

THE RAY TRACING NEWS

Volume 2, Number 2

"Light Makes Right"

June 1988

- | | | |
|----|--|------------------------------|
| 2 | Rectangular Bounding Volumes
for Popular Primitives | Ben Trumbore |
| 3 | Linear-time Voxel Walking
for Octrees | Jim Arvo |
| 6 | Exposure Keys for the
Digital Darkroom | Andrew Glassner |
| 8 | A PostScript Ray Tracer | John Hartman & Paul Heckbert |
| 11 | A Ray Tracing Bibliography | Paul Heckbert & Eric Haines |
| 17 | Solution to Last Issue's Puzzle | Eric Haines |
| 18 | Interactive SIMD Ray Tracing | Russ Tuck |

All contents are copyright © 1988 by the individual authors and Ray Tracing News. Letters to the editor are welcome, and will be considered for publication unless otherwise requested. This issue was sponsored by the Walkthrough Project at UNC-CH, Principal Investigator Dr. Frederick P. Brooks, Jr., NSF Grant CCR-8609588. Please direct all correspondence and submissions to the editor.

Andrew S. Glassner
Editor, The Ray Tracing News
Department of Computer Science
University of North Carolina
Chapel Hill, NC 27599-3175 USA
(919) 962-1803
glassner@cs.unc.edu

Rectangular Bounding Volumes for Popular Primitives

by Ben Trumbore
(wbt@beauty.tn.cornell.edu)

Many efficient ray tracing algorithms require each primitive object in an environment to be bounded by a rectangular volume. While experienced ray tracers have certainly written code to derive tight fitting volumes, many implementors must resort to less efficient bounding techniques. As a reference for those ray tracers who are uncomfortable with differential geometry, this article derives minimal orthogonal rectangular bounding volumes for some popular types of primitives. These derivations could easily be adapted for square bounding volumes or slabs.

The methods described here assume that each object in an environment is derived by transforming some primitive object by a cumulative transformation matrix (CTM). We transform the component parts of these primitives using the CTM and determine the extrema of the transformed components. There are three types of components that concern us: points, circles, and surfaces.

Points are the easiest component to work with. Cubes, prisms, and polygonal objects can be bounded by a convex hull that uses some or all of their vertices. It is simple to take all of the vertices of such a primitive and transform them by a CTM. These transformed points can be checked for extrema in X, Y, and Z to find a rectangular bounding volume.

Cylinders and cones contain circle components. By transforming the circles at both ends of a cylinder and finding the extrema of those circles, we can determine a tight bounding volume. For a cylinder defined as the extrusion of a circle of radius 1.0 along the Y axis from -1.0 to 1.0, our circles J_{top} and J_{bot} have components:

$$J_{top}(t) = [\cos t, 1, \sin t, 1], \quad 0 \leq t \leq 2\pi$$

$$J_{bot}(t) = [\cos t, -1, \sin t, 1]$$

Define our CTM to be:

$$M = \begin{bmatrix} A & B & C & 0 \\ D & E & F & 0 \\ G & H & I & 0 \\ T_x & T_y & T_z & 1 \end{bmatrix}$$

Our transformed circles K_{top} and K_{bot} are defined:

$$K_{top}(t) = J_{top}(t) M = \begin{bmatrix} A \cos t + D + G \sin t + T_x \\ B \cos t + E + H \sin t + T_y \\ C \cos t + F + I \sin t + T_z \\ 1 \end{bmatrix}^t$$

$$K_{bot}(t) = J_{bot}(t) M = \begin{bmatrix} A \cos t - D + G \sin t + T_x \\ B \cos t - E + H \sin t + T_y \\ C \cos t - F + I \sin t + T_z \\ 1 \end{bmatrix}^t$$

Taking the derivative of these equations yields:

$$\frac{dK_{top}}{dt} = \frac{dK_{bot}}{dt} = \begin{bmatrix} G \cos t - A \sin t \\ H \cos t - B \sin t \\ I \cos t - C \sin t \\ 0 \end{bmatrix}^t$$

This equation can be used to find the values of t where the circle has extremes in X, Y, or Z:

X extrema: $t = \text{atan2}(G, A)$

Y extrema: $t = \text{atan2}(H, B)$

Z extrema: $t = \text{atan2}(I, C)$

By substituting these values for t back into the equations for K_{top} and K_{bot} we can find the transformed points that determine our bounding volume. Obviously, a cone's bounding volume involves analyzing the circular component formed by its base and the point component formed by its apex.

A sphere is a simple example of a surface component. When a sphere is transformed by a general CTM, it may become an ellipsoid aligned along any axis. A unit sphere of radius 1.0 centered at the origin is defined:

$$J(u, v) = [\cos u \cos v, \sin v, \sin u \cos v, 1]$$

$$0 \leq u \leq 2\pi, \quad -\frac{\pi}{2} \leq v \leq \frac{\pi}{2}$$

Transforming this sphere by the matrix M we get K:

$$K(u, v) = J(u, v)M =$$

$$\begin{bmatrix} A \cos u \cos v + D \sin v + G \sin u \cos v \\ B \cos u \cos v + E \sin v + H \sin u \cos v \\ C \cos u \cos v + F \sin v + I \sin u \cos v \\ 1 \end{bmatrix}^t$$

We are interested in finding the u and v parameter values for the extrema on this surface. First the partial derivatives of K are found:

$$\frac{\partial K}{\partial u} = \begin{bmatrix} \cos v (G \cos u - A \sin u) \\ \cos v (H \cos u - B \sin u) \\ \cos v (I \cos u - C \sin u) \\ 0 \end{bmatrix}^t$$

$$\frac{\partial K}{\partial v} = \begin{bmatrix} D \cos v - \sin v (A \cos u + G \sin u) \\ E \cos v - \sin v (B \cos u + H \sin u) \\ F \cos v - \sin v (C \cos u + I \sin u) \\ 0 \end{bmatrix}^t$$

The extrema are found by simultaneously finding the roots of one dimension of the partial derivatives. To find the X extrema, for example, solve:

$$0 = \left(\frac{\partial K}{\partial u}\right)_x = \cos v (G \cos u - A \sin u)$$

$$0 = \left(\frac{\partial K}{\partial v}\right)_y = D \cos v - \sin v (A \cos u + G \sin u)$$

$$u = \text{atan2}(G, A)$$

$$v = \text{atan2}(D, A \cos u + G \sin u)$$

Having found this one X extreme, add or subtract π to u and negate v to find the other X extreme. This is repeated for the Y and Z dimensions. Each of these u, v locations is then substituted into $Y(u, v)$ to find the point used to generate the bounding volume.

Finally, define a torus as a vertical circle of radius R that is revolved around the Y axis. The circle's center sweeps out a circle of radius Q in the XZ plane, and $R + Q = 1$. The equation of such a surface is:

$$J(u, v) = \begin{bmatrix} (R + Q \cos v) \cos u \\ Q \sin v \\ (R + Q \cos v) \sin u \\ 1 \end{bmatrix}^t$$

It happens that the u, v coordinates of tori extrema are found using exactly the same equations as used for a sphere. These coordinates are then substituted into $K(u, v) = J(u, v) M$. Alternatively, the u, v coordinates can be found using a circular component of the torus. To do this, transform the horizontal circle J of the torus by M to give K :

$$J(u) = [R \cos u, 0, R \sin u, 1]$$

$$K(u) = J(u) M$$

At each of the dimensional extrema of the circle K , Q can be added or subtracted to find points that will determine the rectangular bounding volume. •

Linear-time Voxel Walking for Octrees

by Jim Arvo

(rutgers!umix!apollo!arvo)

Here is a new way to attack the problem of "voxel walking" in octrees (at least I think it's new). By voxel walking I mean identifying the successive voxels along the path of a ray. This is more for theoretical interest than anything else, though the algorithm described below may actually be practical in some situations. I make no claims about the practicality, however, and stick to theoretical time complexity for the most part.

For this discussion assume that we have recursively subdivided a cubical volume of space into a collection of equal-sized voxels using a BSP tree – i.e. each level imposes a single axis-orthogonal partitioning plane. The algorithm is much easier to describe using BSP trees, and from the point of view of computational complexity, there is basically no difference between BSP trees and octrees. Also, assuming that the subdivision has been carried out to uniform depth throughout simplifies the discussion, but is by no means a prerequisite. This would defeat the whole purpose because we all know how to efficiently walk the voxels along a ray in the context of uniform subdivision — i.e. use a 3DDDA.

Assuming that the leaf nodes form an $N \times N \times N$ array of voxels, any given ray will pierce at most $O(N)$ voxels. The actual bound is something like $3N$, but the point is that it's linear in N . Now, suppose that we use a "re-traversal" technique to move from one voxel to the next along the ray. That is, we create a point that is guaranteed to lie within the next voxel and then traverse the hierarchy from the root node until we find the leaf node, or voxel, containing this point. This requires $O(\log N)$ operations. In real life this is quite insignificant, but here we are talking about the actual time complexity. Therefore, in the worst case situation of following a ray through $O(N)$ voxels, the "re-traversal" scheme requires $O(N \log N)$ operations just to do the "voxel walking." Assuming that there is an upper bound on the number of objects in any voxel (i.e. independent of N), this is also the worst case time complexity for intersecting a ray with the environment.

In this note I propose a new "voxel walking" algorithm for octrees (call it the "partitioning" algorithm for now) which has a worst case time complexity of $O(N)$ under the conditions outlined above. In the best case of finding a hit "right away" (after $O(1)$ voxels), both "re-traversal" and "partitioning" have a time complexity of $O(\log N)$. These results are summarized in Table 1.

The new algorithm proceeds by recursively partitioning the ray into little line segments which intersect the leaf

voxels. The top-down nature of the recursive search ensures that partition nodes are only considered *once per ray*. In addition, the voxels will be visited in the correct order, thereby retaining the $O(\log N)$ best case behavior.

	Best case: $O(1)$ voxels searched before a hit	Worst case: $O(N)$ voxels searched before a hit
Re-traversal	$O(\log N)$	$O(N \log N)$
Partitioning	$O(\log N)$	$O(N)$

Table 1

Algorithm 1 gives a pseudo code description of the "partitioning" algorithm. It is the routine for intersecting a ray with an environment which has been subdivided using a BSP tree. Little things like checking to make sure the intersection is within the appropriate interval have been omitted. The input arguments to this routine are:

Node : A BSP tree node which comes in two flavors — a partition node or a leaf node. A partition node defines a plane and points to two child nodes which further partition the "positive" and "negative" half-spaces. A leaf node points to a list of candidate objects.

P : The ray origin. Actually, think of this as an endpoint of a 3D line segment, since we are constraining

the "ray" to be of finite length.

D : A unit vector indicating the ray direction.

len : The length of the "ray" — or, more appropriately, the line segment. This is measured from the origin, P, along the direction vector, D.

The function `Intersect` is initially passed the root node of the BSP tree, the origin and direction of the ray, and a length, `len`, indicating the maximum distance to intersections which are to be considered. This starts out being the distance to the far end of the original bounding cube.

As the BSP tree is traversed, the line segments are chopped up by the partitioning nodes. The "shrinking" of the line segments is critical to ensure that only relevant branches of the tree will be traversed.

The actual encodings of the intersection data, the partitioning planes, and the nodes of the tree are all irrelevant to this discussion. These are "constant time" details. Granted, they become exceedingly important when considering whether the algorithm is really practical. Let's save this for later.

A naive (and incorrect) proof of the claim that the time complexity of this algorithm is $O(N)$ would go something like this:

The voxel walking that we perform on behalf of a single ray is really just a search of a binary tree with voxels at the leaves. Since each node is only processed once, and since a binary tree with k leaves has $k-1$ internal nodes, the total number of nodes which are processed in

Algorithm 1

```

FUNCTION Intersect( Node, P, D, len )
  RETURNING results of intersection
  IF Node is NIL THEN RETURN( no intersection )
  IF Node is a leaf THEN BEGIN
    intersect ray (P,D) with objects in the candidate list
    RETURN( the closest resulting intersection )
  END IF
  dist := signed distance along ray (P,D) to plane defined by Node
  near := child of Node in half-space which contains P
  IF 0 < dist < len THEN BEGIN / the interval intersects the plane
    hit_data := Intersect( near, P, D, dist )
    IF hit_data <> "no intersection" THEN RETURN( hit_data )
    Q := P + dist * D / 3D coords of point of intersection
    far := child of Node in half-space which does NOT contain P
    RETURN( Intersect( far, Q, D, len - dist ) )
  END IF
  ELSE RETURN( Intersect( near, P, D, len ) )
  END
  
```

the entire operation must be of the same order as the number of leaves. We know that there are $O(N)$ leaves. Therefore, the time complexity is $O(N)$.

But wait! The tree that we search is not truly binary since many of the internal nodes have one NIL branch. This happens when we discover that the entire current line segment is on one side of a partitioning plane and we prune off the branch on the other side. This is essential because there are really N^3 leaves and we need to discard branches leading to all but $O(N)$ of them. Thus, k leaves does not imply that there are only $k-1$ internal nodes. The question is, "Can there be more than $O(k)$ internal nodes?"

Suppose we were to pick N random voxels from the N^3 possible choices, then walk up the BSP tree marking all the nodes in the tree which eventually lead to these N leaves. Let's call this the subtree "generated" by the original N voxels. Clearly this is a tree and it's uniquely determined by the leaves. A very simple argument shows that the generated subtree can have as many as $2(N-1)\log N$ nodes. This puts us right back where we started from, with a time complexity of $O(N \log N)$, even if we visit these nodes only once. This makes sense, because the "re-traversal" method, which is also $O(N \log N)$, treats the nodes as though they were unrelated. That is, it does not take advantage of the fact that

paths leading to neighboring voxels are likely to be almost identical, diverging only very near the leaves. Therefore, if the "partitioning" scheme really does visit only $O(N)$ nodes, it does so because the voxels along a ray are far from random. It must implicitly take advantage of the fact that the voxels are much more likely to be brothers than distant cousins.

This is in fact the case. To prove it I found that all I needed to assume about the voxels was connectedness — provided I made some assumptions about the "niceness" of the BSP tree. To give a careful proof of this is very tedious, so I'll just outline the strategy (which I *think* is correct). But first let's define a couple of convenient terms:

1) Two voxels are "connected" (actually "26-connected") if they meet at a face, an edge, or a corner. We will say that a collection of voxels is connected if there is a path of connected voxels between any two of them.

2) A "regular" BSP tree is one in which each axis-orthogonal partition divides the parent volume in half, and the partitions cycle: X, Y, Z, X, Y, Z, etc. (Actually, we can weaken both of these requirements considerably and still make the proof work. If we're dealing with "standard" octrees, the regularity is automatic.)

Here is a sequence of little theorems which leads to

Algorithm 2

```

FUNCTION BSP_Intersect( Ray, Node, min, max )
  RETURNING intersection results
BEGIN
  IF Node is NIL THEN RETURN( no intersection )
  IF Node is a leaf THEN BEGIN / Do the real intersection checking
    intersect Ray with each object in the candidate list discarding
      those farther away than max.
  RETURN( the closest resulting intersection )
  END IF
  dist := signed distance along Ray to plane defined by Node
  near := child of Node for half-space containing the origin of Ray
  far := the "other" child of Node - i.e. not equal to near.
  IF dist > max OR dist < 0 THEN / Whole interval is on near side
    RETURN( BSP_Intersect( Ray, near, min, max ) )
  ELSE IF dist < min THEN / Whole interval is on far side
    RETURN( BSP_Intersect( Ray, far, min, max ) )
  ELSE BEGIN / the interval intersects the plane
    hit_data := BSP_Intersect( Ray, near, min, dist ) / Test near side
    IF hit_data indicates that there was a hit THEN RETURN( hit_data )
    RETURN( BSP_Intersect( Ray, far, dist, max ) ) / Test far side
  END IF
END
  
```

the main result:

THEOREM 1: A ray pierces $O(N)$ voxels.

THEOREM 2: The voxels pierced by a ray form a connected set.

THEOREM 3: Given a collection of voxels defined by a "regular" BSP tree, any connected subset of K voxels generates a unique subtree with $O(K)$ nodes.

THEOREM 4: The "partitioning" algorithm visits exactly the nodes of the subtree generated by the voxels pierced by a ray. Furthermore, each of these nodes is visited exactly once per ray.

THEOREM 5: The "partitioning" algorithm has a worst case complexity of $O(N)$ for walking the voxels pierced by a ray.

Theorems 1 and 2 are trivial. With the exception of the "uniqueness" part, theorem 3 is a little tricky to prove. I found that if I completely removed either of the "regularity" properties of the BSP tree (as opposed to just weakening them), I could construct a counterexample. I think that theorem 3 is true as stated, but I don't like my "proof" yet. I'm looking for an easy and intuitive proof. Theorem 4 is not hard to prove at all. All the facts become fairly clear if you see what the algorithm is doing. Finally, theorem 5, the main result, follows immediately from theorems 1 through 4.

Some Practical Matters

Since $\log N$ is typically going to be very small — bounded by 10, say — this whole discussion may be purely academic. However, just for the heck of it, I'll mention some things which could make this a (maybe) competitive algorithm for real-life situations (in as much as ray tracing can ever be considered to be "real life").

First of all, it would probably be advisable to avoid recursive procedure calls in the "inner loop" of a voxel walker. This means maintaining an explicit stack. At the very least one should "longjump" out of the recursion once an intersection is found.

The calculation of `dist` is very simple for axis-orthogonal planes, consisting of a subtract and a multiply (assuming that the reciprocals of the direction components are computed once up front, before the recursion begins).

A nice thing which falls out for free is that arbitrary partitioning planes can be used if desired. The only penalty is a more costly distance calculation. The rest of the algorithm works without modification. There may be some situations in which this extra cost is justified.

Algorithm 2 presents a slightly improved version of the algorithm. It turns out that you never need to explicitly compute the points of intersection with the partitioning planes. This makes it a little more attractive. •

Exposure Keys for the Digital Darkroom

by Andrew Glassner
(glassner@unc.cs.edu)

Ray tracers illuminate virtual objects with virtual lights, and then image those objects through a virtual lens in a virtual camera. And then we typically record onto a model of real film! By "a model of real film" I mean an image storage medium with predefined minimum sensitivity and saturation points.

This often isn't disastrous. If we make an image and it's too dark, we can just turn up the monitor brightness and contrast. But sometimes we build an image and all the pixels have RGBs in the teens at highest. Stretching the histogram can give us more range, but the basic quantization of colors is already done and irreversible; the best solution is to re-render with more or brighter virtual illumination in the scene.

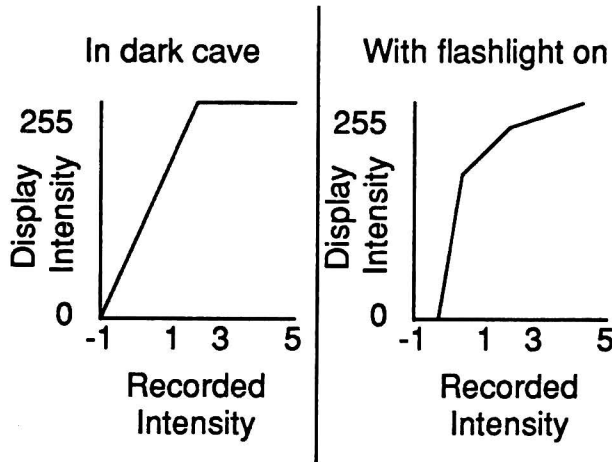
For a single image improper illumination might be no more than inconvenient. But consider an animated scene, where we set up the rendering system and go home for the weekend. What level of illumination do we choose? And how do we choose our virtual camera's aperture? And what should be the "speed" of our virtual film? These are hard questions, and typically not questions that I want to think about answering.

Makers of real films *must* address these questions. Imagine a scene where a character is exploring a dark cave with a flashlight. When the flashlight is off, we want to see enough of what's going on that we can see the character moving around, avoiding various rock formations; this argues for film of high sensitivity or speed (i.e. very responsive to light) and a wide aperture. But then the flashlight comes on, and starts sweeping over the floor, which is littered with shiny, reflective gold coins. Unless we adjust the aperture or the film speed, we're going to wind up with overexposed film. To find the right combination of aperture and speed in each situation requires careful planning and measurement (typically one cannot change the film speed in the middle of a scene, so only the aperture can be adjusted). If the exposures are wrong, the scene must be re-shot.

The analogy to our computer-generated scenes is direct if we clip our final colors (usually computed in "raw" RGB) to the lower and upper limits of 0 and 1 before storing them on disk. In effect, this says that our final recording medium will not image any light with an absolute intensity less than 0 units or greater than 1 unit. Why should we place this arbitrary restriction on our recording media? If we overexpose an entire animated scene, we may get back intensities for every pixel for every frame between 5 and 9 units. If we clip to 1 before storing, then the whole scene comes out white. If we save the raw RGB in its complete dynamic range, then we can adjust the final intensities after the rays have been cast.

What I'm arguing for is one component in a digital darkroom, where we can take the intensity values recorded onto very wide-range negatives and map them into appropriate values for a display device.

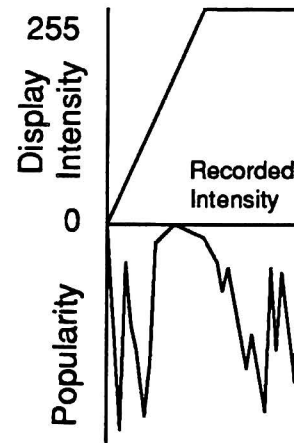
We might describe the mapping for just a single frame; this is helpful for creating images when you're not sure just how much light to use before rendering. For animations, the mapping may change from frame to frame. So for the flashlight in the cave scene, we could map low intensities to the full range of the display before the flashlight comes on, and then change the mapping when the flashlight comes on:



So the scheme is simple: a linearly-interpolated mapping from recorded intensity to display intensity. I use luminance as a measure of intensity, and scale the RGB vector to match the desired luminance. For an animated sequence, I linearly interpolate these maps, which is why I call each one an exposure key. The essential difference between this idea and regular histogram manipulation is that the input data is deliberately kept with the value with which it is computed; as opposed to the typical practice of clamping RGB to 0 and 1.

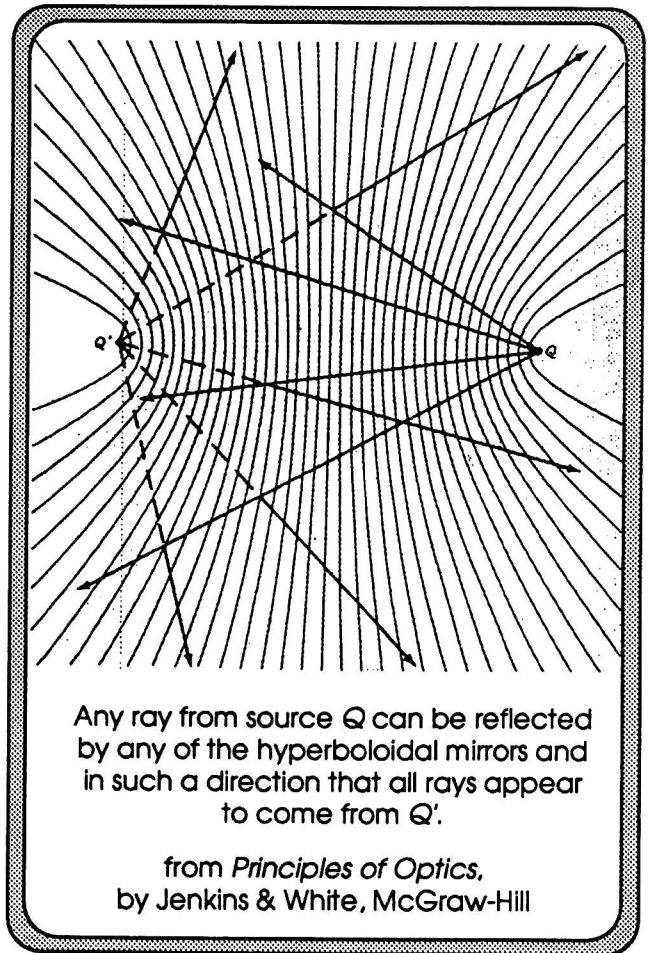
There's no reason to arbitrarily avoid negative values. A common idea (though unpublished, to my knowledge) is to model digital darklights into a scene. These are the opposite of spotlights; they are light sources that simply emit negative light! Although there is no physical justification for such a device, it's an intriguing new tool that computer graphics puts at our disposal. If I focus a powerful darklight onto an object lit only by a weak (regular) light source, then the light striking film that images that object will have negative intensity. If we clamp to a minimum of 0, then we can't get blacker than black; i.e. the region in the darklight is no darker than the surrounding world. If we allow the light intensity to drop below 0 we can let these different areas appear different if we want.

One of the best advantages of exposure keys is that they can be applied interactively after the exposure. Sometimes it's useful to show the intensity histogram in the key:



I propose it would be helpful to have a sequence of key frames on the color monitor, and the corresponding sequence of exposure keys on the animator's workstation. One could adjust the exposure keys until the animation looks good, and then interpolate keys for inbetween frames. Note that the frames chosen for exposure control need not have any relation to which rendered frames are selected from the animation.

Exposure keys can be easily extended to other optical processing, such as colored filters and special effects. •



A Postscript Ray Tracer

by John H. Hartman and Paul Heckbert
 (jhartman@emie.berkeley.edu, ph@miro.berkeley.edu)

Ever worry about all those cycles that are going to waste every night when you shut off your laserwriter? Well, now you can put them to good use. Introducing the world's first PostScript ray tracer. Just start it up, wait a (long) while, and out comes an image that any true graphics type would die laughing over. As it is currently set up it will trace a scene with three spheres and two lights. The image resolution is 16x16 pixels. Warning: the code is a real kludge. I'm not sure there is much you can change without breaking it, but you're welcome to try. If, by chance, you are able to improve the running time please send us the improved version.

This is a PostScript ray tracer. It is not a toy - don't let the kids play with it. Features include: shadows, recursive specular reflection and refraction, and colored surfaces and lights (bet you can't tell!). To use this thing just send it to your favorite Postscript printer. Then take a long nap/walk/coffee break/vacation. Running time for a recursive depth of 3 and a size of 16x16 is about 1000 seconds (roughly 20 minutes) or 4 seconds/pixel. There are a few parameters at the beginning of the file that you can change. The rest of the code is pretty indecipherable. It is translated from a C program written by Paul Heckbert, Darwyn Peachey, and Joe Cychosz for the Minimal Ray Tracing Programming Contest in comp.graphics in May 1987. Some changes have been made to improve the running time. Don't even bother trying to figure out how this works if you are looking for a good example of a ray tracer.

Have fun.

(*Editor's note:* If you don't want to type this in, you can retrieve it from the usenet newsgroup comp.graphics. If you can't do that, send email to John or Paul for a copy.)

```
%!
% Copyright (c) 1988 John H. Hartman and Paul Heckbert
%
% PostScript ray tracer
% This source may be copied, distributed, altered or used, but not sold for
% profit or incorporated into a product except under licence from the authors.
% This notice should remain in the source unaltered.
% John H. Hartman jhartman@ernie.Berkeley.EDU
% Paul Heckbert ph@miro.Berkeley.EDU
%
/starttime usertime def
/DEPTH 3 def % recursion depth
/SIZE 16 def % image resolution
/TIMEOUT SIZE dup mul 10 mul cvi 120 add def % approximately 10 sec/pixel
/NUM_SPHERES 5 def
/AOV 25.0 def % angle of view
/AMB [0.02 0.02 0.02] def % ambient light
% list of spheres/lights in scene
%
% x y z r g b rad kd ks kt kl ir
/SPHERES [[[ 0.0 6.0 0.5] [1.0 1.0 1.0] 0.9 0.05 0.2 0.85 0.0 1.7]
          [[-1.0 8.0 -0.5] [1.0 0.5 0.2] 1.0 0.7 0.3 0.0 0.05 1.2]
          [[ 1.0 8.0 -0.5] [0.1 0.8 0.8] 1.0 0.3 0.7 0.0 0.0 1.2]
          [[ 3.0 -6.0 15.0] [1.0 0.8 1.0] 7.0 0.0 0.0 0.0 0.6 1.5]
          [[-3.0 -3.0 12.0] [0.8 1.0 1.0] 5.0 0.0 0.0 0.0 0.5 1.5]
          ] def
statusdict begin TIMEOUT setjobtimeout /waittimeout TIMEOUT def end
/initpage { /Courier findfont 10 scalefont setfont } def
/X 0 def /Y 1 def /Z 2 def /TOL 5e-4 def
/BLACK [0.0 0.0 0.0] def /WHITE [1.0 1.0 1.0] def
/U 0.0 def /B 0.0 def
```



```

% index of fields in sphere array
/cen 0 def /col 1 def /rad 2 def /kd 3 def
/ks 4 def /kt 5 def /kl 6 def /ir 7 def
/NEG_SIZE SIZE neg def
/MATRIX [SIZE 0 0 NEG_SIZE 0 SIZE] def
/vec {3 array} def /VU vec def /vunit_a 0.0 def
% dot product, two arrays of three reals
/vdot {aload pop 4 -1 roll aload pop 4 -1 roll mul
      2 -1 roll 4 -1 roll mul add 3 -2 roll mul add} def
% vcomb, sa, a, sb, b returns new array of sa*a + sb*b
/vcomb { aload pop          4 -1 roll dup dup          5 1 roll 3 1 roll mul
        5 1 roll mul          4 1 roll mul 3 1 roll 5 -2 roll aload pop
        4 -1 roll dup dup          5 1 roll 3 1 roll mul 5 1 roll mul
        4 1 roll mul 3 1 roll 4 -1 roll add 5 1 roll   3 -1 roll add 4 1 roll
        add 3 1 roll          vec astore } def
/vsub {aload pop          4 -1 roll aload pop          4 -1 roll sub 5 1 roll
        3 -1 roll sub 4 1 roll   exch sub 3 1 roll          vec astore} def
/smul {aload pop          4 -1 roll dup dup          5 1 roll 3 1 roll mul
        5 1 roll mul          4 1 roll mul 3 1 roll vec astore} def
/vunit { /vunit_a exch store 1.0 vunit_a dup vdot sqrt div vunit_a smul} def
/grayscale {
  % convert to ntsc, then to grayscale
  0.11 mul exch 0.59 mul add exch 0.30 mul add 255.0 mul cvi} def
/intersect { % returns best, tmin, rootp
  7 dict begin /d exch def /p exch def /best -1 def /tmin 1e30 def
  /rootp 0 def
  0 1 NUM_SPHERES 1 sub {
    /i exch def /sphere SPHERES i get def/VU sphere cen get p vsub store
    /B d VU vdot store
    /U B dup mul VU dup vdot sub sphere rad get dup mul add storeU 0.0 gt
    { /U B U sqrt sub store U TOL lt
      { /U 2.0 B mul U sub store /B 1.0 store }
      { /B -1.0 store }
      ifelse
      U TOL ge U tmin lt and
      { /best i store /tmin U store /rootp B store } if
    } if
  } for best tmin rootp end
} def
/trace {
  13 dict begin /d exch def /p exch def /level exch def
  /saveobj save def /color AMB vec copy def /level level 1 sub store
  p d intersect /root exch def/v exch def /s exch def -1 s ne
  {
    /sphere SPHERES s get def/p 1.0 p v d vcomb store /n
    sphere cen get p root 0.0 lt { exch } if vsub vunit def
    sphere kd get 0.0 gt
    { 0 1 NUM_SPHERES 1 sub
      { /i exch def /light SPHERES i get def light kl get 0.0 gt
        { /VU light cen get p vsub vunit store /v light kl get
          n VU vdot mul store v 0.0 gt p VU intersect
          /B exch store /nd exch def i eq and
          { /color 1.0 color v light col get vcomb def } if
        } if
      } for
    } if
    color aload pop sphere col get aload vec copy /VU exch store
    4 -1 roll mul 5 1 roll 3 -1 roll mul 4 1 roll mul 3 1 roll
    color astore pop /nd d n vdot neg store /color sphere ks get
    sphere kd get color sphere kl get VU vcomb 0 level eq
  }

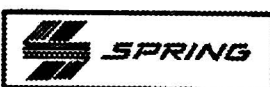
```

```

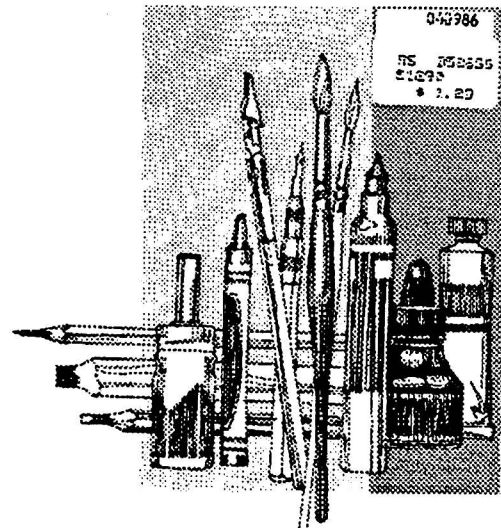
{ BLACK vec copy}
{ level p 1.0 d 2 nd mul n vcomb trace vec astore }
ifelse 1.0 3 -1 roll vcomb storeroot 0.0 gt
{ /v sphere ir get store }
{ /v 1.0 sphere ir get div store }
ifelse /U 1 v dup mul 1 nd dup mul sub mul sub store U 0.0 gt
{ /color 1.0 color sphere kt get 0 level eq
  { BLACK vec copy}
  { level p v d v nd mul U sqrt sub n vcomb trace vec astore }
  ifelse vcomb store
} if
} if color aload pop saveobj restore end
} def
/main {initpage /data SIZE dup mul string def
/half SIZE 0.5 mul def /i 0 def
/dy half AOV cvr 0.5 mul dup cos exch sin div mul def
/temp vec def 0 1 SIZE 1 sub
{ /y exch def 0 1 SIZE 1 sub
  { /x exch def data i /saveobj save def VU X x cvr half sub put
    VU Y dy put VU Z half y cvr sub put DEPTH BLACK VU vunit trace
    grayscale saveobj restore put /i i 1 add store
  } for
} for gsave
% coarsen halftone screen to eliminate contouring you get with default setscreen of 50
currentscreen 3 -1 roll pop 18 3 1 roll setscreen
100 300 translate 400 400 scale SIZE SIZE 8 MATRIX {data} image
grestore 100 200 moveto (Statistics: ) show 100 190 moveto
(Size: ) show SIZE 10 string cvs show 100 180 moveto
(Depth: ) show DEPTH 3 string cvs show 100 170 moveto
(Running time: ) show usertime starttime sub 1000 div 20 string cvs show
showpage
} def /main load bind main stop

```


Low-cost hardware support at last!



QUALITY Tracing Tablet



40 Sheets
 12 in. x 9 in.
 305 mm x 229 mm
 No. 51290



Rising Spring • Top Scholar—Rising Spring, Pa. 18879

A Ray Tracing Bibliography

by Paul Heckbert and Eric Haines

(ph@miro.berkeley.edu,
hpfcrs!eyelerich@hplabs.hp.com)

- Amanatides, John, Alain Fournier, "Ray Casting using Divide and Conquer in Screen Space", *Intl. Conf. on Engineering and Computer Graphics*, Beijing, China, Aug. 1984, similar to SIGGRAPH '84 paper but more emphasis on recursive screen subdivision, extents, {screen subdivision, bounding volume}
- Amanatides, John, "Ray Tracing with Cones", *Computer Graphics (SIGGRAPH '84 Proceedings)*, vol. 18, no. 3, July 1984, pp. 129-135, ray tracing spheres and polygons with circular conical rays, {cone tracing, antialiasing}
- Amanatides, John, "Ray Tracing with Cones", *Proceedings of Graphics Interface '84*, May 1984, pp. 97-98, brief summary of his SIGGRAPH paper, {cone tracing}
- Amanatides, John, "A Fast Voxel Traversal Algorithm for Ray Tracing", *Eurographics '87*, North-Holland, Amsterdam, uniform grid space subdivision
- Appel, Arthur, "Some Techniques for Shading Machine Renderings of Solids", *AFIPS 1968 Spring Joint Computer Conf.*, vol. 32, 1968, pp. 37-45, first ray tracing paper, light ray tracing, b&w pictures on Calcomp plotter
- Arnaldi, Bruno, Thierry Priol, Kadi Bouatouch, "A New Space Subdivision Method for Ray Tracing CSG Modelled Scenes", *Visual Computer*, vol. 3, 1987, pp. 98-108, {CSG}
- Arvo, James, "Backward Ray Tracing", *SIGGRAPH '86 Developments in Ray Tracing seminar notes*, Aug. 1986, light ray tracing
- Arvo, James, David Kirk, "Fast Ray Tracing by Ray Classification", *Computer Graphics (SIGGRAPH '87 Proceedings)*, vol. 21, no. 4, July 1987, pp. 55-64, {octree}, five dimensional space subdivision
- Atherton, Peter R., "A Scanline Hidden Surface Removal Procedure for Constructive Solid Geometry", *Computer Graphics (SIGGRAPH '83 Proceedings)*, vol. 17, no. 3, July 1983, pp. 73-82, {CSG}
- Barr, Alan H., "Decal Projections", *SIGGRAPH '84 Mathematics of Computer Graphics seminar notes*, July 1984, {texture mapping}
- Barr, Alan H., "Ray Tracing Deformed Surfaces", *Computer Graphics (SIGGRAPH '86 Proceedings)*, vol. 20, no. 4, Aug. 1986, pp. 287-296
- Bier, Eric A., "Solidviews, An Interactive Three-Dimensional Illustrator", BS & MS thesis, Dept. of EE&CS, MIT, May 1983, use of Roth's CSG ray tracer as part of an interactive system, {CSG}
- Blinn, James F., Martin E. Newell, "Texture and Reflection in Computer Generated Images", *CACM*, vol. 19, no. 10, Oct. 1976, pp. 542-547, early paper on texture mapping, discusses spherical sky textures, {texture mapping, reflection}
- Blinn, James F., "A Generalization of Algebraic Surface Drawing", *ACM Trans. on Graphics*, vol. 1, no. 3, July 1982, pp. 235-256, ray tracing "blobby" models: finding roots of sums of gaussians, {blob, root finding}
- Bouatouch, Kadi, "A New Algorithm of Space Tracing Using a CSG Model", *Eurographics '87*, North-Holland, Amsterdam
- Bouville, Christian, R. Brusq, J. L. Dubois, I. Marchal, "Image Synthesis by Ray-Casting (in French)", *Acta Electron. (France)*, vol. 26, no. 3-4, 1984, pp. 249-259
- Bouville, Christian, J. L. Dubois, I. Marchal, "Generating High Quality Pictures by Ray Tracing", *Eurographics '84*, Copenhagen, Sept. 1984, pp. 161-177, (also *Computer Graphics Forum*, vol 4, no 2, June 1985, pp. 87-99)
- Bouville, Christian, "Bounding Ellipsoids for Ray-Fractal Intersection", *Computer Graphics (SIGGRAPH '85 Proceedings)*, vol. 19, no. 3, July 1985, pp. 45-52, {bounding volume}
- Bouville, Christian, "Image Synthesis Through Ray Tracing", *Banc-Titre*, France, Mar. 1985, pp. 50, {hardware}
- Bronsvort, Willem F., Fopke Klok, "Ray Tracing General Sweep-Defined Objects", 84-36, Dept. of Mathematics and Informatics, Delft U. of Tech., Delft, Netherlands, 1984
- Bronsvort, Willem F., Frederik W. Jansen, Jarke J. van Wijk, "The Use of Ray Casting in Solid Modeling", *Informatie*, Netherlands, vol. 26, Jan. 1984, pp. 50-59, {CSG}
- Bronsvort, Willem F., Jarke J. van Wijk, Frederik W. Jansen, "Two Methods for Improving the Efficiency of Ray Casting in Solid Modeling", *Computer-Aided Design*, vol. 16, no. 1, Jan. 1984, pp. 110-116, {CSG}, enhancements to Roth: scanline interval enclosures, CSG tree optimization, and recursive screen subdivision
- Bronsvort, Willem F., Fopke Klok, "Ray Tracing Generalized Cylinders", *ACM Transactions on Graphics*, (note corrigendum in ACM TOG July 1987 issue, v.6, no.3, p. 238-239), vol. 4, no. 4, Oct. 1985, pp. 291-303

- Brooks, Joan, "Extension and Adjuncts to the BRL-COMGEOM Program", (for Ballistic Research Laboratories), Aug. 1974, NTIS AD/A-000 897, *MAGI: intersection of ray and ellipsoid, ray tracing in the punch card era*, {CSG, quadric}
- Brooks, Joan, Ragini Murarka, Daniel Onuoha, Frank Rahn, Herbert A. Steinberg, "An Extension of the Combinatorial Geometry Technique for Modeling Vegetation and Terrain Features", (for Ballistic Research Laboratories), June 1974, NTIS AD-782 883, *MAGI: hierarchical bounding boxes, adaptive subsampling, pine tree models*, {CSG, bounding volume, botanical tree}
- Brown, Chris, "Special Purpose Computer Hardware for Mechanical Design Systems", *Proc. 1981 National Computer Graphics Assoc. Conf.*, pp. 403-414
- Bukow, Hans M. T., Michael J. Bailey, Warren H. Stevenson, "Simulation of Reflectance Sensors Using Image Synthesis Techniques", *Computers in Mechanical Engineering*, vol. 3, no. 4, Jan. 1985, pp. 69-74, {CAM}, *simulating assembly line optical sensors*
- Chang, Arthur G., "Parallel Architectural Support for Raytracing Graphics Techniques", Masters thesis, EECS Dept., UC Berkeley
- Chattopadhyay, S., Akira Fujimoto, "Bi-directional Ray Tracing", *Computer Graphics 1987*, Toshiyasu Kunii ed., Springer Verlag, Tokyo, 1987, pp. 335-343
- Chung, W. L., "A New Method of View Synthesis for Solid Modelling", *CAD84*, Butterworth & Co, Guildford, Surrey, UK, Apr. 1984, pp. 470-480, {CSG}
- Cleary, John G., Brian Wyvill, Reddy Vatti, Graham M. Birtwistle, "Design and Analysis of a Parallel Ray Tracing Computer", *Proceedings Graphics Interface '83*, May 1983, pp. 33-34, (also Proceedings XI Association of Simula Users Conference, 1983), {hardware}, *short note describing their project*
- Cleary, John G., Brian Wyvill, Graham M. Birtwistle, Reddy Vatti, "Multiprocessor Ray Tracing", Technical Report No. 83/128/17, Dept. of CS, U of Calgary, Oct. 1983, {hardware}, *analysis of square and cubical processor arrays for ray tracing*
- Cleary, John G., Geoff Wyvill, "An Analysis of an Algorithm for Fast Ray-Tracing using Uniform Space Subdivision", Research Report 87/264/12, U. of Calgary, Dept. of CS, 1987
- Cook, Robert L., Thomas Porter, Loren Carpenter, "Distributed Ray Tracing", *Computer Graphics (SIGGRAPH '84 Proceedings)*, vol. 18, no. 3, July 1984, pp. 137-145, *Monte Carlo distribution of rays to get gloss, translucency, penumbras, depth of field, motion blur*, {probabilistic ray tracing, monte carlo, motion blur, stochastic sampling}
- Cook, Robert L., "Stochastic Sampling in Computer Graphics", *ACM Transactions on Graphics*, vol. 5, no. 1, Jan. 1986, pp. 51-72
- Cook, Robert L., "Practical Aspects of Distributed Ray Tracing", *SIGGRAPH '86 Developments in Ray Tracing seminar notes*, Aug. 1986, {probabilistic ray tracing}
- Coquillart, S., "An Improvement of the Ray-Tracing Algorithm", *Eurographics '85*, Sept. 1985, North-Holland, Amsterdam, pp. 77-88
- Cordonnier, E., C. Bouville, I. Marchal, J. L. Dubois, "Creating CSH Modelled Pictures for Ray-Casting Display", *Eurographics '85*, Sept. 1985, North-Holland, Amsterdam
- Dadoun, Norm, David G. Kirkpatrick, John P. Walsh, "The Geometry of Beam Tracing", *Proc. of the Symp. on Computational Geometry*, June 1985, pp. 55-61, *the use of BSP trees and hierarchical bounding volumes for fast beam intersection testing*
- Davis, J. Roy, Roger Nagel, Walter Guber, "A Model Making and Display Technique for 3-D Pictures", *Proceedings of the 7th Annual Meeting of UAIDE*, San Francisco, Oct. 1968, pp. 47-72, {CSG}, *Synthavision genesis: CSG, primitives, optimization by region adjacency lists and adaptive subdivision for line drawings*
- Davis, Jon E., Michael J. Bailey, David C. Anderson, "Realistic Image Generation and the Modeling of Mechanical Solids", *Computers in Mechanical Engineering*, vol. 1, no. 1, Aug. 1982, *intro to CAD, solid modeling, and Whitted ray tracing, pre-Roth*
- Davis, Jon E., "Recursive Ray Tracing for the Realistic Display of Solid Models", MSME thesis, Dept. of ME, Purdue U., May 1982, {CAD}
- Deguchi, Hiroshi, Hitoshi Nishimura, Hiroshi Yoshimura, Toru Kawata, Isao Shirakawa, Koichi Omura, "A Parallel Processing Scheme for Three-Dimensional Image Creation", *Conf. Proc. Int. Symp. on Circuit and Systems (ISCAS'84)*, 1984, pp. 1285-1288, {hardware}, *LINKS-1 hardware*
- Deguchi, Hiroshi, Hitoshi Nishimura, Toshikazu Tatsumi, Toru Kawata, Isao Shirakawa, Koichi Omura, "Performance Evaluation of Parallel Processing in Computer Graphics System LINKS-1", *submitted to SIGGRAPH '85*, 1985, {hardware}
- Dippé, Mark A. Z., Erling Henry Wold, "Antialiasing Through Stochastic Sampling", *Computer Graphics (SIGGRAPH '85 Proceedings)*, vol. 19, no. 3, July 1985, pp. 69-78, {probabilistic ray tracing, stochastic sampling}

- Dippé, Mark E., John Swensen**, "An Adaptive Subdivision Algorithm and Parallel Architecture for Realistic Image Synthesis", *Computer Graphics (SIGGRAPH '84 Proceedings)*, vol. 18, no. 3, July 1984, pp. 149-158, *3-D network of processors, algorithm for adaptive load distribution*, {hardware}
- Edwards, Bruce E.**, "Implementation of a Ray-Tracing Algorithm for Rendering Superquadric Solids", Masters thesis, TR-82018, Rensselaer Polytechnic Institute, Troy, NY, Dec. 1982, *ray-traced unions and differences of superquadrics*, {superquadric}
- Fitzhorn, Patrick A.**, "Realistic Image Synthesis: A Time Complexity Analysis of Ray Tracing", Masters thesis, Dept. of CS, Colorado State U., Fort Collins, CO, Spring 1982
- Fujimoto, Akira, Kansei Iwata**, "Accelerated Ray Tracing", *Computer Graphics: Visual Technology and Art (Proceedings of Computer Graphics Tokyo '85)*, Tosiyasu Kunii ed., Springer Verlag, Tokyo, 1985, pp. 41-65, {octree}
- Fujimoto, Akira, Takayuki Tanaka, Kansei Iwata**, "ARTS: Accelerated Ray-Tracing System", *IEEE Computer Graphics and Applications*, Apr. 1986, pp. 16-26, {octree}
- Gjoystdal, H., J. E. Reinhardsen, K. Astebol**, "Computer Representation of Complex 3-D Geological Structures Using a New 'Solid Modeling' Technique", *Geophy. Prospect. (Netherlands)*, vol. 33, no. 8, Dec. 1985, pp. 1195-1211, {dynamic ray tracing}
- Glassner, Andrew S.**, "Space Subdivision for Fast Ray Tracing", *IEEE Computer Graphics and Applications*, vol. 4, no. 10, Oct. 1984, pp. 15-22, *use of octrees to speed intersection testing*, {bounding volume, octree}
- Glassner, Andrew S.**, "Spacetime Ray Tracing for Animation", *IEEE Computer Graphics and Applications*, vol. 8, no. 2, March 1988, pp. 60-70
- Goldsmith, Jeffrey, John Salmon**, "A Ray Tracing System for the Hypercube", Caltech, 1984, {parallel processing}
- Goldsmith, Jeffrey, John Salmon**, "Automatic Creation of Object Hierarchies for Ray Tracing", *IEEE Computer Graphics and Applications*, 1987, {ray tracing, bounding volume}
- Goldstein, Robert A.**, "A System for Computer Animation of 3-D Objects", *Proceedings of the 10th Annual UAIDE Meeting*, 1971
- Goldstein, Robert A., Roger Nagel**, "3-D Visual Simulation", *Simulation*, vol. 16, no. 1, Jan. 1971, pp. 25-31, *introduction to CSG, ray tracing, director's language*, {CSG}
- Graham, Eric**, "Graphic Scene Simulations", *Amiga World*, May/June 1987, pp. 18, *C source for sphere ray tracer (runs on Amiga)*
- Haines, Eric A., Donald P. Greenberg**, "The Light Buffer: A Ray Tracer Shadow Testing Accelerator", *IEEE Computer Graphics and Applications*, vol. 6, no. 9, Sept. 1986, pp. 6-16, {shading, ray tracing, shadows}
- Haines, Eric**, "A Proposal for Standard Graphics Environments", *IEEE Computer Graphics and Applications*, vol. 7, no. 11, Nov. 1987, pp. 3-5, {benchmark}, *renderer benchmarking environments and how to obtain them*
- Hall, Roy A.**, "A Methodology for Realistic Image Synthesis", Masters thesis, Cornell U., 1983, {shading, color}
- Hall, Roy A., Donald P. Greenberg**, "A Testbed for Realistic Image Synthesis", *IEEE Computer Graphics and Applications*, vol. 3, no. 8, Nov. 1983, pp. 10-20, *concerns shading and color more than ray tracing, but nice pictures!*, {shading, color}
- Hanrahan, Pat**, "Ray Tracing Algebraic Surfaces", *Computer Graphics (SIGGRAPH '83 Proceedings)*, vol. 17, no. 3, July 1983, pp. 83-90, *numerical techniques for finding roots of polynomials*, {root finding, algebraic surface}
- Hanrahan, Pat, Paul S. Heckbert**, "Introduction to Beam Tracing", *Intl. Conf. on Engineering and Computer Graphics*, Beijing, China, Aug. 1984, pp. 286-289, *early version of their SIGGRAPH paper*
- Hanrahan, Pat**, "Using Caching and Breadth-First Search to Speed Up Ray-Tracing", *Graphics Interface '86*, May 1986, pp. 56-61, {seed fill, coherence}
- Heckbert, Paul S., Pat Hanrahan**, "Beam Tracing Polygonal Objects", *Computer Graphics (SIGGRAPH '84 Proceedings)*, vol. 18, no. 3, July 1984, pp. 119-127, *Weiler-Atherton algorithm applied to ray tracing*, {polygon}
- Heckbert, Paul S.**, "Ray Tracing JELL-O (R) Brand Gelatin", *Computer Graphics (SIGGRAPH '87 Proceedings)*, vol. 21, no. 4, July 1987, pp. 73-74
- Jansen, Frederik**, "Data Structures for Ray Tracing", L. R. A. Kessener ed., F. J. Peters ed., M. L. P. van Lierop ed., *Data Structures for Raster Graphics*, (Eurographic Seminar), New York, 1986, Springer-Verlag, pp. 57-73, {data structures, CSG}, *overview of published algorithms for ray tracing using spatial subdivision*
- Joy, Kenneth I., Murthy N. Bhetanabhotla**, "Ray Tracing Parametric Surface Patches Utilizing Numerical Techniques and Ray Coherence", *Computer Graphics (SIGGRAPH '86 Proceedings)*, vol. 20, no. 4, Aug. 1986, pp. 279-285

- Kajiya, James T.**, "Ray Tracing Parametric Patches", *Computer Graphics (SIGGRAPH '82 Proceedings)*, vol. 16, no. 3, July 1982, pp. 245-254, *ray tracing bivariate polynomial patches*, {patch}
- Kajiya, James T.**, "New Techniques for Ray Tracing Procedurally Defined Objects", *ACM Trans. on Graphics*, vol. 2, no. 3, July 1983, pp. 161-181, (also appeared in SIGGRAPH '83 Proceedings.), *ray tracing fractals, prisms, and surfaces of revolution*, {fractal}
- Kajiya, James T.**, "SIGGRAPH '83 Tutorial on Ray Tracing", *SIGGRAPH '83 State of the Art in Image Synthesis seminar notes*, July 1983, *good survey of ray tracing*
- Kajiya, James T.**, Brian P. Von Herzen, "Ray Tracing Volume Densities", *Computer Graphics (SIGGRAPH '84 Proceedings)*, vol. 18, no. 3, July 1984, pp. 165-174, *ray tracing and meteorological simulation of clouds*, {cloud}
- Kajiya, James T.**, "The Rendering Equation", *Computer Graphics (SIGGRAPH '86 Proceedings)*, vol. 20, no. 4, Aug. 1986, pp. 143-150, {shading, diffuse reflection, radiosity}
- Kaplan, Michael R.**, "Space-Tracing, A Constant Time Ray-Tracer", *SIGGRAPH '85 State of the Art in Image Synthesis seminar notes*, July 1985, *like Glassner*
- Kay, Douglas S.**, Donald P. Greenberg, "Transparency for Computer Synthesized Images", *Computer Graphics (SIGGRAPH '79 Proceedings)*, vol. 13, no. 2, Aug. 1979, pp. 158-164, *2.5-D ray tracing: refraction by warping background image, contains better refraction formula than Whitted*
- Kay, Douglas S.**, "Transparency, Refraction, and Ray Tracing for Computer Synthesized Images", Masters thesis, Cornell U., Jan. 1979
- Kay, Timothy L.**, James T. Kajiya, "Ray Tracing Complex Scenes", *Computer Graphics (SIGGRAPH '86 Proceedings)*, vol. 20, no. 4, Aug. 1986, pp. 269-278, {bounding volume}
- Kedem, G.**, J. L. Ellis, "The Raycasting Machine", *Proc. IEEE Intl. Conf. on Computer Design: VLSI in Computers (ICCD '84)*, (Port Chester, NY 8-11 Oct. 1984), IEEE Computer Society Press, Silver Spring, MD, 1984, pp. 533-538
- Kirk, David B.**, "The Simulation of Natural Features Using Cone Tracing", *Advanced Computer Graphics (Proc. of CG Tokyo '86)*, Tosiyasu Kunii ed., Springer Verlag, Tokyo, 1986, pp. 129-144, {antialiasing}
- Kitaoka, Shoichi**, "KIT: An Experimental Solid Modelling System", MS thesis, University of Utah, April 1985, *Roth-style ray tracer with the addition of several new surfaces including patches, sweeps, etc. Produced "Scene with Corkscrew" on SIGGRAPH '85 back cover*, {solid modeling, primitive shapes}
- Kitaoka, Shoichi**, "KIT: An Experimental Solid Modelling System", *The Visual Computer*, vol. 2, no. 1, Jan. 1986, pp. 9, *Roth-style ray tracer with the addition of several new surfaces including patches, sweeps, etc.*, {solid modeling, primitive shapes}
- Lee, Mark E.**, Richard A. Redner, Samuel P. Usselton, "Statistically Optimized Sampling for Distributed Ray Tracing", *Computer Graphics (SIGGRAPH '85 Proceedings)*, vol. 19, no. 3, July 1985, pp. 61-67, {probabilistic ray tracing, stochastic sampling}
- Levner, G.**, P. Tassinari, D. Marini, "A Simple Method for Ray Tracing Bicubic Surfaces", *Computer Graphics 1987*, Tosiyasu Kunii ed., Springer Verlag, Tokyo, 1987, pp. 285-302
- Martin, R. R.**, "Recent Advances in Graphical Techniques", *1985 European Conference on Solid Modeling*, (London, 9-10 Sept 1985), Oyez Sci. and Tech. Services, London, 1985, {texture mapping}
- Max, Nelson L.**, "Vectorized Procedural Models for Natural Terrain: Waves and Islands in the Sunset", *Computer Graphics (SIGGRAPH '81 Proceedings)*, vol. 15, no. 3, Aug. 1981, pp. 317-324, *ray tracing on a CRAY + many tricks*, {orientation code, colormap animation, hardware, wave}
- Max, Nelson L.**, "An Anti-Aliased Wave Reflection Algorithm", *SIGGRAPH '82 Advanced Image Synthesis seminar notes*, July 1982, *improved ray tracing of waves*, {wave}
- Miller, Gene S.**, C. Robert Hoffman, "Illumination and Reflection Maps: Simulated Objects in Simulated and Real Environments", *SIGGRAPH '84 Advanced Computer Graphics Animation seminar notes*, July 1984, *reflection maps: how to make and use them*, {illumination map}
- Montcel, B.** Tezenas du, A. Nicolas, "An Illumination Model for Ray-Tracing", *Eurographics '85*, Sept. 1985
- Moravec, Hans P.**, "3D Graphics and the Wave Theory", *Computer Graphics (SIGGRAPH '81 Proceedings)*, vol. 15, no. 3, Aug. 1981, pp. 289-296, *illumination by wave fronts, rather than light rays*, {wave theory}
- Murakami, Kouichi**, Hitoshi Matsumoto, "Ray Tracing with Octree Data Structure", *Proc. 28th Information Processing Conf.*, 1983

- Nemoto, Keiji, Takao Omachi**, "An Adaptive Subdivision by Sliding Boundary Surfaces for Fast Ray Tracing", *Graphics Interface '86*, May 1986, pp. 43-48, {adaptive subdivision algorithm on a parallel architecture}
- Nishimura, Hitoshi, Hiroshi Ohno, Toru Kawata, Isao Shirakawa, Koichi Omura**, "Links-1: A Parallel Pipelined Multimicrocomputer System for Image Creation", *Conference Proceedings of the 10th Annual International Symposium on Computer Architecture, SIGARCH*, 1983, pp. 387-394, a parallel hardware architecture being used for ray traced animation; the paper does not discuss ray tracing or their software, {hardware}
- Ohta, Masataka, Mamoru Maekawa**, "Ray Coherence Theorem and Constant Time Ray Tracing Algorithm", *Computer Graphics 1987*, Toshiyasu Kunii ed., Springer Verlag, Tokyo, 1987, pp. 303-314
- Peachey, Darwyn R.**, "PORTRAY - An Image Synthesis System for Realistic Computer Graphics", TR 84-18, Dept. of Computational Science, U. of Saskatchewan, Saskatoon, Saskatchewan, Canada, 1984, {modeling, shading}, excellent survey of image synthesis, system issues
- Peachey, Darwyn R.**, "PORTRAY - An Image Synthesis System", *Graphics Interface '86*, May 1986, pp. 37-42, {modeling, shading}, image formats, condensed version of U of Sask tech report
- Peng, Q. S.**, "A Fast Ray Tracing Algorithm Using Space Indexing Techniques", *Eurographics '87*, North-Holland, Amsterdam
- Peterson, John W.**, "Ray Tracing General B-Splines", *Proceedings of the ACM Mountain Regional Conference*, April, 1986, pp. 87, {B-splines, surfaces}, extensions to Sweeney's patch algorithm to handle a wider range of surfaces
- Plunkett, D. J., M. J. Bailey**, "The Vectorization of a Ray-Tracing Algorithm for Improved Execution Speed", *IEEE Computer Graphics and Applications*, vol. 5, no. 8, Aug. 1985, pp. 52-60
- Potmesil, Michael**, "Generating Three-Dimensional Surface Models of Solid Objects from Multiple Projections", PhD thesis, IPL-TR-033, Oct. 1982, Image Processing Laboratory, RPI, Troy, NY, contains brief description of his ray tracer, camera model and motion blur as post-processes, appendix on ray-patch intersection methods, {computer vision, patch, quadtree}
- Pulleyblank, Ron, John Kapenga**, "The Feasibility of a VLSI Chip for Ray Tracing Bicubic Patches", *IEEE Computer Graphics and Applications*, vol. 7, no. 3, March 1987, pp. 33-44, {bicubic patch}
- Purgathofer, Werner**, "A Statistical Method for Adaptive Stochastic Sampling", *Eurographics '86*, 1986, North-Holland, Amsterdam, pp. 145-152, {probabilistic ray tracing}
- Reddy, D. R., Steven M. Rubin**, "Representation of Three-Dimensional Objects", CMU-CS-78-113, Dept. of CS, Carnegie-Mellon U., Apr. 1978, {bounding volume}
- Rogers, David F.**, "Procedural Elements for Computer Graphics", McGraw-Hill, New York, 1985, {hidden surface}, the only book on image synthesis, good summary of ray tracing
- Roth, Scott D.**, "Ray Casting for Modeling Solids", *Computer Graphics and Image Processing*, vol. 18, no. 2, Feb. 1982, pp. 109-144, the other classic ray tracing paper, {CSG, hidden line}
- Rubin, Steven M., Turner Whitted**, "A 3-Dimensional Representation for Fast Rendering of Complex Scenes", *Computer Graphics (SIGGRAPH '80 Proceedings)*, vol. 14, no. 3, July 1980, pp. 110-116, hierarchical bounding boxes, used to speed up ray tracing & other algs, {bounding volume}
- Sederberg, Thomas W., David C. Anderson**, "Ray Tracing of Steiner Patches", *Computer Graphics (SIGGRAPH '84 Proceedings)*, vol. 18, no. 3, July 1984, pp. 159-164, implicitization of Steiner patch, solution of resulting quartic, {patch, root finding}
- Shinya, Mikio, Tokiichiro Takahashi, Seiichiro Naito**, "Principles and Applications of Pencil Tracing", *Computer Graphics (SIGGRAPH '87 Proceedings)*, vol. 21, no. 4, July 1987, pp. 45-54
- Snyder, John M., Alan H. Barr**, "Ray Tracing Complex Models Containing Surface Tessellations", *Computer Graphics (SIGGRAPH '87 Proceedings)*, vol. 21, no. 4, July 1987, pp. 119-128, {parametric surface, tessellation, 3D grid}
- Speer, L. Richard, Tony D. DeRose, Brian A. Barsky**, "A Theoretical and Empirical Analysis of Coherent Ray-Tracing", *Graphics Interface '85*, May 1985, {coherence}, they conclude that their cylinder-piercing optimization doesn't work
- Steinberg, Herbert A.**, "A Smooth Surface Based on Biquadratic Patches", *IEEE Computer Graphics and Applications*, vol. 4, no. 9, Sept. 1984, pp. 20-23, ray tracing biquadratic and bicubic Coons patches, {patch}
- Steinberg, Herbert A.**, "Ray Tracing and CSG Applications", *SIGGRAPH '85 Introduction to Solid Modeling seminar notes*, July 1985, {CSG}
- Sweeney, Michael A. J.**, "The Waterloo Ray Tracing Package", CS-85-35 (Master's thesis), Dept of CS, U. of Waterloo, Oct. 1985

- Sweeney, Michael, Richard H. Bartels, "Ray Tracing Free-Form B-Spline Surfaces", *IEEE Computer Graphics and Applications*, vol. 6, no. 2, pp. 41, Feb. 1986
- Tamminen, M., O. Karonen, M. Mantyla, "Ray-Casting and Block Model Conversion Using a Spatial Index", *Computer Aided Design*, vol. 16, July 1984, pp. 203-208
- Thomas, Spencer, "Dispersive Refraction in Ray Tracing", *The Visual Computer*, vol. 2, no. 1, Jan. 1986, pp. 3-8, *prismatic effects*
- Toth, Daniel L., "On Ray Tracing Parametric Surfaces", *Computer Graphics (SIGGRAPH '85 Proceedings)*, vol. 19, no. 3, July 1985, pp. 171-179
- Ullner, Mike K., "Parallel Machines for Computer Graphics", PhD thesis, California Institute of Technology, 1983, *hardware for ray tracing*, {hardware}
- Vatti, Bala Rajareddy, "Multiprocessor Ray-Tracing", MS thesis, Dept. of CS, U of Calgary, May 1984, {parallel processing, space subdivision}, *multiprocessor algorithm and uniprocessor simulation results, regular space subdivision to reduce object/ray intersections*
- Verbeck, Channing P., "Extended Geometries and Directional Intensity Variation for Light Sources", *Banc-Titre*, France, Mar. 1985, pp. 53-54, {shading, ray tracing, numerical integration}
- Wallace, John R., Michael F. Cohen, Donald P. Greenberg, "A Two-Pass Solution to the Rendering Equation: A Synthesis of Ray Tracing and Radiosity Methods", *Computer Graphics (SIGGRAPH '87 Proceedings)*, vol. 21, no. 4, July 1987, pp. 311-320, {radiosity, probabilistic ray tracing, z-buffer}
- Warren, Van, "Geometric Hashing for Rendering Complex Scenes", MS thesis, University of Utah, May 1986, *Divides the scene into a volume of regular small cubes, and traces the rays between cubes*, {geometric hashing}
- Weghorst, Hank, Gary Hooper, Donald P. Greenberg, "Improved Computational Methods for Ray Tracing", *ACM Trans. on Graphics*, vol. 3, no. 1, Jan. 1984, pp. 52-69, *discussion of bounding volumes, hierarchical structures and the "item buffer"*, {bounding volume}
- Whelan, Daniel S., "A Multiprocessor Architecture for Real-Time Computer Animation", Computer Science TR 5200, Caltech, 1985, {hardware}
- Whitted, Turner, "Processing Requirements for Hidden Surface Elimination and Realistic Shading", *IEEE Digest of Papers, COMPCON*, Spring '82, pp. 245-250, *discussion of various visible surface and illumination methods, including ray tracing*, {efficiency}
- Whitted, Turner, "An Improved Illumination Model for Shaded Display", *CACM*, vol. 23, no. 6, June 1980, pp. 343-349, *the classic ray tracing paper*
- Whitted, Turner, "The Hacker's Guide to Making Pretty Pictures", *SIGGRAPH '85 Image Rendering Tricks seminar notes*, July 1985, *general tricks for image synthesis, includes C source for simple ray tracer*
- Wijk, Jarke J. van, Frederik W. Jansen, "Realism in Raster Graphics", *Computers and Graphics*, vol. 8, no. 2, 1984, pp. 217-219, {image synthesis}
- Wijk, Jarke J. van, "Ray Tracing Objects Defined by Sweeping Planar Cubic Splines", *ACM Trans. on Graphics*, vol. 3, no. 3, July 1984, pp. 223-237, *ray tracing prisms, cones, and surfaces of revolution*
- Wijk, Jarke J. van, "Ray Tracing Objects Defined by Sweeping A Sphere", *Eurographics '84*, Copenhagen, Sept. 1984, pp. 73-82, (reprinted in *Computers and Graphics*, Vol 9. No 3, 1985, pp. 283-290)
- Wyvill, Geoff, A. Ward, T. Brown, "Sketches by Ray Tracing", Research Report 1/1/87, Dept. of CS, U. of Otago, New Zealand, {hidden line}, *line drawing*
- Wyvill, Geoff, Tosiyasu L. Kunii, Yasuto Shirai, "Space Division for Ray Tracing in CSG", *IEEE Computer Graphics and Applications*, Apr. 1986, pp. 28-34, {CSG}
- Yamamoto, Tsuyoshi, "The Three Dimensional Computer Graphics", CQ Publishing, 1983, *a Japanese book on ray tracing! No English, but some BASIC(!) listings of ray tracing programs*
- Yasuda, T., S. Yokoi, J. I. Toriwaki, S. Tsuruoka, Y. Miyake, "An Improved Ray Tracing Algorithm for Rendering Transparent Objects (in Japanese)", *Trans. Inf. Process. Soc. of Japan*, vol. 25, no. 6, 1984, pp. 953-959
- Yokoi, Shigeki, T. Yasuda, Jun-ichiro Toriwaki, "Simplified Ray Tracing Algorithms for Rendering Transparent Objects", Technical Report, Information Engineering Dept., Nagoya University, Japan
- Youssef, Saul, "A New Algorithm for Object Oriented Ray Tracing", *Computer Vision, Graphics and Image Processing*, vol. 34, 1986, pp. 125-137 •

About this issue...

of the *Ray Tracing News*: Layout: Macintosh II using Pagemaker 3.0. Printing: LaserWriter II at 300dpi. Scanner: Abaton 300/FB. Running heads: Helvetica Bold. Titles: Avant Garde. Text: Times, Bookman, and Courier. Word processing: Word 3.01. Equations: Expressionist 1.11.

Edit, Design, & Layout by Andrew Glassner

Solution to Last Issue's Puzzle

by Eric Haines

(hpfcrsleye@erich@hplabs.hp.com)

The answer to the puzzle: there are a few subtle perceptual bugs which occur when using this method, depending on implementation. Basically, it boils down to "does this 'ambient' light cast a shadow for reflected and refracted ray intersections?" and "does the 'ambient' light create a specular highlight?"

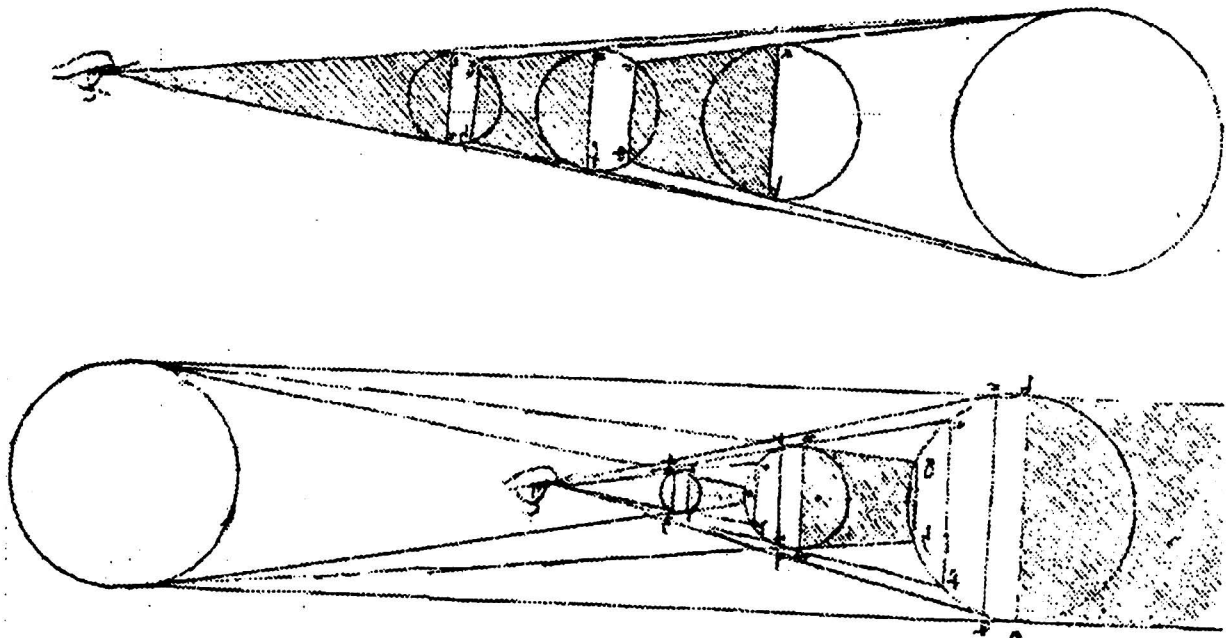
If it does cast a shadow, then you've just added a noticeable expense to your ray tracer. The 'ambient' light is now just another light source for non-eye rays, and has to be tested as such - your little bit of added realism has just cost a fair bit of compute time for scenes which spawn a lot of rays. The 'bug' is simply that the objects now each cast another shadow, and these shadows are only visible in reflections and refractions. For example, a ball on a plane lit by one light and your 'ambient' light seems to cast but one shadow as viewed by the eye. If you look in a mirror behind the ball you see in the reflection that there are two shadows. This is what I call a perceptual bug: it's physically correct, but is almost never seen in the physical

world. A colored glass ball will have a red filter spot appear behind it, and this defect can be seen within the image of the ball itself (my clue about the Hall model was simply that he adds a number of transparency effects to his model).

However, the standard way to implement the 'ambient' light at the eye is to have it never cast a shadow. The major problem here is that this assumption is not physically based. This error gives rise to other perceptual bugs, such as the ball in the mirror now being dark on the far side for no apparent reason (again, this method will cause some odd problems using the Hall model when looking through a colored ball).

Another annoying problem is that the sphere itself will have a specular highlight always at the closest point to the eye (I assume the 'ambient' light was turned on to give definition to objects in shadow, and not for its own sake). Reflected objects will also have highlights in places where they didn't occur when using a simple ambient add-in term. One quick solution is to not perform specular highlighting for the 'ambient' light (this also makes the 'ambient' light faster to compute).

The puzzle question is open ended: these are just my thoughts on what are the pitfalls of this trick. It's a useful trick, but is not superior to a simple ambient term in all cases. Essentially, it all comes down to deciding what effects are desirable. •



From Plate II, illustrating "Third Book on Light and Shade" from *The Notebooks of Leonardo Da Vinci*, Volume I (Dover Publications, NY)

Interactive SIMD Ray Tracing

by Russ Tuck
(tuck@cs.unc.edu)

1. Introduction

Ray Tracing is widely recognized as a very powerful, general, and slow means of generating realistic computer-generated images. This paper will explore the use of massively parallel SIMD (Single-Instruction, Multiple-Data) computers to generate ray traced images at interactive rates (at least 10 frames per second). Real and then hypothetical SIMD machines will be considered.

I assume the reader is familiar with ray tracing algorithms for ordinary sequential computers, including distributed ray tracing and the rendering equation. I also assume the reader is familiar with the basic characteristics of SIMD computers. Familiarity with the Connection Machine (CM) [Hil85, Cor87], Massively Parallel Processor (MPP) [Pot85], and Pixel-Planes (Pxpl) [FGH*85] architectures is helpful, but not required.

The rest of this paper discusses in sequence: terminology, a simple algorithm for SIMD ray tracing, some published algorithms for SIMD ray tracing, and some new algorithms for SIMD ray tracing.

2. Terminology

The scene to be rendered consists of a *database of primitives*. A *ray* is a straight-line segment of the *path* a photon of light might take. A *sample* is a path plus additional *shadow rays*: one from each primitive the ray path intersects to each light source. Kajiya's approximation of the rendering equation is based on computing selected samples at each pixel.

3. One PE per Pixel

The simplest way to do SIMD ray tracing is to assign a processing element (PE) to each pixel and broadcast the scene database repeatedly. Each PE computes a path or sample by sequentially intersecting each ray with the entire database to find which (if any) primitive the ray hits first. This intersection is used to compute the direction of the next ray in the path, and the directions of the shadow rays if a sample is being computed. Each PE, executing in lockstep with all the others, intersects its current ray with the database as follows:

```

set minimum intersection distance to infinity.
for each primitive in database (as it is broadcast):
    find distance along ray to intersection with this primitive.
    if this distance is less than minimum
        set minimum to this distance.
    compute and save surface normal at intersection.
    
```

4. Connection Machine (CM) Results

Franklin Crow [Cro88, Cro87] reports that Karl Sims at the MIT Media Lab has written a ray tracing program for the Connection Machine. It uses the basic algorithm described above to compute a single path three rays deep at each pixel. Some surfaces were texture mapped, with one pixel of the map stored at each PE. The CM's global routing network was used to retrieve the relevant texture pixel during the color calculation for rays that hit textured surfaces. A 128 by 128 PE array was used to compute a 512 by 512 image, with each PE handling 16 pixels. A scene consisting of a few spheres and planes was generated in about 40 seconds.

5. Massively Parallel Processor (MPP) Methods

McAnulty and Wainer [MW86] described an unimplemented ray tracing system for Constructive Solid Geometry (CSG) models on the MPP. It uses the method of section 3, adding a mechanism for evaluating the combining operations of the CSG tree. This is necessary because the first intersection of the ray with the overall object is not necessarily the same as the first intersection of the ray with a primitive.

Dorband [Dor87] presents an unimplemented algorithm for doing ray tracing on the MPP with sort computation, which his paper references but does not fully describe. This algorithm assigns a PE to each ray and each primitive. Each ray intersection operation builds and uses the next level of an implicit octree. In the process, it alternates between making eight copies each of all remaining rays and primitives, and discarding those copies which it determines do not participate in an intersection. Some issues are not dealt with, including the extent to which this data duplication will limit the scene complexity or image resolution.

6. Successive Refinement on Pixel-Planes 5

Roman Kuchkuda, a graduate of the UNC-CH Computer Science Department, has proposed that ray tracing be used for successive refinement of images produced by faster techniques on Pixel-Planes 5 (Pxpl5). Pixel-Planes 4 (Pxpl4) has too little memory per pixel for ray intersection calculations. He suggests that the interactive z-buffer rendering code be slightly extended to compute the first ray. If the image remains static long enough, the image can be improved with additional rays as a path, sample, and multiple samples are computed. This would be a logical extension of the successive refinements already used for anti-aliasing on Pxpl4. In light of the

time estimates presented below, it also appears to be the most practical way of using ray tracing with complex scenes on Pxp15.

7. Time Estimate for Pixel-Planes 5

Intersection calculations dominate the execution time of ray tracers, especially those which naively intersect every ray with every primitive. So, I will use an estimate of the time required for intersection calculations as an estimate of the entire ray tracing time.

Kuchkuda presents a simple but complete ray tracing program [Kuc87]. I counted the operations it uses to intersect a ray with a primitive. This includes finding the distance along the ray to its intersection (if any) with the primitive, and the surface normal at this point. For a sphere, this takes 11 additions, 8 multiplications, 3 divisions, and 1 square root. A triangle requires 17 additions, 18 multiplications, and 1 division.

I obtained rough estimates of bit-serial operations required for single precision (32-bit IEEE format) floating-point calculations on Pixel-Planes from John Eyles. A ray-primitive intersection calculation takes about 52,400 bit-serial operations for a sphere, and about 56,800 for a triangle. Rounding these counts up to 60,000 simplifies calculations. This should more than compensate for the integer overhead computation ignored in these operation counts. Pixel-Planes 5 PE's are designed to execute 40,000,000 bit-serial operations per second. This allows the computation of 666 intersections per second, or 666/ (number of primitives) rays per second.

Let's see how much interactive ray tracing we can do with this computing power. A Pixel-Planes 5 machine can have 256K PE's, one per pixel in a 512 by 512 image. Tracing samples 3 rays deep with one light source takes 6 rays per sample, or 60 rays per second at 10 frames per second. This will be possible for trivial databases with 11 or fewer primitives. The image can be successively refined by computing additional and deeper samples.

8. Multiple PE's per Pixel

This section explores the potential for greater parallelism, using multiple PE's per pixel for faster ray tracing. In many cases, the methods discussed here give a near linear speedup as more PE's are used per pixel. They require inter-PE communication in order to (1) perform reduction operations (eg, maximum, sum) on a group of adjacent PE's, and (2) broadcast a value computed in a particular PE to a group of adjacent PE's. Ideally, both operations should take time logarithmic in group size. A tree or Cube Connected Cycles network [PV81] can provide this. So can a hypercube (binary n-cube), but it uses many more wires. For modest sized groups, an inexpensive 2D grid network is sufficient.

8.1. One PE per Sample

An obvious extension of the method of section 3 is to compute multiple samples for each pixel simultaneously in different PE's. Clumps of adjacent PE's can be assigned to the same pixel. This collapses some of the successive refinement into the original image, and makes further successive refinement faster.

8.2. Multiple PE's per Sample

If there are l light sources in a scene, the l shadow rays at each level can be computed in parallel with the ray to the next level. This allows $l+1$ PE's to compute a sample s rays deep in the time it takes to trace $s+1$ rays. After each ray trace, the shadow rays are combined with a reduction operation, and the begin point of the next set of shadow rays is broadcast from the PE that computed the ray to the next level.

8.3. Multiple PE's per Ray

A group of r PE's can share the work of intersecting a single ray with the database. Let the PE's be numbered $0 \dots r-1$ and each PE p be responsible for every r 'th primitive, beginning with primitive p . Now r primitives can be broadcast sequentially and then intersected in parallel. These different intersections are then combined to find the first primitive the ray hits. Approximately 200 primitives can be broadcast in the time it takes to compute one ray-primitive intersection. Finding the first intersection in a group of 200 PE's should take less than 10% of the time needed to find a ray-primitive intersection. This means 200 PE's assigned to a ray can trace it almost 100 times as fast as a single PE. 11 PE's can trace a ray about 10 times faster than 1.

8.4 Speed and PE's

The previous sections have described several levels at which parallelism can be used to make ray tracing faster. These can be used together or independently. Let's examine the potential speed and number of PE's used if they are all used together. Assume that r PE's are used to compute each ray, that one ray-primitive intersection calculation takes time t , that a primitive can be broadcast in ct , and that s samples per pixel are computed in parallel. Each sample (path) is d rays deep, and the database has l light sources among d primitives. Then P PE's are used to compute a pixel in time T , where $P = rs(l+1)$ and

$$T \approx p(d+1) \left[c + \frac{1}{r} \right] t = \frac{p(d+1)(cr+1)t}{r}$$

(Editor's Note: we repeat the equation here for convenience):

$$T \approx p(d+1) \left[c + \frac{1}{r} \right] t = \frac{p(d+1)(cr+1)t}{r}$$

Note that in these equations, increasing s and l improves image quality and increases P without affecting T . Increasing s also makes the image improve more in each iterative refinement step. Increasing d improves image quality and increases T without affecting P . Increasing r increases P and decreases T , at first linearly but asymptotically approaching a factor of c , but does not affect image quality. In the normal case of $p > r$, p increases T and image complexity without affecting P . For Pixel-Planes 5, the machine-dependant constants have values $c \approx .005$ and $t \approx 1.5$ ms.

Although it is not yet practical to build machines with more than one PE per pixel, chip and packaging technologies are improving rapidly. Over the next 10 years, SIMD machines with 100 or even 1000 PEs per pixel may become practical. These PEs will probably also be 100 or even 1000 times faster than today's.

9 Conclusions

SIMD computers can be used effectively for ray tracing. Pixel-Planes 5 will be able to interactively ray trace trivial images. For complex scenes, ray tracing will be a natural successive refinement process following initial z-buffer rendering.

Essentially unlimited numbers of PEs can be used effectively to speed ray tracing and improve images. Future SIMD machines with more (and faster) PEs can make interactive ray tracing practical for complex scenes and high resolution images. •

References

- [Cor87] Thinking Machines Corporation. *Connection Machine Model CM-2 Technical Summary*. Technical Report HA87-4, Thinking Machines Corporation, April 1987.
- [Cro87] Franklin C. Crow. Concurrent rendering on general-purpose hardware. In *Workshop on Rendering Algorithms and Systems*, pages 47-52, April 1987.
- [Cro88] Franklin C. Crow. 3d image synthesis on the connection machine. In *Parallel Processing for Computer Vision and Display*, January 1988.
- [Dor87] John E. Dorband. 3d graphic generation on the MPP. In *Second International Conference on Supercomputing*, Vol. 1, pages 305- 309, International Supercomputing Institute, Inc., 1987. Also in *The First Symposium on the Frontiers of Massively Parallel Scientific Computation*, J. Fischer, Ed., NASA Goddard Space Flight Center, Sept. 1986.

[FGH*85] Henry Fuchs, Jack Goldfeather, Jeff P. Hultquist, Susan Spach, John D. Austin, Frederick P. Brooks, Jr., John G. Eyles, and John Poulton. Fast spheres, shadows, textures, transparencies, and image enhancements in pixel planes. *Computer Graphics*, 19(3):111- 120, July 1985.

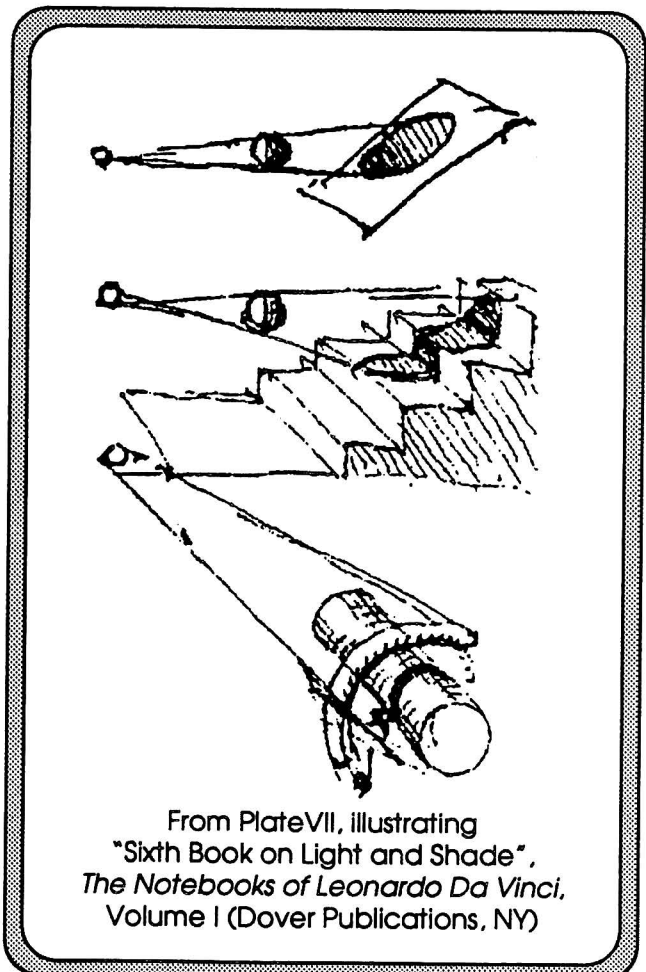
[Hil85] W. Daniel Hillis. *The Connection Machine*. MIT Press Series in Artificial Intelligence, The MIT Press, Cambridge, MA, 1985.

[Kuc87] Roman Kuchkuda. An introduction to ray tracing. October 1987. unpublished paper.

[MW86] Michael A. McAnulty and Michael S. Wainer. Algorithmic commonalities in the parallel environment. In J. Fischer, editor, *The First Symposium on the Frontiers of Massively Parallel Scientific Computation*, NASA Goddard Space Flight Center, Greenbelt, MD, September 1986.

[Pot85] J. L. Potter, editor. *The Massively Parallel Processor*. MIT Press Series in Scientific Computation, The MIT Press, Cambridge, MA, 1985.

[PV81] Franco P. Preparata and Jean Vuillemin. The Cube-Connected Cycles: a versatile network for parallel computation. *Communications of the ACM*, 24(5):300-309, May 1981.



From Plate VII, illustrating "Sixth Book on Light and Shade", *The Notebooks of Leonardo Da Vinci*, Volume I (Dover Publications, NY)