



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 `fk` ($k = 1, 2, \dots$) to denote variables of type `double`, and `i` or `ik` ($k = 1, 2, \dots$) to denote variables of type `int`. We use `v` or `vi` ($i = 1, 2, \dots$) to denote instances of vectors and `M` or `Mi` ($i = 1, 2, \dots$) 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 .

¹see volume I, pages 470-471.

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...	$\log e$	gmSQRTL2	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 [f1, f2]	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, $f \neq 0$	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>i</i> 2D: $i \in \{0, 1\}$ 3D: $i \in \{0, 1, 2\}$	<code>f = v[0];</code>	/=	destructive scalar divide requires: divisor $\neq 0$	<code>v1 /= f;</code>
[i]	change component <i>i</i> 2D: $i \in \{0, 1\}$ 3D: $i \in \{0, 1, 2\}$	<code>v[0] = f;</code>	+	addition	<code>v1 = v2 + v3;</code>
=	assign	<code>v1 = v2;</code>	-	subtraction	<code>v1 = v2 - v3;</code>
+=	destructive add	<code>v1 += v2;</code>	-	negation (unary)	<code>v1 = -v2;</code>
-=	destructive subtract	<code>v1 -= v2;</code>	*	scalar pre-multiply	<code>v1 = f * v2;</code>
*=	destructive scalar multiply	<code>v1 *= f;</code>	*	scalar post-multiply	<code>v1 = v2 * f;</code>
			/	scalar divide requires: divisor $\neq 0$	<code>v1 = v2 / f;</code>
			==	equality (fuzzy)	<code>if (v1 == v2) ...</code>
			!=	inequality (fuzzy)	<code>if (v1 != v2) ...</code>

Table 4. *libgm* Vector Functions

function	description	usage
<code>gmVector2()</code>	default constructor, initializes to (0,0)	<code>gmVector v;</code>
<code>gmVector2(v)</code>	copy constructor	<code>gmVector v1(v2);</code>
<code>gmVector2(f1,f2)</code>	constructor, initializes to (f1,f2)	<code>gmVector v(f1,f2);</code>
<code>assign(f1,f2)</code>	assign components	<code>v.assign(f1,f2);</code>
<code>gmVector3()</code>	default constructor, initializes to (0,0)	<code>gmVector v;</code>
<code>gmVector3(v)</code>	copy constructor	<code>gmVector v1(v2);</code>
<code>gmVector3(f1,f2,f3)</code>	constructor, initializes to (f1,f2,f3)	<code>gmVector v(f1,f2,f3);</code>
<code>assign(f1,f2,f3)</code>	assign components	<code>v.assign(f1,f2,f3);</code>
<code>cross(v1,v2)</code>	cross product of v1 and v2	<code>v = cross(v1,v2);</code>
<code>distance(v1,v2)</code>	distance between points v1 and v2	<code>f = distance(v1,v2);</code>
<code>distanceSquared(v1,v2)</code>	square of distance(v1,v2)	<code>f = distanceSquared(v1,v2);</code>
<code>dot(v1,v2)</code>	dot product of v1 and v2	<code>f = dot(v1,v2);</code>
<code>length()</code>	length of this vector	<code>f = v.length();</code>
<code>lengthSquared()</code>	square of length()	<code>f = v.lengthSquared();</code>
<code>lerp(f,v1,v2)</code>	linear interpolation from v1 (when f = 0) to v2 (when f = 1)	<code>v = lerp(f,v1,v2);</code>
<code>normalize()</code>	normalize this vector v requires: <code>v.length() $\neq 0$</code>	<code>v.normalize();</code>

Table 5. *libgm* Matrix Operators

op	description	usage	op	description	usage
$[i][j]$	access element i, j 3×3 : $i, j \in \{0, 1, 2\}$ 4×4 : $i, j \in \{0, 1, 2, 3\}$	$f = M[0][0];$	+	addition	$M1 = M2 + M3;$
			-	subtraction	$M1 = M2 - M3;$
			-	negation (unary)	$M1 = -M2;$
$[i][j]$	change element i, j 3×3 : $i, j \in \{0, 1, 2\}$ 4×4 : $i, j \in \{0, 1, 2, 3\}$	$M[0][0] = f;$	*	matrix multiply	$M1 = M2 * M3;$
			*	scalar pre-multiply	$M1 = f * M2;$
			*	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	$M1 -= M2;$	/	scalar divide requires: divisor $\neq 0$	$M1 = M2 / f;$
*=	destructive matrix multiply	$M1 *= M2;$	==	equality (fuzzy)	if $(M1 == M2) \dots$
*=	destructive scalar multiply	$M1 *= f;$!=	inequality (fuzzy)	if $(M1 != M2) \dots$
/=	destructive scalar divide requires: divisor $\neq 0$	$M1 /= f;$			

Table 6. *libgm* Matrix Functions

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