The OpenGL Mathematics (GLM) Library

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
void gluLookAt( 0., 0., 3.,     0., 0., 0.,     0., 1., 0. );
gluRotatef( (GLfloat)Yrot, 0., 1., 0. );
gluRotatef( (GLfloat)Xrot, 1., 0., 0. );
gluScalef( (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.);
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) );
gluMultMatrixf( glm::value_ptr( modelview ) );
```

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.
### The Most Useful GLM Variables, Operations, and Functions

#### Viewing volume (assign, not concatenate):

```cpp
glm::mat4 glm::ortho(float left, float right, float bottom, float top, float near, float far);
```

```cpp
glm::mat4 glm::ortho(float left, float right, float bottom, float top);
```

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

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

#### Viewing (assign, not concatenate):

```cpp
glm::mat4 glm::lookAt(glm::vec3 const & eye, glm::vec3 const & look, glm::vec3 const & up);
```

#### Loading matrices into OpenGL:

```cpp
glLoadMatrixf(glm::value_ptr(glm::mat4));
```

```cpp
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

```
g++ ... -I ...
```

"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. #include <glm/glm.hpp>
2. #include <glm/gtc/matrix_transform.hpp>
3. #include <glm/gtc/type_ptr.hpp>

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
#include <GL_PROJECTION>
#include <GL_MODELVIEW>
#include <GL_MODELVIEW>
#include <GL_MODELVIEW>

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

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

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

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

Using Transformations, GLM-style, I

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

// convert degrees to radians:
const float D2R = M_PI/180.f; // 0.01745…

// apply the projection matrix:
 glm::value_ptr( projection )
```

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

// convert degrees to radians:
const float D2R = M_PI/180.f; // 0.01745…

// apply the projection matrix:
 glm::value_ptr( projection )
```
// place the objects into the scene:
glMatrixMode(GL_MODELVIEW);
gLoadIdentity();

// 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:
glMultMatrixf(glm::value_ptr(modelview));

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

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

mat4 PVM = projectionMatrix * viewMatrix * modelMatrix;
gl_Position = PVM * gl_Vertex;

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 = 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);

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 = 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);

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 = 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);

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 = 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);