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 a library — it is all specified in *.hpp header files so that it gets compiled in 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.

You can find it at:

http://glm.g-truc.net/0.9.8.5/
Here’s what that GLM folder looks like

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. A period, indicating that the project folder should also be searched when an `#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

```c
#include <GL/glut.h>
#include <GL/glu.h>
// OpenGL

#include <GL/glew.h>
#include <GL/gl.h>

// GLM

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

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

Using Transformations, GLM-style, I

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

glMatrixMode( GL_PROJECTION );
gLoadIdentity( );
if( WhichProjection == ORTHO )
  projection = glm::ortho( -3., 3.,     -3., 3.,     0.1, 1000. );
else
  projection = glm::perspective( D2R*90., 1., 0.1, 1000. );
// apply the projection matrix:
glMultMatrixf( glm::value_ptr( projection ) );
```
Using Transformations, GLM-style, II

If the objects into the scene:

```c
glMatrixMode(GL_MODELVIEW);
```

If set the eye position, look-at position, and up-vector:

```c
glm::vec3 eye(0., 0., 3.);
glm::vec3 look(0., 0., 0.);
glm::vec3 up(0., 1., 0.);
```

If rotate the scene (warning -- unlike OpenGL’s `glRotatef`,
GLM’s rotate method takes angles in *radians*):

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

If uniformly scale the scene:

```c
if (Scale < MINSCALE)
    Scale = MINSCALE;
```

```c
modelview = glm::scale(modelview, glm::vec3(Scale, Scale, Scale));
```

If apply the modelview matrix:

```c
Pattern->Use();
Pattern->SetUniformVariable("projectionMatrix", projection);
Pattern->SetUniformVariable("viewMatrix", view);
Pattern->SetUniformVariable("modelMatrix", model);
```

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

```c
mat4 PVM = projectionMatrix * viewMatrix * modelMatrix; // must be in this order
```

```
gl_Position = PVM * gl_Vertex; // must be in this order
```

Passing GLM Matrices into a Vertex Shader

In the shader:

```c
uniform mat4 projectionMatrix;
uniform mat4 viewMatrix;
uniform mat4 modelMatrix;
```

```c
mat4 PVM = projectionMatrix * viewMatrix * modelMatrix; // must be in this order
```

```c
gl_Position = PVM * gl_Vertex; // must be in this order
```

In the C/C++ program:

```c
glm::mat4 projection = glm::perspective(D2R*90., 1., 0.1, 1000.);
```

```c
projection[1][1] *= -1.; // Vulkan’s projected Y is inverted from OpenGL’s
```

```c
glm::vec3 eye(0., 0., 3.);
glm::vec3 look(0., 0., 0.);
glm::vec3 up(0., 1., 0.);
```

```c
glm::mat4 view = glm::lookAt(eye, look, up);
```

```c
glm::mat4 model = glm::identity(); // identity
```

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

GLM for Vulkan

```c
glm::mat4 projection = glm::perspective(D2R*90., 1., 0.1, 1000.);
```

```c
projection[1][1] *= -1.; // Vulkan’s projected Y is inverted from OpenGL’s
```

```c
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);
```

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