

# GLM 0.7 Manual

03/22/2008

## 1. Introduction

OpenGL Mathematics (GLM) is a mathematics library for 3D applications based on OpenGL Shading Language (GLSL). The project's goal is to provide to 3D programmer the same tools when he develops under GPU and CPU. This project wasn't limited by GLSL tools, 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 7.1 (2003).

Any feedback are welcome, please send then to [g.truc.creation\[NO\\_SPAM\\_THANKS\]gmail.com](mailto: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. Using extensions features

To take advantages of all extra features of GLM, you have to use the extensions which are included in "glmext.h": `#include <glm/glmext.h>`. Go to section 3 for more information about extensions

## 3. GLM Extensions

### 3.1. Setup

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>`. It is also possible to include extensions one by one.

### 3.2. About GLSL extension conventions

GLSL working group have defined some rules to define GLSL extensions. However, it seems that graphics card companies choose to adapt this rules according there own wishes which actually have some advantages thanks to the possibility to enable or disable GLSL extensions. So far, both conventions are supported by GLM, even if in the futur IHV conventions may overcome ARB conventions as those never been used for GLSL.

### 3.3. ARB conventions

By default, all extensions are available using the working group rules. To use an extension, you just have to include the according header file. To include the quaternion extension (GLM\_GTX\_quaternion), just include "glm/gtx/quaternion.h" You can also ask to use every extension by including glmext.h.

### 3.4. IHV conventions

To use the vendor way of using extensions, it's still required include header files but you also have to explicitly ask for it. Example to use GLM\_GTX\_quaternion with IHV conventions:

```
namespace glm
{
    using GLM_GTX_quaternion;
}
```

It's also possible to automatically setup every extensions defining GLM\_GTX\_INCLUDED before including "glm.h".

## 4. Use samples

### 4.1. GLM core features

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

```

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;
}

```

## 4.2. GLM extensions features through ARB conventions

```

#include <glm/glm.h>
#include <glm/glmext.h>

using namespace glm;

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

    mat4 Model = translateGTX(mat4(1.0), 1.0f, 1.0f, 1.0f);
    vec4 Transformed = Model * Position;

    return 0;
}

```

## 4.3. GLM extensions features through IHV conventions

```

#include <glm/glm.h>
#include <glm/glmext.h>

using namespace glm;
using GLM_GTX_transform;

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

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

    return 0;
}

```

---

```

#include <glm/glm.h>
#include <glm/glmext.h>

namespace glm
{
    using GLM_GTX_transform;
}

int foo()
{

```

```

glm::vec4 Position = glm::vec4(glm::vec3(0.0), 1.0);

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

return 0;
}

```

## 5. Known issues

### 5.1. Swizzle operators

Enabling swizzle operator can result to name collision with the Win32 API.

### 5.2. "not" function

The GLSL keyword "not" isn't well supported under Visual Studio because of Microsoft implementation of the "not" C++ Keyword... it's a #define.

### 5.3. "half" based types

Half based types can't be used with GCC because of multiple component names implementation based on union. To use half based types, define GLM\_SINGLE\_COMP\_NAME before including GLM headers. This disables color and texcoord component names.

## 6. Extensions list

GLM\_GTX\_associated\_min\_max:

Min and max functions that return associated values not the compared ones.

GLM\_GTX\_bit:

Allow to perform bit operations on integer values

GLM\_GTX\_closest\_point:

Find the point on a straight line which is the closet of a point.

GLM\_GTX\_color\_cast:

Conversion between two color types.

GLM\_GTX\_color\_space:

Related to RGB to HSV conversions and operations.

GLM\_GTX\_compatibility:

Provide functions to increase the compatibility with Cg and HLSL languages.

GLM\_GTX\_component\_wise:

Operations between components of a type.

GLM\_GTX\_determinant:

Compute the determinant of a matrix.

GLM\_GTX\_double\_float:

Add support for double precision floating-point types.

GLM\_GTX\_epsilon:

Comparison functions for a user defined epsilon values.

GLM\_GTX\_euler\_angles:

Build matrices from euler angles.

GLM\_GTX\_extend:

Extend a position from a source to a position at a defined length.

GLM\_GTX\_extented\_min\_max:

Min and max functions for 3 to 4 parameters.

GLM\_GTX\_fast\_exponential:

Fast but less accurate implementations of exponential based functions.

GLM\_GTX\_fast\_square\_root:

Fast but less accurate implementations of square root based functions.

GLM\_GTX\_fast\_trigonometry:

Fast but less accurate implementations of trigonometric functions.

GLM\_GTX\_flexible\_mix:

More flexible functions for linear interpolations.

GLM\_GTX\_gpu\_shader4:

Implementation of GL\_EXT\_gpu\_shader4 for GLM.

GLM\_GTX\_half:

Add support for half precision floating-point types.

GLM\_GTX\_handed\_coordinate\_space:

To know if a triadron is right or left handed.

GLM\_GTX\_hyperbolic:

Add hyperbolic functions.

GLM\_GTX\_inertia:

Create inertia matrices.

GLM\_GTX\_integer:

Add support for integer for core functions.

GLM\_GTX\_intersect:

Add intersection functions.

GLM\_GTX\_inverse:

Inverse matrix functions.

GLM\_GTX\_inverse\_transpose:  
Inverse transpose matrix functions.

GLM\_GTX\_mat4x3 (Deprecated):  
mat4x3 matrix type. Use core type instead.

GLM\_GTX\_matrix\_access:  
Set a column or a row of a matrix.

GLM\_GTX\_matrix\_cross\_product:  
Build cross product matrices.

GLM\_GTX\_matrix\_cross\_product:  
Build matrices with specific matrix order, row or column.

GLM\_GTX\_matrix\_projection:  
Various ways to build and operate on projection matrices

GLM\_GTX\_matrix\_query:  
Query to evaluate matrices properties

GLM\_GTX\_matrix\_selection:  
Access to matrix columns or rows

GLM\_GTX\_matx (Work in progress):  
NxN matrix types

GLM\_GTX\_mixed\_product:  
Mixed product of 3 vectors

GLM\_GTX\_mul:  
mul function for Cg and HLSL compatibility

GLM\_GTX\_norm:  
Various way to compute vector norms

GLM\_GTX\_normal:  
Compute the normal of a triangle

GLM\_GTX\_normalize\_dot:  
Dot product of vectors that need to be normalized with a single square root

GLM\_GTX\_number\_precision:  
Defined size types

GLM\_GTX\_optimum\_pow:  
Integer exponentiation of power functions

GLM\_GTX\_orthonormalize:  
Orthonormalize matrices

GLM\_GTX\_outer\_product:  
Product of x extended to a matrix with the y extended to a transposed matrix.  
(Deprecated: included in GLM core)

GLM\_GTX\_perpendicular:  
Perpendicular of a vector from other one

GLM\_GTX\_polar\_coordinates:  
Conversion from Euclidean space to polar space and revert

GLM\_GTX\_projection:  
Projection of a vector to other one

GLM\_GTX\_quaternion:  
Quaternion types and functions

GLM\_GTX\_random:  
Generate random number from various distribution methods

GLM\_GTX\_rotate\_vector:  
Function to directly rotate a vector

GLM\_GTX\_round:  
Computes the round value

GLM\_GTX\_spline:  
Spline functions

GLM\_GTX\_statistics\_operation (Work in progress):  
Statistics functions

GLM\_GTX\_transform:  
Add transformation matrices

GLM\_GTX\_transform2:  
Add extra transformation matrices

GLM\_GTX\_transpose (Deprecated):  
Use GLM core function instead

GLM\_GTX\_unsigned\_int:  
Add support for unsigned integer for core functions

GLM\_GTX\_vector\_access:  
Function to set values to vectors

GLM\_GTX\_vector\_angle:

Compute angle between vectors

GLM\_GTX\_vector\_comp\_mult (Deprecated):

Revert matrix and vector product instead with core operators

GLM\_GTX\_vector\_query:

Query informations of vector types

GLM\_GTX\_vecx (Work in progress):

Add custom size vectors

GLM\_GTX\_verbose\_operator:

Use words to replace operators