Manual

Table of Contents

C R V C V R C

C L R U L L C R V C

C U C

A C C R R A C C L L V R C G C R V R C

G R C C C C R C

G R C C C R L C L G C C R G C

C R G d C C V C V L R C

A C d C O C U V R C

A C d C L G O V G R V V R R C V L R C

C R G d C C L C C C

d L C C V C

A d C L G O A L V C L C A L V C A A L C d C A L C

A C R C C V R C C C R G G L L R C C R C

A V C L V C R C C L C G L C C

C R G C C

C R V C O V L C R C

C V L O V R C

C V R O V C

C L V L C

C C

C G R V C

R C L L V R C

C V C L O C

C R V C L C C

R O V R L C L C C

C U L V C R R V C C

C V V L O V R C C

C O R A G V C

Introduction

1. Getting started

1.1. Setup

<glm/glm.hpp>

<glm/vec2.hpp>: vec2, bvec2, dvec2, ivec2 and uvec2
<glm/vec3.hpp>: vec3, bvec3, dvec3, ivec3 and uvec3
<glm/vec4.hpp>: vec4, bvec4, dvec4, ivec4 and uvec4
<glm/mat2x2.hpp>: mat2, dmat2
<glm/mat2x3.hpp>: mat2x3, dmat2x3
<glm/mat2x4.hpp>: mat2x4, dmat2x4
<glm/mat3x2.hpp>: mat3x2, dmat3x2
<glm/mat3x3.hpp>: mat3, dmat3
<glm/mat3x4.hpp>: mat3x4, dmat2
<glm/mat4x2.hpp>: mat4x2, dmat4x2
<glm/mat4x3.hpp>: mat4x3, dmat4x3
<glm/mat4x4.hpp>: mat4, dmat4
<glm/common.hpp>: all the GLSL common functions
<glm/exponential.hpp>: all the GLSL exponential functions
<glm/geometry.hpp>: all the GLSL geometry functions
<glm/integer.hpp>: all the GLSL integer functions
<glm/matrix.hpp>: all the GLSL matrix functions
<glm/packing.hpp>: all the GLSL packing functions
<glm/trigonometric.hpp>: all the GLSL trigonometric functions
<glm/vector_relational.hpp>: all the GLSL vector relational functions

1.2. Faster program compilation

<glm/fwd.hpp>

```
// Header file
#include <glm/fwd.hpp>

// Source file
#include <glm/glm.hpp>
```

1.3. Use sample of GLM core

```
// Include GLM core features
#include <glm/vec3.hpp>
#include <glm/vec4.hpp>
#include <glm/mat4x4.hpp>
#include <glm/trigonometric.hpp>

// Include GLM extensions
#include <glm/gtc/matrix_transform.hpp>

glm::mat4 transform(
    glm::vec2 const& Orientation,
    glm::vec3 const& Translate,
    glm::vec3 const& Up)
{
    glm::mat4 Proj = glm::perspective(glm::radians(45.f), 1.33f, 0.1f, 10.f);
```

```
    glm::mat4 ViewTranslate = glm::translate(glm::mat4(1.f), Translate);
    glm::mat4 ViewRotateX = glm::rotate(ViewTranslate, Orientation.y, Up);
    glm::mat4 View = glm::rotate(ViewRotateX, Orientation.x, Up);
    glm::mat4 Model = glm::mat4(1.0f);

    return Proj * View * Model;
}
```

1.4. Dependencies

<glm/glm.hpp>

gl.h glcorearb.h gl3.h glu.h

windows.h <boost/static_assert.hpp>

static_assert

2. Swizzle operators

```
#define GLM_FORCE_SWIZZLE
#include <glm/glm.hpp>

void foo()
{
	glm::vec4 ColorRGBA(1.0f, 0.5f, 0.0f, 1.0f);

	// l-value:
	glm::vec4 ColorBGRA = ColorRGBA.bgra;

	// r-value:
	ColorRGBA.bgra = ColorRGBA;

	// Both l-value and r-value
	ColorRGBA.bgra = ColorRGBA.rgba;
	…
}
```

glm::vec2 glm::vec3 glm::vec4

3. Preprocessor options

3.1. Default precision

```
precision mediump int;
precision highp float;
```

glm.hpp

```
#define GLM_PRECISION_MEDIUMP_INT;
#define GLM_PRECISION_HIGHP_FLOAT;
#include <glm/glm.hpp>
```

glm::vec* glm::mat*

GLM_PRECISION_LOWP_FLOAT
GLM_PRECISION_MEDIUMP_FLOAT
GLM_PRECISION_HIGHP_FLOAT

glm::dvec* glm::dmat*

GLM_PRECISION_LOWP_DOUBLE
GLM_PRECISION_MEDIUMP_DOUBLE
GLM_PRECISION_HIGHP_DOUBLE

glm::ivec*

GLM_PRECISION_LOWP_INT
GLM_PRECISION_MEDIUMP_INT
GLM_PRECISION_HIGHP_INT

glm::uvec*

GLM_PRECISION_LOWP_UINT
GLM_PRECISION_MEDIUMP_UINT
GLM_PRECISION_HIGHP_UINT

3.2. Compile-time message system

GLM_FORCE_MESSAGES
<glm/glm.hpp>. The messages are generated only by compiler supporting #program message and only once per project build.

```
#define GLM_FORCE_MESSAGES
#include <glm/glm.hpp>
```

3.3. C++ language detection

GLM_FORCE_CXX98
<glm/glm.hpp>

```
#define GLM_FORCE_CXX98
#include <glm/glm.hpp>
```

GLM_FORCE_CXX11, GLM_FORCE_CXX14.

```
#define GLM_FORCE_CXX11
#include <glm/glm.hpp>
```

GLM_FORCE_CXX14 GLM_FORCE_CXX11 and GLM_FORCE_CXX11 GLM_FORCE_CXX98

3.4. SIMD support

/arch:AVX

GLM_FORCE_SSE2, GLM_FORCE_SSE3, GLM_FORCE_SSSE3, GLM_FORCE_SSE41, GLM_FORCE_SSE42, GLM_FORCE_AVX GLM_FORCE_AVX2 or GLM_FORCE_AVX512.

GLM_FORCE_PURE

```
#define GLM_FORCE_PURE
#include <glm/glm.hpp>

// GLM code will be compiled using pure C++ code
```

```
#define GLM_FORCE_AVX2
#include <glm/glm.hpp>

// If the compiler doesn't support AVX2 instrinsics,
// compiler errors will happen.
```

glm/simd

3.5. Force inline

GLM_FORCE_INLINE
<glm/glm.hpp>

```
#define GLM_FORCE_INLINE
#include <glm/glm.hpp>
```

3.6. Vector and matrix static size

.length()

```
#include <glm/glm.hpp>

void foo(vec4 const & v)
{
	int Length = v.length();
	…
}
```

int size_t

GLM_FORCE_SIZE_T_LENGTH length() size_t

glm::length_t length()

GLM_FORCE_SIZE_T_LENGTH

```
#define GLM_FORCE_SIZE_T_LENGTH
#include <glm/glm.hpp>

void foo(vec4 const & v)
{
	glm::size_t Length = v.length();
	…
}
```

3.7. Disabling default constructor initialization

GLM_FORCE_NO_CTOR_INIT

<glm/glm.hpp>

```
#include <glm/glm.hpp>

void foo()
{
	glm::vec4 v; // v is (0.0f, 0.0f, 0.0f, 0.0f)
	…
}
```

GLM_FORCE_NO_CTOR_INIT

```
#define GLM_FORCE_NO_CTOR_INIT
#include <glm/glm.hpp>

void foo()
{
	glm::vec4 v; // v is fill with garbage
	…
}
```

```
#include <glm/glm.hpp>

void foo()
{
```

```
        glm::vec4 v(glm::uninitialize);
        …
}
```

3.8. Require explicit conversions

ivec4

vec4

GLM_FORCE_EXPLICIT_CTOR

```
#include <glm/glm.hpp>

void foo()
{
        glm::ivec4 a;
        …
        glm::vec4 b(a); // Explicit conversion, OK
        glm::vec4 c = a; // Implicit conversion, OK
        …
}
```

GLM_FORCE_EXPLICIT_CTOR define

```
#define GLM_FORCE_EXPLICIT_CTOR
#include <glm/glm.hpp>

void foo()
{
        glm::ivec4 a;
        …
        glm::vec4 b(a); // Explicit conversion, OK
        glm::vec4 c = a; // Implicit conversion, ERROR
        …
}
```

3.9. Removing genType restriction

genType

```
#include <glm/glm.hpp>

typedef glm::tvec4<float> my_fvec4;
```

GLM_FORCE_UNRESTRICTED_GENTYPE

```
#define GLM_FORCE_UNRESTRICTED_GENTYPE
#include <glm/glm.hpp>
#include "half.hpp" // Define “half” class with equivalent behavior than “float”

typedef glm::tvec4<half> my_hvec4;
```

GLM_FORCE_UNRESTRICTED_GENTYPE GLM_FORCE_SWIZZLE

4. Stable extensions

```
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>

int foo()
{
	glm::vec4 Position = glm::vec4(glm:: vec3(0.0f), 1.0f);
	glm::mat4 Model = glm::translate(
		glm::mat4(1.0f), glm::vec3(1.0f));
	glm::vec4 Transformed = Model * Position;
	…
	return 0;
}
```

4.1. GLM_GTC_bitfield

<glm/gtc/bitfield.hpp>

4.2. GLM_GTC_color_space

<glm/gtc/color_space.hpp>

4.3. GLM_GTC_constants

<glm/gtc/constants.hpp>

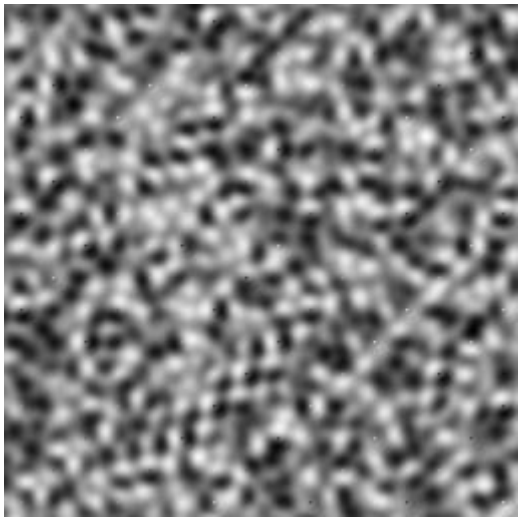
4.4. GLM_GTC_epsilon

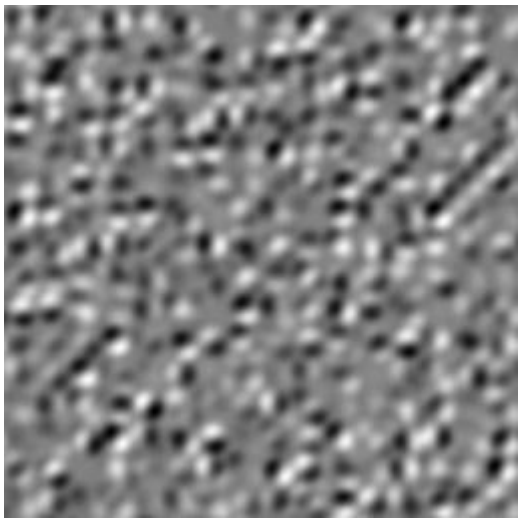
<glm/gtc/epsilon.hpp>

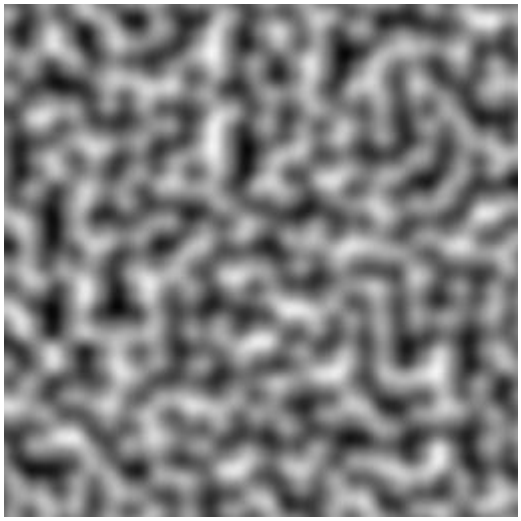
4.5. GLM_GTC_functions

<glm/gtc/functions.hpp>

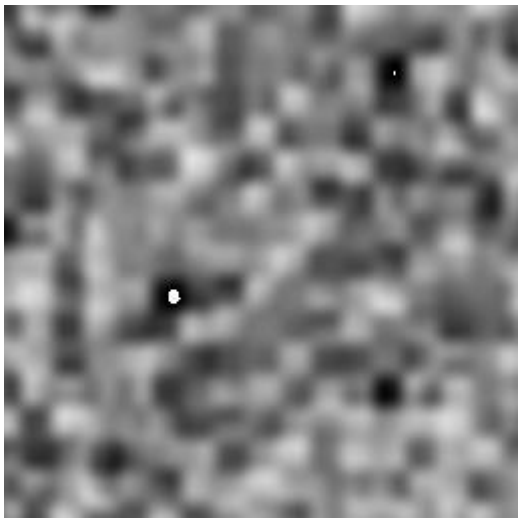
4.6. GLM_GTC_integer



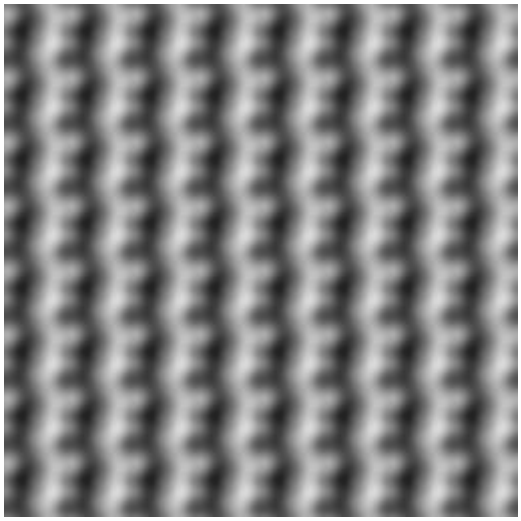




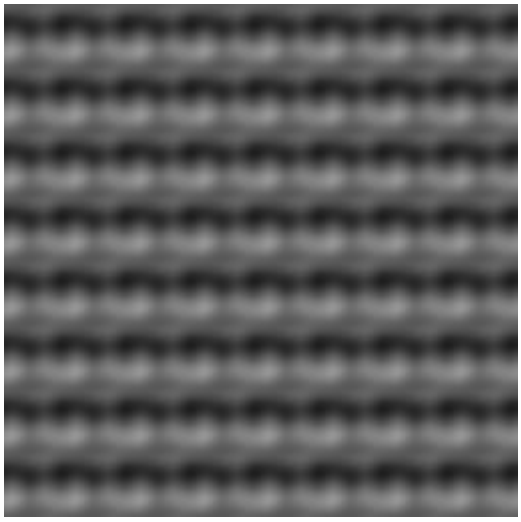
glm::perlin(glm::vec3(x / 16.f, y / 16.f, 0.5f));



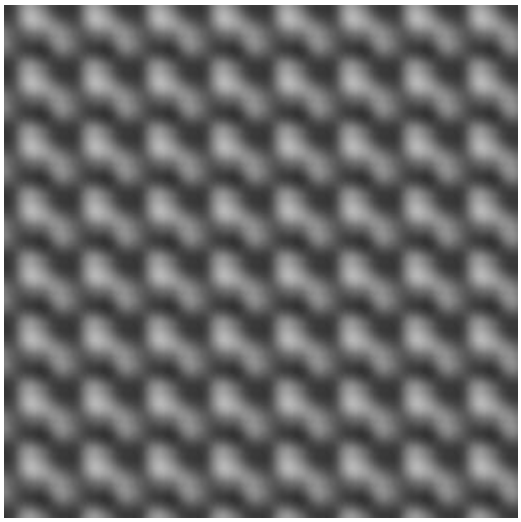
glm::perlin(glm::vec4(x / 16.f, y / 16.f, 0.5f, 0.5f)));



glm::perlin(glm::vec2(x / 16.f, y / 16.f), glm::vec2(2.0f));



glm::perlin(glm::vec3(x / 16.f, y / 16.f, 0.5f), glm::vec3(2.0f));



glm::perlin(glm::vec4(x / 16.f, y / 16.f, glm::vec2(0.5f)), glm::vec4(2.0f));

4.12. GLM_GTC_packing

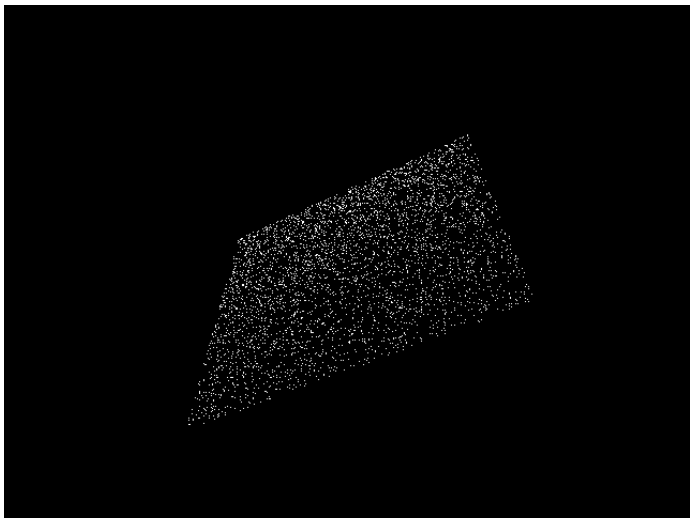
<glm/gtc/packing.hpp>

4.13. GLM_GTC_quaternion

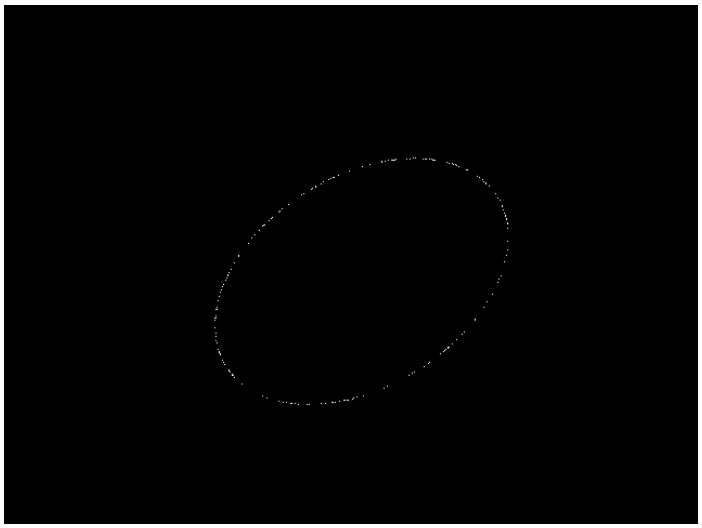
<glm/gtc/quaternion.hpp>

4.14. GLM_GTC_random

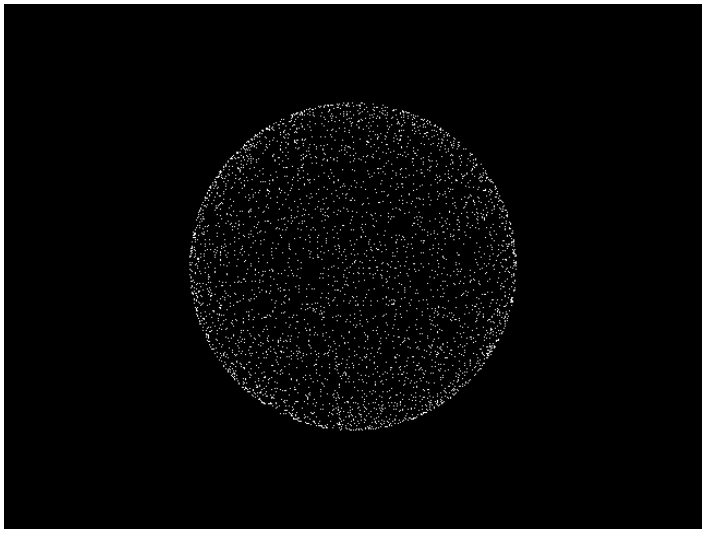
<glm/gtc/random.hpp>



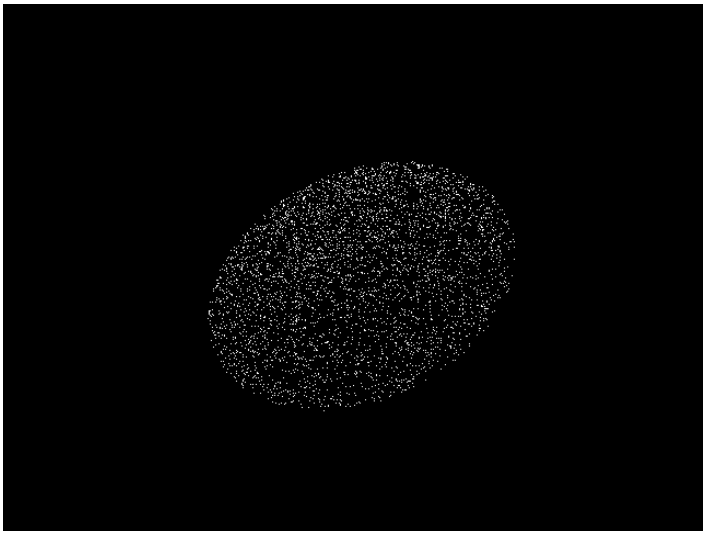
glm::vec4(glm::linearRand(glm::vec2(-1), glm::vec2(1)), 0, 1);



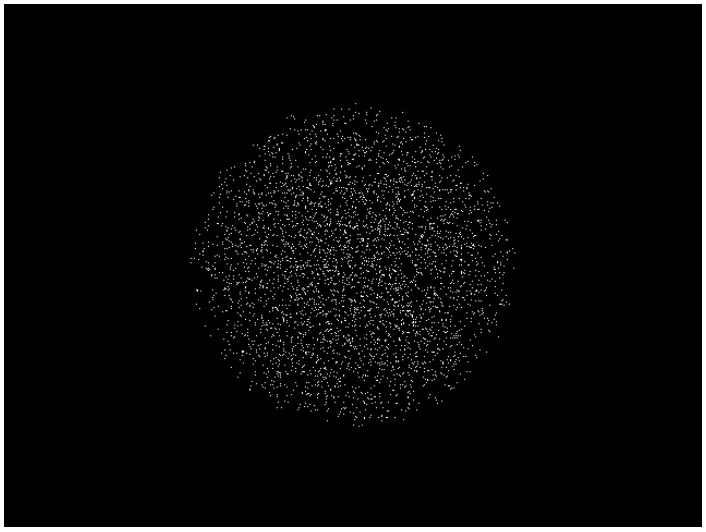
glm::vec4(glm::circularRand(1.0f), 0, 1);



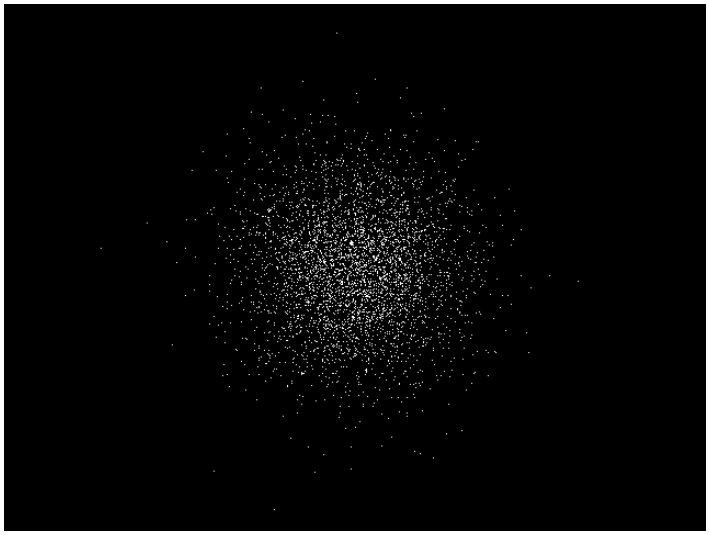
glm::vec4(glm::sphericalRand(1.0f), 1);



glm::vec4(glm::diskRand(1.0f), 0, 1);



glm::vec4(glm::ballRand(1.0f), 1);



glm::vec4(glm::gaussRand(glm::vec3(0), glm::vec3(1)), 1);

4.15. GLM_GTC_reciprocal

<glm/gtc/reciprocal.hpp>

4.16. GLM_GTC_round

<glm/gtc/round.hpp>

4.17. GLM_GTC_type_aligned

<glm/gtc/type_aligned.hpp>

4.18. GLM_GTC_type_precision

i8vec4

glm::i8vec*

GLM_PRECISION_LOWP_INT8
GLM_PRECISION_MEDIUMP_INT8
GLM_PRECISION_HIGHP_INT8

glm::u8vec*

GLM_PRECISION_LOWP_UINT8
GLM_PRECISION_MEDIUMP_UINT8
GLM_PRECISION_HIGHP_UINT8

glm::i16vec*

GLM_PRECISION_LOWP_INT16
GLM_PRECISION_MEDIUMP_INT16
GLM_PRECISION_HIGHP_INT16

glm::u16vec*

GLM_PRECISION_LOWP_UINT16
GLM_PRECISION_MEDIUMP_UINT16
GLM_PRECISION_HIGHP_UINT16

glm::i32vec*

GLM_PRECISION_LOWP_INT32
GLM_PRECISION_MEDIUMP_INT32
GLM_PRECISION_HIGHP_INT32

glm::u32vec*

GLM_PRECISION_LOWP_UINT32
GLM_PRECISION_MEDIUMP_UINT32
GLM_PRECISION_HIGHP_UINT32

glm::i64vec*

GLM_PRECISION_LOWP_INT64
GLM_PRECISION_MEDIUMP_INT64
GLM_PRECISION_HIGHP_INT64

glm::u64vec*

GLM_PRECISION_LOWP_UINT64
GLM_PRECISION_MEDIUMP_UINT64
GLM_PRECISION_HIGHP_UINT64

glm::f32vec* glm::f32mat* glm::f32quat

GLM_PRECISION_LOWP_FLOAT32
GLM_PRECISION_MEDIUMP_FLOAT32
GLM_PRECISION_HIGHP_FLOAT32

glm::f64vec* glm::f64mat* glm::f64quat

GLM_PRECISION_LOWP_FLOAT64
GLM_PRECISION_MEDIUMP_FLOAT64
GLM_PRECISION_HIGHP_FLOAT64

<glm/gtc/type_precision.hpp>

4.19. GLM_GTC_type_ptr

glm::value_ptr

vec3 mat4

```
// GLM_GTC_type_ptr extension provides a safe solution:
#include <glm/glm.hpp>
#include <glm/gtc/type_ptr.hpp>

void foo()
{
	glm::vec4 v(0.0f);
	glm::mat4 m(1.0f);
	...
	glVertex3fv(glm::value_ptr(v))
	glLoadMatrixfv(glm::value_ptr(m));
}

// Another solution inspired by STL:
#include <glm/glm.hpp>

void foo()
{
	glm::vec4 v(0.0f);
	glm::mat4 m(1.0f);
	...
	glVertex3fv(&v[0]);
	glLoadMatrixfv(&m[0][0]);
```

Note: It would be possible to implement glVertex3fv(glm::vec3(0)) in C++ with the appropriate cast operator that would result as an implicit cast in this example. However cast operators may produce programs running with unexpected behaviours without build error or any form of notification.

<glm/gtc/type_ptr.hpp>

4.20. GLM_GTC_ulp

ULP

<glm/gtc/ulp.hpp>

4.21. GLM_GTC_vec1

*vec1

<glm/gtc/vec1.hpp>

5. OpenGL interoperability

5.1. GLM replacements for deprecated OpenGL functions

glRotate{f, d}:

```
glm::mat4 glm::rotate(
        glm::mat4 const & m,
        float angle,
        glm::vec3 const & axis);

glm::dmat4 glm::rotate(
        glm::dmat4 const & m,
        double angle,
        glm::dvec3 const & axis);
```

GLM_GTC_matrix_transform <glm/gtc/matrix_transform.hpp>

glScale{f, d}:

```
glm::mat4 glm::scale(
        glm::mat4 const & m,
        glm::vec3 const & factors);

glm::dmat4 glm::scale(
        glm::dmat4 const & m,
        glm::dvec3 const & factors);
```

GLM_GTC_matrix_transform <glm/gtc/matrix_transform.hpp>

glTranslate{f, d}:C

```
glm::mat4 glm::translate(
        glm::mat4 const & m,
        glm::vec3 const & translation);

glm::dmat4 glm::translate(
        glm::dmat4 const & m,
        glm::dvec3 const & translation);
```

GLM_GTC_matrix_transform <glm/gtc/matrix_transform.hpp>

glLoadIdentity:C

```
glm::mat4(1.0) or glm::mat4();
glm::dmat4(1.0) or glm::dmat4();
```

<glm/glm.hpp>

glMultMatrix{f, d}: C

```
glm::mat4() * glm::mat4();
glm::dmat4() * glm::dmat4();
```

<glm/glm.hpp>

glLoadTransposeMatrix{f, d}: C

```
glm::transpose(glm::mat4());
glm::transpose(glm::dmat4());
```

<glm/glm.hpp>

glMultTransposeMatrix{f, d}: C

```
glm::mat4() * glm::transpose(glm::mat4());
glm::dmat4() * glm::transpose(glm::dmat4());
```

<glm/glm.hpp>

glFrustum: C

```
glm::mat4 glm::frustum(
        float left, float right,
        float bottom, float top,
```

```
        float zNear, float zFar);

glm::dmat4 glm::frustum(
        double left, double right,
        double bottom, double top,
        double zNear, double zFar);
```

GLM_GTC_matrix_transform <glm/gtc/matrix_transform.hpp>

glOrtho: C

```
glm::mat4 glm::ortho(
        float left, float right,
        float bottom, float top,
        float zNear, float zFar);

glm::dmat4 glm::ortho(
        double left, double right,
        double bottom, double top,
        double zNear, double zFar);
```

GLM_GTC_matrix_transform <glm/gtc/matrix_transform.hpp>

5.2. GLM replacements for GLU functions

gluLookAt: C

```
glm::vec3 project(
        glm::vec3 const & obj,
        glm::mat4 const & model,
        glm::mat4 const & proj,
        glm::{i, ' '}vec4 const & viewport);

glm::dvec3 project(
        glm::dvec3 const & obj,
        glm::dmat4 const & model,
        glm::dmat4 const & proj,
```

C

GLM_GTC_matrix_transform <glm/gtc/matrix_transform.hpp>

gluUnProject: **C**

```
glm::vec3 unProject(
        glm::vec3 const & win,
        glm::mat4 const & model,
        glm::mat4 const & proj,
        glm::{i, ' '}vec4 const & viewport);

glm::dvec3 unProject(
        glm::dvec3 const & win,
        glm::dmat4 const & model,
        glm::dmat4 const & proj,
        glm::{i, ' ', d}vec4 const & viewport);
```

GLM_GTC_matrix_transform <glm/gtc/matrix_transform.hpp>

6. Known issues

6.1. not function

not_

6.2. Precision qualifiers support

vec4 lowp_vec4 mediump_vec4 highp_vec4

```
// Using precision qualifier in GLSL:

ivec3 foo(in vec4 v)
{
	highp vec4 a = v;
	mediump vec4 b = a;
	lowp ivec3 c = ivec3(b);

	return c;
}

// Using precision qualifier in GLM:
#include <glm/glm.hpp>

ivec3 foo(const vec4 & v)
{
	highp_vec4 a = v;
	medium_vec4 b = a;
	lowp_ivec3 c = glm::ivec3(b);

	return c;
```

7. FAQ

7.1 Why GLM follows GLSL specification and conventions?

7.2. Does GLM run GLSL program?

7.3. Does a GLSL compiler build GLM codes?

7.4. Should I use 'GTX' extensions?

7.5. Where can I ask my questions?

7.6. Where can I find the documentation of extensions?

7.7. Should I use 'using namespace glm;'?

```
using namespace glm;
```

using namespace glm;

7.8. Is GLM fast?

lowp mediump highp

7.9. When I build with Visual C++ with /W4 warning level, I have warnings...

7.10. Why some GLM functions can crash because of division by zero?

`glm::normalize`

7.11. What unit for angles is used in GLM?

8. Code samples

8.1. Compute a triangle normal

```
#include <glm/glm.hpp> // vec3 normalize cross

glm::vec3 computeNormal
(
	glm::vec3 const & a,
	glm::vec3 const & b,
	glm::vec3 const & c
)
{
	return glm::normalize(glm::cross(c - a, b - a));
}

// A much faster but less accurate alternative:
#include <glm/glm.hpp> // vec3 cross
#include <glm/gtx/fast_square_root.hpp> // fastNormalize

glm::vec3 computeNormal
(
	glm::vec3 const & a,
	glm::vec3 const & b,
	glm::vec3 const & c
)
{
	return glm::fastNormalize(glm::cross(c - a, b - a));
}
```

8.2. Matrix transform

```
// vec3, vec4, ivec4, mat4
#include <glm/glm.hpp>
// translate, rotate, scale, perspective
#include <glm/gtc/matrix_transform.hpp>
// value_ptr
#include <glm/gtc/type_ptr.hpp>

void setUniformMVP
(
	GLuint Location,
	glm::vec3 const & Translate,
	glm::vec3 const & Rotate
)
{
	glm::mat4 Projection =
	glm::perspective(45.0f, 4.0f / 3.0f, 0.1f, 100.f);
	glm::mat4 ViewTranslate = glm::translate(
	glm::mat4(1.0f),
	Translate);
	glm::mat4 ViewRotateX = glm::rotate(
		ViewTranslate,
		Rotate.y, glm::vec3(-1.0f, 0.0f, 0.0f));
	glm::mat4 View = glm::rotate(
		ViewRotateX,
		Rotate.x, glm::vec3(0.0f, 1.0f, 0.0f));
	glm::mat4 Model = glm::scale(
		glm::mat4(1.0f),
		glm::vec3(0.5f));
	glm::mat4 MVP = Projection * View * Model;
	glUniformMatrix4fv(Location, 1, GL_FALSE, glm::value_ptr(MVP));
}
```

8.3. Vector types

```
#include <glm/glm.hpp> //vec2
#include <glm/gtc/type_precision.hpp> //hvec2, i8vec2, i32vec2

std::size_t const VertexCount = 4;

// Float quad geometry
std::size_t const PositionSizeF32 = VertexCount * sizeof(glm::vec2);
glm::vec2 const PositionDataF32[VertexCount] =
{
glm::vec2(-1.0f,-1.0f),
glm::vec2( 1.0f,-1.0f),
glm::vec2( 1.0f, 1.0f),
glm::vec2(-1.0f, 1.0f)
};
// Half-float quad geometry
std::size_t const PositionSizeF16 = VertexCount * sizeof(glm::hvec2);
glm::hvec2 const PositionDataF16[VertexCount] =
{
glm::hvec2(-1.0f, -1.0f),
glm::hvec2( 1.0f, -1.0f),
glm::hvec2( 1.0f, 1.0f),
glm::hvec2(-1.0f, 1.0f)
};
// 8 bits signed integer quad geometry
std::size_t const PositionSizeI8 = VertexCount * sizeof(glm::i8vec2);
glm::i8vec2 const PositionDataI8[VertexCount] =
{
glm::i8vec2(-1,-1),
glm::i8vec2( 1,-1),
glm::i8vec2( 1, 1),
glm::i8vec2(-1, 1)
};
// 32 bits signed integer quad geometry
std::size_t const PositionSizeI32 = VertexCount * sizeof(glm::i32vec2);
glm::i32vec2 const PositionDataI32[VertexCount] =
{
	glm::i32vec2(-1,-1),
	glm::i32vec2( 1,-1),
	glm::i32vec2( 1, 1),
	glm::i32vec2(-1, 1)
};
```

8.4. Lighting

```
#include <glm/glm.hpp> // vec3 normalize reflect dot pow
#include <glm/gtx/random.hpp> // vecRand3

// vecRand3, generate a random and equiprobable normalized vec3
glm::vec3 lighting
(
	intersection const & Intersection,
	material const & Material,
	light const & Light,
	glm::vec3 const & View
)
{
	glm::vec3 Color = glm::vec3(0.0f);
	glm::vec3 LightVertor = glm::normalize(
		Light.position() - Intersection.globalPosition() +
		glm::vecRand3(0.0f, Light.inaccuracy());
	if(!shadow(
		Intersection.globalPosition(),
		Light.position(),
		LightVertor))
	{
		float Diffuse = glm::dot(Intersection.normal(), LightVector);
		if(Diffuse <= 0.0f)
			return Color;
		if(Material.isDiffuse())
			Color += Light.color() * Material.diffuse() * Diffuse;
		if(Material.isSpecular())
```

```
		{
			glm::vec3 Reflect = glm::reflect(
				-LightVector,
				Intersection.normal());
			float Dot = glm::dot(Reflect, View);
			float Base = Dot > 0.0f ? Dot : 0.0f;
			float Specular = glm::pow(Base, Material.exponent());
			Color += Material.specular() * Specular;
		}
	}
	return Color;
}
```

9. References

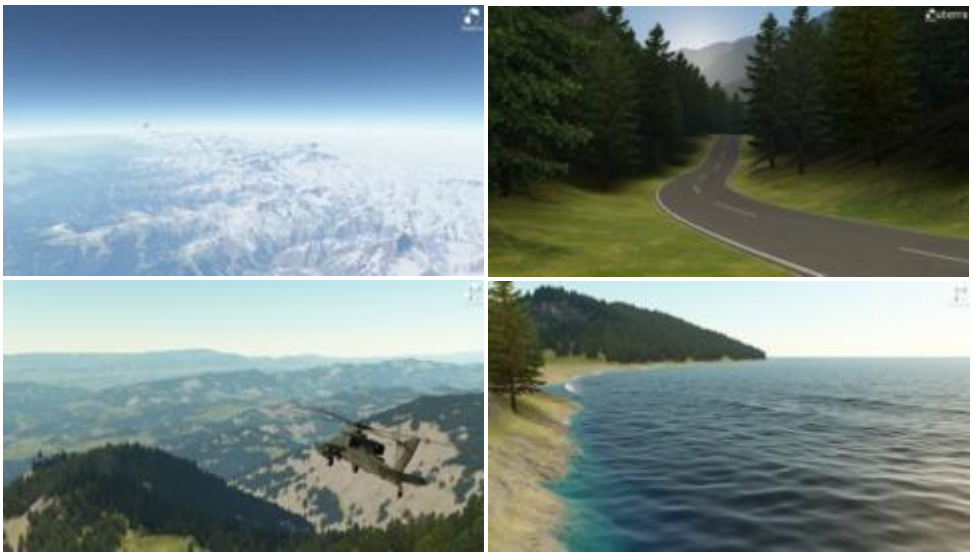
9.1. GLM development

9.2. OpenGL specifications

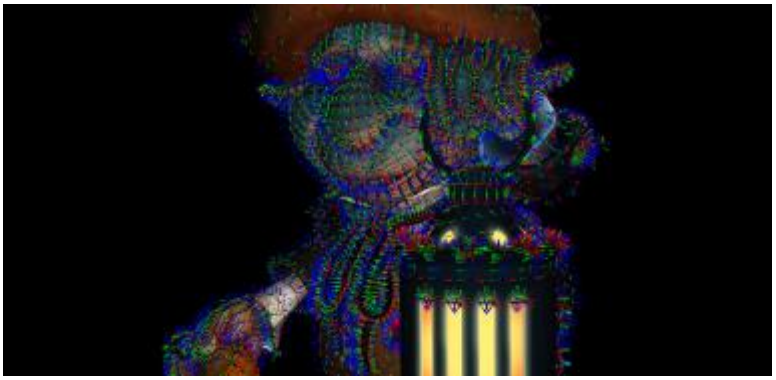
9.3. External links

9.4. Projects using GLM

R tt C

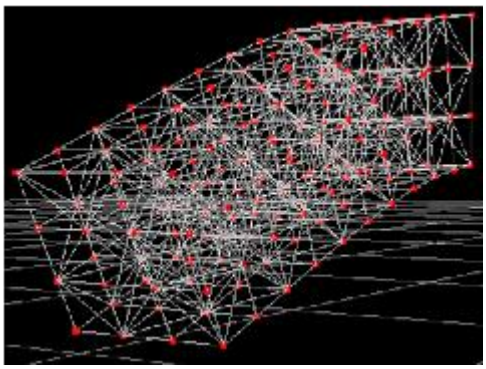


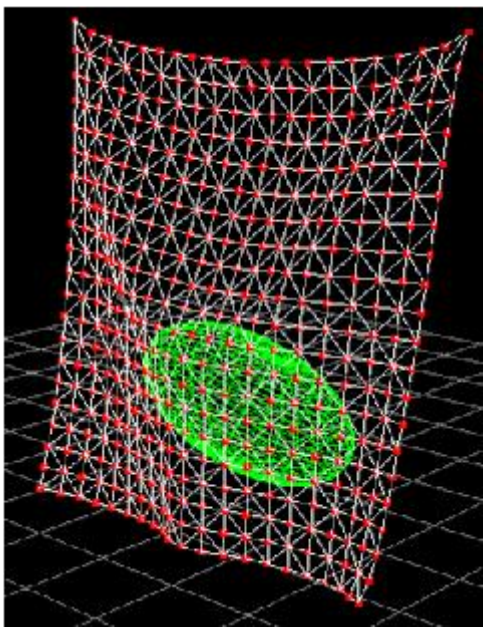
r C



k C

r rtj C





Cr C k C Ck CC r ttC

9.5. OpenGL tutorials using GLM

9.6. Equivalent for other languages

9.7. Alternatives to GLM

9.8. Acknowledgements