

Manual

Version 0.8.0

22 November 2015



The Happy Bunny License (Modified MIT License)

Copyright (c) 2008 - 2015 G-Truc Creation

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

Restrictions: By making use of the Software for military purposes, you choose to make a Bunny unhappy.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.



The MIT License

Copyright (c) 2008 - 2015 G-Truc Creation

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.



**Table of Contents**

Introduction 5

1. Getting started 6

1.1. Setup 6

1.2. Faster program compilation 6

1.3. Use sample of GLM core 6

1.4. Dependencies 7

2. Features 8

2.1. Loading texture files 8

2.2. Loading texture in GPU memory 8

2.3. Manipulating texture data on CPU 8

2.4. Generating, accessing, modifying and sampling texture data on CPU 8

3. Texture container formats 9

3.1. Khronos Image Format (KMG) 9

3.2. Khronos Texture Format (KTX) 9

3.3. Direct Draw Surface (DDS) 9

4. Graphics APIs interoperability 10

4.1. OpenGL 10

4.2. OpenGL ES 10

4.3. Vulkan 10

4.4. Direct3D 10

5. Known issues 11

5.1. not function 11

5.2. Precision qualifiers support 11

7. FAQ 12

7.1 Why GLM follows GLSL specification and conventions? 12

7.2. Does GLM run GLSL program? 12

7.3. Does a GLSL compiler build GLM codes? 12

7.4. Should I use ‘GTX’ extensions? 12

7.5. Where can I ask my questions? 12

7.6. Where can I find the documentation of extensions? 12

8. Code samples 13

8.1. Compute a triangle normal 13

8.2. Matrix transform 13

Introduction

OpenGL Mathematics (GLM) is a C++ mathematics library for graphics C++ programs based on the [OpenGL Shading Language](http://www.opengl.org/registry/) (GLSL) specifications.

GLM provides classes and functions designed and implemented with the same naming conventions and functionalities than GLSL so that when a programmer knows GLSL, he knows GLM as well which makes it really easy to use.

This project isn't limited to GLSL features. An extension system, based on the GLSL extension conventions, provides extended capabilities: matrix transformations, quaternions, data packing, random numbers, noise, etc...

This library works perfectly with [OpenGL](http://www.opengl.org) but it also ensures interoperability with other third party libraries and SDK. It is a good candidate for software rendering (raytracing / rasterisation), image processing, physic simulations and any development context that requires a simple and convenient mathematics library.

GLM is written in C++98 but can take advantage of C++11 when supported by the compiler. It is a platform independent library with no dependence and it officially supports the following compilers:

- [Apple Clang](https://developer.apple.com/Library/mac/documentation/CompilerTools/Conceptual/LLVMCompilerOverview/index.html) 4.0 and higher  
- [GCC](http://gcc.gnu.org/) 4.2 and higher

- [Intel C++ Composer](https://software.intel.com/en-us/intel-compilers) XE 2013 and higher  
- [LLVM](http://llvm.org/) 3.0 and higher

- [Visual C++](http://www.visualstudio.com/) 2010 and higher

- [CUDA](https://developer.nvidia.com/about-cuda) 4.0 and higher (experimental)

- Any conform C++98 or C++11 compiler

The source code and the documentation, including this manual, are licensed under [the Happy Bunny License (Modified MIT) and the MIT License](http://glm.g-truc.net/copying.txt).

Thanks for contributing to the project by [submitting reports](https://github.com/g-truc/glm/issues) for bugs and feature requests. Any feedback is welcome at [glm@g-truc.net](mailto:glm@g-truc.net).

1. Getting started

1.1. Setup

GLM is a header only library. Hence, there is nothing to build to use it. To use GLM, a programmer only has to include <glm/glm.hpp> in his program. This include provides all the GLSL features implemented by GLM.

Core GLM features can be included using individual headers to allow faster user program compilations.

<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 is a header only library that makes a heavy usage of C++ templates. This design may significantly increase the compile time for files that use GLM. Hence, it is important to limit GLM inclusion to header and source files that actually use it. Likewise, GLM extensions should be included only in program sources using them.

To further help compilation time, GLM 0.9.5 introduced <glm/fwd.hpp> that provides forward declarations of GLM types.

|  |
| --- |
| // Header file  #include <glm/fwd.hpp>  // Source file  #include <glm/glm.hpp> |

1.3. Use sample of GLI

|  |
| --- |
| // Include GLM core features  #include <glm/vec3.hpp>  #include <glm/vec4.hpp>  #include <glm/mat4x4.hpp>  // Include GLM extensions  #include <glm/gtc/matrix\_transform.hpp>  glm::mat4 transform(  glm::vec2 const & Orientation,  glm::vec3 const & Translate,  glm::vec2 const & Up)  {  glm::mat4 Projection = glm::perspective(45.0f, 4.0f / 3.0f, 0.1f, 100.0f);  glm::mat4 ViewTranslate = glm::translate(glm::mat4(1.0f), 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 Projection \* View \* Model;  } |

1.4. Dependencies

When <glm/glm.hpp> is included, GLM provides all the GLSL features it implements in C++.

There is no dependence with external libraries or external headers such as gl.h, [glcorearb.h](http://www.opengl.org/registry/api/GL/glcorearb.h), gl3.h, glu.h or windows.h. However, if <boost/static\_assert.hpp> is included, [Boost static assert](http://www.boost.org/doc/libs/1_52_0/doc/html/boost_staticassert.html) will be used all over GLM code to provide compiled time errors unless GLM is built with a C++ 11 compiler in which case [static\_assert](http://en.cppreference.com/w/cpp/language/static_assert). If neither are detected, GLM will rely on its own implementation of static assert.

2. Features

2.1. Loading texture files

Blabla

2.2. Loading texture in GPU memory

IBlabla.

2.3. Manipulating texture data on CPU

Blabla.

2.4. Generating, accessing, modifying and sampling texture data on CPU

Blabla.

3. Texture container formats

3.1. Khronos Image Format (KMG)

Blabla

3.2. Khronos Texture Format (KTX)

IBlabla.

3.3. Direct Draw Surface (DDS)

Blabla.

4. Graphics APIs interoperability

4.1. OpenGL

OpenGL 3.1 specification has deprecated some features that have been removed from OpenGL 3.2 core profile specification. GLM provides some replacement 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); |

From GLM\_GTC\_matrix\_transform extension: <glm/gtc/matrix\_transform.hpp>

4.2. OpenGL ES

**gluOrtho2D:**

|  |
| --- |
| glm::mat4 glm::ortho(  float left, float right, float bottom, float top);    glm::dmat4 glm::ortho(  double left, double right, double bottom, double top); |

From GLM\_GTC\_matrix\_transform extension: <glm/gtc/matrix\_transform.hpp>

4.3. Vulkan

**gluLookAt:**

|  |
| --- |
| glm::mat4 glm::lookAt(  glm::vec3 const & eye,  glm::vec3 const & center,  glm::vec3 const & up); |

From GLM\_GTC\_matrix\_transform extension: <glm/gtc/matrix\_transform.hpp>

4.4. Direct3D

**gluOrtho2D:**

|  |
| --- |
| glm::mat4 glm::ortho(  float left, float right, float bottom, float top);    glm::dmat4 glm::ortho(  double left, double right, double bottom, double top); |

From GLM\_GTC\_matrix\_transform extension: <glm/gtc/matrix\_transform.hpp>

5. Known issues

This section reports the divergences of GLM with GLSL.

5.1. not function

The GLSL keyword not is also a keyword in C++. To prevent name collisions, ensure cross compiler support and a high API consistency, the GLSL not function has been implemented with the name not\_.

5.2. Precision qualifiers support

GLM supports GLSL precision qualifiers through prefixes instead of qualifiers. For example, additionally to vec4, GLM exposes lowp\_vec4, mediump\_vec4 and highp\_vec4 types.

Similarly to GLSL, GLM precision qualifiers are used to handle trade-off between performances and precisions of operations in term of [ULPs](http://en.wikipedia.org/wiki/Unit_in_the_last_place).

By default, all the types use high precision.

|  |
| --- |
| // 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. Where can I ask my questions?

A good place is the [OpenGL Toolkits](http://www.opengl.org/discussion_boards/ubbthreads.php?ubb=postlist&Board=10&page=1) forum on [OpenGL.org](http://www.opengl.org/).

7.2 How should I report my issues?

Following GLSL conventions is a really strict policy of GLM. It has been designed following the idea that everyone does its own math library with his own conventions. The idea is that brilliant developers (the OpenGL ARB) worked together and agreed to make GLSL. Following GLSL conventions is a way to find consensus. Moreover, basically when a developer knows GLSL, he knows GLM.

7.2. Does GLM run GLSL program?

No, GLM is a C++ implementation of a subset of GLSL.

7.3. Does a GLSL compiler build GLM codes?

No, this is not what GLM attends to do.

7.4. Should I use ‘GTX’ extensions?

GTX extensions are qualified to be experimental extensions. In GLM this means that these extensions might change from version to version without any restriction. In practice, it doesn’t really change except time to time. GTC extensions are stabled, tested and perfectly reliable in time. Many GTX extensions extend GTC extensions and provide a way to explore features and implementations and APIs and then are promoted to GTC extensions. This is fairly the way OpenGL features are developed; through extensions.

7.6. Where can I find the documentation of extensions?

The Doxygen generated documentation includes a complete list of all extensions available. Explore this [API documentation](http://glm.g-truc.net/html/index.html) to get a complete view of all GLM capabilities!

8. Code samples

This series of samples only shows various GLM features without consideration of any sort.

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));  } |