GLM: Manual

Version 0.8.3
May 5, 2009

Summary

1. Introduction

OpenGL Mathematics (GLM) is a C++ mathematics library for 3D applications based on the OpenGL Shading Language (GLSL) specification.

The goal of the project is to provide to 3D programmers math classes and functions that miss in C++ when we use to program with GLSL or any high level GPU language. With GLM, the idea is to have a library that works the same way that GLSL which imply a strict following of GLSL specification for the implementation.

However, this project isn't limited by GLSL features. An extension system based on GLSL extensions development conventions allows to extend GLSL capabilities.

GLM is release under MIT license and available for all version of GCC from version 3.4 and Visual Studio from version 8.0 as a platform independent library.

Any feedback is welcome, please send them to g.truc.creation[NO_SPAM_THANKS]gmail.com.

---

2. Installation

2.1. Compiler setup

It's not required to build GLM, it's a header library. You use have to indicate where is the "glm" directory to your compiler. The only files present in this directory that matter for your own projects are header files. You can whether copy this directory in your "include" project directory or add this directory to your compiler header directories list (-I with GCC).

GLM is a header library. It implies that it could significantly increase the compiling time of your software. Consequently, it’s recommended to use precompiled headers.

2.2. Core features

When you have setup your compiler or project, all core features of GLM (basically, GLSL features) will be available to your project if you include "glm.h": #include <glm/glm.h>.

The directory of the file might change according your compiler or project setup. There are no restrictions or dependences with “gl.h”, “glu.h” or “windows.h”.

2.3. Setup of swizzle operators

Swizzle operators are disabled by default. It's possible to enable each component types by defining GLM_SWIZZLE to GLM_SWIZZLE_XYZW, GLM_SWIZZLE_RGBA or GLM_SWIZZLE_STQP. To enable all swizzle names, use GLM_SWIZZLE_FULL.

This setup is done using defines included in "glmsetup.h". You can directly edit this file but you take the risk to replace "glmsetup.h" and lose your settings when you will update GLM. Other way is to include "glmsetup.h" in a third-party file and then add your settings before including "glm.h". GLM will use default settings if "glmsetup.h" isn't included before "glm.h".

2.4. Use sample of GLM core

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

using namespace glm;

int foo()
{
	vec4 Position = vec4(vec3(0.0), 1.0);

	mat4 Model = mat4(1.0);
	Model[4] = vec4(1.0, 1.0, 0.0, 1.0);
	vec4 Transformed = Model * Position;

	return 0;
}
```

---

3. GLM Extensions

3.1. Description

GLM extends GLSL features with a large number of extensions for quaternion, transformation, spline, matrix inverse, color spaces, etc.

GLM provides two methods to use these extensions.

Some extensions could involve name collisions. It means that at least two incompatible extensions have been used.

3.2. Include method

This method requires to include each extension needed from the directory where it is implemented before being used. The extension features become available in “glm” namespace.

```
#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(1.0f, 1.0f, 1.0f);
	glm::vec4 Transformed = Model * Position;

	return 0;
```

```
}
```

3.3. Extension name method (Deprecated)

This method only requires to include each extension needed before being used using the extension name. The extension features become available in “glm” namespace.

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

int foo()
{
	glm::vec4 Position = glm::vec4(glm:: vec3(0.0f), 1.0f);

	glm::mat4 Model = glm::translate(glm::mat4(1.0f), 1.0f, 1.0f, 1.0f);
	glm::vec4 Transformed = Model * Position;

	return 0;
}
```

3.4. Namespace method (Deprecated)

To take advantages of all extra features of GLM, you have to use the extensions which are all included in "glmext.h": #include <glm/glmext.h>.

However, it's also required to explicitly use each single extension.

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

namespace glm
{
	using GLM_GTX_transform;
}

int foo()
{
	glm::vec4 Position = glm::vec4(glm::vec3 (0.0f), 1.0f);

	glm::mat4 Model = glm::translate(glm::mat4(1.0f), 1.0f, 1.0f, 1.0f);
	glm::vec4 Transformed = Model * Position;

	return 0;
}
```

---

4. Dependencies

4.1. Internal dependencies

Including <glm/glm.h> provides all the GLSL features implemented in GLM for C++ programmers. It’s a totally independent part; no extension is involved in any way into it.

Including <glm/glmext.h> provides all the features of all extensions. Each extension depends of GLM core, <glm/glm.h>, and possibly of other extension. You can include extension one by one so that if any extension depend of others extensions, those extensions will be automatically included.

4.2. External dependencies

GLM 0.7.3 introduces dependency to external libraries with the extension GLM_VIRTREV_gl which depend on GLEW. To prevent the dependence of external libraries when extensions aren't even used a new mechanism have been introduced in GLM 0.7.4.

The define GLM_DEPENDENCE must be declared to allow the automatic inclusion of extensions with external dependencies. For example:

```
#define GLM_DEPENDENCE GLM_DEPENDENCE_GLEW | GLM_DEPENDENCE_BOOST
#include <glm/glm.h>
#include <glm/glmext.h>
```

This will include all extensions that depend on GLEW and BOOST but obviously every extension with not external dependency.

---

5. Debugging

5.1. Build message system

GLM 0.7.5 introduces a new message system allowing to information at build GLM configurations and build. By default this system is disable (#define GLM_MESSAGE GLM_MESSAGE_QUIET).

This is the list of options for the build message system:
- GLM_MESSAGE_QUIET: No information display at build.
- GLM_MESSAGE_WARNING: Display warning messages, something is wrong.
- GLM_MESSAGE_NOTIFICATION: Display notifications about what happen at build
- GLM_MESSAGE_CORE: Display notifications or warnings from the core library
- GLM_MESSAGE_EXTS: Display notifications or warnings from the extensions libraries
- GLM_MESSAGE_SETUP: Display notifications or warnings from setup
- GLM_MESSAGE_ALL: Display everything

5.2. Static assert

If you use GLM_DEPENDENCE_BOOST to define GLM_DEPENDENCE or if you include <boost/static_assert.hpp> before including GLM then the setup system will automatically enable the use of static assert through GLM code to reduce the cases of bad use of templated functions or types

---

6. Known issues

6.1. Swizzle operators

Enabling swizzle operator can result to name collision with the Win32 API. To prevent these issues you can access to the internal swizzle operator functions without making swizzle operator enable. This is done with
#define GLM_SWIZZLE GLM_SWIZZLE_FUNC

6.2. "not" function

The GLSL keyword "not" is also a keyword in C++. To prevent name collisions, the GLSL not function has been implemented with the name “not_”

6.3. "half" based types

GLM support half float numbers through the extension GLM_GTX_half. This extension provides types half, hvec*, hmat*x* and hquat*.

Unfortunately, C++ norm doesn't support anonymous union so that hvec* vector components could be access only with x, y, z and w.

However, Visual C++ does support anonymous union. To enable the support of all component names (x,y,z,w;r,g,b,a;s,t,p,q) define GLM_USE_ANONYMOUS_UNION. With GCC it will result as build errors.

---

7. Extensions list

The Doxygen documentation includes a complete list of all extensions available. Please report to it to discover all GLM capabilities!