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

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 Vulkan and the mobile world of OpenGL-ES, it means those functions are gone. You might as well become familiar with how to live without them. So, instead of saying:

```c
    glLoadIdentity( 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, you would now say:

```c
    glm::mat4 modelview;
    glm::vec3 eye(0.,0.,3.);
    glm::vec3 look(0.,0.,0.);
    glm::vec3 up(0.,1.,0.);
    glm::mat4 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) );
    glm::mat4 modelview; // identity matrix
    glm::vec4 view = glm::value_ptr( modelview );
```

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.

// multiplications – the * operator has been overloaded:

```c
    glm::mat4 * glm::mat4
    glm::mat4 * glm::vec4
    glm::mat4 * glm::vec3( glm::vec3, 1. ) // promote vec3 to a vec4 via a constructor
```

// emulating OpenGL transformations with concatenation:

```c
    glm::mat4 glm::translate( glm::mat4 const & m, glm::vec3 const & translation );
    glm::mat4 glm::scale( glm::mat4 const & m, glm::vec3 const & factors );
    glm::mat4 glm::rotate( glm::mat4 const & m, glm::vec3 const & axis );
```

Exactly the same concept, but a different expression of it. Read on for details...
The Most Useful GLM Variables, Operations, and Functions

// viewing volume (assign, not concatenate):
glm::mat4 glm::ortho( float left, float right, float bottom, float top, float near, float far );

// viewing (assign, not concatenate):
glm::mat4 glm::lookAt( glm::vec3 const & eye, glm::vec3 const & look, glm::vec3 const & up );

// loading matrices into opengl:
glLoadMatrix( glm::value_ptr( glm::mat4 ) );

glm::mat4 glm::frustum( float left, float right, float bottom, float top, float near, float far );

glm::mat4 glm::perspective( float fovy, float aspect, float near, float far );

// loading matrices into opengl:
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.

Telling Linux about where the GLM folder is

"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.

Here's what that GLM folder looks like
Telling Visual Studio about where the GLM folder is

1. Add the GLM folder to your project's include path.
2. Update your project's build settings to include the GLM library.

A period, indicating that the project folder should also be searched when a #include <xxx> is encountered. If you put it somewhere else, enter that full or relative path instead.

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

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

#include <GL/glut.h>
#include <GL/glu.h>

// Convert degrees to radians:
const float D2R = M_PI/180.f; // 0.01745..."
// place the objects into the scene:
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();

// set the eye position, look-at position, and up-vector:
glm::vec3 eye(0.,0.,3.);
glm::vec3 look(0.,0.,0.);
glm::vec3 up(0.,1.,0.);
glm::mat4 modelview = glm::lookAt(eye, look, up);

// rotate the scene (warning -- unlike OpenGL's glRotatef, 
//        GLM's rotate method takes angles in *radians*):
modelview = glm::rotate(modelview, D2R*Yrot, glm::vec3(0.,1.,0.));
modelview = glm::rotate(modelview, D2R*Xrot, glm::vec3(1.,0.,0.));

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

// apply the modelview matrix:
glu MultMatrixf(glm::value_ptr(modelview));

// GLM for Vulkan

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::vec3 eye(0.,0.,3.);
glm::vec3 look(0.,0.,0.);
glm::vec3 up(0.,1.,0.);
glm::mat4 view = glm::lookAt(eye, look, up);

glm::mat4 model(1.); // identity
model = glm::rotate(model, D2R*Yrot, glm::vec3(0.,1.,0.));
model = glm::rotate(model, D2R*Xrot, glm::vec3(1.,0.,0.));

in the shader:
uniform mat4 projectionMatrix;
uniform mat4 viewMatrix;
uniform mat4 modelMatrix;
mat4 PVM = projectionMatrix * viewMatrix * modelMatrix;
gl_Position = PVM * gl_Vertex;

in the C++ program:
glm::mat4 projection = glm::perspective(D2R*90., 1., 0.1, 1000.);
glm::vec3 eye(0.,0.,3.);
glm::vec3 look(0.,0.,0.);
glm::vec3 up(0.,1.,0.);
glm::mat4 view = glm::lookAt(eye, look, up);

glm::mat4 model(1.); // identity
model = glm::rotate(model, D2R*Yrot, glm::vec3(0.,1.,0.));
model = glm::rotate(model, D2R*Xrot, glm::vec3(1.,0.,0.));

Pattern.Use();
Pattern.SetUniformVariable(“projectionMatrix”, projection);
Pattern.SetUniformVariable(“viewMatrix”, view);
Pattern.SetUniformVariable(“modelMatrix”, model);