8 Using Viewing and Camera Transforms, and gluLookAt()

8.010 How does the camera work in OpenGL?

As far as OpenGL is concerned, there is no camera. More specifically, the camera is always located at the eye space coordinate (0., 0., 0.). To give the appearance of moving the camera, your OpenGL application must move the scene with the inverse of the camera transformation.

8.020 How can I move my eye, or camera, in my scene?

OpenGL doesn't provide an interface to do this using a camera model. However, the GLU library provides the gluLookAt() function, which takes an eye position, a position to look at, and an up vector, all in object space coordinates. This function computes the inverse camera transform according to its parameters and multiplies it onto the current matrix stack.

8.030 Where should my camera go, the ModelView or Projection matrix?

The GL_PROJECTION matrix should contain only the projection transformation calls it needs to transform eye space coordinates into clip coordinates.

The GL_MODELVIEW matrix, as its name implies, should contain modeling and viewing transformations, which transform object space coordinates into eye space coordinates. Remember to place the camera transformations on the GL_MODELVIEW matrix and never on the GL_PROJECTION matrix.

Think of the projection matrix as describing the attributes of your camera, such as field of view, focal length, fish eye lens, etc. Think of the ModelView matrix as where you stand with the camera and the direction you point it.

The game dev FAQ has good information on these two matrices.

Read Steve Baker's article on projection abuse (local mirror). This article is highly recommended and well-written. It's helped several new OpenGL programmers.

8.040 How do I implement a zoom operation?

A simple method for zooming is to use a uniform scale on the ModelView matrix. However, this often results in clipping by the *zNear* and *zFar* clipping planes if the model is scaled too large.

A better method is to restrict the width and height of the view volume in the Projection matrix.

For example, your program might maintain a zoom factor based on user input, which is a floating-point number. When set to a value of 1.0, no zooming takes place. Larger values result in greater zooming or a more restricted field of view, while smaller values cause the opposite to occur. Code to create this effect might look like:

```
static float zoomFactor; /* Global, if you want. Modified by user input. Initially 1.0 */

/* A routine for setting the projection matrix. May be called from a resize
   event handler in a typical application. Takes integer width and height
   dimensions of the drawing area. Creates a projection matrix with correct
   aspect ratio and zoom factor. */
void setProjectionMatrix (int width, int height)
{
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
```

```
    gluPerspective (50.0*zoomFactor, (float)width/(float)height, zNear, zFar);
    /* ...Where 'zNear' and 'zFar' are up to you to fill in. */
}
```

Instead of gluPerspective(), your application might use glFrustum(). This gets tricky, because the *left, right, bottom,* and *top* parameters, along with the *zNear* plane distance, also affect the field of view. Assuming you desire to keep a constant *zNear* plane distance (a reasonable assumption), glFrustum() code might look like this:

```
glFrustum(left*zoomFactor, right*zoomFactor,
    bottom*zoomFactor, top*zoomFactor,
    zNear, zFar);
```

glOrtho() is similar.

8.050 Given the current ModelView matrix, how can I determine the object-space location of the camera?

The "camera" or viewpoint is at (0., 0., 0.) in eye space. When you turn this into a vector [0 0 0 1] and multiply it by the inverse of the ModelView matrix, the resulting vector is the object-space location of the camera.

OpenGL doesn't let you inquire (through a glGet* routine) the inverse of the ModelView matrix. You'll need to compute the inverse with your own code.

8.060 How do I make the camera "orbit" around a point in my scene?

You can simulate an orbit by translating/rotating the scene/object and leaving your camera in the same place. For example, to orbit an object placed somewhere on the Y axis, while continuously looking at the origin, you might do this:

```
gluLookAt(camera[0], camera[1], camera[2], /* look from camera XYZ */
               0, 0, 0, /* look at the origin */
               0, 1, 0); /* positive Y up vector */
glRotatef(orbitDegrees, 0.f, 1.f, 0.f);/* orbit the Y axis */
/* ...where orbitDegrees is derived from mouse motion */

glCallList(SCENE); /* draw the scene */
```

If you insist on physically orbiting the camera position, you'll need to transform the current camera position vector before using it in your viewing transformations.

In either event, I recommend you investigate gluLookAt() (if you aren't using this routine already).

8.070 How can I automatically calculate a view that displays my entire model? (I know the bounding sphere and up vector.)

The following is from a posting by Dave Shreiner on setting up a basic viewing system:

First, compute a bounding sphere for all objects in your scene. This should provide you with two bits of information: the center of the sphere (let (c.x, c.y, c.z) be that point) and its diameter (call it "diam").

Next, choose a value for the *zNear* clipping plane. General guidelines are to choose something larger than, but close to 1.0. So, let's say you set

```
zNear = 1.0;
zFar = zNear + diam;
```

Structure your matrix calls in this order (for an Orthographic projection):

```
GLdouble left = c.x - diam;
GLdouble right = c.x + diam;
```

```
  GLdouble bottom c.y - diam;
  GLdouble top = c.y + diam;

  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  glOrtho(left, right, bottom, top, zNear, zFar);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
```

This approach should center your objects in the middle of the window and stretch them to fit (i.e., its assuming that you're using a window with aspect ratio = 1.0). If your window isn't square, compute *left, right, bottom,* and *top,* as above, and put in the following logic before the call to glOrtho():

```
  GLdouble aspect = (GLdouble) windowWidth / windowHeight;

  if ( aspect < 1.0 ) { // window taller than wide
     bottom /= aspect;
     top /= aspect;
  } else {
     left *= aspect;
     right *= aspect;
  }
```

The above code should position the objects in your scene appropriately. If you intend to manipulate (i.e. rotate, etc.), you need to add a viewing transform to it.

A typical viewing transform will go on the ModelView matrix and might look like this:

```
  GluLookAt (0., 0., 2.*diam,
                  c.x, c.y, c.z,
                  0.0, 1.0, 0.0);
```

8.080 Why doesn't gluLookAt work?

This is usually caused by incorrect transformations.

Assuming you are using gluPerspective() on the Projection matrix stack with *zNear* and *zFar* as the third and fourth parameters, you need to set gluLookAt on the ModelView matrix stack, and pass parameters so your geometry falls between *zNear* and *zFar.*

It's usually best to experiment with a simple piece of code when you're trying to understand viewing transformations. Let's say you are trying to look at a unit sphere centered on the origin. You'll want to set up your transformations as follows:

```
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(50.0, 1.0, 3.0, 7.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  gluLookAt(0.0, 0.0, 5.0,
            0.0, 0.0, 0.0,
            0.0, 1.0, 0.0);
```

It's important to note how the Projection and ModelView transforms work together.

In this example, the Projection transform sets up a 50.0-degree field of view, with an aspect ratio of 1.0. The *zNear* clipping plane is 3.0 units in front of the eye, and the *zFar* clipping plane is 7.0 units in front of the eye. This leaves a Z volume distance of 4.0 units, ample room for a unit sphere.

The ModelView transform sets the eye position at (0.0, 0.0, 5.0), and the look-at point is the origin in the center of our unit sphere. Note that the eye position is 5.0 units away from the look at point. This is important, because a distance of 5.0 units in front of the eye is in the middle of the Z volume that the

Projection transform defines. If the gluLookAt() call had placed the eye at (0.0, 0.0, 1.0), it would produce a distance of 1.0 to the origin. This isn't long enough to include the sphere in the view volume, and it would be clipped by the *zNear* clipping plane.

Similarly, if you place the eye at (0.0, 0.0, 10.0), the distance of 10.0 to the look at point will result in the unit sphere being 10.0 units away from the eye and far behind the *zFar* clipping plane placed at 7.0 units.

If this has confused you, read up on transformations in the OpenGL red book or OpenGL Specification. After you understand object coordinate space, eye coordinate space, and clip coordinate space, the above should become clear. Also, experiment with small test programs. If you're having trouble getting the correct transforms in your main application project, it can be educational to write a small piece of code that tries to reproduce the problem with simpler geometry.

8.090 How do I get a specified point (XYZ) to appear at the center of the scene?

gluLookAt() is the easiest way to do this. Simply set the X, Y, and Z values of your point as the fourth, fifth, and sixth parameters to gluLookAt().

8.100 I put my gluLookAt() call on my Projection matrix and now fog, lighting, and texture mapping don't work correctly. What happened?

Look at question 8.030 for an explanation of this problem.

8.110 How can I create a stereo view?

Paul Bourke has assembled information on stereo OpenGL viewing.

- 3D Stereo Rendering Using OpenGL
- Creating Anaglyphs using OpenGL

Column Header
Column Footer