The OpenGL Mathematics (GLM) Library

Mike Bailey
mjb@cs.oregonstate.edu

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License

---

What is GLM?
GLM is a set of C++ classes and functions to fill in the programming gaps in writing the basic vector and matrix mathematics for OpenGL applications. GLM isn’t really a library – it is all specified in *.hpp header files so that it gets compiled with your source code.

You can find it at:
http://glm.g-truc.net/0.9.8.5/

You invoke GLM like this:
```cpp
#define GLM_FORCE_RADIANS
#include <glm/glm.hpp>
#include  <glm/gtc/matrix_transform.hpp>
```

Or, you can #include only the specific GLM .hpp files you need.

If GLM is not installed in a system place, put it somewhere you can get access to. Later on, these notes will show you how to use it from there.

---

Why are we even talking about this?
The OpenGL overlords have “deprecated” some of the OpenGL functions we have been using to perform transformations. In the desktop world, it means that the use of such functions is discouraged. In the tablet and mobile world of OpenGL, it means those functions are gone. You might as well become familiar with how to live without them. So, instead of saying:
```
    gluLookAt( 0., 0., 3.,     0., 0., 0.,     0., 1., 0. );
    glRotatef( (GLfloat)Yrot, 0., 1., 0. );
    glRotatef( (GLfloat)Xrot, 1., 0., 0. );
    glScalef( (GLfloat)Scale, (GLfloat)Scale, (GLfloat)Scale );
```
for OpenGL,

```
    glm::mat4 modelview;
    glm::vec3 eye(0.,0.,3.);
    glm::vec3 look(0.,0.,0.);
    glm::vec3 up(0.,1.,0.);
    modelview = glm::lookAt( eye, look, up );
    modelview = glm::rotate( modelview, D2R*Yrot, glm::vec3(0.,1.,0.) );
    modelview = glm::rotate( modelview, D2R*Xrot, glm::vec3(1.,0,0.) );
    modelview = glm::scale( modelview, glm::vec3(Scale,Scale,Scale) );
    glMultMatrixf(   glm::value_ptr( modelview )   );
```

Exactly the same concept, but a different expression of it. Read on for details …

---

The Most Useful GLM Variables, Operations, and Functions

GLM recommends that you use the “glm:” syntax and not use “using namespace” syntax because they have not made any effort to create unique function names.

If constructor:
```
glm::mat4( 1. ); // identity matrix
glm::vec4( );
glm::vec3( );
```

If multiplications – the * operator has been overloaded:
```
glm::mat4
glm::rotate
( glm::mat4 const & m, float angle, glm::vec3 const & axis );
glm::mat4
glm::scale
( glm::mat4 const & m, glm::vec3 const & factors );
glm::mat4
glm::translate
( glm::mat4 const & m, glm::vec3 const & translation );
```

If viewing volume (assign, not concatenate):
```
glm::mat4 glm::ortho( float left, float right, float bottom, float top );
glm::mat4 glm::ortho( float left, float right, float bottom, float top );
glm::mat4 glm::perspective( float fovy, float aspect, float near, float far );
```

If loading matrices into OpenGL:
```
glLoadMatrix( glm::value_ptr( glm::mat4 )
);
glUniformMatrix4fv( Location, 1, GL_FALSE, glm::value_ptr( glm::mat4 )
);
```

---

Installing GLM into your own space
I like to just put the whole thing under my Visual Studio project folder so I can zip up a complete project and give it to someone else.
Here’s what that GLM folder looks like

Telling Linux about where the GLM folder is

```
g++ ...
```

“minus-capital-eye-period” means “also look for the < > includes in this folder”

Instead of the period, you can list a full or relative pathname.

Telling Visual Studio about where the GLM folder is

1.

2.

3.

4.

5.

Using Transformations, OpenGL-style, like in the sample.cpp Program

```
gMatrixMode(GL_PROJECTION);
gLoadIdentity();

if( WhichProjection == ORTHO )
gOrtho( -3., 3., -3., 3., 0.1, 1000. );
else
  gluPerspective( 90., 1., 0.1, 1000. );

// place the objects into the scene:
gMatrixMode(GL_MODELVIEW);
gLoadIdentity();

// set the eye position, look-at position, and up-vector:
gluLookAt( 0., 0., 3., 0., 0., 0., 0., 1., 0. );

// rotate the scene:
gRotatef( (GLfloat)Yrot, 0., 1., 0. );
gRotatef( (GLfloat)Xrot, 1., 0., 0. );

// uniformly scale the scene:
if( Scale < MINSCALE )
  Scale = MINSCALE;
gScalef( (GLfloat)Scale, (GLfloat)Scale, (GLfloat)Scale );
```

Using Transformations, GLM-style, I

```
#include <glm/vec3.hpp>
#include <glm/mat4x4.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>

const float D2R = M_PI/180.f; // 0.01745…

// convert degrees to radians:

// apply the projection matrix:
gMat[4][4] = glm::value_ptr( projection );
```

```
Using Transformations, GLM-style, II

// place the objects into the scene:
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();

// set the eye position, look-at position, and up-vector:
float eye[] = {0.,0.,3.};
float look[] = {0.,0.,0.};
float up[] = {0.,1.,0.};

glm::mat4 modelview = glm::lookAt(glm::vec3(eye[0], eye[1], eye[2]), glm::vec3(look[0], look[1], look[2]), glm::vec3(up[0], up[1], up[2]));

// rotate the scene (warning -- unlike OpenGL's glRotatef, GLM's rotate method takes angles in radians):
modelview = glm::rotate(modelview, Yrot, glm::vec3(0.,1.,0.));
modelview = glm::rotate(modelview, Xrot, glm::vec3(1.,0.,0.));

// uniformly scale the scene:
if (Scale < MINSCALE)
    Scale = MINSCALE;
modelview = glm::scale(modelview, glm::vec3(Scale, Scale, Scale));

// apply the modelview matrix:
Pattern->SetUniformVariable("modelMatrix", modelview);

In the shader:

uniform mat4 modelMatrix;
uniform mat4 viewMatrix;
uniform mat4 projectionMatrix;

mat4 PVM = projectionMatrix * viewMatrix * modelMatrix;

gl_Position = PVM * gl_Vertex;

In the C/C++ program:

glm::mat4 projection = glm::perspective(D2R*90., 1., 0.1, 1000.);
projection[1][1] *= -1.; // Vulkan's projected Y is inverted from OpenGL's

glm::mat4 model = glm::rotate(model, Yrot, glm::vec3(0.,1.,0.));
model = glm::rotate(model, Xrot, glm::vec3(1.,0.,0.));