OpenGL Notes [a]

Stu Pomerantz

smp@psc.edu

`http://www.psc.edu/~smp`

September 29, 2004

[a] Most material is adapted from: OpenGL ARB, et. al, "The OpenGL Programming Guide", Third Ed., Reading: Addison-Wesley, 1999

Vector Matrix Math Review

Suppose $p = (x, y, z)$ (a point).

And Suppose $M = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ (the identity matrix)

Then $p \cdot M = (x, y, z)$

Vector Matrix Math Review

Recall the formula for rotation about the z-axis is:

$$
\begin{aligned}
x' &= x\cos\theta - y\sin\theta \\
y' &= x\sin\theta + y\cos\theta \\
z' &= z
\end{aligned}
$$

Expressed in matrix form, this rotation looks like:

$$
(x, y, z) \cdot \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix}
$$

Vector Matrix Math Review

If $\theta = \frac{\pi}{2}$ then

$$\begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

becomes

$$\begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

a rotation of 90° counter-clockwise.

Vector Matrix Math Review

For example, if $p = (1, 0, 5)$ then

$$(1,0,5) \cdot \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 5 \end{pmatrix}$$

z remains fixed and x and y are rotated 90 degrees. Generally,

$$(x,y,z) \cdot \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} -y \\ x \\ z \end{pmatrix}$$

Vector Matrix Math Review

Translation p by $(-1,-2,-3)$ and then rotate $\frac{\pi}{2}$:

$$(x-1, y-2, z-3) \cdot \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} -y+2 \\ x-1 \\ z-3 \end{pmatrix}$$

Rotate p by $\frac{\pi}{2}$ and then translate p by $(-1,-2,-3)$:

$$(x, y, z) \cdot \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} -y-1 \\ x-2 \\ z-3 \end{pmatrix}$$

Not the same answer.

Can translation and rotation of a point be accomplished in 1 step ?

Homogeneous Coordinates

Yes. Using homogenious coordinates.

Suppose $p = (x, y, z, 1)$ (a homogenious point).

And Suppose $M = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ (the 4x4 identity matrix)

Then $p \cdot M = (x, y, z, 1)$

Homogeneous Coordinates

Using homogeneous coordinates:

the rotation about z matrix is: $$\begin{pmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

If $\theta = \frac{\pi}{2}$ then this matrix is: $$\begin{pmatrix} 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

a rotation of 90° counter-clockwise.

Homogeneous Coordinates

Using homogenious coordinates to rotate p by $\frac{\pi}{2}$,

$$(x, y, z, 1) \cdot \begin{pmatrix} 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} -y \\ x \\ z \\ 1 \end{pmatrix}$$

will generate the same answer as the non-homogenious version. Wouldn't want to use homogenious coordinates otherwise.

- The last coordinate, which has always been 1 so far, is called w.
- 3D homogenious points are specified as (x, y, z, w).
- 2D homogenious points are specified as (x, y, w).

Homogeneous Coordinates

Using homogenious coordinates to *translate* p by $(-1, -2, -3)$:

$$(x, y, z, 1) \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -1 & -2 & -3 & 1 \end{pmatrix} = \begin{pmatrix} x - 1 \\ y - 2 \\ z - 3 \\ 1 \end{pmatrix}$$

Note the role of the new $w = 1$ coordinate in the vector-matrix multiplication.

Homogeneous Coordinates

Using homogenious coordinates to rotate p by $\frac{\pi}{2}$ and then translate p by $(-1,-2,-3)$:

$$(x,y,z,1)\cdot\begin{pmatrix}0 & 1 & 0 & 0\\ -1 & 0 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{pmatrix}\cdot\begin{pmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ -1 & -2 & -3 & 1\end{pmatrix}=$$

$$(x,y,z,1)\cdot\begin{pmatrix}0 & 1 & 0 & 0\\ -1 & 0 & 0 & 0\\ 0 & 0 & 1 & 0\\ -1 & -2 & -3 & 1\end{pmatrix}=\begin{pmatrix}-y-1\\ x-2\\ z-3\\ 1\end{pmatrix}$$

Homogeneous Coordinates

Using homogenious coordinates to translate p by $(-1,-2,-3)$ and then rotate p by $\frac{\pi}{2}$:

$$(x,y,z,1)\cdot\begin{pmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ -1 & -2 & -3 & 1\end{pmatrix}\cdot\begin{pmatrix}0 & 1 & 0 & 0\\ -1 & 0 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{pmatrix}=$$

$$(x,y,z,1)\cdot\begin{pmatrix}0 & 1 & 0 & 0\\ -1 & 0 & 0 & 0\\ 0 & 0 & 1 & 0\\ 2 & -1 & -3 & 1\end{pmatrix}=\begin{pmatrix}-y+2\\ x-1\\ z-3\\ 1\end{pmatrix}$$

Homogeneous Coordinates

Note:

- Order dependence of matrix multiplication.
- Translation & rotation collapsed to a matrix multiply.
- OpenGL points and transformations use homogenious coordinates.

Homogeneous Matrices inside OpenGL

The following discussion assumes:

```
GLfloat mat[16] ;
```

To set mat to be the 4x4 identity matrix:

```
mat[0]  = 1; mat[1]  = 0; mat[2]  = 0; mat[3]  = 0;
mat[4]  = 0; mat[5]  = 1; mat[6]  = 0; mat[7]  = 0;
mat[8]  = 0; mat[9]  = 0; mat[10] = 1; mat[11] = 0;
mat[12] = 0; mat[13] = 0; mat[14] = 0; mat[15] = 1;
```

Note the order in which the rows and columns are specified.

Homogeneous Matrices inside OpenGL

Query OpenGL state using `glGet*()`. Here are its 4 forms:

```
void glGetBooleanv( GLenum pname, GLboolean *params )
void glGetDoublev( GLenum pname, GLdouble *params )
void glGetFloatv( GLenum pname, GLfloat *params )
void glGetIntegerv( GLenum pname, GLint *params )
```

- `pname` is a defined constant. There are a bunch of them.
- `params` is where OpenGL will return information.

Homogeneous Matrices inside OpenGL

Example 1: Consider this code fragment:

```
glLoadIdentity() ;
glRotatef(90,0,0,1) ;
glGetFloatv(GL_MODELVIEW_MATRIX, mat) ;
```

Printing the values in `mat` :

```
m[0 ] = 0.0    m[1 ] = 1.0    m[2 ] = 0.0    m[3 ] = 0.0
m[4 ] =-1.0    m[5 ] = 0.0    m[6 ] = 0.0    m[7 ] = 0.0
m[8 ] = 0.0    m[9 ] = 0.0    m[10] = 1.0    m[11] = 0.0
m[12] = 0.0    m[13] = 0.0    m[14] = 0.0    m[15] = 1.0
```

Surprised ?

Homogeneous Matrices inside OpenGL

Example 2: Consider this code fragment:

```
glLoadIdentity() ;
glTranslatef(-1,-2,-3) ;
glGetFloatv(GL_MODELVIEW_MATRIX, mat) ;
```

Printing the values in `mat` :

```
m[0 ] = 1.0    m[1 ] = 0.0    m[2 ] = 0.0    m[3 ] = 0.0
m[4 ] = 0.0    m[5 ] = 1.0    m[6 ] = 0.0    m[7 ] = 0.0
m[8 ] = 0.0    m[9 ] = 0.0    m[10] = 1.0    m[11] = 0.0
m[12] =-1.0    m[13] =-2.0    m[14] =-3.0    m[15] = 1.0
```

So far, so good.

Homogeneous Matrices inside OpenGL

Example 3: Consider this code fragment:

```
glLoadIdentity() ;
glTranslatef(-1,-2,-3) ;
glRotatef(90,0,0,1) ;
glGetFloatv(GL_MODELVIEW_MATRIX, mat) ;
```

Printing the values in `mat` :

```
m[0 ] = 0.0    m[1 ] = 1.0    m[2 ] = 0.0    m[3 ] = 0.0
m[4 ] =-1.0    m[5 ] = 0.0    m[6 ] = 0.0    m[7 ] = 0.0
m[8 ] = 0.0    m[9 ] = 0.0    m[10] = 1.0    m[11] = 0.0
m[12] =-1.0    m[13] =-2.0    m[14] =-3.0    m[15] = 1.0
```

Rotation *followed* by translation!

Homogeneous Matrices inside OpenGL

Example 4: Consider this code fragment:

```
glLoadIdentity() ;
glRotatef(90,0,0,1) ;
glTranslatef(-1,-2,-3) ;
glGetFloatv(GL_MODELVIEW_MATRIX, mat) ;
```

Printing the values in `mat` :

```
m[0 ] = 0.0    m[1 ] = 1.0    m[2 ] = 0.0    m[3 ] = 0.0
m[4 ] =-1.0    m[5 ] = 0.0    m[6 ] = 0.0    m[7 ] = 0.0
m[8 ] = 0.0    m[9 ] = 0.0    m[10] = 1.0    m[11] = 0.0
m[12] = 2.0    m[13] =-1.0    m[14] =-3.0    m[15] = 1.0
```

Translation *followed* by rotation.

Homogeneous Matrices inside OpenGL

Example 5: Consider this code fragment:

```
glLoadIdentity() ;
glScalef(2,3,4) ;
glGetFloatv(GL_MODELVIEW_MATRIX, mat) ;
```

Printing the values in `mat` :

```
m[0 ] = 2.0    m[1 ] = 0.0    m[2 ] = 0.0    m[3 ] = 0.0
m[4 ] = 0.0    m[5 ] = 3.0    m[6 ] = 0.0    m[7 ] = 0.0
m[8 ] = 0.0    m[9 ] = 0.0    m[10] = 4.0    m[11] = 0.0
m[12] = 0.0    m[13] = 0.0    m[14] = 0.0    m[15] = 1.0
```

Scaling space in OpenGL using `glScalef()`.

Homogeneous Matrices inside OpenGL

`glMultMatrix*()` multiplies the current matrix state by the specified matrix. It has two forms:

```
void glMultMatrixd( const GLdouble *m )
void glMultMatrixf( const GLfloat *m )
```

- Note that, like the OpenGL transformation function `glMultMatrix*()` multiplies into the current matrix state, *it does not replace the current matrix*.
- The programmer can specify the modelview matrix directly.
- The programmer can specify the projection matrix directly.

Saving and Restoring Matrix State

Recall,

- Saving the Modelview state:

  `glPushMatrix()` pushes the current matrix on to a stack, duplicating the current matrix. That is, after a glPushMatrix call, the matrix on top of the stack is identical to the one below it.

- Restoring the Modelview state:

  `glPopMatrix()` pops the current matrix stack, replacing the current matrix with the one below it on the stack.

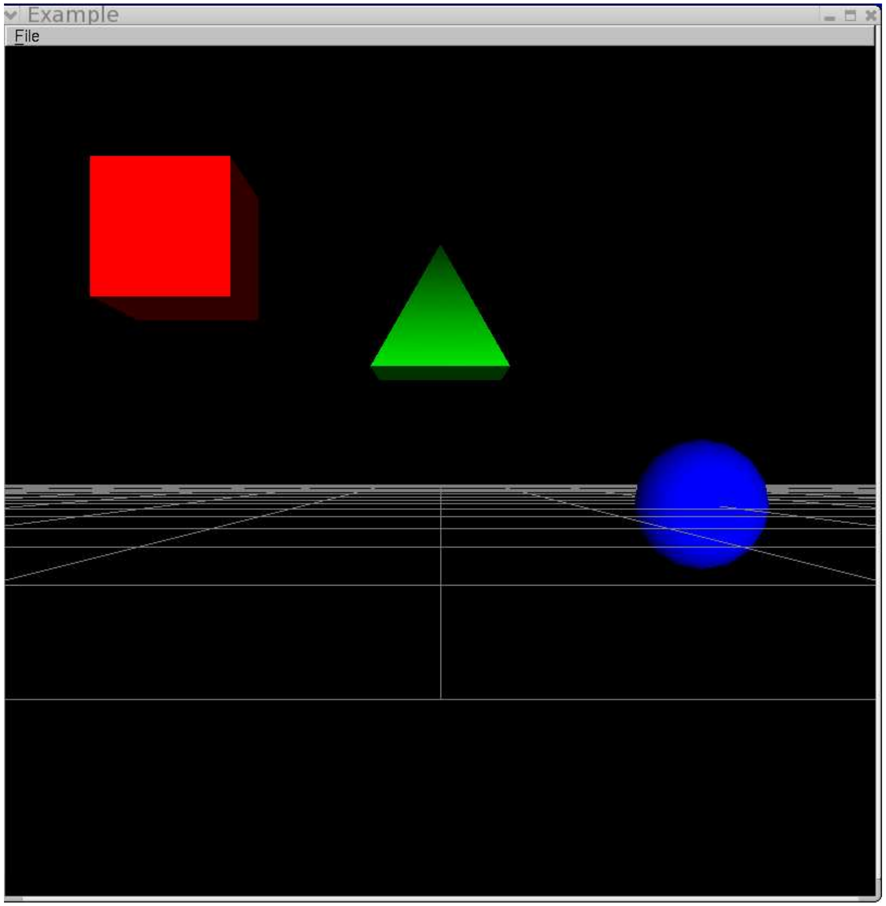
Saving and Restoring Matrix State

Consider this code fragment:

```
glTranslatef(0,1,0) ;
glPushMatrix() ;
  glTranslatef(2,-1,0) ;
  draw_blue_sphere() ;
glPopMatrix() ;
glPushMatrix() ;
  glTranslatef(-2,1,0) ;
  draw_red_cube() ;
glPopMatrix() ;
draw_green_tetrahedron() ;
```

Saving and Restoring Matrix State

The previous code fragment generates:



GLU Projection functions

`gluProject()` maps object coordinates to window coordinates.

```
gluProject(
    GLdouble objX, // point in world space
    GLdouble objY,
    GLdouble objZ,
    const GLdouble *model, // how world maps to screen
    const GLdouble *proj,
    const GLint    *view,
    GLdouble* winX, // point in screen space
    GLdouble* winY,
    GLdouble* winZ ) ;
```

GLU Projection functions

- `gluProject()` will return where a point in world space (object coordinates) projects back to the window (window coordinates).
- `gluProject()` runs a point (objX, objY, objZ) through the OpenGL pipeline and returns the projection in (winX, winY, winZ)
- Note the *winZ* coordinate.
  - winZ $= 0.0 \rightarrow$ near clip plane.
  - winZ $= 1.0 \rightarrow$ far clip plane.
- Not terribly efficient.

GLU Projection functions

`gluProject()` example:

1. Query the OpenGL environment for the modelview, projection, and viewport matrices:

```
GLdouble model[16] ;
GLdouble proj[16] ;
GLint    view[4] ;

glGetDoublev(GL_MODELVIEW_MATRIX,  model) ;
glGetDoublev(GL_PROJECTION_MATRIX, proj) ;
glGetIntegerv(GL_VIEWPORT, view) ;
```

GLU Projection functions

`gluProject()` example continued:

2. Call `gluProject()`.

```
GLdouble objX = 1, objY = 2, objZ = 3 ;
GLdouble winX, winY, winZ ;

gluProject(objX, objY, objZ,
           model, proj, view,
           &winX, &winY, &winZ ) ;
```

This code returns the location in window coordinates where the point (1,2,3) projects.

GLU Projection functions

`gluProject()` map window coordinates to object coordinates.

```
gluUnProject(
     GLdouble winX, // point in screen space
     GLdouble winY,
     GLdouble winZ,
     const GLdouble *model, // how world maps to screen
     const GLdouble *proj,
     const GLint *view,
     GLdouble* objX, // point in world space
     GLdouble* objY,
     GLdouble* objZ )
```

GLU Projection functions

`gluUnProject()` *reverses* the projection transformation.

- `gluUnProject()` will return where a point in the window (window coordinates) projects into world space (object coordinates).
- `gluUnProject()` runs a point (winX, winY, winZ) through the OpenGL pipeline "in reverse" and returns its projection into world space (objX, objY, objZ).
- Not terribly efficient.

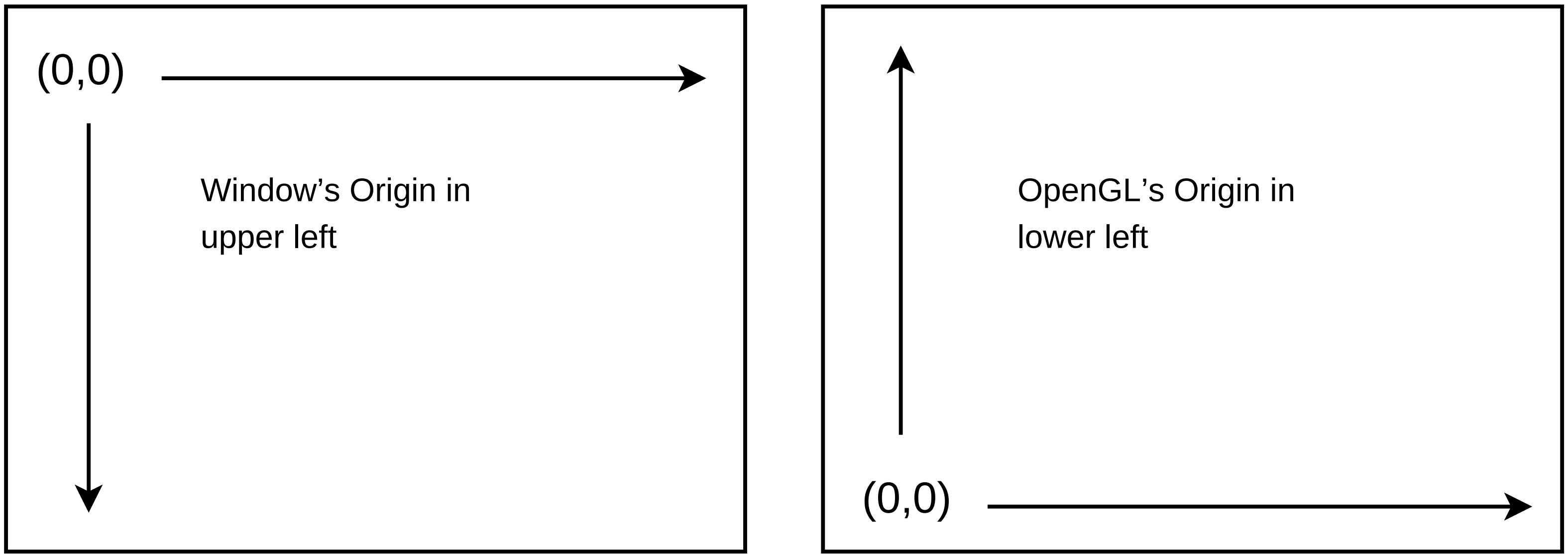
GLU Projection functions

`gluUnProject()` is only occasionally directly useful.

- Consider the earlier Push/Pop example.
  - Intermediate transformations multiplied into the modelview state won't be available without special effort to save them.
- Calculations outside OpenGL are not known to `gluUnProject()` without special effort.

Mapping the Mouse to the OpenGL World

To map the mouse location into the world, note:



Must account for the differences in the assumed origin point.

Mapping the Mouse to the OpenGL World

Suppose:

```
glMatrixMode(GL_PROJECTION) ;
glLoadIdentity() ;
glOrtho(left,right,bottom,top,near,far) ;
```

Also known is:

```
int mouse_x, mouse_y ;   // mouse position
int window_w, window_h ; // window width & height
```

Then

```
float xratio = (float)mouse_x/(float)window_w ;
float yratio = (float)mouse_y/(float)window_h ;
```

Mapping the Mouse to the OpenGL World

Knowing,

```
float xratio = (float)mouse_x/(float)window_w ;
float yratio = (float)mouse_y/(float)window_h ;
```

Then use *interpolation* to find the point in space:

```
float xspace = xratio*right + (1-xratio)*left ;
float yspace = yratio*top   + (1-yratio)*bottom  ;
```

Finally, flip y to account for the difference in origins:

```
yspace *= -1 ;
```