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 (with one small exception which can be worked around).

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:

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

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.

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
    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 have to 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, glm::vec3(0.,1.,0.) );
    modelview = glm::rotate( modelview, glm::vec3(1.,0.,0.) );
    modelview = glm::scale( modelview, glm::vec3(Scale,Scale,Scale) );
```

Exactly the same concept, but a different expression of it. Read on for details …

The Most Useful GLM Variables, Operations, and Functions

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

// multiplications:
```
    glm::mat4    * glm::mat4
    glm::mat4    * glm::vec4
    glm::mat4    * glm::vec4( glm::vec3 ) // promote 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 );
```

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

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.

GLM in the Vulkan sample.cpp Program

Telling Visual Studio about where the GLM folder is

Your Sample2017.zip File Contains GLM Already

How Does this Matrix Stuff Really Work?

$x' = Ax + By + Cz + D$

$y' = Ex + Fy + Gz + H$

$z' = Ix + Jy + Kz + L$

This is called a "Linear Transformation" because all of the coordinates are raised to the 1st power, that is, there are no $x^2, y^2, etc., terms.

Or, in matrix form:

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

Translation

\[
\begin{bmatrix}
1 & 0 & 0 & T_x \\
0 & 1 & 0 & T_y \\
0 & 0 & 1 & T_z \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

Rotation about X

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

Scaling

\[
\begin{bmatrix}
S_x & 0 & 0 & 0 \\
0 & S_y & 0 & 0 \\
0 & 0 & S_z & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

Rotation about Y

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

Rotation about Z

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

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

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

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.

Matrix Multiplication is not Commutative

Rotate, then translate

Translate, then rotate

Matrix Multiplication is Associative

One matrix – the Current Transformation Matrix, or CTM
### One Matrix to Rule Them All

\[
\begin{pmatrix}
  x' \\
  y' \\
  z'
\end{pmatrix} = \begin{bmatrix}
  T_{x,y,z} \\
  R_{x,y,z} \\
  N\end{bmatrix} \begin{bmatrix}
  x \\
  y \\
  1
\end{bmatrix}
\]

- \( T_{x,y,z} = \text{translation matrix} \)
- \( R_{x,y,z} = \text{rotation matrix} \)
- \( N = \text{normal matrix} \)

### Why Isn't The Normal Matrix just 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.

Original object and normal

Wrong!

Right!

```cpp
glm::mat3 Model = glm::translate(Model, glm::vec3(-A, -B, 0.));
glm::mat3 Model = glm::rotate(Model, thetaRadians, glm::vec3(Ax, Ay, Az));
glm::mat3 Model = glm::translate(Model, glm::vec3(A, B, 0.));
```

```cpp
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.;
```

```cpp
glm::mat3 NormalMatrix = glm::inverseTransposed(glm::mat3(Model));
```

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