GLM: Manual

Version 0.8.4
September 30, 2009

Christophe Riccio
g.truc.creation[at]gmail.com

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 3D programmers with math classes in C++ that are similar to GLSL or any high level GPU language. With GLM, the idea is to have a library that has identical naming conventions and functionality as GLSL, which requires a strict following of the GLSL specification for implementation.

However, this project isn't limited by GLSL features. An extension system based on the GLSL extension conventions allows for extended GLSL capabilities.

GLM is written as a platform independent library and supports the following compilers:

- GNU GCC 3.4 and higher
- Microsoft Visual Studio 8.0 and higher

GLM source code is licensed under the MIT licence.

Any feedback is welcome and can be sent to **g.truc.creation[at]gmail.com.**

2. Installation

2.1. Compiler setup

GLM is a pure header library and therefore does not require to be built separately. GLM usage is achieved by simply directing the compiler to add the GLM install path to the include search paths. (-I option with GCC) Another option is to copy the GLM files directly into the project's source directory.

As GLM is a pure header library that also makes heavy usages of templates, there may be a significant increase in a project's compile time for files that use GLM. For this reason, usage of GLM as a precompiled header is recommended.

2.2. Core features

After initial compiler setup, all core features of GLM (core GLSL features) can be accessed by including the "glm.hpp" header. The line: #include <glm/glm.hpp> is used for a typical compiler setup.

Note that by default there are no dependencies on external headers like "gl.h", "gl3.h", "glu.h" or "windows.h".

2.3. Setup of swizzle operators

A common feature of shader languages like GLSL is swizzling. This involves being able to select what components of a vector are used and in what order. For example "variable.x","variable.xxy", "variable.zxyy" are examples of swizzling.
However in GLM, swizzling operators are disabled by default. To enable swizzling the define "GLM_SWIZZLE" must be defined to one of GLM_SWIZZLE_XYZW, GLM_SWIZZLE_RGBA , GLM_SWIZZLE_STQP or GLM_SWIZZLE_FULL depending on what swizzle syntax is required.

The swizzle defines are supplied in the file "setup.hpp" and a simple way of enabling swizzling is to edit this file. However to avoid settings being lost on future GLM upgrades, it is suggested that "setup.hpp" be included first, then custom settings and finally "glm.hpp". For example:

```
#include  <glm/setup.hpp>
#define GLM_SWIZZLE  GLM_SWIZZLE_FULL
#include  <glm/glm.hpp>
```

These custom setup lines can then be placed in a common project header or pre-compiled header.

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 the core GLSL feature set with extensions. These extensions include: quaternion, transformation, spline, matrix inverse, color spaces, etc.
Note that some extensions are incompatible with other extension as and may result in C++ name collisions when used together.

GLM provides two methods to use these extensions.

3.2. Use example

This method simply requires the inclusion of the extension implementation filename. The extension features are added to the "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;
}
```

4. Dependencies

4.1. Internal dependencies

Including <glm/glm.hpp> provides all the GLSL features implemented in GLM for C++ programmers. This is a standalone header with no dependencies on other files.

By including <glm/ext.hpp> all the features of all extensions to GLM are included.

If you are only including the extensions needed, each extension will automatically included other extensions it depends on. (all extensions depend on and include GLM core <glm/glm.hpp>)

4.2. External dependencies

GLM 0.7.3 introduced dependency to external libraries with the extension GLM_VIRTREV_gl which depends on GLEW. To prevent the dependence of external libraries when extensions aren't even used, a new mechanism has 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:

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

This will include all extensions that depend on GLEW and BOOST and obviously every other extension with no external dependencies.

5. Debugging

5.1. Build message system

GLM 0.7.5 introduced a message system allowing for information to be generated at build time about the GLM configurations and GLM warnings/status. By default this system is disabled (#define GLM_MESSAGE GLM_MESSAGE_QUIET).

This is the list of options that can be enabled 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. Static assert

If the GLM_DEPENDENCE define has the flag GLM_DEPENDENCE_BOOST added (or <boost/static_assert.hpp> is included before GLM) then the setup system will automatically enable the use of static asserts throughout the GLM code to reduce the usage of bad templated functions or types.

6. Known issues

6.1. Swizzle operators

Enabling the swizzle operator can result in name collisions with the Win32 API. To prevent these issues, you can access to the internal swizzle operator functions without enabling the swizzle operator itself. This is done by defining:

```
#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 supports the half float number type through the extension GLM_GTC_half_float. This extension provides the types half, hvec*, hmat*x* and hquat*.

Unfortunately, C++ norm does not support anonymous unions to allow hvec* vector components to be accessed with only x, y, z and w.

However, Visual C++ does support anonymous unions. 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 in build errors.

7. References

OpenGL 3.2 core:

http://www.opengl.org/registry/doc/glspec32.core.20090803.pdf

GLSL 1.5:

http://www.opengl.org/registry/doc/GLSLangSpec.1.50.09.pdf

GLEW:

http://glew.sourceforge.net

Boost:

http://www.boost.org

8. Extensions list

The Doxygen generated documentation includes a complete list of all extensions available. Explore this documentation to get a complete overview of all GLM capabilities!