Computer Graphics Notes-3D

When rotating about an axis in OpenGL you can use the right hand rule to determine the + direction (thumb points in axis, finger indicate + rotation direction).

We can think of transformations as changing the coordinate system, where (u, v, n) is the new basis and O is the origin.

$$\begin{pmatrix} u_x & v_x & n_x & O_x \\ u_y & v_y & n_y & O_y \\ u_z & v_z & n_z & O_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

This kind of transformation in is known as a local to world transform. This is useful for defining objects which are made up of many smaller objects. It also means to transform the object we just have to change the local to world transform instead of changing the coordinates of each individual vertex. A series of local to world transformations on objects builds up a scene graph, useful for drawing a scene with many distinct models.

Matrix Stacks

OpenGL has MODELVIEW, PROJECTION, VIEWPORT, and TEXTURE matrix modes.

- glLoadIdentity() – puts the Identity matrix on the top of the stack
- glPushMatrix() – copies the top of the matrix stack and puts it on top
- glPopMatrix()
  For MODELVIEW operations include glTranslate, glScaled, glRotated… These are post multiplied to the top of the stack, so the last call is done first (ie. a glTranslate then glScaled will scale then translate.).

  Any glVertex() called have the value transformed by matrix on the top of the MODELVIEW stack.

  Usually the hardware only supports projection and viewport stacks of size 2, whereas the modelview stack should have at least a size of 32.

The View Volume

Can set the view volume using,(after setting the the current matrix stack to the PROJECTION stack

- glOrtho(left, right, bottom, top, near, far)

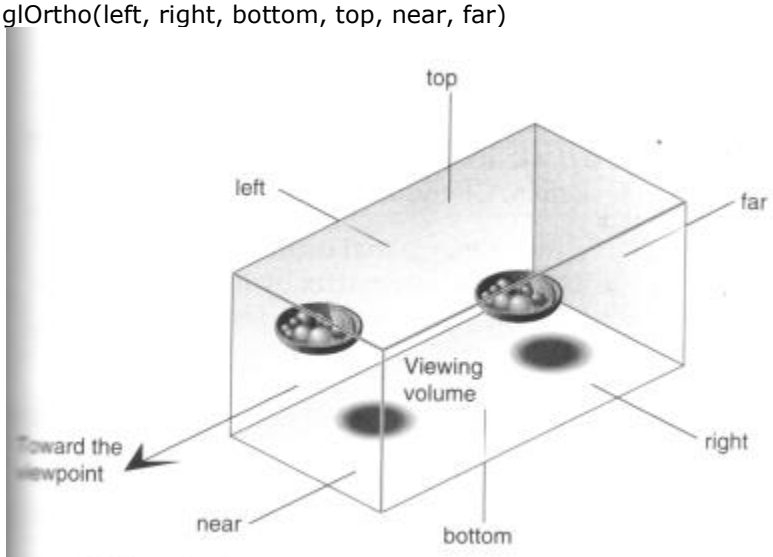

Figure 3-15 Orthographic Viewing Volume

(Source: Unknown)

- glFrustum(left, right, bottom, top, near, far)

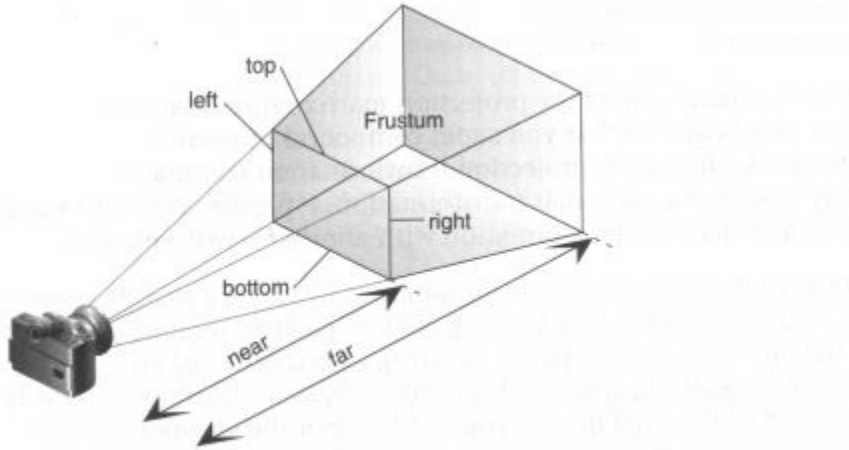

Figure 3-13 Perspective Viewing Volume Specified by glFrustum()

(Source: Unknown)

- gluPerspective(fovy, aspect, zNear, zFar)

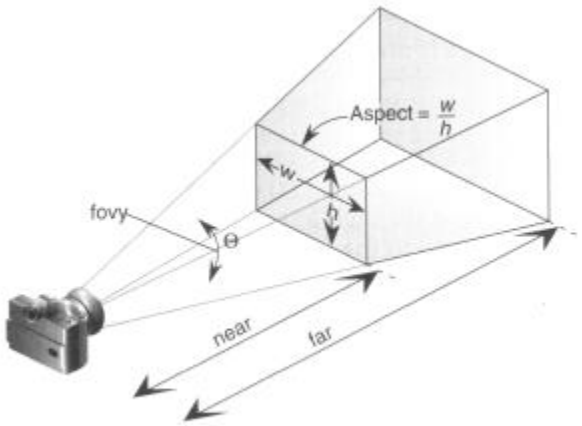

Figure 3-14 Perspective Viewing Volume Specified by gluPerspective()

(Source: Unknown)

In OpenGL the projection method just determines how to squish the 3D space into the canonical view volume.

Then you can set the direction using gluLookAt (after calling one of the above) where you set the eye location, a forward look at vector and an up vector.

When using perspective the view volume will be a frustum, but this is more complicated to clip against than a cube. So we convert the view volume into the canonical view volume which is just a transformation to make the view volume a cube at 0,0,0 of width 2. Yes this introduces distortion but this will be compensated by the final window to viewport transformation.

Remember we can set the viewport with glViewport(left, bottom, width, height) where x and y are a location in the screen (I think this means window, but also this stuff is probably older that modern window management so I'm not worrying about the details here.)

Visible Surface Determination (Hidden Surface Removal)

First clip to the view volume then do back face culling.

Could just sort the polygons and draw the ones further away first (painter's algorithm/depth sorting). But this fails for those three overlapping triangles.

Can fix by splitting the polygons.

BSP (Binary Space Partitioning)

For each polygon there is a region in front and a region behind the polygon. Keep subdividing the space for all the polygons.

Can then use this BSP tree to draw.

```
void drawBSP(BSPTree m, Point myPos{
   if (m.poly.inFront(myPos)) {
      drawBSP(m.behind, myPos);
```

```
            draw(m.poly);
            drawBSP(m.front, myPos);
        }else{
            drawBSP(m.front, myPos);
            draw(m.poly);
            drawBSP(m.behind, myPos);
        }
    }
```

If one polygon's plane cuts another polygon, need to split the polygon.

You get different tree structures depending on the order you select the polygons. This does not matter, but some orders will give a more efficient result.

Building the BSP tree is slow, but it does not need to be recalculated when the viewer moves around. We would need to recalculate the tree if the polygons move or new ones are added.

BSP trees are not so common anymore, instead the Z buffer is used.

Z Buffer

Before we fill in a pixel into the framebuffer, we check the z buffer and only fill that pixel is the z value (can be a pseudo-depth) is less (large values for further away) than the one in the z buffer. If we fill then we must also update the z buffer value for that pixel.

Try to use the full range of values for each pixel element in the z buffer.

To use in OpenGL just do gl.glEnable(GL.GL_DEPTH_TEST) and to clear the z-buffer use gl.glClear(GL.GL_DEPTH_BUFFER_BIT).

Fractals

L-Systems

Line systems. eg. koch curve

Self-similarity

- Exact (eg. sierpinski trangle)
- Stochastic (eg. mandelbrot set)

IFS – Iterated Function System

Source: http://andrewharvey4.wordpress.com/2009/12/02/computer-graphics-notes/