◊ .1

A Graphics Math Library

Ferdi Scheepers and Stephen F. May
Advanced Computing Center for the Arts and Design
Department of Computer and Information Science
The Ohio State University
Columbus, Ohio, USA

◊ Description ◊

libgm provides fundamental data types often required by graphics-related programs. Included are data types for 2D and 3D vectors, and for 3×3 and 4×4 matrices. In addition, *libgm* provides a set of constants and utility functions which are useful for graphics programming and a boolean data type `bool` with boolean constants `false` and `true`.

The constants defined by *libgm* are shown in Table ??. All constants are double precision, floating point numbers defined to 20 digits of precision, although only 6 digits are shown in the table. Tables ??—?? show the scalar utility functions, and the vector and matrix operations provided by *libgm*. Notationally, we use `f` or `f`k $(k = 1, 2, \ldots)$ to denote variables of type `double`, and `i` or `i`k $(k = 1, 2, \ldots)$ to denote variables of type `int`. We use `v` or `v`i $(i = 1, 2, \ldots)$ to denote instances of vectors and `M` or `M`i $(i = 1, 2, \ldots)$ to denote instances of matrices. Matrix elements are stored in row-major order and can be accessed or changed by using two, zero-indexed subscript operators. Note that the adjoint matrix $M^* = (1/|M|)M^{-1}$ is used to determine the inverse of a non-singular matrix M[1].

[1]see volume I, pages 470-471.

ISBN 0-12-XXXXXX-X

Table 1. *libgm* Constants

gm2PI	6.283185...	2π	gmPIDIV4	0.785398...	$\pi/4$
gmDEGTORAD	0.017453...	$\pi/180$	gmPIDIV4	0.785398...	$\pi/4$
gmE	2.718281...	e	gmRADTODEG	57.295779...	$180/\pi$
gmEEXPPI	23.140692...	e^{π}	gmSQRT2	1.414213...	$\sqrt{2}$
gmGOLDEN	1.618033...	golden ratio	gmSQRT3	1.732050...	$\sqrt{3}$
gmINVPI	0.318309...	π^{-1}	gmSQRT10	3.162277...	$\sqrt{10}$
gmLN10	2.302585...	$\ln 10$	gmSQRTE	1.648721...	$\sqrt{e}$
gmLN2	0.693147...	$\ln 2$	gmSQRTHALF	0.707106...	$\sqrt{0.5}$
gmLOG10E	0.434294...	$\log e$	gmSQRTLN2	0.832554...	$\sqrt{\ln 2}$
gmLOG2E	1.442695...	$\lg e$	gmSQRTPI	1.772453...	$\sqrt{\pi}$
gmPI	3.141592...	π	gmEPSILON	1.0e-10	next `double` > 0
gmPIDIV2	1.570796...	$\pi/2$	gmGOOGOL	1.0e50	large `double`

Table 2. *libgm* Utility Functions

function	description	usage
`gmAbs(f)`	absolute value of `f`	`f1 = gmAbs(f2);`
`gmCeil(f)`	least integer greater than or equal to `f`	`f1 = gmCeil(f2);`
`gmClamp(f,f1,f2)`	clamp `f` to `[f2,f3]`	`gmClamp(f,f1,f2);`
`gmCube(f)`	f^3	`f1 = gmCube(f2);`
`gmDegrees(f)`	convert angle in radians, `f`, to angle in degrees	`f1 = gmDegrees(f2);`
`gmFloor(f)`	greatest integer less than or equal to `f`	`f1 = gmFloor(f2);`
`gmFuzEQ(f1,f2)`	`true` iff `f1` is fuzzy equal to `f2`	`if (gmFuzEQ(f1,f2)) ...`
`gmFuzGEQ(f1,f2)`	`true` iff `f1` is fuzzy greater than or equal to `f2`	`if (gmFuzGEQ(f1,f2)) ...`
`gmFuzLEQ(f1,f2)`	`true` iff `f1` is fuzzy less than or equal to `f2`	`if (gmFuzLEQ(f1,f2)) ...`
`gmInv(f)`	inverse of `f`, $f \not\approx 0$	`f1 = gmInv(f2);`
`gmIsZero(f)`	`true` iff `f` is fuzzy equal to 0	`if (gmIsZero(f)) ...`
`gmLerp(f,f1,f2)`	linear interpolation from `f1` (when $f = 0$) to `f2` (when $f = 1$)	`f3 = gmLerp(f,f1,f2);`
`gmMax(f1,f2)`	maximum of `f1` and `f2`	`f = gmMax(f1,f2);`
`gmMax(f1,f2,f3)`	maximum of `f1` and `f2` and `f3`	`f = gmMax(f1,f2,f3);`
`gmMin(f1,f2)`	minimum of `f1` and `f2`	`f = gmMin(f1,f2);`
`gmMin(f1,f2,f3)`	minimum of `f1` and `f2` and `f3`	`f = gmMin(f1,f2,f3);`
`gmRadians(f)`	convert angle in degrees, `f`, to angle in radians	`f1 = gmRadians(f2);`
`gmRound(f)`	`f` rounded to nearest integer	`f1 = gmRound(f2);`
`gmSign(f)`	sign of `f` (-1 iff $f < 0$)	`f1 = gmSign(f2);`
`gmSlide(f,f1,f2)`	hermite interpolation from `f1` (when $f = 0$) to `f2` (when $f = 1$)	`f3 = gmSlide(f,f1,f2);`
`gmSmooth(f)`	smooth hermite interpolate of `f`	`f1 = gmSmooth(f2);`
`gmSqr(f)`	f^2	`f1 = gmSqr(f2);`
`gmSwap(f1,f2)`	swap `f1` and `f2`	`gmSwap(f1,f2);`
`gmSwap(i1,i2)`	swap `i1` and `i2`	`gmSwap(i1,i2);`
`gmTrunc(f)`	`f` truncated	`f1 = gmTrunc(f2);`
`gmZSign(f)`	zero or sign of `f` (-1, 0, or 1)	`f1 = gmZSign(f2);`

Table 3. *libgm* Vector Operators

op	description	usage	op	description	usage
$[i]$	access component i	`f = v[0];`	/=	destructive scalar divide	`v1 /= f;`
	2D: $i \in \{0,1\}$			requires: divisor $\not\approx 0$	
	3D: $i \in \{0,1,2\}$		+	addition	`v1 = v2 + v3;`
$[i]$	change component i	`v[0] = f;`	−	subtraction	`v1 = v2 - v3;`
	2D: $i \in \{0,1\}$		−	negation (unary)	`v1 = -v2;`
	3D: $i \in \{0,1,2\}$		*	scalar pre-multiply	`v1 = f * v2;`
=	assign	`v1 = v2;`	*	scalar post-multiply	`v1 = v2 * f;`
+=	destructive add	`v1 += v2;`	/	scalar divide	`v1 = v2 / f;`
−=	destructive subtract	`v1 -= v2;`		requires: divisor $\not\approx 0$	
*=	destructive scalar multiply	`v1 *= f;`	==	equality (fuzzy)	`if (v1 == v2) ...`
			!=	inequality (fuzzy)	`if (v1 != v2) ...`

Table 4. *libgm* Vector Functions

function	description	usage
`gmVector2()`	default constructor, initializes to `(0,0)`	`gmVector v;`
`gmVector2(v)`	copy constructor	`gmVector v1(v2);`
`gmVector2(f1,f2)`	constructor, initializes to `(f1,f2)`	`gmVector v(f1,f2);`
`assign(f1,f2)`	assign components	`v.assign(f1,f2);`
`gmVector3()`	default constructor, initializes to `(0,0)`	`gmVector v;`
`gmVector3(v)`	copy constructor	`gmVector v1(v2);`
`gmVector3(f1,f2,f3)`	constructor, initializes to `(f1,f2,f3)`	`gmVector v(f1,f2,f3);`
`assign(f1,f2,f3)`	assign components	`v.assign(f1,f2,f3);`
`cross(v1,v2)`	cross product of `v1` and `v2`	`v = cross(v1,v2);`
`distance(v1,v2)`	distance between points `v1` and `v2`	`f = distance(v1,v2);`
`distanceSquared(v1,v2)`	square of `distance(v1,v2)`	`f = distanceSquared(v1,v2);`
`dot(v1,v2)`	dot product of `v1` and `v2`	`f = dot(v1,v2);`
`length()`	length of this vector	`f = v.length();`
`lengthSquared()`	square of `length()`	`f = v.lengthSquared();`
`lerp(f,v1,v2)`	linear interpolation from `v1` (when $\texttt{f} = 0$) to `v2` (when $\texttt{f} = 1$)	`v = lerp(f,v1,v2);`
`normalize()`	normalize this vector `v` requires: `v.length()` $\not\approx 0$	`v.normalize();`

Table 5. *libgm* Matrix Operators

op	description	usage	op	description	usage
`[i][j]`	access element i, j	`f = M[0][0];`	+	addition	`M1 = M2 + M3;`
	3×3: $i, j \in \{0, 1, 2\}$		−	subtraction	`M1 = M2 - M3;`
	4×4: $i, j \in \{0, 1, 2, 3\}$		−	negation (unary)	`M1 = -M2;`
`[i][j]`	change element i, j	`M[0][0] = f;`	*	matrix multiply	`M1 = M2 * M3;`
	3×3: $i, j \in \{0, 1, 2\}$		*	scalar pre-multiply	`M1 = f * M2;`
	4×4: $i, j \in \{0, 1, 2, 3\}$		*	scalar post-multiply	`M1 = M2 * f;`
=	assign	`M1 = M2;`	*	row vector pre-multiply	`M1 = v * M2;`
+=	destructive add	`M1 += M2;`	*	column vector post-multiply	`M1 = M2 * v;`
−=	destructive subtract	`M1 -= M2;`			
*=	destructive matrix multiply	`M1 *= M2;`	/	scalar divide	`M1 = M2 / f;`
*=	destructive scalar multiply	`M1 *= f;`		requires: divisor $\not\approx \mathbf{0}$	
/=	destructive scalar divide	`M1 /= f;`	==	equality (fuzzy)	`if (M1 == M2) ...`
	requires: divisor $\not\approx \mathbf{0}$		!=	inequality (fuzzy)	`if (M1 != M2) ...`

Table 6. *libgm* Matrix Functions

function	description	usage
gmMatrix3()	default 3×3 matrix constructor, initializes to all zeroes	gmMatrix M;
gmMatrix3(M)	copy constructor	gmMatrix M1(M2);
gmMatrix3(f00,f01,f02, f10,f11,f12, f20,f21,f22)	constructor	gmMatrix M(f00,f01,f02, f10,f11,f12, f20,f21,f22)
assign(f00,f01,f02, f10,f11,f12, f20,f21,f22)	assign elements	M.assign(f00,f01,f02, f10,f11,f12, f20,21,22)
identity()	3×3 identity matrix	M = gmMatrix3::identity();
gmMatrix4()	default 4×4 matrix constructor, initializes to all zeroes	gmMatrix M;
gmMatrix4(M)	copy constructor	gmMatrix M1(M2);
gmMatrix4(f00,f01,f02,f03, f10,f11,f12,f13, f20,f21,f22,f23, f30,f31,f32,f33)	constructor	gmMatrix M(f00,f01,f02,f03, f10,f11,f12,f13, f20,f21,f22,f23, f30,f31,f32,f33)
assign(f00,f01,f02,f03, f10,f11,f12,f13, f20,f21,f22,f23, f30,f31,f32,f33)	assign elements	M.assign(f00,f01,f02,f03, f10,f11,f12,f13, f20,f21,f22,f23, f30,f31,f32,f33)
identity()	4×4 identity matrix	M = gmMatrix4::identity();
adjoint()	adjoint of this matrix	M1 = M2.adjoint();
determinant()	determinant of this matrix	f = M.determinant();
inverse()	inverse of this matrix M requires: !M.isSingular()	M1 = M2.inverse();
isSingular()	true iff this matrix is singular	if (M.isSingular()) ...
transpose()	transpose of this matrix	M1 = M2.transpose();