) const
{
    assert(!gmIsZero(d));
    double di = 1 / d;
    return gmMatrix4(m_[0][0] * di, m_[0][1] * di, m_[0][2] * di, m_[0][3] * di,
        m_[1][0] * di, m_[1][1] * di, m_[1][2] * di, m_[1][3] * di,
        m_[2][0] * di, m_[2][1] * di, m_[2][2] * di, m_[2][3] * di,
        m_[3][0] * di, m_[3][1] * di, m_[3][2] * di, m_[3][3] * di);
}

gmMatrix4 operator *(double d, const gmMatrix4& M)
{
    return gmMatrix4(M[0][0] * d, M[0][1] * d, M[0][2] * d, M[0][3] * d,
        M[1][0] * d, M[1][1] * d, M[1][2] * d, M[1][3] * d,
        M[2][0] * d, M[2][1] * d, M[2][2] * d, M[2][3] * d,
        M[3][0] * d, M[3][1] * d, M[3][2] * d, M[3][3] * d);
}

bool gmMatrix4::operator ==(const gmMatrix4& M) const
{
    return (gmFuzEQ(m_[0][0], M[0][0]) && gmFuzEQ(m_[0][1], M[0][1]) &&
        gmFuzEQ(m_[0][2], M[0][2]) && gmFuzEQ(m_[0][3], M[0][3]) &&

        gmFuzEQ(m_[1][0], M[1][0]) && gmFuzEQ(m_[1][1], M[1][1]) &&
        gmFuzEQ(m_[1][2], M[1][2]) && gmFuzEQ(m_[1][3], M[1][3]) &&

        gmFuzEQ(m_[2][0], M[2][0]) && gmFuzEQ(m_[2][1], M[2][1]) &&
        gmFuzEQ(m_[2][2], M[2][2]) && gmFuzEQ(m_[2][3], M[2][3]) &&

        gmFuzEQ(m_[3][0], M[3][0]) && gmFuzEQ(m_[3][1], M[3][1]) &&
        gmFuzEQ(m_[3][2], M[3][2]) && gmFuzEQ(m_[3][3], M[3][3]));
}

bool gmMatrix4::operator !=(const gmMatrix4& M) const
{
    return (!(*this == M));
}

gmVector4 gmMatrix4::operator *(const gmVector4& v) const
{
    return gmVector4(
        m_[0][0] * v[0] + m_[0][1] * v[1] + m_[0][2] * v[2] + m_[0][3] * v[3],
        m_[1][0] * v[0] + m_[1][1] * v[1] + m_[1][2] * v[2] + m_[1][3] * v[3],
        m_[2][0] * v[0] + m_[2][1] * v[1] + m_[2][2] * v[2] + m_[2][3] * v[3],
        m_[3][0] * v[0] + m_[3][1] * v[1] + m_[3][2] * v[2] + m_[3][3] * v[3]);
}
```



```
gmVector4 operator *(const gmVector4& v, const gmMatrix4& M)
{
    return gmVector4(
        v[0] * M[0][0] + v[1] * M[1][0] + v[2] * M[2][0] + v[3] * M[3][0],
        v[0] * M[0][1] + v[1] * M[1][1] + v[2] * M[2][1] + v[3] * M[3][1],
        v[0] * M[0][2] + v[1] * M[1][2] + v[2] * M[2][2] + v[3] * M[3][2],
        v[0] * M[0][3] + v[1] * M[1][3] + v[2] * M[2][3] + v[3] * M[3][3]);
}

// OPERATIONS

gmMatrix4 gmMatrix4::transpose() const
{
    return gmMatrix4(m_[0][0], m_[1][0], m_[2][0], m_[3][0],
                     m_[0][1], m_[1][1], m_[2][1], m_[3][1],
                     m_[0][2], m_[1][2], m_[2][2], m_[3][2],
                     m_[0][3], m_[1][3], m_[2][3], m_[3][3]);
}

gmMatrix4 gmMatrix4::inverse() const
{
    assert(!isSingular());
    return adjoint() * gmInv(determinant());
}

gmMatrix4 gmMatrix4::adjoint() const
{
    gmMatrix4 A;

    A[0][0] = MINOR(*this, 1, 2, 3, 1, 2, 3);
    A[0][1] = -MINOR(*this, 0, 2, 3, 1, 2, 3);
    A[0][2] = MINOR(*this, 0, 1, 3, 1, 2, 3);
    A[0][3] = -MINOR(*this, 0, 1, 2, 1, 2, 3);
    A[1][0] = -MINOR(*this, 1, 2, 3, 0, 2, 3);
    A[1][1] = MINOR(*this, 0, 2, 3, 0, 2, 3);
    A[1][2] = -MINOR(*this, 0, 1, 3, 0, 2, 3);
    A[1][3] = MINOR(*this, 0, 1, 2, 0, 2, 3);
    A[2][0] = MINOR(*this, 1, 2, 3, 0, 1, 3);
    A[2][1] = -MINOR(*this, 0, 2, 3, 0, 1, 3);
    A[2][2] = MINOR(*this, 0, 1, 3, 0, 1, 3);
    A[2][3] = -MINOR(*this, 0, 1, 2, 0, 1, 3);
    A[3][0] = -MINOR(*this, 1, 2, 3, 0, 1, 2);
    A[3][1] = MINOR(*this, 0, 2, 3, 0, 1, 2);
    A[3][2] = -MINOR(*this, 0, 1, 3, 0, 1, 2);
    A[3][3] = MINOR(*this, 0, 1, 2, 0, 1, 2);

    return A;
}

double gmMatrix4::determinant() const
{
    return m_[0][0] * MINOR(*this, 1, 2, 3, 1, 2, 3) -
        m_[0][1] * MINOR(*this, 1, 2, 3, 0, 2, 3) +
        m_[0][2] * MINOR(*this, 1, 2, 3, 0, 1, 3) -
        m_[0][3] * MINOR(*this, 1, 2, 3, 0, 1, 2);
}

bool gmMatrix4::isSingular() const
{
    return gmIsZero(determinant());
}
```

```
}

gmVector3 gmMatrix4::transform(const gmVector3& v) const
{
    return gmVector3(
        v[0] * m_[0][0] + v[1] * m_[1][0] + v[2] * m_[2][0] + m_[3][0],
        v[0] * m_[0][1] + v[1] * m_[1][1] + v[2] * m_[2][1] + m_[3][1],
        v[0] * m_[0][2] + v[1] * m_[1][2] + v[2] * m_[2][2] + m_[3][2]);
}

////////////////////////////////////
// TRANSFORMATION MATRICES

gmMatrix4 gmMatrix4::identity()
{
    return gmMatrix4(1, 0, 0, 0,
                     0, 1, 0, 0,
                     0, 0, 1, 0,
                     0, 0, 0, 1);
}

gmMatrix4 gmMatrix4::rotate(double angle, const gmVector3& axis)
{
    gmMatrix4 m_;
    double length = axis.length();
    double a = axis[0] / length;
    double b = axis[1] / length;
    double c = axis[2] / length;
    double aa = a * a;
    double bb = b * b;
    double cc = c * c;
    double sine = sin(gmRadians(angle));
    double cosine = cos(gmRadians(angle));
    double omcos = 1 - cosine;

    m_[0][0] = aa + (1 - aa) * cosine;
    m_[1][1] = bb + (1 - bb) * cosine;
    m_[2][2] = cc + (1 - cc) * cosine;
    m_[0][1] = a * b * omcos + c * sine;
    m_[0][2] = a * c * omcos - b * sine;
    m_[1][0] = a * b * omcos - c * sine;
    m_[1][2] = b * c * omcos + a * sine;
    m_[2][0] = a * c * omcos + b * sine;
    m_[2][1] = b * c * omcos - a * sine;
    m_[0][3] = m_[1][3] = m_[2][3] = m_[3][0] = m_[3][1] = m_[3][2] = 0;
    m_[3][3] = 1;

    return m_;
}

gmMatrix4 gmMatrix4::scale(double x, double y, double z)
{
    return gmMatrix4(x, 0, 0, 0,
                     0, y, 0, 0,
                     0, 0, z, 0,
                     0, 0, 0, 1);
}

gmMatrix4 gmMatrix4::translate(double x, double y, double z)
{
    return gmMatrix4(1, 0, 0, 0,
```

```
        0, 1, 0, 0,
        0, 0, 1, 0,
        x, y, z, 1);
}

////////////////////////////////////
// CUBIC BASIS MATRICES

gmMatrix4 gmMatrix4::bezierBasis()
{
    return gmMatrix4(-1,  3, -3,  1,
                     3, -6,  3,  0,
                    -3,  3,  0,  0,
                     1,  0,  0,  0);
}

gmMatrix4 gmMatrix4::bsplineBasis()
{
    return gmMatrix4(-1,  3, -3,  1,
                     3, -6,  3,  0,
                    -3,  0,  3,  0,
                     1,  4,  1,  0) / 6;
}

gmMatrix4 gmMatrix4::catmullromBasis()
{
    return gmMatrix4(-1,  3, -3,  1,
                     2, -5,  4, -1,
                    -1,  0,  1,  0,
                     0,  2,  0,  0) / 2;
}

gmMatrix4 gmMatrix4::hermiteBasis()
{
    return gmMatrix4( 2,  1, -2,  1,
                    -3, -2,  3, -1,
                     0,  1,  0,  0,
                     1,  0,  0,  0);
}

gmMatrix4 gmMatrix4::tensedBSplineBasis(double tension)
{
    gmMatrix4 m;
    double sixth = 1.0 / 6.0;

    m[0][0] = sixth * (-tension);
    m[0][1] = sixth * (12.0 - 9.0 * tension);
    m[0][2] = sixth * (9.0 * tension - 12.0);
    m[0][3] = sixth * tension;

    m[1][0] = sixth * 3.0 * tension;
    m[1][1] = sixth * (12.0 * tension - 18.0);
    m[1][2] = sixth * (18.0 - 15.0 * tension);
    m[1][3] = 0.0;

    m[2][0] = sixth * -3.0 * tension;
    m[2][1] = 0.0;
    m[2][2] = sixth * 3.0 * tension;
    m[2][3] = 0.0;

    m[3][0] = sixth * tension;
```

```
m[3][1] = sixth * (6.0 - 2.0 * tension);
m[3][2] = sixth * tension;
m[3][3] = 0.0;
```

```
return m;
```

```
}
```

```
gmMatrix4 gmMatrix4::cardinalBasis(double tension)
```

```
{
```

```
    gmMatrix4 m;
```

```
m[0][0] = -tension;
m[0][1] = 2.0 - tension;
m[0][2] = tension - 2.0;
m[0][3] = tension;
```

```
m[1][0] = 2.0 * tension;
m[1][1] = tension - 3.0;
m[1][2] = 3 - 2.0 * tension;
m[1][3] = -tension;
```

```
m[2][0] = -tension;
m[2][1] = 0.0;
m[2][2] = tension;
m[2][3] = 0.0;
```

```
m[3][0] = 0.0;
m[3][1] = 1.0;
m[3][2] = 0.0;
m[3][3] = 0.0;
```

```
return m;
```

```
}
```

```
gmMatrix4 gmMatrix4::tauBasis(double bias, double tension)
```

```
{
```

```
    gmMatrix4 m;
```

```
m[0][0] = tension * (bias - 1.0);
m[0][1] = 2.0 - tension * bias;
m[0][2] = tension * (1.0 - bias) - 2.0;
m[0][3] = tension * bias;
```

```
m[1][0] = tension * 2.0 * (1.0 - bias);
m[1][1] = tension * (3.0 * bias - 1.0) - 3.0;
m[1][2] = 3.0 - tension;
m[1][3] = -tension * bias;
```

```
m[2][0] = tension * (bias - 1.0);
m[2][1] = tension * (1.0 - 2.0 * bias);
m[2][2] = tension * bias;
m[2][3] = 0.0;
```

```
m[3][0] = 0.0;
m[3][1] = 1.0;
m[3][2] = 0.0;
m[3][3] = 0.0;
```

```
return m;
```

```
}
```

```
gmMatrix4 gmMatrix4::betaSplineBasis(double bias, double tension)
{
    gmMatrix4 m;
    double bias2, bias3, d;

    bias2 = bias * bias;
    bias3 = bias * bias2;
    d = 1.0 / (tension + 2.0 * bias3 + 4.0 * bias2 + 4.0 * bias + 2.0);

    m[0][0] = -d * 2.0 * bias3;
    m[0][1] = d * 2.0 * (tension + bias3 + bias2 + bias);
    m[0][2] = -d * 2.0 * (tension + bias2 + bias + 1.0);
    m[0][3] = d * 2.0;

    m[1][0] = d * 6.0 * bias3;
    m[1][1] = -d * 3.0 * (tension + 2.0 * bias3 + 2.0 * bias2);
    m[1][2] = d * 3.0 * (tension + 2 * bias2);
    m[1][3] = 0.0;

    m[2][0] = -d * 6.0 * bias3;
    m[2][1] = d * 6.0 * (bias3 - bias);
    m[2][2] = d * 6.0 * bias;
    m[2][3] = 0.0;

    m[3][0] = d * 2.0 * bias3;
    m[3][1] = d * (tension + 4 * (bias2 + bias));
    m[3][2] = d * 2.0;
    m[3][3] = 0.0;

    return m;
}
```

```
// gmMatrix4.h - 4x4 element matrix class
//
// libgm++: Graphics Math Library
// Ferdi Scheepers and Stephen F May
// 15 June 1994

#ifndef GMMATRIX4_H
#define GMMATRIX4_H

#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <gmUtils.h>

class gmVector3;
class gmVector4;

class gmMatrix4 {

protected:
    double m_[4][4];

public:
    gmMatrix4();
    gmMatrix4(const gmMatrix4&);
    gmMatrix4(double, double, double, double,
              double, double, double, double,
              double, double, double, double,
              double, double, double, double);

    // array access

    double* operator [](int);
    const double* operator [](int) const;

    // assignment

    gmMatrix4& assign(double, double, double, double,
                    double, double, double, double,
                    double, double, double, double,
                    double, double, double, double);
    gmMatrix4& operator =(const gmMatrix4&);

    // operators

    gmMatrix4& operator +=(const gmMatrix4&);
    gmMatrix4& operator -=(const gmMatrix4&);
    gmMatrix4& operator *=(const gmMatrix4&);
    gmMatrix4& operator *=(double);
    gmMatrix4& operator /=(double);

    gmMatrix4 operator +(const gmMatrix4&) const;
    gmMatrix4 operator -(const gmMatrix4&) const;
    gmMatrix4 operator -() const;
    gmMatrix4 operator *(const gmMatrix4&) const;
    gmMatrix4 operator *(double) const;
    gmMatrix4 operator /(double) const;

friend gmMatrix4 operator *(double, const gmMatrix4&);

    bool operator ==(const gmMatrix4&) const;
```

```
bool operator !=(const gmMatrix4&) const;

gmVector4 operator *(const gmVector4&) const;

friend gmVector4 operator *(const gmVector4&, const gmMatrix4&);

// operations

gmMatrix4 inverse() const;
gmMatrix4 transpose() const;
gmMatrix4 adjoint() const;

double determinant() const;
bool isSingular() const;

gmVector3 transform(const gmVector3&) const;

void copyTo(float [4][4]) const;
void copyTo(double [4][4]) const;

// transformation matrices

static gmMatrix4 identity();
static gmMatrix4 rotate(double, const gmVector3& axis);
static gmMatrix4 scale(double, double, double);
static gmMatrix4 translate(double, double, double);

// cubic basis matrices

static gmMatrix4 bezierBasis();
static gmMatrix4 bsplineBasis();
static gmMatrix4 catmullromBasis();
static gmMatrix4 hermiteBasis();

static gmMatrix4 tensedBSplineBasis(double);
static gmMatrix4 cardinalBasis(double);
static gmMatrix4 tauBasis(double, double);
static gmMatrix4 betaSplineBasis(double, double);

};

// ARRAY ACCESS

inline double* gmMatrix4::operator [](int i)
{
    assert(i == 0 || i == 1 || i == 2 || i == 3);
    return &m_[i][0];
}

inline const double* gmMatrix4::operator [](int i) const
{
    assert(i == 0 || i == 1 || i == 2 || i == 3);
    return &m_[i][0];
}

inline void gmMatrix4::copyTo(float f[4][4]) const
{
    f[0][0] = m_[0][0]; f[0][1] = m_[0][1];
    f[0][2] = m_[0][2]; f[0][3] = m_[0][3];
    f[1][0] = m_[1][0]; f[1][1] = m_[1][1];
    f[1][2] = m_[1][2]; f[1][3] = m_[1][3];
```

```
f[2][0] = m_[2][0]; f[2][1] = m_[2][1];
f[2][2] = m_[2][2]; f[2][3] = m_[2][3];
f[3][0] = m_[3][0]; f[3][1] = m_[3][1];
f[3][2] = m_[3][2]; f[3][3] = m_[3][3];
}

inline void gmMatrix4::copyTo(double f[4][4]) const
{
    f[0][0] = m_[0][0]; f[0][1] = m_[0][1];
    f[0][2] = m_[0][2]; f[0][3] = m_[0][3];
    f[1][0] = m_[1][0]; f[1][1] = m_[1][1];
    f[1][2] = m_[1][2]; f[1][3] = m_[1][3];
    f[2][0] = m_[2][0]; f[2][1] = m_[2][1];
    f[2][2] = m_[2][2]; f[2][3] = m_[2][3];
    f[3][0] = m_[3][0]; f[3][1] = m_[3][1];
    f[3][2] = m_[3][2]; f[3][3] = m_[3][3];
}

#endif // GMMATRIX4_H
```



```
// gmUtils.h - graphics math utility functions
//
// libgm++: Graphics Math Library
// Ferdi Scheepers and Stephen F May
// 15 June 1994

#ifndef GMUTILS_H
#define GMUTILS_H

#include <iostream.h>
#include <math.h>
#include <bool.h>
#include <gmConst.h>

// ONE-ARGUMENT UTILITY FUNCTIONS

inline double gmAbs(double f)
{
    return (f >= 0) ? f : -f;
}

inline double gmCeil(double f)
{
    return (f == int(f)) ? f : (f > 0) ? double(int(f) + 1) : double(int(f));
}

inline double gmCube(double f)
{
    return f * f * f;
}

inline double gmDegrees(double f)
{
    return f * gmRADTODEG;
}

inline double gmFloor(double f)
{
    return (f == int(f)) ? f : (f > 0) ? double(int(f)) : double(int(f) - 1);
}

inline double gmInv(double f)
{
    return 1.0 / f;
}

inline bool gmIsZero(double f)
{
    return (gmAbs(f) < gmEPSILON);
}

inline double gmRadians(double f)
{
    return f * gmDEGTORAD;
}

inline double gmRound(double f)
{
    return (f >= 0) ? double(int(f + 0.5)) : double(- int(0.5 - f));
}
```

```
inline double gmSign(double f)
{
    return (f < 0) ? -1.0 : 1.0;
}

inline double gmSmooth(double f)
{
    return (3.0 - 2.0 * f) * f * f;
}

inline double gmSqr(double f)
{
    return f * f;
}

inline double gmTrunc(double f)
{
    return double(int(f));
}

inline double gmZSign(double f)
{
    return (f > 0) ? 1.0 : (f < 0) ? -1.0 : 0.0;
}

// TWO-ARGUMENT UTILITY FUNCTIONS

inline bool gmFuzEQ(double f, double g)
{
    return (f <= g) ? (f >= g - gmEPSILON) : (f <= g + gmEPSILON);
}

inline bool gmFuzGEQ(double f, double g)
{
    return (f >= g - gmEPSILON);
}

inline bool gmFuzLEQ(double f, double g)
{
    return (f <= g + gmEPSILON);
}

inline double gmMax(double f, double g)
{
    return (f > g) ? f : g;
}

inline double gmMin(double f, double g)
{
    return (f < g) ? f : g;
}

inline void gmSwap(double& f, double& g)
{
    double gmTmp = f; f = g; g = gmTmp;
}

inline void gmSwap(int& i, int& j)
{
    int gmTmp = i; i = j; j = gmTmp;
}
```

```
// MULTI-ARGUMENT UTILITY FUNCTIONS

inline void gmClamp(double &f, double l, double h)
{
    if(f < l) f = l;
    if(f > h) f = h;
}

inline double gmLerp(double f, double l, double h)
{
    return l + ((h - l) * f );
}

inline double gmMax(double f, double g, double h)
{
    return (f > g) ? gmMax(f, h) : gmMax(g, h);
}

inline double gmMin(double f, double g, double h)
{
    return (f < g) ? gmMin(f, h) : gmMin(g, h);
}

inline double gmSlide(double f, double l, double h)
{
    return (f < 0) ? l : (f > 1) ? h : gmLerp(gmSmooth(f), l, h);
}

#endif
```

```
// gmVector2.h - 2 element vector class
//
// libgm++: Graphics Math Library
// Ferdi Scheepers and Stephen F May
// 15 June 1994

#ifndef GMVECTOR2_H
#define GMVECTOR2_H

#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <gmUtils.h>

class gmVector2 {
protected:
    double v_[2];

public:
    gmVector2();
    gmVector2(const gmVector2&);
    gmVector2(double, double);

    // array access

    double& operator [](int);
    const double& operator [](int) const;

    // assignment

    gmVector2& assign(double, double);
    gmVector2& operator =(const gmVector2&);

    // math operators

    gmVector2& operator +=(const gmVector2&);
    gmVector2& operator -=(const gmVector2&);
    gmVector2& operator *=(double);
    gmVector2& operator /=(double);

    gmVector2 operator +(const gmVector2&) const;
    gmVector2 operator -(const gmVector2&) const;
    gmVector2 operator -() const;
    gmVector2 operator *(double) const;
    gmVector2 operator /(double) const;

friend gmVector2 operator *(double, const gmVector2&);

    bool operator ==(const gmVector2&) const;
    bool operator !=(const gmVector2&) const;

    // operations

    gmVector2& clamp(double, double);
    double length() const;
    double lengthSquared() const;
    gmVector2& normalize();

    void copyTo(float [2]) const;
    void copyTo(double [2]) const;
```

```
friend double distance(const gmVector2&, const gmVector2&);
friend double distanceSquared(const gmVector2&, const gmVector2&);
friend double dot(const gmVector2&, const gmVector2&);
friend gmVector2 lerp(double, const gmVector2&, const gmVector2&);

// output

friend ostream & operator << ( ostream &, const gmVector2 & );
};

// CONSTRUCTORS

inline gmVector2::gmVector2()
{
    v_[0] = v_[1] = 0;
}

inline gmVector2::gmVector2(const gmVector2& v)
{
    v_[0] = v.v_[0]; v_[1] = v.v_[1];
}

inline gmVector2::gmVector2(double x, double y)
{
    v_[0] = x; v_[1] = y;
}

// ARRAY ACCESS

inline double& gmVector2::operator [](int i)
{
    assert(i == 0 || i == 1);
    return v_[i];
}

inline const double& gmVector2::operator [](int i) const
{
    assert(i == 0 || i == 1);
    return v_[i];
}

// ASSIGNMENT

inline gmVector2& gmVector2::assign(double x, double y)
{
    v_[0] = x; v_[1] = y;
    return *this;
}

inline gmVector2& gmVector2::operator =(const gmVector2& v)
{
    v_[0] = v[0]; v_[1] = v[1];
    return *this;
}

// MATH OPERATORS

inline gmVector2& gmVector2::operator +=(const gmVector2& v)
{
    v_[0] += v[0]; v_[1] += v[1];
}
```

```
    return *this;
}

inline gmVector2& gmVector2::operator -=(const gmVector2& v)
{
    v_[0] -= v[0]; v_[1] -= v[1];
    return *this;
}

inline gmVector2& gmVector2::operator *=(double c)
{
    v_[0] *= c; v_[1] *= c;
    return *this;
}

inline gmVector2& gmVector2::operator /=(double c)
{
    assert(!gmIsZero(c));
    v_[0] /= c; v_[1] /= c;
    return *this;
}

inline gmVector2 gmVector2::operator +(const gmVector2& v) const
{
    return gmVector2(v_[0] + v[0], v_[1] + v[1]);
}

inline gmVector2 gmVector2::operator -(const gmVector2& v) const
{
    return gmVector2(v_[0] - v[0], v_[1] - v[1]);
}

inline gmVector2 gmVector2::operator -() const
{
    return gmVector2(-v_[0], -v_[1]);
}

inline gmVector2 gmVector2::operator *(double c) const
{
    return gmVector2(v_[0] * c, v_[1] * c);
}

inline gmVector2 gmVector2::operator /(double c) const
{
    assert(!gmIsZero(c));
    return gmVector2(v_[0] / c, v_[1] / c);
}

inline gmVector2 operator *(double c, const gmVector2& v)
{
    return gmVector2(c * v[0], c * v[1]);
}

inline bool gmVector2::operator ==(const gmVector2& v) const
{
    return (gmFuzEQ(v_[0], v[0]) && gmFuzEQ(v_[1], v[1]));
}

inline bool gmVector2::operator !=(const gmVector2& v) const
{
    return (!(*this == v));
}
```

```
}

// OPERATIONS

inline gmVector2& gmVector2::clamp(double lo, double hi)
{
    gmClamp(v_[0], lo, hi); gmClamp(v_[1], lo, hi);
    return *this;
}

inline double gmVector2::length() const
{
    return sqrt(gmSqr(v_[0]) + gmSqr(v_[1]));
}

inline double gmVector2::lengthSquared() const
{
    return gmSqr(v_[0]) + gmSqr(v_[1]);
}

inline gmVector2& gmVector2::normalize()
{
    double len = length();
    assert(!gmIsZero(len));
    *this /= len;
    return *this;
}

inline void gmVector2::copyTo(float f[2]) const
{
    f[0] = v_[0]; f[1] = v_[1];
}

inline void gmVector2::copyTo(double f[2]) const
{
    f[0] = v_[0]; f[1] = v_[1];
}

inline double distance(const gmVector2& v1, const gmVector2& v2)
{
    return sqrt(gmSqr(v1[0] - v2[0]) + gmSqr(v1[1] - v2[1]));
}

inline double distanceSquared(const gmVector2& v1, const gmVector2& v2)
{
    return gmSqr(v1[0] - v2[0]) + gmSqr(v1[1] - v2[1]);
}

inline double dot(const gmVector2& v1, const gmVector2& v2)
{
    return v1[0] * v2[0] + v1[1] * v2[1];
}

inline gmVector2 lerp(double f, const gmVector2& v1, const gmVector2& v2)
{
    return v1 + ((v2 - v1) * f);
}

// OUTPUT

inline ostream & operator << ( ostream& os, const gmVector2& v)
```

```
{
    os << "< " << v[0] << " " << v[1] << " >";
    return os;
}

#endif
```



```
// gmVector3.h - 3 element vector class
//
// libgm++: Graphics Math Library
// Ferdi Scheepers and Stephen F May
// 15 June 1994

#ifndef GMVECTOR3_H
#define GMVECTOR3_H

#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <gmUtils.h>

class gmVector3 {

protected:
    double v_[3];

public:
    gmVector3();
    gmVector3(const gmVector3&);
    gmVector3(double, double, double);

    // array access

    double& operator [](int);
    const double& operator [](int) const;

    // assignment

    gmVector3& assign(double, double, double);
    gmVector3& operator =(const gmVector3&);

    // math operators

    gmVector3& operator +=(const gmVector3&);
    gmVector3& operator -=(const gmVector3&);
    gmVector3& operator *=(double);
    gmVector3& operator /=(double);

    gmVector3 operator +(const gmVector3&) const;
    gmVector3 operator -(const gmVector3&) const;
    gmVector3 operator -() const;
    gmVector3 operator *(double) const;
    gmVector3 operator /(double) const;

friend gmVector3 operator *(double, const gmVector3&);

    bool operator ==(const gmVector3&) const;
    bool operator !=(const gmVector3&) const;

    // operations

    gmVector3& clamp(double, double);
    double length() const;
    double lengthSquared() const;
    gmVector3& normalize();

    void copyTo(float [3]) const;
    void copyTo(double [3]) const;
```

```
friend gmVector3 cross(const gmVector3&, const gmVector3&);
friend double distance(const gmVector3&, const gmVector3&);
friend double distanceSquared(const gmVector3&, const gmVector3&);
friend double dot(const gmVector3&, const gmVector3&);
friend gmVector3 lerp(double, const gmVector3&, const gmVector3&);

// output

friend ostream & operator << ( ostream &, const gmVector3 & );
};

// CONSTRUCTORS

inline gmVector3::gmVector3()
{
    v_[0] = v_[1] = v_[2] = 0;
}

inline gmVector3::gmVector3(const gmVector3& v)
{
    v_[0] = v.v_[0]; v_[1] = v.v_[1]; v_[2] = v.v_[2];
}

inline gmVector3::gmVector3(double x, double y, double z)
{
    v_[0] = x; v_[1] = y; v_[2] = z;
}

// ARRAY ACCESS

inline double& gmVector3::operator [](int i)
{
    assert(i == 0 || i == 1 || i == 2);
    return v_[i];
}

inline const double& gmVector3::operator [](int i) const
{
    assert(i == 0 || i == 1 || i == 2);
    return v_[i];
}

// ASSIGNMENT

inline gmVector3& gmVector3::assign(double x, double y, double z)
{
    v_[0] = x; v_[1] = y; v_[2] = z;
    return *this;
}

inline gmVector3& gmVector3::operator =(const gmVector3& v)
{
    v_[0] = v[0]; v_[1] = v[1]; v_[2] = v[2];
    return *this;
}

// MATH OPERATORS

inline gmVector3& gmVector3::operator +=(const gmVector3& v)
{

```

```
    v_[0] += v[0]; v_[1] += v[1]; v_[2] += v[2];
    return *this;
}

inline gmVector3& gmVector3::operator -=(const gmVector3& v)
{
    v_[0] -= v[0]; v_[1] -= v[1]; v_[2] -= v[2];
    return *this;
}

inline gmVector3& gmVector3::operator *=(double c)
{
    v_[0] *= c; v_[1] *= c; v_[2] *= c;
    return *this;
}

inline gmVector3& gmVector3::operator /=(double c)
{
    assert(!gmIsZero(c));
    v_[0] /= c; v_[1] /= c; v_[2] /= c;
    return *this;
}

inline gmVector3 gmVector3::operator +(const gmVector3& v) const
{
    return gmVector3(v_[0] + v[0], v_[1] + v[1], v_[2] + v[2]);
}

inline gmVector3 gmVector3::operator -(const gmVector3& v) const
{
    return gmVector3(v_[0] - v[0], v_[1] - v[1], v_[2] - v[2]);
}

inline gmVector3 gmVector3::operator -() const
{
    return gmVector3(-v_[0], -v_[1], -v_[2]);
}

inline gmVector3 gmVector3::operator *(double c) const
{
    return gmVector3(v_[0] * c, v_[1] * c, v_[2] * c);
}

inline gmVector3 gmVector3::operator /(double c) const
{
    assert(!gmIsZero(c));
    return gmVector3(v_[0] / c, v_[1] / c, v_[2] / c);
}

inline gmVector3 operator *(double c, const gmVector3& v)
{
    return gmVector3(c * v[0], c * v[1], c * v[2]);
}

inline bool gmVector3::operator ==(const gmVector3& v) const
{
    return
        (gmFuzEQ(v_[0], v[0]) && gmFuzEQ(v_[1], v[1]) && gmFuzEQ(v_[2], v[2]));
}

inline bool gmVector3::operator !=(const gmVector3& v) const
```

```
{
    return (!(*this == v));
}

// OPERATIONS

inline gmVector3& gmVector3::clamp(double lo, double hi)
{
    gmClamp(v_[0], lo, hi); gmClamp(v_[1], lo, hi); gmClamp(v_[2], lo, hi);
    return *this;
}

inline double gmVector3::length() const
{
    return sqrt(gmSqr(v_[0]) + gmSqr(v_[1]) + gmSqr(v_[2]));
}

inline double gmVector3::lengthSquared() const
{
    return gmSqr(v_[0]) + gmSqr(v_[1]) + gmSqr(v_[2]);
}

inline gmVector3& gmVector3::normalize()
{
    double len = length();
    assert(!gmIsZero(len));
    *this /= len;
    return *this;
}

inline void gmVector3::copyTo(float f[3]) const
{
    f[0] = v_[0]; f[1] = v_[1]; f[2] = v_[2];
}

inline void gmVector3::copyTo(double f[3]) const
{
    f[0] = v_[0]; f[1] = v_[1]; f[2] = v_[2];
}

inline gmVector3 cross(const gmVector3& v1, const gmVector3& v2)
{
    return gmVector3(v1[1] * v2[2] - v1[2] * v2[1],
                     v1[2] * v2[0] - v1[0] * v2[2],
                     v1[0] * v2[1] - v1[1] * v2[0]);
}

inline double distance(const gmVector3& v1, const gmVector3& v2)
{
    return
        sqrt(gmSqr(v1[0] - v2[0]) + gmSqr(v1[1] - v2[1]) + gmSqr(v1[2] - v2[2]));
}

inline double distanceSquared(const gmVector3& v1, const gmVector3& v2)
{
    return gmSqr(v1[0] - v2[0]) + gmSqr(v1[1] - v2[1]) + gmSqr(v1[2] - v2[2]);
}

inline double dot(const gmVector3& v1, const gmVector3& v2)
{
    return v1[0] * v2[0] + v1[1] * v2[1] + v1[2] * v2[2];
}
```

```
}

inline gmVector3 lerp(double f, const gmVector3& v1, const gmVector3& v2)
{
    return v1 + ((v2 - v1) * f);
}

// OUTPUT

inline ostream & operator << ( ostream& os, const gmVector3& v)
{
    os << "< " << v[0] << " " << v[1] << " " << v[2] << " >";
    return os;
}

#endif
```

```
// gmVector4.h - 4 element vector class
//
// libgm++: Graphics Math Library
// Ferdi Scheepers and Stephen F May
// 15 June 1994

#ifndef GMVECTOR4_H
#define GMVECTOR4_H

#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <gmUtils.h>

class gmVector4 {
protected:
    double v_[4];

public:
    gmVector4();
    gmVector4(const gmVector4&);
    gmVector4(double, double, double, double);

    // array access

    double& operator [](int);
    const double& operator [](int) const;

    // assignment

    gmVector4& assign(double, double, double, double);
    gmVector4& operator =(const gmVector4&);

    // math operators

    gmVector4& operator +=(const gmVector4&);
    gmVector4& operator -=(const gmVector4&);
    gmVector4& operator *=(double);
    gmVector4& operator /=(double);

    gmVector4 operator +(const gmVector4&) const;
    gmVector4 operator -(const gmVector4&) const;
    gmVector4 operator -() const;
    gmVector4 operator *(double) const;
    gmVector4 operator /(double) const;

friend gmVector4 operator *(double, const gmVector4&);

    bool operator ==(const gmVector4&) const;
    bool operator !=(const gmVector4&) const;

    // operations

    gmVector4& clamp(double, double);
    double length() const;
    double lengthSquared() const;
    gmVector4& normalize();

    void copyTo(float [4]) const;
    void copyTo(double [4]) const;
```

```
friend double distance(const gmVector4&, const gmVector4&);
friend double distanceSquared(const gmVector4&, const gmVector4&);
friend double dot(const gmVector4&, const gmVector4&);
friend gmVector4 lerp(double, const gmVector4&, const gmVector4&);

// output

friend ostream & operator << ( ostream &, const gmVector4 & );
};

// CONSTRUCTORS

inline gmVector4::gmVector4()
{
    v_[0] = v_[1] = v_[2] = v_[3] = 0;
}

inline gmVector4::gmVector4(const gmVector4& v)
{
    v_[0] = v.v_[0]; v_[1] = v.v_[1]; v_[2] = v.v_[2]; v_[3] = v.v_[3];
}

inline gmVector4::gmVector4(double x, double y, double z, double a)
{
    v_[0] = x; v_[1] = y; v_[2] = z; v_[3] = a;
}

// ARRAY ACCESS

inline double& gmVector4::operator [](int i)
{
    assert(i == 0 || i == 1 || i == 2 || i == 3);
    return v_[i];
}

inline const double& gmVector4::operator [](int i) const
{
    assert(i == 0 || i == 1 || i == 2 || i == 3);
    return v_[i];
}

// ASSIGNMENT

inline gmVector4& gmVector4::assign(double x, double y, double z, double a)
{
    v_[0] = x; v_[1] = y; v_[2] = z; v_[3] = a;
    return *this;
}

inline gmVector4& gmVector4::operator =(const gmVector4& v)
{
    v_[0] = v[0]; v_[1] = v[1]; v_[2] = v[2]; v_[3] = v[3];
    return *this;
}

// MATH OPERATORS

inline gmVector4& gmVector4::operator +=(const gmVector4& v)
{
    v_[0] += v[0]; v_[1] += v[1]; v_[2] += v[2]; v_[3] += v[3];
}
```

```
    return *this;
}

inline gmVector4& gmVector4::operator -=(const gmVector4& v)
{
    v_[0] -= v[0]; v_[1] -= v[1]; v_[2] -= v[2]; v_[3] -= v[3];
    return *this;
}

inline gmVector4& gmVector4::operator *=(double c)
{
    v_[0] *= c; v_[1] *= c; v_[2] *= c; v_[3] *= c;
    return *this;
}

inline gmVector4& gmVector4::operator /=(double c)
{
    assert(!gmIsZero(c));
    v_[0] /= c; v_[1] /= c; v_[2] /= c; v_[3] /= c;
    return *this;
}

inline gmVector4 gmVector4::operator +(const gmVector4& v) const
{
    return gmVector4(v_[0] + v[0], v_[1] + v[1], v_[2] + v[2], v_[3] + v[3]);
}

inline gmVector4 gmVector4::operator -(const gmVector4& v) const
{
    return gmVector4(v_[0] - v[0], v_[1] - v[1], v_[2] - v[2], v_[3] - v[3]);
}

inline gmVector4 gmVector4::operator -() const
{
    return gmVector4(-v_[0], -v_[1], -v_[2], -v_[3]);
}

inline gmVector4 gmVector4::operator *(double c) const
{
    return gmVector4(v_[0] * c, v_[1] * c, v_[2] * c, v_[3] * c);
}

inline gmVector4 gmVector4::operator /(double c) const
{
    assert(!gmIsZero(c));
    return gmVector4(v_[0] / c, v_[1] / c, v_[2] / c, v_[3] / c);
}

inline gmVector4 operator *(double c, const gmVector4& v)
{
    return gmVector4(c * v[0], c * v[1], c * v[2], c * v[3]);
}

inline bool gmVector4::operator ==(const gmVector4& v) const
{
    return (gmFuzEQ(v_[0], v[0]) && gmFuzEQ(v_[1], v[1]) &&
            gmFuzEQ(v_[2], v[2]) && gmFuzEQ(v_[3], v[3]));
}

inline bool gmVector4::operator !=(const gmVector4& v) const
{

```



```
    return (!(*this == v));
}

// OPERATIONS

inline gmVector4& gmVector4::clamp(double lo, double hi)
{
    gmClamp(v_[0], lo, hi); gmClamp(v_[1], lo, hi);
    gmClamp(v_[2], lo, hi); gmClamp(v_[3], lo, hi);
    return *this;
}

inline double gmVector4::length() const
{
    return sqrt(gmSqr(v_[0]) + gmSqr(v_[1]) + gmSqr(v_[2]) + gmSqr(v_[3]));
}

inline double gmVector4::lengthSquared() const
{
    return gmSqr(v_[0]) + gmSqr(v_[1]) + gmSqr(v_[2]) + gmSqr(v_[3]);
}

inline gmVector4& gmVector4::normalize()
{
    double len = length();
    assert(!gmIsZero(len));
    *this /= len;
    return *this;
}

inline void gmVector4::copyTo(float f[4]) const
{
    f[0] = v_[0]; f[1] = v_[1]; f[2] = v_[2]; f[3] = v_[3];
}

inline void gmVector4::copyTo(double f[4]) const
{
    f[0] = v_[0]; f[1] = v_[1]; f[2] = v_[2]; f[3] = v_[3];
}

inline double distance(const gmVector4& v1, const gmVector4& v2)
{
    return sqrt(gmSqr(v1[0] - v2[0]) + gmSqr(v1[1] - v2[1]) +
                gmSqr(v1[2] - v2[2]) + gmSqr(v1[3] - v2[3]));
}

inline double distanceSquared(const gmVector4& v1, const gmVector4& v2)
{
    return gmSqr(v1[0] - v2[0]) + gmSqr(v1[1] - v2[1]) +
        gmSqr(v1[2] - v2[2]) + gmSqr(v1[3] - v2[3]);
}

inline double dot(const gmVector4& v1, const gmVector4& v2)
{
    return v1[0] * v2[0] + v1[1] * v2[1] + v1[2] * v2[2] + v1[3] * v2[3];
}

inline gmVector4 lerp(double f, const gmVector4& v1, const gmVector4& v2)
{
    return v1 + ((v2 - v1) * f);
}
```

```
// OUTPUT
```

```
inline ostream & operator << ( ostream& os, const gmVector4& v)
{
    os << "< " << v[0] << " " << v[1] << " " << v[2] << " " << v[3] << " >";
    return os;
}

#endif
```

```
/* ----- */
MAT2.H :

Definition and manipulation of a 2D matrix (either integers or reals)

by Christophe Schlick (1 June 1992)
/* ----- */

#ifndef _MAT2_
#define _MAT2_

#include "tool.h"
#include "vec2.h"

/*
** Creation, duplication, swapping
*/

#define MAKE_MAT2(M,A,B)\
    (COPY_VEC2 ((M)[0],(A)),\
     COPY_VEC2 ((M)[1],(B)))

#define COPY_MAT2(M,A)\
    (COPY_VEC2 ((M)[0],(A)[0]),\
     COPY_VEC2 ((M)[1],(A)[1]))

#define SWAP_MAT2(A,B,t)\
    (SWAP_VEC2 ((A)[0], (B)[0], t),\
     SWAP_VEC2 ((A)[1], (B)[1], t))

/*
** Addition, subtraction, multiplication, division (by a matrix element)
*/

#define INC_MAT2(M,A)\
    (INC_VEC2 ((M)[0],(A)[0]),\
     INC_VEC2 ((M)[1],(A)[1]))

#define DEC_MAT2(M,A)\
    (DEC_VEC2 ((M)[0],(A)[0]),\
     DEC_VEC2 ((M)[1],(A)[1]))

#define ADD_MAT2(M,A,B)\
    (ADD_VEC2 ((M)[0],(A)[0],(B)[0]),\
     ADD_VEC2 ((M)[1],(A)[1],(B)[1]))

#define SUB_MAT2(M,A,B)\
    (SUB_VEC2 ((M)[0],(A)[0],(B)[0]),\
     SUB_VEC2 ((M)[1],(A)[1],(B)[1]))

#define MUL_MAT2(M,A,B)\
    (MUL_VEC2 ((M)[0],(A)[0],(B)[0]),\
     MUL_VEC2 ((M)[1],(A)[1],(B)[1]))

#define DIV_MAT2(M,A,B)\
    (DIV_VEC2 ((M)[0],(A)[0],(B)[0]),\
     DIV_VEC2 ((M)[1],(A)[1],(B)[1]))

/*
** Addition, subtraction, multiplication, division (by a scalar element)
*/
```

```
#define ADDS_MAT2(M,A,s)\
    (ADDS_VEC2 ((M)[0],(A)[0],s),\
     ADDS_VEC2 ((M)[1],(A)[1],s))

#define SUBS_MAT2(M,A,B)\
    (SUBS_VEC2 ((M)[0],(A)[0],s),\
     SUBS_VEC2 ((M)[1],(A)[1],s))

#define MULS_MAT2(M,A,B)\
    (MULS_VEC2 ((M)[0],(A)[0],s),\
     MULS_VEC2 ((M)[1],(A)[1],s))

#define DIVS_MAT2(M,A,B)\
    (DIVS_VEC2 ((M)[0],(A)[0],s),\
     DIVS_VEC2 ((M)[1],(A)[1],s))

/*
** Determinant, transposition, adjunction, inversion
*/

#define DELTA_MAT2(M)\
    (DELTA_VEC2 ((M)[0],(M)[1]))

#define TRANS_MAT2(M,A)\
    ((M)[0].x = (A)[0].x, (M)[0].y = (A)[1].x,\
     (M)[1].x = (A)[0].y, (M)[1].y = (A)[1].y)

#define ADJ_MAT2(M,A)\
    ((M)[0].x = (A)[1].y, (M)[0].y = -(A)[0].y,\
     (M)[1].x = -(A)[1].x, (M)[1].y = (A)[0].x)

#define INV_MAT2(M,A,s)\
    (ADJ_MAT2 (M,A), (s) = DOT_VEC2 ((M)[0],(A)[0]),\
     ZERO (s) ? (DIVS_MAT2 (M,M,s), TRUE) : FALSE)

/*
** Matrix product, left vector product, right vector product
*/

#define ROW_VEC2(V,M,n)\
    ((V).x*(M)[0].n + (V).y*(M)[1].n)

#define PROD_MAT2(M,A,B)\
    ((M)[0].x = ROW_VEC2 ((A)[0],(B),x),\
     (M)[0].y = ROW_VEC2 ((A)[0],(B),y),\
     (M)[1].x = ROW_VEC2 ((A)[1],(B),x),\
     (M)[1].y = ROW_VEC2 ((A)[1],(B),y))

#define LMAT_VEC2(V,A,M)\
    ((V).x = ROW_VEC2 ((A),(M),x),\
     (V).y = ROW_VEC2 ((A),(M),y))

#define RMAT_VEC2(V,M,A)\
    ((V).x = DOT_VEC2 ((A),(M)[0]),\
     (V).y = DOT_VEC2 ((A),(M)[1]))

/*
** MAKE_FRAME2(F,O,A,B,s) = Create a local frame F where O is the origin,
**                          (A,B) are the axis and 's' a dummy real variable
** LOCAL_FRAME2(V,F,A) = Transform A from world frame into local frame F
*/
```

```
** WORLD_FRAME2(V,F,A) = Transform A from local frame F into world frame
*/
```

```
#define MAKE_FRAME2(F,O,A,B,s)\
    (COPY_VEC2 ((F)[0],(A)),\
    COPY_VEC2 ((F)[1],(B)),\
    COPY_VEC2 ((F)[2],(O)),\
    INV_MAT2 ((F)+6,(F),s))
```

```
#define LOCAL_FRAME2(V,F,A)\
    (DEC_VEC2 ((A),(F)[2]),\
    LMAT_VEC2 ((V),(A),(F)+6),\
    INC_VEC2 ((A),(F)[2]))
```

```
#define WORLD_FRAME2(V,F,A)\
    (LMAT_VEC2 ((V),(A),(F)),\
    INC_VEC2 ((V),(F)[2]))
```

```
#endif
```

```
/* ----- */
```

```
/* ----- */
MAT3.H :

Definition and manipulation of a 3D matrix (either integers or reals)

by Christophe Schlick (1 June 1992)
/* ----- */

#ifndef _MAT3_
#define _MAT3_

#include "tool.h"
#include "vec3.h"

/*
** Creation, duplication, swapping
*/

#define MAKE_MAT3(M,A,B,C)\
    (COPY_VEC3 ((M)[0],(A)),\
     COPY_VEC3 ((M)[1],(B)),\
     COPY_VEC3 ((M)[2],(C)))

#define COPY_MAT3(M,A)\
    (COPY_VEC3 ((M)[0],(A)[0]),\
     COPY_VEC3 ((M)[1],(A)[1]),\
     COPY_VEC3 ((M)[2],(A)[2]))

#define SWAP_MAT3(A,B,t)\
    (SWAP_VEC3 ((A)[0], (B)[0], t),\
     SWAP_VEC3 ((A)[1], (B)[1], t),\
     SWAP_VEC3 ((A)[2], (B)[2], t))

/*
** Addition, subtraction, multiplication, division (by a matrix element)
*/

#define INC_MAT3(M,A)\
    (INC_VEC3 ((M)[0],(A)[0]),\
     INC_VEC3 ((M)[1],(A)[1]),\
     INC_VEC3 ((M)[2],(A)[2]))

#define DEC_MAT3(M,A)\
    (DEC_VEC3 ((M)[0],(A)[0]),\
     DEC_VEC3 ((M)[1],(A)[1]),\
     DEC_VEC3 ((M)[2],(A)[2]))

#define ADD_MAT3(M,A,B)\
    (ADD_VEC3 ((M)[0],(A)[0],(B)[0]),\
     ADD_VEC3 ((M)[1],(A)[1],(B)[1]),\
     ADD_VEC3 ((M)[2],(A)[2],(B)[2]))

#define SUB_MAT3(M,A,B)\
    (SUB_VEC3 ((M)[0],(A)[0],(B)[0]),\
     SUB_VEC3 ((M)[1],(A)[1],(B)[1]),\
     SUB_VEC3 ((M)[2],(A)[2],(B)[2]))

#define MUL_MAT3(M,A,B)\
    (MUL_VEC3 ((M)[0],(A)[0],(B)[0]),\
     MUL_VEC3 ((M)[1],(A)[1],(B)[1]),\
     MUL_VEC3 ((M)[2],(A)[2],(B)[2]))
```

```
#define DIV_MAT3(M,A,B)\
    (DIV_VEC3 ((M)[0],(A)[0],(B)[0]),\
     DIV_VEC3 ((M)[1],(A)[1],(B)[1]),\
     DIV_VEC3 ((M)[2],(A)[2],(B)[2]))

/*
** Addition, subtraction, multiplication, division (by a scalar element)
*/

#define ADDS_MAT3(M,A,s)\
    (ADDS_VEC3 ((M)[0],(A)[0],s),\
     ADDS_VEC3 ((M)[1],(A)[1],s),\
     ADDS_VEC3 ((M)[2],(A)[2],s))

#define SUBS_MAT3(M,A,B)\
    (SUBS_VEC3 ((M)[0],(A)[0],s),\
     SUBS_VEC3 ((M)[1],(A)[1],s),\
     SUBS_VEC3 ((M)[2],(A)[2],s))

#define MULS_MAT3(M,A,B)\
    (MULS_VEC3 ((M)[0],(A)[0],s),\
     MULS_VEC3 ((M)[1],(A)[1],s),\
     MULS_VEC3 ((M)[2],(A)[2],s))

#define DIVS_MAT3(M,A,B)\
    (DIVS_VEC3 ((M)[0],(A)[0],s),\
     DIVS_VEC3 ((M)[1],(A)[1],s),\
     DIVS_VEC3 ((M)[2],(A)[2],s))

/*
** Determinant, transposition, adjunction, inversion
*/

#define DELTA_MAT3(M)\
    (DELTA_VEC3 ((M)[0],(M)[1],(M)[2]))

#define TRANS_MAT3(M,A)\
    ((M)[0].x = (A)[0].x, (M)[0].y = (A)[1].x, (M)[0].z = (A)[2].x,\
     (M)[1].x = (A)[0].y, (M)[1].y = (A)[1].y, (M)[1].z = (A)[2].y,\
     (M)[2].x = (A)[0].z, (M)[2].y = (A)[1].z, (M)[2].z = (A)[2].z)

#define ADJ_MAT3(M,A)\
    ((M)[0].x = (A)[1].y * (A)[2].z - (A)[1].z * (A)[2].y,\
     (M)[1].x = (A)[2].y * (A)[0].z - (A)[2].z * (A)[0].y,\
     (M)[2].x = (A)[0].y * (A)[1].z - (A)[0].z * (A)[1].y,\
     (M)[0].y = (A)[1].z * (A)[2].x - (A)[1].x * (A)[2].z,\
     (M)[1].y = (A)[2].z * (A)[0].x - (A)[2].x * (A)[0].z,\
     (M)[2].y = (A)[0].z * (A)[1].x - (A)[0].x * (A)[1].z,\
     (M)[0].z = (A)[1].x * (A)[2].y - (A)[1].y * (A)[2].x,\
     (M)[1].z = (A)[2].x * (A)[0].y - (A)[2].y * (A)[0].x,\
     (M)[2].z = (A)[0].x * (A)[1].y - (A)[0].y * (A)[1].x)

#define INV_MAT3(M,A,s)\
    (ADJ_MAT3 (M,A), (s) = DOT_VEC3((M)[0],(A)[0]),\
     ZERO (s) ? (DIVS_MAT3 (M,M,s), TRUE) : FALSE)

/*
** Matrix product, left vector product, right vector product
*/
```

```
#define ROW_VEC3(V,M,n)\
    ((V).x*(M)[0].n + (V).y*(M)[1].n + (V).z*(M)[2].n)

#define PROD_MAT3(M,A,B)\
    ((M)[0].x = ROW_VEC3 ((A)[0],(B),x),\
    (M)[0].y = ROW_VEC3 ((A)[0],(B),y),\
    (M)[0].z = ROW_VEC3 ((A)[0],(B),z),\
    (M)[1].x = ROW_VEC3 ((A)[1],(B),x),\
    (M)[1].y = ROW_VEC3 ((A)[1],(B),y),\
    (M)[1].z = ROW_VEC3 ((A)[1],(B),z),\
    (M)[2].x = ROW_VEC3 ((A)[2],(B),x),\
    (M)[2].y = ROW_VEC3 ((A)[2],(B),y),\
    (M)[2].z = ROW_VEC3 ((A)[2],(B),z))

#define LMAT_VEC3(V,A,M)\
    ((V).x = ROW_VEC3 ((A),(M),x),\
    (V).y = ROW_VEC3 ((A),(M),y),\
    (V).z = ROW_VEC3 ((A),(M),z))

#define RMAT_VEC3(V,M,A)\
    ((V).x = DOT_VEC3 ((A),(M)[0]),\
    (V).y = DOT_VEC3 ((A),(M)[1]),\
    (V).z = DOT_VEC3 ((A),(M)[2]))

/*
** MAKE_FRAME3(F,O,A,B,C,s) = Create a local frame F where O is the origin,
**                          (A,B,C) are the axis and 's' a dummy real variable
** LOCAL_FRAME3(V,F,A) = Transform A from world frame into local frame F
** WORLD_FRAME3(V,F,A) = Transform A from local frame F into world frame
*/

#define MAKE_FRAME3(F,O,A,B,C,s)\
    (COPY_VEC3 ((F)[0],(A)),\
    COPY_VEC3 ((F)[1],(B)),\
    COPY_VEC3 ((F)[2],(C)),\
    COPY_VEC3 ((F)[3],(O)),\
    INV_MAT3 ((F)+12,(F),s))

#define LOCAL_FRAME3(V,F,A)\
    (DEC_VEC3 ((A),(F)[3]),\
    LMAT_VEC3 ((V),(A),(F)+12),\
    INC_VEC3 ((A),(F)[3]))

#define WORLD_FRAME3(V,F,A)\
    (LMAT_VEC3 ((V),(A),(F)),\
    INC_VEC3 ((V),(F)[3]))

#endif

/* ----- */
```



```
/* ----- */
MAT4.H :

Definition and manipulation of a 4D matrix (either integers or reals)

by Christophe Schlick (1 June 1992)
/* ----- */

#ifndef _MAT4_
#define _MAT4_

#include "tool.h"
#include "vec4.h"

/*
** Creation, duplication, swapping
*/

#define MAKE_MAT4(M,A,B,C,D)\
    (COPY_VEC4 ((M)[0],(A)),\
     COPY_VEC4 ((M)[1],(B)),\
     COPY_VEC4 ((M)[2],(C)),\
     COPY_VEC4 ((M)[3],(D)))

#define COPY_MAT4(M,A)\
    (COPY_VEC4 ((M)[0],(A)[0]),\
     COPY_VEC4 ((M)[1],(A)[1]),\
     COPY_VEC4 ((M)[2],(A)[2]),\
     COPY_VEC4 ((M)[3],(A)[3]))

#define SWAP_MAT4(A,B,t)\
    (SWAP_VEC4 ((A)[0], (B)[0], t),\
     SWAP_VEC4 ((A)[1], (B)[1], t),\
     SWAP_VEC4 ((A)[2], (B)[2], t),\
     SWAP_VEC4 ((A)[3], (B)[3], t))

/*
** Addition, subtraction, multiplication, division (by a matrix element)
*/

#define INC_MAT4(M,A)\
    (INC_VEC4 ((M)[0],(A)[0]),\
     INC_VEC4 ((M)[1],(A)[1]),\
     INC_VEC4 ((M)[2],(A)[2]),\
     INC_VEC4 ((M)[3],(A)[3]))

#define DEC_MAT4(M,A)\
    (DEC_VEC4 ((M)[0],(A)[0]),\
     DEC_VEC4 ((M)[1],(A)[1]),\
     DEC_VEC4 ((M)[2],(A)[2]),\
     DEC_VEC4 ((M)[3],(A)[3]))

#define ADD_MAT4(M,A,B)\
    (ADD_VEC4 ((M)[0],(A)[0],(B)[0]),\
     ADD_VEC4 ((M)[1],(A)[1],(B)[1]),\
     ADD_VEC4 ((M)[2],(A)[2],(B)[2]),\
     ADD_VEC4 ((M)[3],(A)[3],(B)[3]))

#define SUB_MAT4(V,A,B)\
    (SUB_VEC4 ((M)[0],(A)[0],(B)[0]),\
     SUB_VEC4 ((M)[1],(A)[1],(B)[1]),\

```

```
SUB_VEC4 ((M)[2],(A)[2],(B)[2]),\
SUB_VEC4 ((M)[3],(A)[3],(B)[3]))
```

```
#define MUL_MAT4(V,A,B)\
(MUL_VEC4 ((M)[0],(A)[0],(B)[0]),\
MUL_VEC4 ((M)[1],(A)[1],(B)[1]),\
MUL_VEC4 ((M)[2],(A)[2],(B)[2]),\
MUL_VEC4 ((M)[3],(A)[3],(B)[3]))
```

```
#define DIV_MAT4(V,A,B)\
(DIV_VEC4 ((M)[0],(A)[0],(B)[0]),\
DIV_VEC4 ((M)[1],(A)[1],(B)[1]),\
DIV_VEC4 ((M)[2],(A)[2],(B)[2]),\
DIV_VEC4 ((M)[3],(A)[3],(B)[3]))
```

```
/*
** Addition, subtraction, multiplication, division (by a scalar element)
*/
```

```
#define ADDS_MAT4(M,A,s)\
(ADDS_VEC4 ((M)[0],(A)[0],s),\
ADDS_VEC4 ((M)[1],(A)[1],s),\
ADDS_VEC4 ((M)[2],(A)[2],s),\
ADDS_VEC4 ((M)[3],(A)[3],s))
```

```
#define SUBS_MAT4(V,A,B)\
(SUBS_VEC4 ((M)[0],(A)[0],s),\
SUBS_VEC4 ((M)[1],(A)[1],s),\
SUBS_VEC4 ((M)[2],(A)[2],s),\
SUBS_VEC4 ((M)[3],(A)[3],s))
```

```
#define MULS_MAT4(V,A,B)\
(MULS_VEC4 ((M)[0],(A)[0],s),\
MULS_VEC4 ((M)[1],(A)[1],s),\
MULS_VEC4 ((M)[2],(A)[2],s),\
MULS_VEC4 ((M)[3],(A)[3],s))
```

```
#define DIVS_MAT4(V,A,B)\
(DIVS_VEC4 ((M)[0],(A)[0],s),\
DIVS_VEC4 ((M)[1],(A)[1],s),\
DIVS_VEC4 ((M)[2],(A)[2],s),\
DIVS_VEC4 ((M)[3],(A)[3],s))
```

```
/*
** Determinant, transposition, adjunction, inversion --> YET TO DO
*/
```

```
/*
#define DELTA_MAT4(M)
```

```
#define TRANS_MAT4(M,A)
```

```
#define ADJ_MAT4(M,A)
```

```
#define INV_MAT4(M,A,s)
*/
```

```
/*
** Matrix product, left vector product, right vector product
*/
```

```
#define ROW_VEC4(M,n,V)\
    ((M)[0].n*(V).x + (M)[1].n*(V).y + (M)[2].n*(V).z + (M)[3].n*(V).w)

#define PROD_MAT4(M,A,B)\
    ((M)[0].x = ROW_VEC4 ((A),x,(B)[0]),\
    (M)[0].y = ROW_VEC4 ((A),y,(B)[0]),\
    (M)[0].z = ROW_VEC4 ((A),z,(B)[0]),\
    (M)[0].w = ROW_VEC4 ((A),w,(B)[0]),\
    (M)[1].x = ROW_VEC4 ((A),x,(B)[1]),\
    (M)[1].y = ROW_VEC4 ((A),y,(B)[1]),\
    (M)[1].z = ROW_VEC4 ((A),z,(B)[1]),\
    (M)[1].w = ROW_VEC4 ((A),w,(B)[1]),\
    (M)[2].x = ROW_VEC4 ((A),x,(B)[2]),\
    (M)[2].y = ROW_VEC4 ((A),y,(B)[2]),\
    (M)[2].z = ROW_VEC4 ((A),z,(B)[2]),\
    (M)[2].w = ROW_VEC4 ((A),w,(B)[2]),\
    (M)[3].x = ROW_VEC4 ((A),x,(B)[3]),\
    (M)[3].y = ROW_VEC4 ((A),y,(B)[3]),\
    (M)[3].z = ROW_VEC4 ((A),z,(B)[3]),\
    (M)[3].w = ROW_VEC4 ((A),w,(B)[3]))

#define LMAT_VEC4(V,A,M)\
    ((V).x = ROW_VEC4 ((M),x,(A)),\
    (V).y = ROW_VEC4 ((M),y,(A)),\
    (V).z = ROW_VEC4 ((M),z,(A)),\
    (V).w = ROW_VEC4 ((M),w,(A)))

#define RMAT_VEC4(V,M,A)\
    ((V).x = DOT_VEC4 ((A),(M)[0]),\
    (V).y = DOT_VEC4 ((A),(M)[1]),\
    (V).z = DOT_VEC4 ((A),(M)[2]),\
    (V).w = DOT_VEC4 ((A),(M)[3]))

#endif

/* ----- */
```

```
/* ----- */
REAL.H :

Definition of a real number type (and related types)

by Christophe Schlick (1 June 1992)
/* ----- */

#ifndef _REAL_
#define _REAL_

#include "tool.h"

/*
** By default, real numbers are defined in double precision (ie double)
** To get single precision, simply add "#define SINGLE_REAL" in your program
*/

/*
** Scalar type definition (single or double precision)
*/

#ifdef SINGLE_REAL

typedef float real;

#define REAL1FILE "%g\n"
#define REAL2FILE "%g %g\n"
#define REAL3FILE "%g %g %g\n"
#define REAL4FILE "%g %g %g %g\n"

#else

typedef double real;

#define REAL1FILE "%lg\n"
#define REAL2FILE "%lg %lg\n"
#define REAL3FILE "%lg %lg %lg\n"
#define REAL4FILE "%lg %lg %lg %lg\n"

#endif

/*
** Vector type definition
*/

typedef struct {
    real x,y;
} realvec2;                                /* 2D vector of reals */

typedef struct {
    real x,y,z;
} realvec3;                                /* 3D vector of reals */

typedef struct {
    real x,y,z,w;
} realvec4;                                /* 4D vector of reals */

/*
** Matrix type definition
*/
```

```
typedef realvec2 realmat2[2];      /* 2D matrix of reals */

typedef realvec3 realmat3[3];      /* 3D matrix of reals */

typedef realvec4 realmat4[4];      /* 4D matrix of reals */

/*
** Frame type definition
*/

typedef realvec2 frame2[5];        /* 2D cartesian frame */

typedef realvec3 frame3[7];        /* 3D cartesian frame */

typedef realvec4 frame4[9];        /* 4D cartesian frame */

/*
** Aliases for lazy programmers
*/

typedef realvec2 rv2;
typedef realvec3 rv3;
typedef realvec4 rv4;
typedef realmat2 rm2;
typedef realmat3 rm3;
typedef realmat4 rm4;

/*
** Get values from file
*/

#define GET_REAL(File,Var)\
    (fscanf (File, REAL1FILE, &(Var)))

#define GET_REALVEC2(File,Var)\
    (fscanf (File, REAL2FILE, &(Var).x, &(Var).y))

#define GET_REALVEC3(File,Var)\
    (fscanf (File, REAL3FILE, &(Var).x, &(Var).y, &(Var).z))

#define GET_REALVEC4(File,Var)\
    (fscanf (File, REAL4FILE, &(Var).x, &(Var).y, &(Var).z, &(Var).w))

#define GET_REALMAT2(File,Var)\
    (GET_REALVEC2(File,(Var)[0]),\
     GET_REALVEC2(File,(Var)[1]))

#define GET_REALMAT3(File,Var)\
    (GET_REALVEC3(File,(Var)[0]),\
     GET_REALVEC3(File,(Var)[1]),\
     GET_REALVEC3(File,(Var)[2]))

#define GET_REALMAT4(File,Var)\
    (GET_REALVEC4(File,(Var)[0]),\
     GET_REALVEC4(File,(Var)[1]),\
     GET_REALVEC4(File,(Var)[2]),\
     GET_REALVEC4(File,(Var)[3]))

/*
** Put values in file

```

\*/

```
#define PUT_REAL(File,Var)\
    (fprintf (File, REAL1FILE, (Var)))

#define PUT_REALVEC2(File,Var)\
    (fprintf (File, REAL2FILE, (Var).x, (Var).y))

#define PUT_REALVEC3(File,Var)\
    (fprintf (File, REAL3FILE, (Var).x, (Var).y, (Var).z))

#define PUT_REALVEC4(File,Var)\
    (fprintf (File, REAL4FILE, (Var).x, (Var).y, (Var).z, (Var).w))

#define PUT_REALMAT2(File,Var)\
    (PUT_REALVEC2 (File,(Var)[0]),\
     PUT_REALVEC2 (File,(Var)[1]))

#define PUT_REALMAT3(File,Var)\
    (PUT_REALVEC3(File,(Var)[0]),\
     PUT_REALVEC3(File,(Var)[1]),\
     PUT_REALVEC3(File,(Var)[2]))

#define PUT_REALMAT4(File,Var)\
    (PUT_REALVEC4(File,(Var)[0]),\
     PUT_REALVEC4(File,(Var)[1]),\
     PUT_REALVEC4(File,(Var)[2]),\
     PUT_REALVEC4(File,(Var)[3]))

#endif
```

/\* ----- \*/

```
/* ----- */
SINT.H :

Definition of a signed integer type (and related types)

by Christophe Schlick (1 June 1992)
/* ----- */

#ifndef _SINT_
#define _SINT_

#include "tool.h"

/*
** By default, signed integers are defined in double precision (ie long)
** To get single precision, simply add "#define SINGLE_SINT" in your program
*/

/*
** Scalar type definition (single or double precision)
*/

#ifdef SINGLE_SINT

typedef short sint;

#define SINT1FILE "%d\n"
#define SINT2FILE "%d %d\n"
#define SINT3FILE "%d %d %d\n"
#define SINT4FILE "%d %d %d %d\n"

#else

typedef long sint;

#define SINT1FILE "%ld\n"
#define SINT2FILE "%ld %ld\n"
#define SINT3FILE "%ld %ld %ld\n"
#define SINT4FILE "%ld %ld %ld %ld\n"

#endif

/*
** Vector type definition
*/

typedef struct {
    sint x,y;
} sintvec2;                                /* 2D vector of signed integers */

typedef struct {
    sint x,y,z;
} sintvec3;                                /* 3D vector of signed integers */

typedef struct {
    sint x,y,z,w;
} sintvec4;                                /* 4D vector of signed integers */

/*
** Matrix type definition
*/
```

```
typedef sintvec2    sintmat2[2];    /* 2D matrix of signed integers */

typedef sintvec3    sintmat3[3];    /* 3D matrix of signed integers */

typedef sintvec4    sintmat4[4];    /* 4D matrix of signed integers */

/*
** Aliases for lazy programmers
*/

typedef sintvec2 sv2;
typedef sintvec3 sv3;
typedef sintvec4 sv4;
typedef sintmat2 sm2;
typedef sintmat3 sm3;
typedef sintmat4 sm4;

/*
** Get values from file
*/

#define GET_SINT(File,Var)\
    (fscanf (File, SINT1FILE, &(Var)))

#define GET_SINTVEC2(File,Var)\
    (fscanf (File, SINT2FILE, &(Var).x, &(Var).y))

#define GET_SINTVEC3(File,Var)\
    (fscanf (File, SINT3FILE, &(Var).x, &(Var).y, &(Var).z))

#define GET_SINTVEC4(File,Var)\
    (fscanf (File, SINT4FILE, &(Var).x, &(Var).y, &(Var).z, &(Var).w))

#define GET_SINTMAT2(File,Var)\
    (GET_SINTVEC2(File,(Var)[0]),\
     GET_SINTVEC2(File,(Var)[1]))

#define GET_SINTMAT3(File,Var)\
    (GET_SINTVEC3(File,(Var)[0]),\
     GET_SINTVEC3(File,(Var)[1]),\
     GET_SINTVEC3(File,(Var)[2]))

#define GET_SINTMAT4(File,Var)\
    (GET_SINTVEC4(File,(Var)[0]),\
     GET_SINTVEC4(File,(Var)[1]),\
     GET_SINTVEC4(File,(Var)[2]),\
     GET_SINTVEC4(File,(Var)[3]))

/*
** Put values in file
*/

#define PUT_SINT(File,Var)\
    (fprintf (File, SINT1FILE, (Var)))

#define PUT_SINTVEC2(File,Var)\
    (fprintf (File, SINT2FILE, (Var).x, (Var).y))

#define PUT_SINTVEC3(File,Var)\
    (fprintf (File, SINT3FILE, (Var).x, (Var).y, (Var).z))
```



```
#define PUT_SINTVEC4(File,Var)\
    (fprintf (File, SINT4FILE, (Var).x, (Var).y, (Var).z, (Var).w))

#define PUT_SINTMAT2(File,Var)\
    (PUT_SINTVEC2 (File,(Var)[0]),\
     PUT_SINTVEC2 (File,(Var)[1]))

#define PUT_SINTMAT3(File,Var)\
    (PUT_SINTVEC3(File,(Var)[0]),\
     PUT_SINTVEC3(File,(Var)[1]),\
     PUT_SINTVEC3(File,(Var)[2]))

#define PUT_SINTMAT4(File,Var)\
    (PUT_SINTVEC4(File,(Var)[0]),\
     PUT_SINTVEC4(File,(Var)[1]),\
     PUT_SINTVEC4(File,(Var)[2]),\
     PUT_SINTVEC4(File,(Var)[3]))

#endif

/* ----- */
```

```
/* ----- */
TOOL.H :

Basic definitions (based on A. Glassner in Graphics Gems I)

by Christophe Schlick (1 June 1992)

"A Toolbox of Macro Functions for Computer Graphics"
in Graphics Gems V (edited by A. Paeth), Academic Press
/* ----- */

#ifndef _TOOL_
#define _TOOL_

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

/* ----- */
/* ----- TYPES AND CONSTANTS ----- */
/* ----- */

/*
** Type definitions (boolean, byte and file)
*/

typedef unsigned int  bool;
typedef unsigned char byte;
typedef FILE *file;

/*
** Boolean constants
*/

#define TRUE      (1)
#define FALSE     (0)
#define ON        (1)
#define OFF       (0)

/*
** Mathematical constants
*/

#ifdef SINGLE_REAL

#define LO_TOL      ((float) 0.000010)
#define TOL         ((float) 0.000100)
#define HI_TOL      ((float) 0.001000)
#define E           ((float) 2.718282)
#define PI          ((float) 3.141593)
#define HALF_PI     ((float) 1.570796)
#define DBLE_PI     ((float) 6.283185)
#define SQRT2       ((float) 1.414214)
#define HALF_SQRT2  ((float) 0.707107)
#define DBLE_SQRT2  ((float) 2.828427)
#define SQRT3       ((float) 1.732051)
#define HALF_SQRT3  ((float) 0.860254)
#define DBLE_SQRT3  ((float) 3.464102)
#define GOLD_RATIO  ((float) 1.618340)
#define DEG_TO_RAD  ((float) 0.017453)
#define RAD_TO_DEG  ((float) 57.29578)
```

```
#else

#define LO_TOL      ((double) 0.00000000000000010)
#define TOL         ((double) 0.000000000000000100)
#define HI_TOL      ((double) 0.000000000000001000)
#define E           ((double) 2.7182818284590498)
#define PI          ((double) 3.1415926535897929)
#define HALF_PI     ((double) 1.5707963267948965)
#define DBLE_PI     ((double) 6.2831853071795858)
#define SQRT2       ((double) 1.4142135623730951)
#define HALF_SQRT2  ((double) 0.7071067811865476)
#define DBLE_SQRT2  ((double) 2.8284271248461902)
#define SQRT3       ((double) 1.7320508075688772)
#define HALF_SQRT3  ((double) 0.8602540378443859)
#define DBLE_SQRT3  ((double) 3.4641016151377544)
#define GOLD_RATIO  ((double) 1.6183398874989489)
#define DEG_TO_RAD  ((double) 0.0174532925199433)
#define RAD_TO_DEG  ((double) 57.295779513082322)

#endif

/* ----- * \
                                     MATHEMATICAL MACROS
\* ----- */

/*
** ABS(t) = Absolute value of 't'
** SGN(t) = Sign value of 't'
** FLOOR(t) = Map 't' to the default integer
** ROUND(t) = Map 't' to the nearest integer
*/

#define ABS(t)      ((t) < 0 ? -(t) : (t))
#define SGN(t)      ((t) < 0 ? -1 : (t) > 0 ? 1 : 0)
#define FLOOR(t)    ((t) < 0 ? (int) ((t)-1.0) : (int) (t))
#define ROUND(t)    ((t) < 0 ? (int) ((t)-0.5) : (int) ((t)+0.5))

/*
** ZERO(a)      = Test if a = 0 with a given tolerance
** SAME(a,b)    = Test if a = b with a given tolerance
** LESS(a,b)    = Test if a < b with a given tolerance
** MORE(a,b)    = Test if a > b with a given tolerance
** !LESS(a,b)   = Test if a >= b with a given tolerance
** !MORE(a,b)   = Test if a <= b with a given tolerance
** XXXX_TOL(a,b,t) = Same thing but with a user-provided tolerance
*/

#define ZERO(a)      ((a) > -TOL && (a) < TOL)
#define ZERO_TOL(a,t) ((a) > -(t) && (a) < (t))
#define SAME(a,b)    ((a) > (b)-TOL && (a) < (b)+TOL)
#define SAME_TOL(a,b,t) ((a) > (b)-(t) && (a) < (b)+(t))
#define LESS(a,b)    ((a) < (b)-TOL)
#define LESS_TOL(a,b,t) ((a) < (b)-(t))
#define MORE(a,b)    ((a) > (b)+TOL)
#define MORE_TOL(a,b,t) ((a) > (b)+(t))

/*
** IN(t,lo,hi)    = Test if t > lo and t < hi
** !IN(t,lo,hi)   = Test if t <= lo or t >= hi
** OUT(t,lo,hi)   = Test if t < lo or t > hi

```

```
** !OUT(t,lo,hi) = Test if t >= lo and t <= hi
** CLAMP(t,lo,hi) = Clamp a value 't' to the range [lo,hi]
*/

#define IN(t,lo,hi)      ((t) > (lo) && (t) < (hi))
#define OUT(t,lo,hi)     ((t) < (lo) || (t) > (hi))
#define CLAMP(t,lo,hi)   ((t) < (lo) ? (lo) : (t) > (hi) ? (hi) : (t))

/*
** MIN(a,b) = Minimum of values 'a' and 'b'
** MAX(a,b) = Maximum of values 'a' and 'b'
** MINMIN(a,b,c) = Minimum of values 'a', 'b' and 'c'
** MAXMAX(a,b,c) = Maximum of values 'a', 'b' and 'c'
*/

#define MIN(a,b)         ((a) < (b) ? (a) : (b))
#define MAX(a,b)         ((a) > (b) ? (a) : (b))
#define MINMIN(a,b,c)    ((a) < (b) ? MIN (a,c) : MIN (b,c))
#define MAXMAX(a,b,c)    ((a) > (b) ? MAX (a,c) : MAX (b,c))

/*
** LERP(t,a,b) = Linear interpolation between 'a' and 'b' using 't' (0<=t<=1)
** ==> LERP(0) = a, LERP(1) = b
**
** HERP(t,a,b) = Hermite interpolation between 'a' and 'b' using 't' (0<=t<=1)
** ==> HERP(0) = a, HERP'(0) = 0, HERP(1) = b, HERP'(1) = 0
**
** CERP(t,a,b) = Cardinal interpolation between 'a' and 'b' using 't' (0<=t<=1)
** ==> CERP(0) = a, CERP'(0) = 0.5*(b-a), HERP(1) = b, HERP'(1) = 0.5*(b-a)
*/

#define LERP(t,a,b)       ((a) + ((b)-(a))*(t))
#define HERP(t,a,b)       ((a) + ((b)-(a))*(t)*(t)*(3.0-(t)-(t)))
#define CERP(t,aa,a,b,bb) ((a) + 0.5*(t)*((b)-(aa)+(t)*(2.0*(aa)-5.0*(a)\
                        +4.0*(b)-(bb)+(t)*((bb)-3.0*(b)+3.0*(a)-(aa)))))

/*
** BIAS(t,p) = Rational bias operator (0 <= t <= 1 and 0 < p < 1)
** GAIN(t,p) = Rational gain operator (0 <= t <= 1 and 0 < p < 1)
**
** Note : For details, see the Gem given by C. Schlick in "Graphics Gems IV"
*/

#define BIAS(t,p)         ((p)*(t)/((p)+((p)+(p)-1.0)*((t)-1.0)))
#define GAIN(t,p)         ((t)<0.5 ? (p)*(t)/((p)+((p)+(p)-1.0)*((t)+(t)-1.0)):\
                        (p)*((t)-1.0)/((p)-((p)+(p)-1.0)*((t)+(t)-1.0))+1.0)

/*
** SWAP(a,b,t) = Swap 'a' and 'b' using 't' as temporary variable
**
** Warning : The Gem given by B. Wyvill in "Graphics Gems I" should not be used
**           because it is compiler-dependent when using non-integer variables!
*/

#define SWAP(a,b,t)       ((t) = (a), (a) = (b), (b) = (t))

/*
** Resolution (real solutions only) of a quadratic equation: a*x^2+b*x+c = 0
**
** Warning : This is a hacker-like implementation of a quadratic solver. It
**           is intended to be optimal in terms of floating point operations

```

```
**          but warn that values of parameters a, b and c are destroyed!
**
** Return :   a = Smallest solution (if it exists)
**           b = Greatest solution (if it exists)
**           Return value = Number of solutions (0, 1, or 2)
**
*/

#define QUADRATIC(a,b,c)\
    (ZERO (a) ? ZERO (b) ? 0 : (a = -c / b, b = a, 1) :\
    (a = -0.5 / a, b *= a, c *= a + a, c += b * b,\
    (c >  TOL) ? (c = sqrt (c), a = b - c, b = b + c, 2) :\
    (c < -TOL) ? 0 : (a = b, 1)))

/* ----- */
/*          MEMORY MANAGEMENT MACROS          */
/* ----- */

/*
** USR_INIT_MEM = User-provided memory allocation routine (default = malloc)
** USR_EXIT_MEM = User-provided memory deallocation routine (default = free)
**
** Note: If you do not like standard (de)allocation routines, simply redefine
**       USR_INIT_MEM and USR_EXIT_MEM with your favorite functions
**
*/

#ifndef USR_INIT_MEM
#define USR_INIT_MEM  malloc
#endif

#ifndef USR_EXIT_MEM
#define USR_EXIT_MEM  free
#endif

/*
** INIT_MEM(Var,Type,Nbr) = Allocation for 'Nbr' elements of type 'Type'
** EXIT_MEM(Var) = Deallocation of variable 'Var'
** ZERO_MEM(Var,Type,Nbr) = Fill 'Nbr' elements of type 'Type' with zero
** COPY_MEM(Var,Mem,Type,Nbr) = Copy 'Nbr' elements from 'Mem' to 'Var'
** SAME_MEM(Var,Mem,Type,Nbr) = Test if 'Var=Mem' (only 'Nbr' first elements)
** LESS_MEM(Var,Mem,Type,Nbr) = Test if 'Var<Mem' (only 'Nbr' first elements)
** MORE_MEM(Var,Mem,Type,Nbr) = Test if 'Var>Mem' (only 'Nbr' first elements)
**
*/

#define INIT_MEM(Var,Type,Nbr)      (Var = (Type*) (malloc(sizeof(Type)*(Nbr))))
#define EXIT_MEM(Var)              (free (Var))
#define ZERO_MEM(Var,Type,Nbr)     (bzero (Var,sizeof(Type)*(Nbr)))
#define COPY_MEM(Var,Mem,Type,Nbr) (bcopy (Mem,Var,sizeof(Type)*(Nbr)))
#define SAME_MEM(Var,Mem,Type,Nbr) (bcmp (Var,Mem,sizeof(Type)*(Nbr)) == 0)
#define LESS_MEM(Var,Mem,Type,Nbr) (bcmp (Var,Mem,sizeof(Type)*(Nbr)) < 0)
#define MORE_MEM(Var,Mem,Type,Nbr) (bcmp (Var,Mem,sizeof(Type)*(Nbr)) > 0)

/* ----- */
/*          FILE MANAGEMENT MACROS          */
/* ----- */

/*
** Symbolic constants for standard streams and file modes
**
*/

#define SOF stdout      /* Standard output file */
#define SIF stdin       /* Standard input file */
```

```
#define SEF stderr          /* Standard error file */

#define RFILE "r"          /* Open an existing file in read mode */
#define WFILE "a"          /* Open an existing file in write mode */
#define NFILE "w"          /* Create a new file (or recreate an existing one) */

/*
** INIT_FILE(File,Name,Mode) = Open or create file 'Name' in a given 'Mode'
** EXIT_FILE(File) = Close file 'File'
** TEST_FILE(File) = Test if the file pointer is within 'File'
** HEAD_FILE(File) = Set the file pointer at the head of 'File'
** TAIL_FILE(File) = Set the file pointer at the tail of 'File'
*/

#define INIT_FILE(File,Name,Mode)      ((File) = fopen (Name, Mode))
#define EXIT_FILE(File)                (fclose (File))
#define TEST_FILE(File)                (!feof (File))
#define HEAD_FILE(File)                (fseek (File, (long)0, 0))
#define TAIL_FILE(File)                (fseek (File, (long)0, 2))

/*
** GET_BYTE(File,Var) = Get a byte in 'File' and put it in 'Var'
** PUT_BYTE(File,Var) = Put a byte 'Var' at the current position in 'File'
** GET_STR(File,Var) = Get a string (until the next blank character) from 'File'
** PUT_STR(File,Var) = Put a null-terminated string 'Var' in 'File'
** GET_LINE(File,Var,Max) = Get a line of at most 'Max' characters from 'File'
** PUT_LINE(File,Var) = Put a string 'Var' ended with a newline char in 'File'
*/

#define GET_BYTE(File,Var)             ((Var) = getc (File))
#define PUT_BYTE(File,Var)             (putc (Var, File))
#define GET_STR(File,Var)              (fscanf (File, "%s\n", Var))
#define PUT_STR(File,Var)              (fprintf (File, "%s", Var))
#define GET_LINE(File,Var,Max)         (fgets (Var, Max, File))
#define PUT_LINE(File,Var)             (fprintf (File, "%s\n", Var))

/* ----- */
/*                               ERROR MANAGEMENT MACROS                               */
/* ----- */

/*
** ASSERT(Test) = If 'Test' is false, display a standard error message
** WARN_ERROR(Test,Mesg,Name) = If 'Test' is false, display an error 'Mesg'
**                               including an eventual identifier 'Name'
** STOP_ERROR(Test,Mesg,Name) = If 'Test' is false, display an error 'Mesg'
**                               including an identifier 'Name' and stop
** CURE_ERROR(Test,Cure,Var) = If 'Test' is false, try to 'Cure' the error
*/

#define ASSERT(Test)\
    ((Test) ? TRUE :\
    (fprintf (SEF, "Assertion \"%s\" failed in file %s at line %d\n",\
    #Test, __FILE__, __LINE__), exit (0)))

#define WARN_ERROR(Test,Mesg,Name) (Test ? : fprintf (SEF,Mesg,Name))
#define STOP_ERROR(Test,Mesg,Name) (Test ? : (fprintf (SEF,Mesg,Name),exit(0)))
#define CURE_ERROR(Test,Cure,Var)  (Test ? : (Cure) ((void *) Var))

#endif

/* ----- */
```

```
/* ----- */
UINT.H :

Definition of an unsigned integer type (and related types)

by Christophe Schlick (1 June 1992)
/* ----- */

#ifndef _UINT_
#define _UINT_

#include "tool.h"

/*
** By default, unsigned integers are defined in double precision (ie ulong)
** To get single precision, simply add "#define SINGLE_UINT" in your program
*/

/*
** Scalar type definition (single or double precision)
*/

#ifdef SINGLE_UINT

typedef unsigned short uint;

#define UINT1FILE "%u\n"
#define UINT2FILE "%u %u\n"
#define UINT3FILE "%u %u %u\n"
#define UINT4FILE "%u %u %u %u\n"

#else

typedef unsigned long uint;

#define UINT1FILE "%lu\n"
#define UINT2FILE "%lu %lu\n"
#define UINT3FILE "%lu %lu %lu\n"
#define UINT4FILE "%lu %lu %lu %lu\n"

#endif

/*
** Vector type definition
*/

typedef struct {
    uint x,y;
} uintvec2;                                /* 2D vector of signed integers */

typedef struct {
    uint x,y,z;
} uintvec3;                                /* 3D vector of signed integers */

typedef struct {
    uint x,y,z,w;
} uintvec4;                                /* 4D vector of signed integers */

/*
** Matrix type definition
*/
```

```
typedef uintvec2 uintmat2[2];          /* 2D matrix of signed integers */
typedef uintvec3 uintmat3[3];          /* 3D matrix of signed integers */
typedef uintvec4 uintmat4[4];          /* 4D matrix of signed integers */

/*
** Aliases for lazy programmers
*/

typedef uintvec2 uv2;
typedef uintvec3 uv3;
typedef uintvec4 uv4;
typedef uintmat2 um2;
typedef uintmat3 um3;
typedef uintmat4 um4;

/*
** Get values from file
*/

#define GET_UINT(File,Var)\
    (fscanf (File, UINT1FILE, &(Var)))

#define GET_UINTVEC2(File,Var)\
    (fscanf (File, UINT2FILE, &(Var).x, &(Var).y))

#define GET_UINTVEC3(File,Var)\
    (fscanf (File, UINT3FILE, &(Var).x, &(Var).y, &(Var).z))

#define GET_UINTVEC4(File,Var)\
    (fscanf (File, UINT4FILE, &(Var).x, &(Var).y, &(Var).z, &(Var).w))

#define GET_UINTMAT2(File,Var)\
    (GET_UINTVEC2(File,(Var)[0]),\
     GET_UINTVEC2(File,(Var)[1]))

#define GET_UINTMAT3(File,Var)\
    (GET_UINTVEC3(File,(Var)[0]),\
     GET_UINTVEC3(File,(Var)[1]),\
     GET_UINTVEC3(File,(Var)[2]))

#define GET_UINTMAT4(File,Var)\
    (GET_UINTVEC4(File,(Var)[0]),\
     GET_UINTVEC4(File,(Var)[1]),\
     GET_UINTVEC4(File,(Var)[2]),\
     GET_UINTVEC4(File,(Var)[3]))

/*
** Put values in file
*/

#define PUT_UINT(File,Var)\
    (fprintf (File, UINT1FILE, (Var)))

#define PUT_UINTVEC2(File,Var)\
    (fprintf (File, UINT2FILE, (Var).x, (Var).y))

#define PUT_UINTVEC3(File,Var)\
    (fprintf (File, UINT3FILE, (Var).x, (Var).y, (Var).z))
```



```
#define PUT_UINTVEC4(File,Var)\
    (fprintf (File, UINT4FILE, (Var).x, (Var).y, (Var).z, (Var).w))

#define PUT_UINTMAT2(File,Var)\
    (PUT_UINTVEC2 (File,(Var)[0]),\
     PUT_UINTVEC2 (File,(Var)[1]))

#define PUT_UINTMAT3(File,Var)\
    (PUT_UINTVEC3(File,(Var)[0]),\
     PUT_UINTVEC3(File,(Var)[1]),\
     PUT_UINTVEC3(File,(Var)[2]))

#define PUT_UINTMAT4(File,Var)\
    (PUT_UINTVEC4(File,(Var)[0]),\
     PUT_UINTVEC4(File,(Var)[1]),\
     PUT_UINTVEC4(File,(Var)[2]),\
     PUT_UINTVEC4(File,(Var)[3]))

#endif

/* ----- */
```

```
/* ----- */
VEC2.H :

Definition and manipulation of a 2D vector (either integers or reals)

by Christophe Schlick (1 June 1992)
/* ----- */

#ifndef _VEC2_
#define _VEC2_

#include "tool.h"

/*
** These macros are equivalent (for 2D vectors) to the ones given in tool.h
*/

#define MAKE_VEC2(V,X,Y)\
    ((V).x = X,\
     (V).y = Y)

#define COPY_VEC2(V,A)\
    ((V).x = (A).x,\
     (V).y = (A).y)

#define SWAP_VEC2(A,B,t)\
    (SWAP ((A).x, (B).x, t),\
     SWAP ((A).y, (B).y, t))

#define ABS_VEC2(V,A)\
    ((V).x = ABS ((A).x),\
     (V).y = ABS ((A).y))

#define SGN_VEC2(V,A)\
    ((V).x = SGN ((A).x),\
     (V).y = SGN ((A).y))

#define NEG_VEC2(V,A)\
    ((V).x = -(A).x,\
     (V).y = -(A).y)

#define ROUND_VEC2(V,A)\
    ((V).x = ROUND ((A).x),\
     (V).y = ROUND ((A).y))

#define ZERO_VEC2(A)\
    (ZERO ((A).x) &&\
     ZERO ((A).y))

#define ZERO_TOL_VEC2(A,t)\
    (ZERO_TOL ((A).x,t) &&\
     ZERO_TOL ((A).y,t))

#define SAME_VEC2(A,B)\
    (SAME ((A).x,(B).x) &&\
     SAME ((A).y,(B).y))

#define SAME_TOL_VEC2(A,B,t)\
    (SAME_TOL ((A).x,(B).x,t) &&\
     SAME_TOL ((A).y,(B).y,t))
```

```
#define IN_VEC2(V,A,B)\
    (IN ((V).x,(A).x,(B).x) &&\
     IN ((V).y,(A).y,(B).y))

#define OUT_VEC2(V,A,B)\
    (OUT ((V).x,(A).x,(B).x) ||\
     OUT ((V).y,(A).y,(B).y))

#define CLAMP_VEC2(V,A,B)\
    ((V).x = CLAMP ((A).x,(B).x),\
     (V).y = CLAMP ((A).y,(B).y))

#define MIN_VEC2(V,A,B)\
    ((V).x = MIN ((A).x,(B).x),\
     (V).y = MIN ((A).y,(B).y))

#define MAX_VEC2(V,A,B)\
    ((V).x = MAX ((A).x,(B).x),\
     (V).y = MAX ((A).y,(B).y))

#define MINMIN_VEC2(V,A,B,C)\
    ((V).x = MINMIN ((A).x,(B).x,(C).x),\
     (V).y = MINMIN ((A).y,(B).y,(C).y))

#define MAXMAX_VEC2(V,A,B,C)\
    ((V).x = MAXMAX ((A).x,(B).x,(C).x),\
     (V).y = MAXMAX ((A).y,(B).y,(C).y))

/*
** Addition, subtraction, multiplication, division (by a vector element)
**/

#define INC_VEC2(V,A)\
    ((V).x += (A).x,\
     (V).y += (A).y)

#define DEC_VEC2(V,A)\
    ((V).x -= (A).x,\
     (V).y -= (A).y)

#define ADD_VEC2(V,A,B)\
    ((V).x = (A).x + (B).x,\
     (V).y = (A).y + (B).y)

#define SUB_VEC2(V,A,B)\
    ((V).x = (A).x - (B).x,\
     (V).y = (A).y - (B).y)

#define MUL_VEC2(V,A,B)\
    ((V).x = (A).x * (B).x,\
     (V).y = (A).y * (B).y)

#define DIV_VEC2(V,A,B)\
    ((V).x = (A).x / (B).x,\
     (V).y = (A).y / (B).y)

/*
** Addition, subtraction, multiplication, division (by a scalar element)
**/

#define INCS_VEC2(V,s)\
```

```
((V).x += (s), \
(V).y += (s))
```

```
#define DECS_VEC2(V,s)\
((V).x -= (s), \
(V).y -= (s))
```

```
#define ADDS_VEC2(V,A,s)\
((V).x = (A).x + (s), \
(V).y = (A).y + (s))
```

```
#define SUBS_VEC2(V,A,s)\
((V).x = (A).x - (s), \
(V).y = (A).y - (s))
```

```
#define MULS_VEC2(V,A,s)\
((V).x = (A).x * (s), \
(V).y = (A).y * (s))
```

```
#define DIVS_VEC2(V,A,s)\
((V).x = (A).x / (s), \
(V).y = (A).y / (s))
```

```
/*
** Linear combinations, linear/hermite/cardinal interpolations (see "tool.h")
*/
```

```
#define COMB2_VEC2(V,a,A,b,B)\
((V).x = (a) * (A).x + (b) * (B).x, \
(V).y = (a) * (A).y + (b) * (B).y)
```

```
#define COMB3_VEC2(V,a,A,b,B,c,C)\
((V).x = (a) * (A).x + (b) * (B).x + (c) * (C).x, \
(V).y = (a) * (A).y + (b) * (B).y + (c) * (C).y)
```

```
#define COMB4_VEC2(V,a,A,b,B,c,C,d,D)\
((V).x = (a) * (A).x + (b) * (B).x + (c) * (C).x + (d) * (D).x, \
(V).y = (a) * (A).y + (b) * (B).y + (c) * (C).y + (d) * (D).y)
```

```
#define LERP_VEC2(V,t,A,B)\
((V).x = LERP ((t), (A).x, (B).x), \
(V).y = LERP ((t), (A).y, (B).y))
```

```
#define HERP_VEC2(V,t,A,B)\
((V).x = HERP ((t), (A).x, (B).x), \
(V).y = HERP ((t), (A).y, (B).y))
```

```
#define CERP_VEC2(V,t,AA,A,B,BB)\
((V).x = CERP ((t), (AA).x, (A).x, (B).x, (BB).x), \
(V).y = CERP ((t), (AA).y, (A).y, (B).y, (BB).y))
```

```
/*
** Perpendicular, determinant, dot product, length, normalization
*/
```

```
#define PERP_VEC2(V,A)\
((V).x = -(A).y, \
(V).y = (A).x)
```

```
#define DELTA_VEC2(A,B)\
((A).x * (B).y - (A).y * (B).x)
```

```
#define DOT_VEC2(A,B)\
    ((A).x * (B).x + (A).y * (B).y)

#define LEN_VEC2(A)\
    (sqrt ((double) DOT_VEC2 (A,A)))

#define UNIT_VEC2(V,A,s)\
    ((s) = LEN_VEC2(A), (s) ? (DIVS_VEC2 (V,A,s), TRUE) : FALSE)

#endif

/* ----- */
```

```
/* ----- */
VEC3.H :

Definition and manipulation of a 3D vector (either integers or reals)

by Christophe Schlick (1 June 1992)
/* ----- */

#ifndef _VEC3_
#define _VEC3_

#include "tool.h"

/*
** These macros are equivalent (for 3D vectors) to the ones given in tool.h
*/

#define MAKE_VEC3(V,X,Y,Z)\
    ((V).x = X,\
    (V).y = Y,\
    (V).z = Z)

#define COPY_VEC3(V,A)\
    ((V).x = (A).x,\
    (V).y = (A).y,\
    (V).z = (A).z)

#define SWAP_VEC3(A,B,t)\
    (SWAP ((A).x, (B).x, t),\
    SWAP ((A).y, (B).y, t),\
    SWAP ((A).z, (B).z, t))

#define ABS_VEC3(V,A)\
    ((V).x = ABS ((A).x),\
    (V).y = ABS ((A).y),\
    (V).z = ABS ((A).z))

#define SGN_VEC3(V,A)\
    ((V).x = SGN ((A).x),\
    (V).y = SGN ((A).y),\
    (V).z = SGN ((A).z))

#define NEG_VEC3(V,A)\
    ((V).x = -(A).x,\
    (V).y = -(A).y,\
    (V).z = -(A).z)

#define ROUND_VEC3(V,A)\
    ((V).x = ROUND ((A).x),\
    (V).y = ROUND ((A).y),\
    (V).z = ROUND ((A).z))

#define ZERO_VEC3(A)\
    (ZERO ((A).x) &&\
    ZERO ((A).y) &&\
    ZERO ((A).z))

#define ZERO_TOL_VEC3(A,t)\
    (ZERO_TOL ((A).x,t) &&\
    ZERO_TOL ((A).y,t) &&\
    ZERO_TOL ((A).z,t))
```

```
#define SAME_VEC3(A,B)\
    (SAME ((A).x,(B).x) &&\
     SAME ((A).y,(B).y) &&\
     SAME ((A).z,(B).z))

#define SAME_TOL_VEC3(A,B,t)\
    (SAME_TOL ((A).x,(B).x,t) &&\
     SAME_TOL ((A).y,(B).y,t) &&\
     SAME_TOL ((A).z,(B).z,t))

#define IN_VEC3(V,A,B)\
    (IN ((V).x,(A).x,(B).x) &&\
     IN ((V).y,(A).y,(B).y) &&\
     IN ((V).z,(A).z,(B).z))

#define OUT_VEC3(V,A,B)\
    (OUT ((V).x,(A).x,(B).x) ||\
     OUT ((V).y,(A).y,(B).y) ||\
     OUT ((V).z,(A).z,(B).z))

#define CLAMP_VEC3(V,A,B)\
    ((V).x = CLAMP ((A).x,(B).x),\
     (V).y = CLAMP ((A).y,(B).y),\
     (V).z = CLAMP ((A).z,(B).z))

#define MIN_VEC3(V,A,B)\
    ((V).x = MIN ((A).x,(B).x),\
     (V).y = MIN ((A).y,(B).y),\
     (V).z = MIN ((A).z,(B).z))

#define MAX_VEC3(V,A,B)\
    ((V).x = MAX ((A).x,(B).x),\
     (V).y = MAX ((A).y,(B).y),\
     (V).z = MAX ((A).z,(B).z))

#define MINMIN_VEC3(V,A,B,C)\
    ((V).x = MINMIN ((A).x,(B).x,(C).x),\
     (V).y = MINMIN ((A).y,(B).y,(C).y),\
     (V).z = MINMIN ((A).z,(B).z,(C).z))

#define MAXMAX_VEC3(V,A,B,C)\
    ((V).x = MAXMAX ((A).x,(B).x,(C).x),\
     (V).y = MAXMAX ((A).y,(B).y,(C).y),\
     (V).z = MAXMAX ((A).z,(B).z,(C).z))

/*
** Addition, subtraction, multiplication, division (by a vector element)
*/

#define INC_VEC3(V,A)\
    ((V).x += (A).x,\
     (V).y += (A).y,\
     (V).z += (A).z)

#define DEC_VEC3(V,A)\
    ((V).x -= (A).x,\
     (V).y -= (A).y,\
     (V).z -= (A).z)

#define ADD_VEC3(V,A,B)\
```

```
((V).x = (A).x + (B).x,\n(V).y = (A).y + (B).y,\n(V).z = (A).z + (B).z)
```

```
#define SUB_VEC3(V,A,B)\n((V).x = (A).x - (B).x,\n(V).y = (A).y - (B).y,\n(V).z = (A).z - (B).z)
```

```
#define MUL_VEC3(V,A,B)\n((V).x = (A).x * (B).x,\n(V).y = (A).y * (B).y,\n(V).z = (A).z * (B).z)
```

```
#define DIV_VEC3(V,A,B)\n((V).x = (A).x / (B).x,\n(V).y = (A).y / (B).y,\n(V).z = (A).z / (B).z)
```

```
/*\n** Addition, subtraction, multiplication, division (by a scalar element)\n*/
```

```
#define INCS_VEC3(V,s)\n((V).x += (s),\n(V).y += (s),\n(V).z += (s))
```

```
#define DECS_VEC3(V,s)\n((V).x -= (s),\n(V).y -= (s),\n(V).z -= (s))
```

```
#define ADDS_VEC3(V,A,s)\n((V).x = (A).x + (s),\n(V).y = (A).y + (s),\n(V).z = (A).z + (s))
```

```
#define SUBS_VEC3(V,A,s)\n((V).x = (A).x - (s),\n(V).y = (A).y - (s),\n(V).z = (A).z - (s))
```

```
#define MULS_VEC3(V,A,s)\n((V).x = (A).x * (s),\n(V).y = (A).y * (s),\n(V).z = (A).z * (s))
```

```
#define DIVS_VEC3(V,A,s)\n((V).x = (A).x / (s),\n(V).y = (A).y / (s),\n(V).z = (A).z / (s))
```

```
/*\n** Linear combinations, linear/hermite/cardinal interpolations (see "tool.h")\n*/
```

```
#define COMB2_VEC3(V,a,A,b,B)\n((V).x = (a) * (A).x + (b) * (B).x,\n(V).y = (a) * (A).y + (b) * (B).y,\n(V).z = (a) * (A).z + (b) * (B).z)
```



```
#define COMB3_VEC3(V,a,A,b,B,c,C)\
((V).x = (a) * (A).x + (b) * (B).x + (c) * (C).x,\
 (V).y = (a) * (A).y + (b) * (B).y + (c) * (C).y,\
 (V).z = (a) * (A).z + (b) * (B).z + (c) * (C).z)

#define COMB4_VEC3(V,a,A,b,B,c,C,d,D)\
((V).x = (a) * (A).x + (b) * (B).x + (c) * (C).x + (d) * (D).x,\
 (V).y = (a) * (A).y + (b) * (B).y + (c) * (C).y + (d) * (D).y,\
 (V).z = (a) * (A).z + (b) * (B).z + (c) * (C).z + (d) * (D).z)

#define LERP_VEC3(V,t,A,B)\
((V).x = LERP ((t), (A).x, (B).x),\
 (V).y = LERP ((t), (A).y, (B).y),\
 (V).z = LERP ((t), (A).z, (B).z))

#define HERP_VEC3(V,t,A,B)\
((V).x = HERP ((t), (A).x, (B).x),\
 (V).y = HERP ((t), (A).y, (B).y),\
 (V).z = HERP ((t), (A).z, (B).z))

#define CERP_VEC3(V,t,AA,A,B,BB)\
((V).x = CERP ((t), (AA).x, (A).x, (B).x, (BB).x),\
 (V).y = CERP ((t), (AA).y, (A).y, (B).y, (BB).y),\
 (V).z = CERP ((t), (AA).z, (A).z, (B).z, (BB).z))

/*
** Perpendicular, determinant, dot product, length, normalization
*/

#define PERP_VEC3(V,A,B)\
((V).x = (A).y * (B).z - (A).z * (B).y,\
 (V).y = (A).z * (B).x - (A).x * (B).z,\
 (V).z = (A).x * (B).y - (A).y * (B).x)

#define CROSS_VEC3(V,A,B) PERP_VEC3 (V,A,B) /* Perp is also cross product */

#define DELTA_VEC3(A,B,C)\
((A).x * ((B).y * (C).z - (B).z * (C).y)\
+(A).y * ((B).z * (C).x - (B).x * (C).z)\
+(A).z * ((B).x * (C).y - (B).y * (C).x))

#define DOT_VEC3(A,B)\
((A).x * (B).x + (A).y * (B).y + (A).z * (B).z)

#define LEN_VEC3(A)\
(sqrt ((double) DOT_VEC3 (A, A)))

#define UNIT_VEC3(V,A,s)\
((s) = LEN_VEC3 (A), (s) ? (DIVS_VEC3 (V,A,s), TRUE) : FALSE)

#endif

/* ----- */
```

```
/* ----- */
VEC4.H :

Definition and manipulation of a 4D vector (either integers or reals)

by Christophe Schlick (1 June 1992)
/* ----- */

#ifndef _VEC4_
#define _VEC4_

#include "tool.h"

/*
** These macros are equivalent (for 3D vectors) to the ones given in tool.h
*/

#define MAKE_VEC4(V,X,Y,Z,W)\
    ((V).x = X,\
     (V).y = Y,\
     (V).z = Z,\
     (V).w = W)

#define COPY_VEC4(V,A)\
    ((V).x = (A).x,\
     (V).y = (A).y,\
     (V).z = (A).z,\
     (V).w = (A).w)

#define SWAP_VEC4(A,B,t)\
    (SWAP ((A).x, (B).x, t),\
     SWAP ((A).y, (B).y, t),\
     SWAP ((A).z, (B).z, t),\
     SWAP ((A).w, (B).w, t))

#define ABS_VEC4(V,A)\
    ((V).x = ABS ((A).x),\
     (V).y = ABS ((A).y),\
     (V).z = ABS ((A).z),\
     (V).w = ABS ((A).w))

#define SGN_VEC4(V,A)\
    ((V).x = SGN ((A).x),\
     (V).y = SGN ((A).y),\
     (V).z = SGN ((A).z),\
     (V).w = SGN ((A).w))

#define NEG_VEC4(V)\
    ((V).x = -(V).x,\
     (V).y = -(V).y,\
     (V).z = -(V).z,\
     (V).w = -(V).w)

#define ROUND_VEC4(V,A)\
    ((V).x = ROUND ((A).x),\
     (V).y = ROUND ((A).y),\
     (V).z = ROUND ((A).z),\
     (V).w = ROUND ((A).w))

#define ZERO_VEC4(A)\
    (ZERO ((A).x) &&\
```

```
ZERO ((A).y) &&\
ZERO ((A).z) &&\
ZERO ((A).w)
```

```
#define ZERO_TOL_VEC4(A,t)\
(ZERO_TOL ((A).x,t) &&\
ZERO_TOL ((A).y,t) &&\
ZERO_TOL ((A).z,t) &&\
ZERO_TOL ((A).w,t))
```

```
#define SAME_VEC4(A,B)\
(SAME ((A).x,(B).x) &&\
SAME ((A).y,(B).y) &&\
SAME ((A).z,(B).z) &&\
SAME ((A).w,(B).w))
```

```
#define SAME_TOL_VEC4(A,B,t)\
(SAME_TOL ((A).x,(B).x,t) &&\
SAME_TOL ((A).y,(B).y,t) &&\
SAME_TOL ((A).z,(B).z,t) &&\
SAME_TOL ((A).w,(B).w,t))
```

```
#define IN_VEC4(V,A,B)\
(IN ((V).x,(A).x,(B).x) &&\
IN ((V).y,(A).y,(B).y) &&\
IN ((V).z,(A).z,(B).z) &&\
IN ((V).w,(A).w,(B).w))
```

```
#define OUT_VEC4(V,A,B)\
(OUT ((V).x,(A).x,(B).x) ||\
OUT ((V).y,(A).y,(B).y) ||\
OUT ((V).z,(A).z,(B).z) ||\
OUT ((V).w,(A).w,(B).w))
```

```
#define CLAMP_VEC4(V,A,B)\
((V).x = CLAMP ((A).x,(B).x),\
(V).y = CLAMP ((A).y,(B).y),\
(V).z = CLAMP ((A).z,(B).z),\
(V).w = CLAMP ((A).w,(B).w))
```

```
#define MIN_VEC4(V,A,B)\
((V).x = MIN ((A).x,(B).x),\
(V).y = MIN ((A).y,(B).y),\
(V).z = MIN ((A).z,(B).z),\
(V).w = MIN ((A).w,(B).w))
```

```
#define MAX_VEC4(V,A,B)\
((V).x = MAX ((A).x,(B).x),\
(V).y = MAX ((A).y,(B).y),\
(V).z = MAX ((A).z,(B).z),\
(V).w = MAX ((A).w,(B).w))
```

```
#define MINMIN_VEC4(V,A,B,C)\
((V).x = MINMIN ((A).x,(B).x,(C).x),\
(V).y = MINMIN ((A).y,(B).y,(C).y),\
(V).z = MINMIN ((A).z,(B).z,(C).z),\
(V).w = MINMIN ((A).w,(B).w,(C).w))
```

```
#define MAXMAX_VEC4(V,A,B,C)\
((V).x = MAXMAX ((A).x,(B).x,(C).x),\
(V).y = MAXMAX ((A).y,(B).y,(C).y),\
```

```
(V).z = MAXMAX ((A).z, (B).z, (C).z), \
(V).w = MAXMAX ((A).w, (B).w, (C).w))
```

```
/*
** Addition, subtraction, multiplication, division (by a vector element)
*/
```

```
#define INC_VEC4(V,A)\
((V).x += (A).x, \
(V).y += (A).y, \
(V).z += (A).z, \
(V).w += (A).w)
```

```
#define DEC_VEC4(V,A)\
((V).x -= (A).x, \
(V).y -= (A).y, \
(V).z -= (A).z, \
(V).w -= (A).w)
```

```
#define ADD_VEC4(V,A,B)\
((V).x = (A).x + (B).x, \
(V).y = (A).y + (B).y, \
(V).z = (A).z + (B).z, \
(V).w = (A).w + (B).w)
```

```
#define SUB_VEC4(V,A,B)\
((V).x = (A).x - (B).x, \
(V).y = (A).y - (B).y, \
(V).z = (A).z - (B).z, \
(V).w = (A).w - (B).w)
```

```
#define MUL_VEC4(V,A,B)\
((V).x = (A).x * (B).x, \
(V).y = (A).y * (B).y, \
(V).z = (A).z * (B).z, \
(V).w = (A).w * (B).w)
```

```
#define DIV_VEC4(V,A,B)\
((V).x = (A).x / (B).x, \
(V).y = (A).y / (B).y, \
(V).z = (A).z / (B).z, \
(V).w = (A).w / (B).w)
```

```
/*
** Addition, subtraction, multiplication, division (by a scalar element)
*/
```

```
#define INCS_VEC4(V,s)\
((V).x += (s), \
(V).y += (s), \
(V).z += (s), \
(V).w += (s))
```

```
#define DECS_VEC4(V,s)\
((V).x -= (s), \
(V).y -= (s), \
(V).z -= (s), \
(V).w -= (s))
```

```
#define ADDS_VEC4(V,A,s)\
((V).x = (A).x + (s), \
```

```
(V).y = (A).y + (s),\  
(V).z = (A).z + (s),\  
(V).w = (A).w + (s))
```

```
#define SUBS_VEC4(V,A,s)\  
((V).x = (A).x - (s),\  
(V).y = (A).y - (s),\  
(V).z = (A).z - (s),\  
(V).w = (A).w - (s))
```

```
#define MULS_VEC4(V,A,s)\  
((V).x = (A).x * (s),\  
(V).y = (A).y * (s),\  
(V).z = (A).z * (s),\  
(V).w = (A).w * (s))
```

```
#define DIVS_VEC4(V,A,s)\  
((V).x = (A).x / (s),\  
(V).y = (A).y / (s),\  
(V).z = (A).z / (s),\  
(V).w = (A).w / (s))
```

```
/*  
** Linear combination  
*/
```

```
#define COMB2_VEC4(V,a,A,b,B)\  
((V).x = (a) * (A).x + (b) * (B).x,\  
(V).y = (a) * (A).y + (b) * (B).y,\  
(V).z = (a) * (A).z + (b) * (B).z,\  
(V).w = (a) * (A).w + (b) * (B).w)
```

```
#define COMB3_VEC4(V,a,A,b,B,c,C)\  
((V).x = (a) * (A).x + (b) * (B).x + (c) * (C).x,\  
(V).y = (a) * (A).y + (b) * (B).y + (c) * (C).y,\  
(V).z = (a) * (A).z + (b) * (B).z + (c) * (C).z,\  
(V).w = (a) * (A).w + (b) * (B).w + (c) * (C).w)
```

```
#define COMB4_VEC4(V,a,A,b,B,c,C,d,D)\  
((V).x = (a) * (A).x + (b) * (B).x + (c) * (C).x + (d) * (D).x,\  
(V).y = (a) * (A).y + (b) * (B).y + (c) * (C).y + (d) * (D).y,\  
(V).z = (a) * (A).z + (b) * (B).z + (c) * (C).z + (d) * (D).z,\  
(V).w = (a) * (A).w + (b) * (B).w + (c) * (C).w + (d) * (D).w)
```

```
#define LERP_VEC3(V,t,A,B)\  
((V).x = LERP ((t), (A).x, (B).x),\  
(V).y = LERP ((t), (A).y, (B).y),\  
(V).z = LERP ((t), (A).z, (B).z),\  
(V).w = LERP ((t), (A).w, (B).w))
```

```
#define HERP_VEC3(V,t,A,B)\  
((V).x = HERP ((t), (A).x, (B).x),\  
(V).y = HERP ((t), (A).y, (B).y),\  
(V).z = HERP ((t), (A).z, (B).z),\  
(V).w = HERP ((t), (A).w, (B).w))
```

```
#define CERP_VEC3(V,t,AA,A,B,BB)\  
((V).x = CERP ((t), (AA).x, (A).x, (B).x, (BB).x),\  
(V).y = CERP ((t), (AA).y, (A).y, (B).y, (BB).y),\  
(V).z = CERP ((t), (AA).z, (A).z, (B).z, (BB).z),\  
(V).w = CERP ((t), (AA).w, (A).w, (B).w, (BB).w))
```

```
/*
** Dot product, length, normalization
*/

#define DOT_VEC4(A,B)\
    ((A).x * (B).x + (A).y * (B).y + (A).z * (B).z + (A).w * (B).w)

#define LEN_VEC4(A)\
    (sqrt ((double) DOT_VEC4 (A,A)))

#define UNIT_VEC4(V,A,s)\
    (s = LEN_REAL2 (A), (s) ? (DIVS_REAL2 (V,A,s), TRUE) : FALSE)

#endif

/* ----- */
```

|         |                                                        |
|---------|--------------------------------------------------------|
| README  | this file                                              |
| vec_h.c | library (.h file) generator                            |
| vec.h   | as produced by "cc vec_h.c -o vec_h; ./vec_h 4 >vec.h" |

```
/*
 * vec_h.c -- vec.h generator.
 *
 * Outputs a file containing vector macros for dimensions 2..maxdim,
 * where maxdim is specified on the command line.
 *
 * Usage: vec_h <maxdim>
 */

char *intro[] = {
    " vec.h -- Vector macros for %2..d-1, and %d dimensions,",
    "           for any combination of C scalar types.",
    "",
    " Author:           Don Hatch (hatch@sgi.com)",
    " Last modified:    Fri Sep 30 03:23:02 PDT 1994",
    "",
    " General description:",
    "",
    "     The macro name describes its arguments; e.g.",
    "         MXS3 is \"matrix times scalar in 3 dimensions\";",
    "         VMV2 is \"vector minus vector in 2 dimensions\".",
    "",
    "     If the result of an operation is a scalar, then the macro \"returns\",
    "     the value; e.g.",
    "         result = DOT3(v,w);",
    "         result = DET4(m);",
    "",
    "     If the result of an operation is a vector or matrix, then",
    "     the first argument is the destination; e.g.",
    "         SET2(tovec, fromvec);",
    "         MXM3(result, m1, m2);",
    "",
    " WARNING: For the operations that are not done \"componentwise\",
    "         (e.g. vector cross products and matrix multiplies),
    "         the destination should not be either of the arguments,
    "         for obvious reasons. For example, the following is wrong:",
    "         VXM2(v,v,m);",
    "     For such \"unsafe\" macros, there are safe versions provided,
    "     but you have to specify a type for the temporary",
    "     result vector or matrix. For example, the safe versions",
    "     of VXM2 are:",
    "         VXM2d(v,v,m)    if v's scalar type is double or float",
    "         VXM2i(v,v,m)    if v's scalar type is int or char",
    "         VXM2l(v,v,m)    if v's scalar type is long",
    "         VXM2r(v,v,m)    if v's scalar type is real",
    "         VXM2safe(type,v,v,m) for other scalar types.",
    "     These \"safe\" macros do not evaluate to C expressions",
    "     (so, for example, they can't be used inside the parentheses of",
    "     a for(...)).",
    "",
    " Specific descriptions:",
    "",
    "     The \"?\"'s in the following can be 2, 3, or 4.",
    "",
    "     SET?(to,from)           to = from",
    "     SETMAT?(to,from)        to = from",
    "     ROUNDVEC?(to,from)      to = from with entries rounded",
    "                               to nearest integer",
    "     ROUNDMAT?(to,from)      to = from with entries rounded",
    "                               to nearest integer",

```



```
"      FILLVEC?(v,s)          set each entry of vector v to be s",
"      FILLMAT?(m,s)         set each entry of matrix m to be s",
"      ZEROVEC?(v)           v = 0",
"      ISZEROVEC?(v)         v == 0",
"      EQVEC?(v,w)           v == w",
"      EQMAT?(m1,m2)         m1 == m2",
"      ZEROMAT?(m)           m = 0",
"      IDENTMAT?(m)          m = 1",
"      TRANSPOSE?(to,from)    (matrix to) = (transpose of matrix from)",
"      ADJOINT?(to,from)      (matrix to) = (adjoint of matrix from)",
"                             i.e. its determinant times its inverse",
" ",
"      V{P,M}V?(to,v,w)      to = v {+,-} w",
"      M{P,M}M?(to,m1,m2)    to = m1 {+,-} m2",
"      SX{V,M}?(to,s,from)   to = s * from",
"      M{V,M}?(to,from)      to = -from",
"      {V,M}{X,D}S?(to,from,s) to = from {*,/} s",
"      MXM?(to,m1,m2)        to = m1 * m2",
"      VXM?(to,v,m)          (row vec to) = (row vec v) * m",
"      MXV?(to,m,v)          (column vec to) = m * (column vec v)",
"      LERP?(to,v0,v1,t)     to = v0 + t*(v1-v0)",
" ",
"      DET?(m)               determinant of m",
"      TRACE?(m)             trace (sum of diagonal entries) of m",
"      DOT?(v,w)             dot (scalar) product of v and w",
"      NORMSQRD?(v)          square of |v|",
"      DISTSQRD?(v,w)        square of |v-w|",
" ",
"      XV2(to,v)             to = v rotated by 90 degrees",
"      VXV3(to,v1,v2)        to = cross (vector) product of v1 and v2",
"      VXVXV4(to,v1,v2,v3)   to = 4-dimensional vector cross product",
"                             of v1,v2,v3 (a vector orthogonal to",
"                             v1,v2,v3 whose length equals the",
"                             volume of the spanned parallelotope)",
"      VXV2(v0,v1)           determinant of matrix with rows v0,v1",
"      VXVXV3(v0,v1,v2)      determinant of matrix with rows v0,v1,v2",
"      VXVXVXV4(v0,v1,v2,v3) determinant of matrix with rows v0,...,v3",
" ",
"      The following macros mix objects from different dimensions.",
"      For example, V3XM4 would be used to apply a composite",
"      4x4 rotation-and-translation matrix to a 3d vector.",
" ",
"      SET3from2(to,from,pad) (3d vec to) = (2d vec from) with pad",
"      SET4from3(to,from,pad) (4d vec to) = (3d vec from) with pad",
"      SETMAT3from2(to,from,pad0,pad1) (3x3 mat to) = (2x2 mat from)",
"                                     padded with pad0 on the sides",
"                                     and pad1 in the corner",
"      SETMAT4from3(to,from,pad0,pad1) (4x4 mat to) = (3x3 mat from)",
"                                     padded with pad0 on the sides",
"                                     and pad1 in the corner",
"      V2XM3(to2,v2,m3)       (2d row vec to2) = (2d row vec v2) * (3x3 mat m3)",
"      V3XM4(to3,v3,m4)       (3d row vec to3) = (3d row vec v2) * (4x4 mat m4)",
"      M3XV2(to2,m3,v2)       (2d col vec to2) = (3x3 mat m3) * (2d col vec v2)",
"      M4XV3(to3,m4,v3)       (3d col vec to3) = (4x4 mat m4) * (3d col vec v3)",
"      M2XM3(to3,m2,m3)       (3x3 mat to3) = (2x2 mat m2) * (3x3 mat m3)",
"      M3XM4(to4,m3,m4)       (4x4 mat to4) = (3x3 mat m3) * (4x4 mat m4)",
"      M3XM2(to3,m3,m2)       (3x3 mat to3) = (3x3 mat m3) * (2x2 mat m2)",
"      M4XM3(to4,m4,m3)       (4x4 mat to4) = (4x4 mat m4) * (3x3 mat m3)",
" ",
" ",
"      This file is machine-generated and can be regenerated",
```

```
"    for any number of dimensions.",
"    The program that generated it is available upon request.",
0
};

struct {
    char *abbr, *full;
} types[] = {
    {"d", "double"},
    {"i", "int"},
    {"l", "long"},
    {"r", "real"},
};

struct definition {
    int safety; /* 0 means already safe, i.e. returns a scalar or is done
                  "componentwise". 1 means unsafe and returns a vector.
                  2 means unsafe and returns a matrix. */
    char *name_and_args, *all_but_last, *sep, *last;
} defs[] = {
    {0, "SET%d(to,from)",      "(to)[%i] = (from)[%i]",      "", ""},
    {0, "SETMAT%d(to,from)",  "SET%d((to)[%i], (from)[%i])",  "", ""},
    {0, "ROUNDVEC%d(to,from)", "(to)[%i] = floor((from)[%i]+.5)",  "", ""},
    {0, "ROUNDMAT%d(to,from)", "ROUNDVEC%d((to)[%i], (from)[%i])",  "", ""},
    {0, "FILLVEC%d(v,s)",      "(v)[%i] = (s)",      "", ""},
    {0, "FILLMAT%d(m,s)",      "FILLVEC%d((m)[%i], s)",  "", ""},
    {0, "ZEROVEC%d(v)",        "(v)[%i] = 0",        "", ""},
    {0, "ISZEROVEC%d(v)",      "(v)[%i] == 0",        " &&"},
    {0, "EQVEC%d(v,w)",        "(v)[%i] == (w)[%i]",  " &&"},
    {0, "EQMAT%d(m1,m2)",      "EQVEC%d((m1)[%i], (m2)[%i])",  " &&"},
    {0, "ZEROMAT%d(m)",        "ZEROVEC%d((m)[%i])",  "", ""},
    {0, "IDENTMAT%d(m)",       "ZEROVEC%d((m)[%i]), (m)[%i][%i]=1",  "", ""},
    {2, "TRANPOSE%d(to,from)", "_SETcol%d((to)[%i], from, %i)",  "", ""},
    {0, "VPV%d(to,v,w)",       "(to)[%i] = (v)[%i] + (w)[%i]",  "", ""},
    {0, "VMV%d(to,v,w)",       "(to)[%i] = (v)[%i] - (w)[%i]",  "", ""},
    {0, "MPM%d(to,m1,m2)",     "VPV%d((to)[%i], (m1)[%i], (m2)[%i])",  "", ""},
    {0, "MMM%d(to,m1,m2)",     "VMV%d((to)[%i], (m1)[%i], (m2)[%i])",  "", ""},
    {0, "SXV%d(to,s,from)",    "(to)[%i] = (s) * (from)[%i]",  "", ""},
    {0, "SXM%d(to,s,from)",    "SXV%d((to)[%i], s, (from)[%i])",  "", ""},
    {0, "MV%d(to,from)",       "(to)[%i] = -(from)[%i]",  "", ""},
    {0, "MM%d(to,from)",       "MV%d((to)[%i], (from)[%i])",  "", ""},
    {0, "VXS%d(to,from,s)",    "(to)[%i] = (from)[%i] * (s)",  "", ""},
    {0, "VDS%d(to,from,s)",    "(to)[%i] = (from)[%i] / (s)",  "", ""},
    {0, "MXS%d(to,from,s)",    "VXS%d((to)[%i], (from)[%i], s)",  "", ""},
    {0, "MDS%d(to,from,s)",    "VDS%d((to)[%i], (from)[%i], s)",  "", ""},
    {2, "MXM%d(to,m1,m2)",     "VXM%d((to)[%i], (m1)[%i], m2)",  "", ""},
    {1, "VXM%d(to,v,m)",       "(to)[%i] = _DOTcol%d(v, m, %i)",  "", ""},
    {1, "MXV%d(to,m,v)",       "(to)[%i] = DOT%d((m)[%i], v)",  "", ""},
    {0, "LERP%d(to,v0,v1,t)",  "(to)[%i]=(v0)[%i]+(t)*((v1)[%i]-(v0)[%i])",  "", ""},
    {0, "TRACE%d(m)",          "(m)[%i][%i]",          " "+"},
    {0, "DOT%d(v,w)",          "(v)[%i] * (w)[%i]",      " "+"},
    {0, "NORMSQRD%d(v)",       "(v)[%i] * (v)[%i]",      " "+"},
    {0, "DISTSQRD%d(v,w)",     "((v)[%i]-(w)[%i])*((v)[%i]-(w)[%i])",  " "+"},
    {0, "_DOTcol%d(v,m,j)",    "(v)[%i] * (m)[%i][j]",  " "+"},
    /* following two aren't really "safe", but shouldn't be used anyway */
    {0, "_SETcol%d(v,m,j)",    "(v)[%i] = (m)[%i][j]",  "", ""},
    {0, "_MXVcol%d(to,m,M,j)", "(to)[%i][j] = _DOTcol%d((m)[%i],M,j)",  "", ""},
    {0, "_DET%d(v%0..d-1,i%0..d-1)", "(v0)[i%0..d-1]*%_DET%d-1(v%1..d-1,i%~i)", " "+"},
    {1, "%XV%d(to,v%1..d-1)", "(to)[%i] = %-%_DET%d-1(v%1..d-1, %~i)", " "+"},
    /* careful! don't use v%0..d-1 for the above or the hack utility routines
       for making "safe" macros won't be able to find the first arg. */
};
```

```

/*
 * dimension-mixing macros for which d (= the dimension of the destination)
 * is the larger of the two dimensions. This is deduced from
 * the fact that the macro name contains %d-1.
 */
{0, "SET%dfrom%d-1(to,from,pad)", "(to)[%i] = (from)[%i]", ",",
    "(to)[%i] = (pad)"}},
{0, "SETMAT%dfrom%d-1(to,from,pad0,pad1)",
    "SET%dfrom%d-1((to)[%i], (from)[%i], pad0)", ",",
    "FILLVEC%d-1((to)[%i], (pad0)), (to)[%i][%i] = (pad1)"}},
{2, "M%d-1XM%d(to%d,m%d-1,m%d)",
    "_MXVcol%d-1(to%d,m%d-1,m%d,%i), (to%d)[%d-1][%i]=(m%d)[%d-1][%i]", ",",
    "VXM%d-1((to%d)[%i],(m%d)[%i],m%d-1), (to%d)[%i][%d-1]=(m%d)[%i][%d-1]", ",",
    "VXM%d-1((to%d)[%i],(m%d)[%i],m%d-1), (to%d)[%i][%d-1]=(m%d)[%i][%d-1]", ",",

/*
 * dimension-mixing macros for which d (= the dimension of the destination)
 * is the smaller of the two dimensions. This is deduced from
 * the fact that the macro name contains %d+1.
 */
{1, "V%dXM%d+1(to%d,v%d,m%d+1)",
    "(to%d)[%i] = _DOTcol%d(v%d,m%d+1,%i) + (m%d+1)[%d][%i]", ",",
    "M%d+1XV%d(to%d,m%d+1,v%d)",
    "(to%d)[%i] = DOT%d((m%d+1)[%i],v%d) + (m%d+1)[%i][%d]", ",",

/*
 * definitions that don't get vector-expanded.
 * This is deduced from the fact that the "sep" field is empty.
 */
{0, "_DET1(v0,i0)", "(v0)[i0]"},
{0, "%VXV%d(v%0..d-1)", "_DET%d(v%0..d-1,%0..d-1)"},
{0, "DET%d(m)", "%VXV%d((m)[%0..d-1])"},

/*
 * New adjoint stuff.
 */
{2, "ADJOINT%d(to,m)", "%__ADJOINTcol%d(to,%i,m,%~i)", ",",
    "_ADJOINTcol%d(to,col,m,i%1..d-1)",
    "(to)[%i][col] = %-_DET%d-1(m[i%1..d-1], %~i)", ",",
    "__ADJOINTcol%d(to,col,m,i%1..d-1)",
    "(to)[%i][col] = %--_DET%d-1(m[i%1..d-1], %~i)", ",",
};

#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <ctype.h>
#include <string.h>
#define MIN(a,b) ((a)<(b)?(a):(b))
#define MAX(a,b) ((a)>(b)?(a):(b))
#define numberof(sextoys) (sizeof(sextoys) / sizeof(*(sextoys)))

/*
 * s is pointing to the last char of the prefix;
 * to a space, ',' or a '(', taking into account nesting parens
 */
char *getprefix(s)
char *s;
{
    static char buf[100];

```

```
char *bufptr = buf + sizeof(buf);
int i, nest = 0;
*--bufptr = 0;
for (i = 0; nest || !strchr(" ,(", s[i]); --i) {
    nest += (s[i] == ')') - (s[i] == '(');
    *--bufptr = s[i];
}
return bufptr;
}

/*
 * s is pointing to the beginning of the suffix;
 * get the suffix up to but not including a space, ',' or ')'.
 */
char *getsuffix(s)
char *s;
{
    static char buf[100];
    int i;
    for (i = 0; !strchr(" ,)", s[i]); ++i)
        buf[i] = s[i];
    buf[i] = 0;
    return buf;
}

/*
 * Print the string s with the following substitutions:
 * %d turns into d
 * %i turns into i
 * %- turns into "-" if i is odd, " " otherwise
 * %-- turns into "--" if i is even, " " otherwise
 * %_ turns into "_" if i is odd, " " otherwise
 * %%- turns into "--" if d+i is even, " " otherwise (sorry this is stupid)
 * %i+1 turns into (i+1)%d etc.
 * %d-1 turns into d-1 etc.
 * %% turns into %
 * blah%0..d-1bleh turns into blah0bleh,blah1bleh,...,blahd-1bleh
 * blah%~ibleh is same as blah%0..d-1bleh but excluding i
 * %XV turns into VXVX...XV with d-1 V's (or XV if d==2)
 * %VXV turns into VXVX...XV with d V's
 */
void printformatted_to(s, tochar, d, i)
char *s;
int tochar, d, i;
{
    char *prefix, *suffix;
    int num, j, lastj;

    for (; *s != tochar; s++) {
        if (*s == '%') {
            ++s;

            prefix = NULL;
            if (isdigit(*s) && !strncmp(s+1, "..d-1", 5)) {
                prefix = getprefix(s-2); /* one char before the % */
                suffix = getsuffix(s+6);
            } else if (!strncmp(s, "~i", 2)) {
                prefix = getprefix(s-2); /* one char before the % */
                suffix = getsuffix(s+2);
            }
            if (prefix) {
```

```
        lastj = d-1 - (*s == '~');
        for (j = atoi(s); j <= lastj; ++j) {
            printf("%d", j + (*s == '~' && j >= i));
            printf("%s", suffix);
            if (j < lastj)
                printf(",%s", prefix);
        }
        s += (*s == '~' ? 2 : 6) + strlen(suffix) - 1;
    } else {
        switch(*s) {
            case '-': case '_':
                if (!strcmp(s, "-%", 3)) {
                    printf("%c", (i+d)%2==0 ? *s : ' ');
                    s += 2;
                } else if (!strcmp(s, "--", 2)) {
                    printf("%c", i%2==0 ? *s : ' ');
                    s++;
                } else {
                    printf("%c", i%2==1 ? *s : ' ');
                }
                break;
            case 'i': case 'd':
                num = (*s == 'i' ? i : d);
                if (strchr("+-", s[1])) {
                    num += (s[1] == '-' ? -1 : 1) * atoi(s+2);
                    if (*s == 'i')
                        num = (num+d) % d;
                    s += 2;
                    /* s is now pointing to the first digit */
                    while (isdigit(s[1]))
                        s++;
                    /* s is now pointing to the last digit */
                }
                printf("%d", num);
                break;
            case 'X': case 'V':
                if (*s == 'X' && d == 2)
                    printf("X");
                for (j = 0; j < d-2; ++j)
                    printf("VX");
                if (*s == 'V')
                    printf("V");
                break;
            case '%':
                printf("%%");
                break;
            default:
                assert(0);
        }
    }
} else
    putchar(*s);
}
```

```
void printformatted(s, d, i)
char *s;
int d, i;
{
    printformatted_to(s, '\\0', d, i);
}
```

```
void define(d, name_and_args, all_but_last, sep, last)
int d;
char *name_and_args, *all_but_last, *sep, *last;
{
    int i;

    printf("#define ");
    printf(name_and_args, d, 0);
    printf("\t\\\n\t\t");
    printf("(");

    if (sep)
        for (i = 0; i < d-1; ++i) {           /* loop for all but last */
            printf(all_but_last, d, i);
            printf(sep, d, i);
            printf(" \\\n\t\t");
        }
    printf(last, d, i);

    printf(")\n");
}

int is_substring(a,b)
char *a, *b;
{
    for (; *b; b++)
        if (strncmp(a, b, strlen(a)) == 0)
            return 1;
    return 0;
}

/*
 * Hack utility routines...
 */
void print_name_and_args_with_stuff_inserted(name_and_arg, suff, arg0, d)
char *name_and_arg, *suff, *arg0;
int d;
{
    printf(name_and_arg, '(', d, 0);
    printf(suff, d, 0);
    printf("(");
    if (arg0) {
        printf(arg0, d, 0);
        printf(",");
    }
    printf(strchr(name_and_arg, '(' /*)*/ ) + 1, d, 0);
}

void print_name_and_args_with_first_arg_changed(name_and_arg, arg0, d)
char *name_and_arg, *arg0;
int d;
{
    printf(name_and_arg, '(', d, 0);
    printf("(");
    printf(arg0, d, 0);
    printf(strchr(name_and_arg, ','), d, 0);
}

void print_first_arg(name_and_arg, d)
```

```
char *name_and_arg;
int d;
{
    printf("formatted_to(strchr(name_and_arg, '(' /*)*/ ) + 1, ',', d, 0);
}

#define MINDIM 2

main(argc, argv)
char **argv;
{
    int i, d, maxdim;

    if (argc != 2 || ! (maxdim = atoi(argv[argc-1])))
        fprintf(stderr, "Usage: %s [<maxdim>]\n", argv[0]), exit(1);

    printf("/*\n");
    for (i = 0; intro[i]; ++i) {
        printf(" *");
        printf("formatted(intro[i], maxdim, 0);
        printf("\n");
    }
    printf(" */\n\n");

    printf("#ifndef VEC_H\n");
    printf("#define VEC_H %d\n", maxdim);
    printf("#include <math.h> /* for definition of floor() */\n");

    for (d = MINDIM; d <= maxdim; ++d)
        for (i = 0; i < numberof(defs); ++i) {
            if (is_substring("%d-1", defs[i].name_and_args) && d-1 < MINDIM
                || is_substring("%d+1", defs[i].name_and_args) && d+1 > maxdim)
                continue;
            if (!is_substring("%d", defs[i].name_and_args) && d != MINDIM)
                continue; /* don't redefine if it's the same */
            define(d, defs[i].name_and_args,
                defs[i].all_but_last,
                defs[i].sep,
                defs[i].last ? defs[i].last : defs[i].all_but_last);
        }
    for (d = MINDIM; d <= maxdim; ++d)
        for (i = 0; i < numberof(defs); ++i) {
            if (is_substring("%d-1", defs[i].name_and_args) && d-1 < MINDIM
                || is_substring("%d+1", defs[i].name_and_args) && d+1 > maxdim)
                continue;
            if (!is_substring("%d", defs[i].name_and_args) && d != MINDIM)
                continue; /* don't redefine if it's the same */
            if (defs[i].safety > 0) {
                int t;
                printf("#define ");
                print_name_and_args_with_stuff_inserted(defs[i].name_and_args,
  "safe", "type", d);

                printf(" \\n\t\t");
                printf("formatted(\"do {type _vec_h_temp_[%d]\", d, 0);
                if (defs[i].safety > 1)
                    printf("formatted(\"[%d]\", d, 0);
                printf("; \\n\t\t");

                print_name_and_args_with_first_arg_changed(
```

```
        defs[i].name_and_args, "_vec_h_temp_", d);
printf("; \\n\\t\\t    ");

if (defs[i].safety > 1)
    printf("SETMAT%d(", d, 0);
else
    printf("SET%d(", d, 0);
print_first_arg(defs[i].name_and_args, d);
printf(", _vec_h_temp_); \\n\\t\\t} while (0)\\n");

for (t = 0; t < numberof(types); ++t) {
    printf("#define ");
    print_name_and_args_with_stuff_inserted(
        defs[i].name_and_args, types[t].abbr, (char *)NULL, d);
    printf(" ");
    print_name_and_args_with_stuff_inserted(
        defs[i].name_and_args, "safe", types[t].full, d);
    printf("\\n");
}
}
```

```
printf("#endif /* VEC_H */\\n");
```

```
return 0;
```

```
}
```