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/

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.

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:

```c
    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 );
    modelview = glm::rotate( modelview, (GLfloat)Yrot, glm::vec3(0.,1.,0.) );
    modelview = glm::rotate( modelview, (GLfloat)Xrot, glm::vec3(1.,0.,0.) );
    modelview = glm::scale( modelview, glm::vec3(Scale,Scale,Scale) );
```

This is 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 avoid “using namespace” syntax because they have not made any effort to create unique function names.

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

    // multiplications:
    glm::mat4
    * 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 );
```
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 );

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 Visual Studio about where the GLM folder is

1.
2.
Telling Visual Studio about where the GLM folder is

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.

4. Telling Visual Studio about where the GLM folder is

5. GLM in the Vulkan sample.cpp Program

How Does this Matrix Stuff Really Work?

\[
\begin{bmatrix}
 x' \\
 y' \\
 z'
\end{bmatrix} = \begin{bmatrix}
 A & B & C & D \\
 E & F & G & H \\
 I & J & K & L \\
 0 & 0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
 x \\
 y \\
 z \\
 1
\end{bmatrix}
\]

This is called a “Linear Transformation” because all of the coordinates are raised to the 1st power, that is, there are no \( x^2, x^3, \) etc. terms.

Or, in matrix form:

\[
\begin{align*}
x' &= Ax + By + Cz + D \\
y' &= Ex + Fy + Gz + H \\
z' &= Ix + Jy + Kz + L
\end{align*}
\]

Transformation Matrices

Translation

\[
\begin{bmatrix}
 1 & 0 & 0 & 0 \\
 0 & 1 & 0 & 0 \\
 0 & 0 & 1 & 0 \\
 0 & 0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
 x \\
 y \\
 z \\
 1
\end{bmatrix}
\]

Rotation about X

\[
\begin{bmatrix}
 1 & 0 & 0 & 0 \\
 0 & \cos \theta - \sin \theta & 0 & 0 \\
 0 & \sin \theta & \cos \theta & 0 \\
 0 & 0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
 x \\
 y \\
 z \\
 1
\end{bmatrix}
\]

Rotation about Y

\[
\begin{bmatrix}
 \cos \theta & 0 & \sin \theta & 0 \\
 0 & 1 & 0 & 0 \\
 -\sin \theta & 0 & \cos \theta & 0 \\
 0 & 0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
 x \\
 y \\
 z \\
 1
\end{bmatrix}
\]

Rotation about Z

\[
\begin{bmatrix}
 \cos \theta & -\sin \theta & 0 & 0 \\
 \sin \theta & \cos \theta & 0 & 0 \\
 0 & 0 & 1 & 0 \\
 0 & 0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
 x \\
 y \\
 z \\
 1
\end{bmatrix}
\]
How it Really Works :-)  

\[
\begin{bmatrix}
\cos 90^\circ & \sin 90^\circ \\
-sin 90^\circ & \cos 90^\circ
\end{bmatrix}
\begin{bmatrix}
\alpha_1 \\
\alpha_2
\end{bmatrix}
= \begin{bmatrix}
0 \\
0
\end{bmatrix}
\]

http://xkcd.com

The Rotation Matrix for an Angle (θ) about an Arbitrary Axis (Ax, Ay, Az)  

\[
M = \begin{bmatrix}
A_A - \cos \theta (1-A_A) & A_A - \cos \theta (A_A) - \sin \theta A & A_A - \cos \theta (A_A) + \sin \theta A \\
A_A - \cos \theta (A_A) + \sin \theta A & A_A + \cos \theta (1-A_A) & A_A + \cos \theta (A_A) - \sin \theta A \\
A_A - \cos \theta (A_A) - \sin \theta A & A_A - \cos \theta (A_A) + \sin \theta A & A_A + \cos \theta (1-A_A)
\end{bmatrix}
\]

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

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

\[
\begin{bmatrix}
x' \\
y' \\
z'
\end{bmatrix}
= \begin{bmatrix}
T_{A+B} & \\
R_\theta & T_{A-B}
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z
\end{bmatrix}
\]

Say it

Matrix Multiplication is not Commutative

Rotate, then translate

Translate, then rotate
Matrix Multiplication is Associative

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

One matrix – the Current Transformation Matrix, or CTM

\[
\begin{pmatrix}
 x' \\
 y' \\
 z' \\
 1
\end{pmatrix} = \begin{pmatrix}
 R_x \\
 R_y \\
 R_z
\end{pmatrix} \begin{pmatrix}
 T_{x,A+B} \\
 T_{y,A-B} \\
 T_{z,A-B}
\end{pmatrix} \begin{pmatrix}
 x \\
 y \\
 z \\
 1
\end{pmatrix}
\]

One Matrix to Rule Them All

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

glm::mat4 Model = glm::mat4(1.);
Model = glm::translate(Model, glm::vec3(A, B, 0.));
Model = glm::rotate(Model, thetaRadians, glm::vec3(Ax, Ay, Az));
Model = glm::translate(Model, glm::vec3(-A, -B, 0.));

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

glm::mat4 Projection = glm::perspective(FOV, (double)Width/(double)Height, 0.1, 1000.);
Projection[1][1] *= -1.;

 glm::mat3 Matrix = Projection * View * Model;
 glm::mat3 NormalMatrix = glm::inverseTranspose(glm::mat3(Model));

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

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

Wrong!

Right!

glm::mat3 NormalMatrix = glm::mat3(Model);

glm::mat3 NormalMatrix = glm::inverseTranspose(glm::mat3(Model));