Vulkan

GLM

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

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:

```
#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:

```
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 );
```

Deprecated OpenGL

you would now have to say:

```
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

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

The Most Useful GLM Variables, Operations, and Functions

// constructor:

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

The Most Useful GLM Variables, Operations, and Functions

5

// 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);

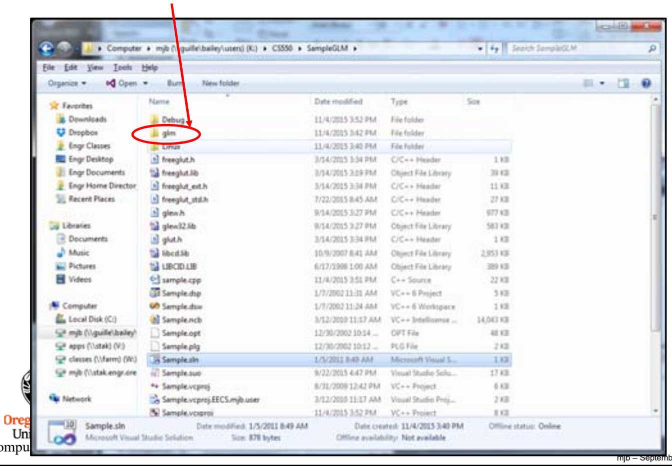
// viewing (assign, not concatenate):

glm::mat4 **glm::lookAt**(glm::vec3 const & eye, glm::vec3 const & look, glm::vec3 const & up);

Installing GLM into your own space

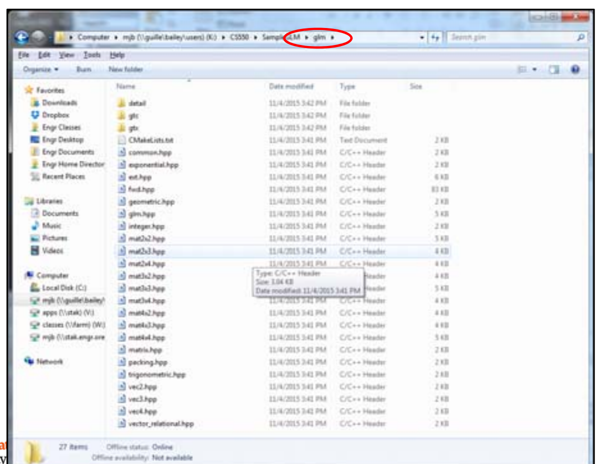
6

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.



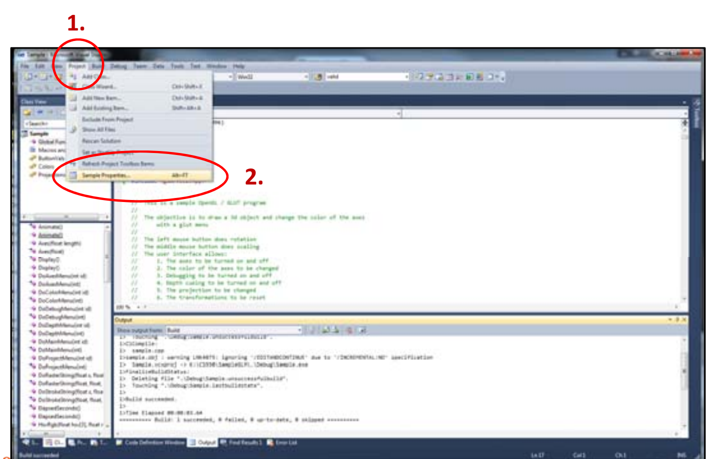
Here's what that GLM folder looks like

7



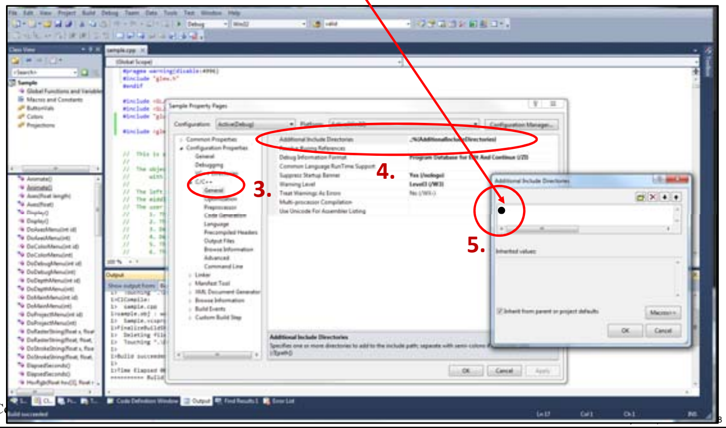
Telling Visual Studio about where the GLM folder is

8



Telling Visual Studio about where the GLM folder is

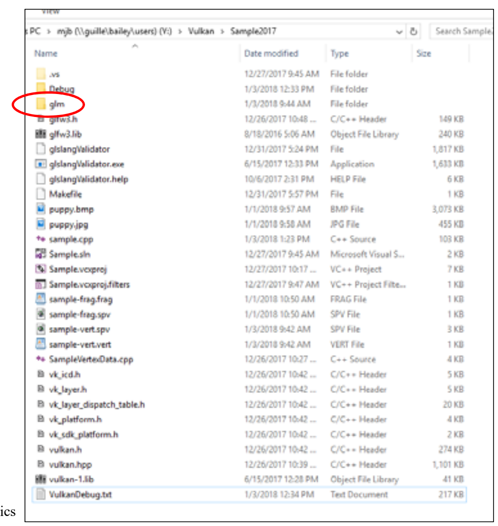
A ***period***, indicating that the **project folder** should also be searched when a
#include <xxx>
is encountered. If you put it somewhere else, enter that full or relative path instead.



GLM in the Vulkan sample.cpp Program


```
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.) );
                    // done this way, the Xrot is applied first, then the Yrot, then the Scale
}
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.uProjectionMatrix = glm::perspective( FOV, (double)Width/(double)Height, 0.1, 1000. );
Matrices.uProjectionMatrix[1][1] *= -1.;

Matrices.uNormalMatrix = glm::inverseTranspose(  glm::mat3( Matrices.uModelMatrix )  );
Fill05DataBuffer( MyMatrixUniformBuffer, (void *) &Matrices );

Misc.uTime = (float)Time;
Misc.uMode = Mode;
Fill05DataBuffer( MyMiscUniformBuffer, (void *) &Misc );
```


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, x^3, etc. terms.

Or, in matrix form:

x consuming column
y consuming column
z consuming column
constant column

$$\begin{array}{r} \text{x' producing row} \\ \text{y' producing row} \\ \text{z' producing row} \\ \end{array} \begin{Bmatrix} x' \\ y' \\ z' \\ 1 \end{Bmatrix} = \begin{bmatrix} A & B & C & D \\ E & F & G & H \\ I & J & K & L \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{Bmatrix} x \\ y \\ z \\ 1 \end{Bmatrix}$$

Transformation Matrices

Translation

$$\begin{Bmatrix} x' \\ y' \\ z' \\ 1 \end{Bmatrix} = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{Bmatrix} x \\ y \\ z \\ 1 \end{Bmatrix}$$

Scaling

$$\begin{Bmatrix} x' \\ y' \\ z' \\ 1 \end{Bmatrix} = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{Bmatrix} x \\ y \\ z \\ 1 \end{Bmatrix}$$

Rotation about X

$$\begin{Bmatrix} x' \\ y' \\ z' \\ 1 \end{Bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{Bmatrix} x \\ y \\ z \\ 1 \end{Bmatrix}$$

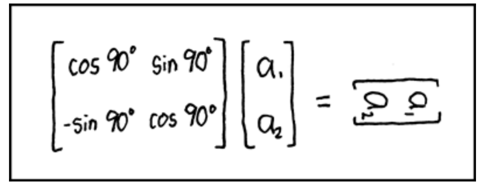
Rotation about Y

$$\begin{Bmatrix} x' \\ y' \\ z' \\ 1 \end{Bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{Bmatrix} x \\ y \\ z \\ 1 \end{Bmatrix}$$

Rotation about Z

$$\begin{Bmatrix} x' \\ y' \\ z' \\ 1 \end{Bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{Bmatrix} x \\ y \\ z \\ 1 \end{Bmatrix}$$

How it Really Works :-)



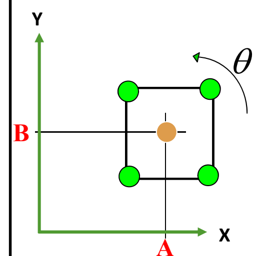
http://xkcd.com

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

$$[M] = \begin{bmatrix} A_xA_x + \cos\theta(1 - A_xA_x) & A_xA_y - \cos\theta(A_xA_y) - \sin\theta A_z & A_xA_z - \cos\theta(A_xA_z) + \sin\theta A_y \\ A_yA_x - \cos\theta(A_yA_x) + \sin\theta A_z & A_yA_y + \cos\theta(1 - A_yA_y) & A_yA_z - \cos\theta(A_yA_z) - \sin\theta A_x \\ A_zA_x - \cos\theta(A_zA_x) - \sin\theta A_y & A_zA_y - \cos\theta(A_zA_y) + \sin\theta A_x & A_zA_z + \cos\theta(1 - A_zA_z) \end{bmatrix}$$

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.

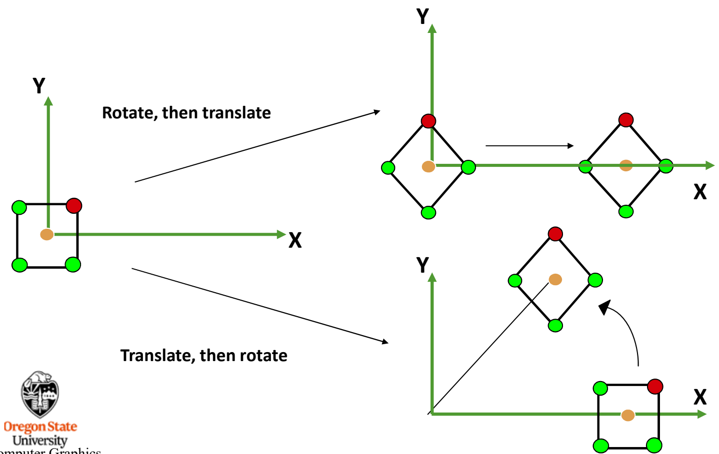
Write it →

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \left(\underset{3}{\left[T_{+A,+B}\right]} \cdot \left(\underset{2}{\left[R_\theta\right]} \cdot \left(\underset{1}{\left[T_{-A,-B}\right]} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \right)\right)\right)$$

← Say it

Matrix Multiplication *is not* Commutative

17



Oregon State University Computer Graphics

Matrix Multiplication *is* Associative

18

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

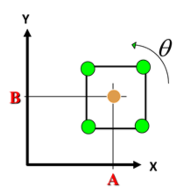
$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \underbrace{\left(\left[T_{+A,+B} \right] \cdot \left[R_\theta \right] \cdot \left[T_{-A,-B} \right] \right)}_{\text{One matrix – the Current Transformation Matrix, or CTM}} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

One matrix –
the Current Transformation Matrix, or CTM

Oregon State University Computer Graphics

One Matrix to Rule Them All

19



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

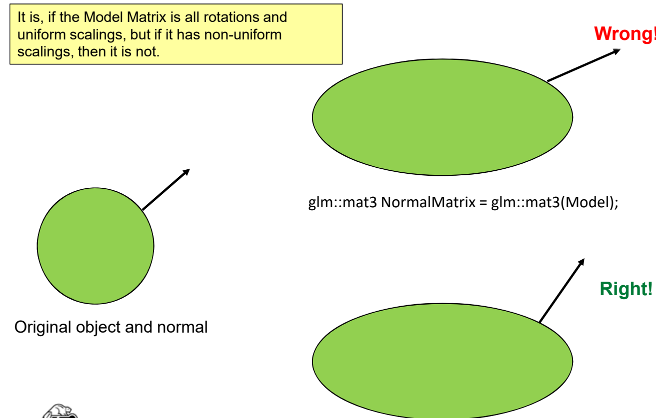
```
glm::mat4  Model = glm::mat4( );
Model = glm::translate(Model, glm::vec3( -A, -B, 0. )  );
Model = glm::rotate(Model, thetaRadians, glm::vec3( Ax, Ay, Az )  );
Model = glm::translate(Model, glm::vec3( A, B, 0. )  );

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)  );
```

Oregon State University Computer Graphics

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

20

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



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

Oregon State University Computer Graphics