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:

```cpp
#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:

```cpp
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:

```cpp
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) );
```

```cpp
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) );
```

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

The Most Useful GLM Variables, Operations, and Functions

```cpp
// constructor:
glm::mat4(); // identity matrix
glm::vec4();

// multiplications:
glm::mat4 glm::mat4
glm::mat4 glm::vec4( glm::vec3, 1.0 ); // 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 );
```
The Most Useful GLM Variables, Operations, and Functions

// viewing volume (assign, not concatenate):

void glm::ortho( float left, float right, float bottom, float top, float near, float far );
void glm::ortho( float left, float right, float bottom, float top );

void glm::frustum( float left, float right, float bottom, float top, float near, float far );

void glm::perspective( float fovy, float aspect, float near, float far );

// viewing (assign, not concatenate):

void 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. 

These steps will allow you to use the GLM library in your Visual Studio project.
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

```cpp
if( UseMouse )
{
  if( Scale < MINSCALE )
    Scale = MINSCALE;

  Matrices.uModelMatrix = glm::mat4(); // identity
  Matrices.uModelMatrix = glm::scale( Matrices.uModelMatrix, glm::vec3(Scale, Scale, Scale) );
  Matrices.uModelMatrix = glm::rotate( Matrices.uModelMatrix, Yrot, glm::vec3(0., 1., 0.) );
  Matrices.uModelMatrix = glm::rotate( Matrices.uModelMatrix, Xrot, glm::vec3(1., 0., 0.) );
}
else
{
  if( !Paused )
  {
    const glm::vec3 axis = glm::vec3(0., 1., 0.);
    Matrices.uModelMatrix = glm::rotate( glm::mat4(), (float)glm::radians(360.f*Time/SECONDS_PER_CYCLE), axis );
  }
}
Matrices.uProjectionMatrix = glm::perspective(FOV, (double)Width/(double)Height, 0.1, 1000.);
Matrices.uProjectionMatrix[1][1] *= -1.; // Vulkan's projected Y is inverted from OpenGL
Matrices.uNormalMatrix = glm::inverseTranspose(glm::mat3(Matrices.uModelMatrix));
Matrices.uNormalMatrix = glm::inverseTranspose(glm::mat3(Matrices.uModelMatrix));
}
Misc.uTime = (float)Time;
Misc.uMode = Mode; Fill05DataBuffer(MyMiscUniformBuffer, (void *)&Misc);
```

### Your Sample2017.zip File Contains GLM Already

- GLM

### How Does this Matrix Stuff Really Work?

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:

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

Or, in matrix form:

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

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

#### Translation
\[
\begin{bmatrix}
    x' \\
    y' \\
    z'
\end{bmatrix} =
\begin{bmatrix}
    1 & 0 & 0 & T_x \\
    0 & 1 & 0 & T_y \\
    0 & 0 & 1 & T_z \\
    1 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
    x \\
    y \\
    z \\
    1
\end{bmatrix}
\]

#### Rotation about X
\[
\begin{bmatrix}
    x' \\
    y' \\
    z'
\end{bmatrix} =
\begin{bmatrix}
    1 & 0 & 0 & 0 \\
    0 & \cos \theta & -\sin \theta & 0 \\
    0 & \sin \theta & \cos \theta & 0 \\
    0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
    x \\
    y \\
    z \\
    1
\end{bmatrix}
\]

#### Scaling
\[
\begin{bmatrix}
    x' \\
    y' \\
    z'
\end{bmatrix} =
\begin{bmatrix}
    s_x & 0 & 0 & 0 \\
    0 & s_y & 0 & 0 \\
    0 & 0 & s_z & 0 \\
    0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
    x \\
    y \\
    z \\
    1
\end{bmatrix}
\]

#### Rotation about Y
\[
\begin{bmatrix}
    x' \\
    y' \\
    z'
\end{bmatrix} =
\begin{bmatrix}
    \cos \theta & 0 & \sin \theta & 0 \\
    0 & 1 & 0 & 0 \\
    -\sin \theta & 0 & \cos \theta & 0 \\
    0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
    x \\
    y \\
    z \\
    1
\end{bmatrix}
\]

#### Rotation about Z
\[
\begin{bmatrix}
    x' \\
    y' \\
    z'
\end{bmatrix} =
\begin{bmatrix}
    \cos \theta & -\sin \theta & 0 & 0 \\
    \sin \theta & \cos \theta & 0 & 0 \\
    0 & 0 & 1 & 0 \\
    0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
    x \\
    y \\
    z \\
    1
\end{bmatrix}
\]

### The Rotation Matrix for an Angle (θ) about an Arbitrary Axis (Ax, Ay, Az)
\[
\begin{bmatrix}
    A_x, A_y + \cos \theta(1 - A_x, A_y) & A_x, A_z - \cos \theta(1 - A_x, A_y) & \sin \theta & A_x, A_x - \cos \theta(1 - A_x, A_y) & \sin \theta & A_x, A_z + \cos \theta(1 - A_x, A_y) \\
    A_x, A_y + \cos \theta(1 - A_x, A_y) & A_y, A_z - \cos \theta(1 - A_x, A_y) & -\sin \theta & A_y, A_y + \cos \theta(1 - A_x, A_y) & -\sin \theta & A_y, A_z + \cos \theta(1 - A_x, A_y) \\
    A_x, A_z - \cos \theta(1 - A_x, A_y) & A_y, A_z + \cos \theta(1 - A_x, A_y) & 0 & A_x, A_z - \cos \theta(1 - A_x, A_y) & 0 & A_x, A_z + \cos \theta(1 - A_x, A_y)
\end{bmatrix}
\]

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**
\[
\begin{bmatrix}
    x' \\
    y' \\
    z'
\end{bmatrix} =
\begin{bmatrix}
    A_x, A_y + \cos \theta(1 - A_x, A_y) & A_x, A_z - \cos \theta(1 - A_x, A_y) & \sin \theta & A_x, A_x - \cos \theta(1 - A_x, A_y) & \sin \theta & A_x, A_z + \cos \theta(1 - A_x, A_y) \\
    A_x, A_y + \cos \theta(1 - A_x, A_y) & A_y, A_z - \cos \theta(1 - A_x, A_y) & -\sin \theta & A_y, A_y + \cos \theta(1 - A_x, A_y) & -\sin \theta & A_y, A_z + \cos \theta(1 - A_x, A_y) \\
    A_x, A_z - \cos \theta(1 - A_x, A_y) & A_y, A_z + \cos \theta(1 - A_x, A_y) & 0 & A_x, A_z - \cos \theta(1 - A_x, A_y) & 0 & A_x, A_z + \cos \theta(1 - A_x, A_y)
\end{bmatrix}
\]

**Say it**
Matrix Multiplication is not Commutative

\[
\begin{pmatrix}
X' \\
Y' \\
Z'
\end{pmatrix} =
\begin{pmatrix}
T_{\alpha, A + B} \\
R_y \\
T_{\alpha, A - B}
\end{pmatrix}
\begin{pmatrix}
X \\
Y \\
Z
\end{pmatrix}
\]

One matrix – the Current Transformation Matrix, or CTM

Matrix Multiplication is Associative

\[
\begin{pmatrix}
X' \\
Y' \\
Z'
\end{pmatrix} =
\begin{pmatrix}
T_{\alpha, A + B} \\
R_y \\
T_{\alpha, A - B}
\end{pmatrix}
\begin{pmatrix}
X \\
Y \\
Z
\end{pmatrix}
\]

One matrix – the Current Transformation Matrix, or CTM

One Matrix to Rule Them All

\[
\begin{pmatrix}
X' \\
Y' \\
Z'
\end{pmatrix} =
\begin{pmatrix}
T_{\alpha, A + B} \\
R_y \\
T_{\alpha, A - B}
\end{pmatrix}
\begin{pmatrix}
X \\
Y \\
Z
\end{pmatrix}
\]

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. These diagrams show you why.

Wrong!

Right!

\[
\text{glm::mat3 NormalMatrix = glm::mat3(Model);}
\]

\[
\text{glm::mat3 NormalMatrix = glm::inverseTranspose(glm::mat3(Model));}
\]