



GLM



Oregon State University
Mike Bailey
mbj@cs.oregonstate.edu




GLM.ods
mb - December 21, 2022

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. However, even though it was written for OpenGL, it works fine with Vulkan.

Even though GLM looks like a library, it actually isn't – 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:

```
#define GLM_FORCE_RADIANS
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/matrix_inverse.hpp>
```

OpenGL treats all angles as given in *degrees*. This line forces GLM to treat all angles as given in *radians*. I recommend this so that *all* angles you create in *all* programming will be in radians.

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.



mb - December 21, 2022

Why are we even talking about this?


All of the things that we have talked about being *deprecated* in OpenGL are *really deprecated* in Vulkan – built-in pipeline transformations, begin-end, fixed-function, etc. So, where you might have said in OpenGL:

```
glMatrixMode( GL_MODELVIEW );
glLoadIdentity();
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 );
```

you would now say:

```
glm::mat4 modelview = glm::mat4( 1. ); // identity
glm::vec3 eye(0.,0.,3.);
glm::vec3 look(0.,0.,0.);
glm::vec3 up(0.,1.,0.);
modelview = glm::lookAt( eye, look, up ); // {x,y,z} = [v]*{x,y,z}
modelview = glm::rotate( modelview, D2R*Yrot, glm::vec3(0.,1.,0.) ); // {x,y,z} = [v]*[r]*{x,y,z}
modelview = glm::rotate( modelview, D2R*Xrot, glm::vec3(1.,0.,0.) ); // {x,y,z} = [v]*[r]*[x]*{x,y,z}
modelview = glm::scale( modelview, glm::vec3(Scale,Scale,Scale) ); // {x,y,z} = [v]*[r]*[s]*{x,y,z}
```

This is exactly the same concept as OpenGL, but a different expression of it. Read on for details ...



mb - December 21, 2022


The Most Useful GLM Variables, Operations, and Functions

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

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

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

```
// emulating OpenGL transformations with concatenation:
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 );
```




mb - December 21, 2022

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

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 );
```

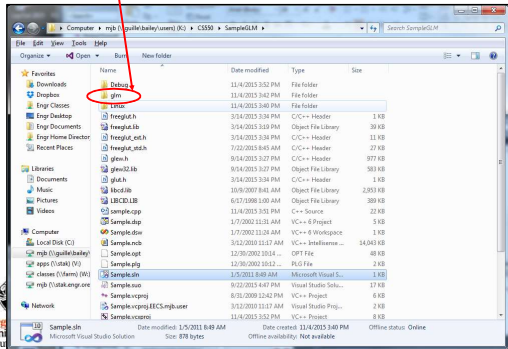

```
// viewing (assign, not concatenate):
glm::mat4 glm::lookAt( glm::vec3 const & eye, glm::vec3 const & look, glm::vec3 const & up );
```



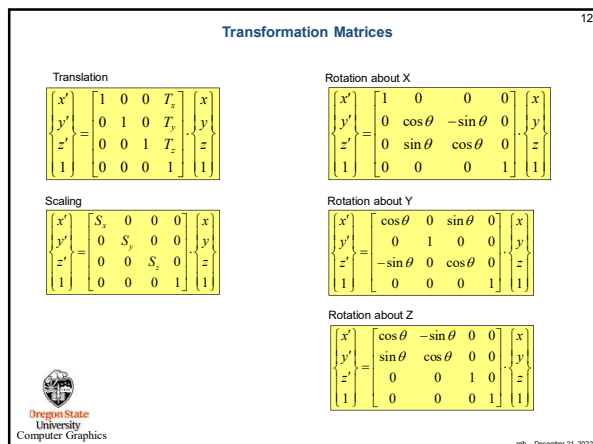
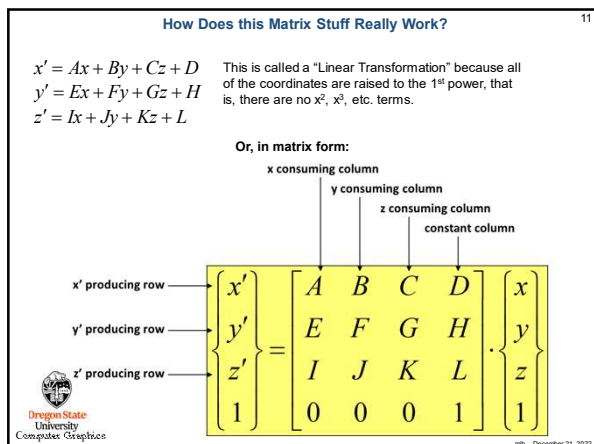
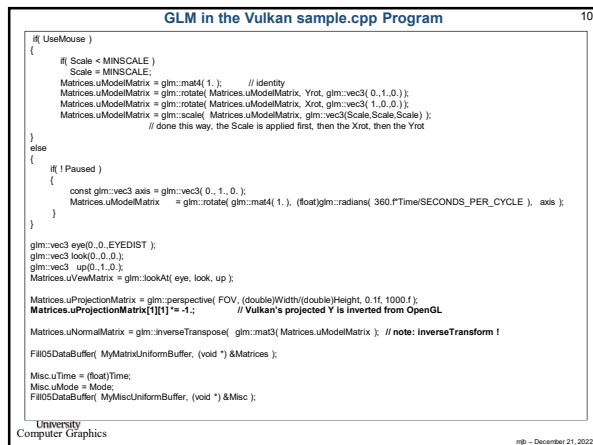
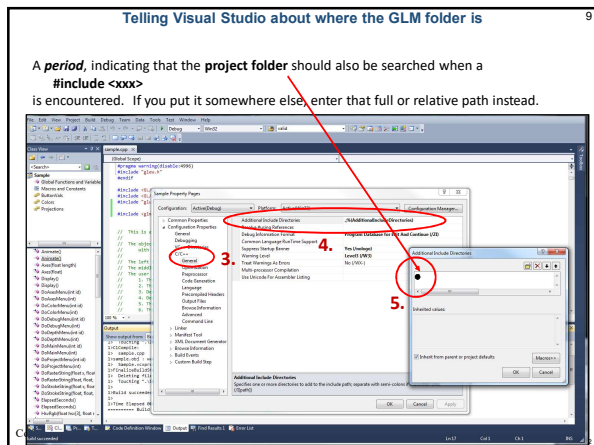
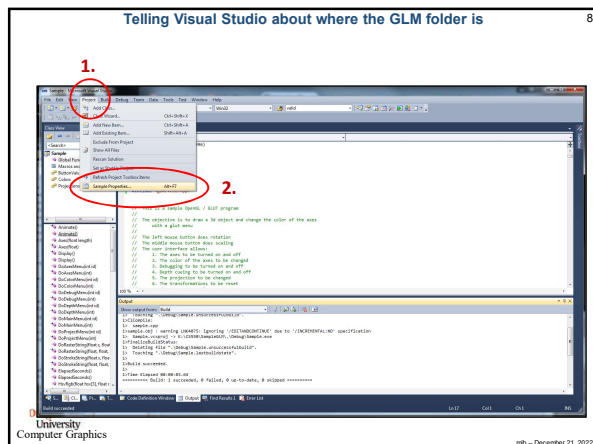
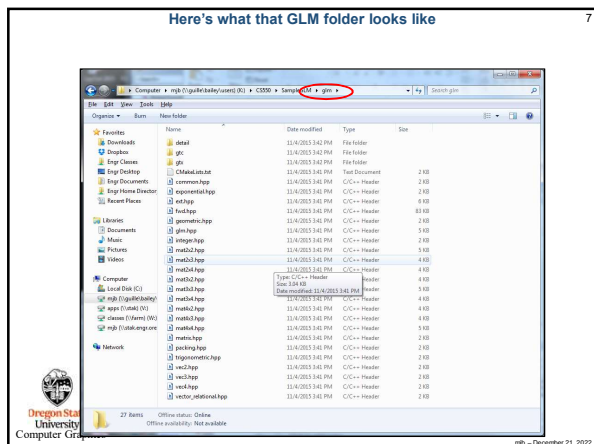
mb - December 21, 2022

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.


mb - December 21, 2022



How it Really Works :-)

$$\begin{bmatrix} \cos 90^\circ & \sin 90^\circ \\ -\sin 90^\circ & \cos 90^\circ \end{bmatrix} \begin{bmatrix} A_1 \\ A_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

<http://xkcd.com>




13

The Rotation Matrix for an Angle (θ) about an Arbitrary Axis (A_x, A_y, A_z)

$$[M] = \begin{bmatrix} A_x A_x + \cos \theta (1 - A_x A_x) & A_x A_y - \cos \theta (A_x A_y) - \sin \theta A_z & A_x A_z - \cos \theta (A_x A_z) + \sin \theta A_y \\ A_x A_y - \cos \theta (A_x A_y) + \sin \theta A_z & A_y A_y + \cos \theta (1 - A_y A_y) & A_y A_z - \cos \theta (A_y A_z) - \sin \theta A_x \\ A_x A_z - \cos \theta (A_x A_z) - \sin \theta A_y & A_y A_z - \cos \theta (A_y A_z) + \sin \theta A_x & A_z A_z + \cos \theta (1 - A_z A_z) \end{bmatrix}$$

For this to be correct, A must be a unit vector



14

Compound Transformations


Q: Our rotation matrices only work around the origin? What if we want to rotate about an arbitrary point (A,B)?

A: We create more than one matrix.

Write it

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 3 \\ [T_{+A,+B}] \cdot [R_\theta] \cdot [T_{-A,-B}] \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Say it

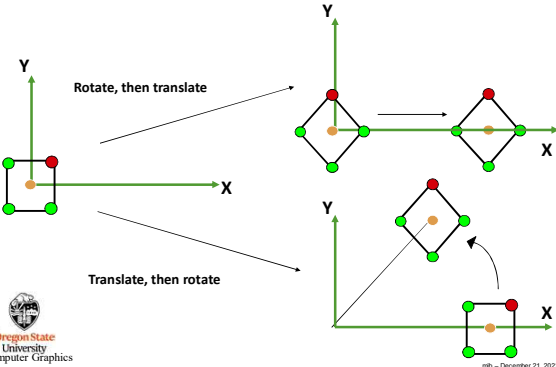



15

Matrix Multiplication is *not* Commutative

Rotate, then translate

Translate, then rotate


16

Matrix Multiplication is Associative

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} [T_{+A,+B}] \cdot [R_\theta] \cdot [T_{-A,-B}] \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} [T_{+A,+B}] \cdot [R_\theta] \cdot [T_{-A,-B}] \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

One matrix to rule them all – the Current Transformation Matrix, or CTM



17

From the Data Buffer Noteset

Here's the vertex shader code to use the matrices:


```

layout( std140, set = 0, binding = 0 ) uniform sceneMatBuf
{
    mat4 uProjectionMatrix;
    mat4 uViewMatrix;
    mat4 uSceneMatrix;
} SceneMatrices;

layout( std140, set = 1, binding = 0 ) uniform objectMatBuf
{
    mat4 uModelMatrix;
    mat4 uNormalMatrix;
} ObjectMatrices;

vNormal = uNormalMatrix * aNormal;
gl_Position = uProjectMatrix * uViewMatrix * uSceneMatrix * uModelMatrix * aVertex;
    
```

"CTM"



18

Why Isn't The Normal Matrix exactly the same as the Model Matrix? 19

```

glm::mat4 Model = uViewMatrix*uSceneMatrix*uModelMatrix;
uNormalMatrix = glm::inverseTranspose( glm::mat3(Model) );
    
```

It is, if the Model Matrices are all rotations and uniform scalings, but if it has non-uniform scalings, then it is not. These diagrams show you why.

Original object and normal:

Wrong!

$uNormalMatrix = glm::mat3(Model);$

Right!

$uNormalMatrix = glm::inverseTranspose(glm::mat3(Model));$

Dragoon State University
Computer Graphics

19 - December 21, 2022