Lab #7, 3D Viewing

Rashid Kaleem

27 October, 2007

Introduction
In this lab, we would be looking at displaying simple objects in 3D, and the displaying them. Whenever we are displaying them, we would need to set up the camera in order to decide what part of the screen we would like to display.

Drawing simple 3D shapes
In order to draw three dimensional shapes, we do not need to change much of our code. All we need to do is to pass 3 arguments to the vertices of whatever basic shape we would like to draw and the shape is drawn in 3D. For instance, we can draw two triangles using the following snippet:

```c
glBegin(GL_TRIANGLES);
    glVertex3f(-0.5,-0.5,0.0);
    glVertex3f(0.5,0.0,0.0);
    glVertex3f(0.0,0.5,0.0);
    glVertex3f(0.0,-0.5,-0.5);
    glVertex3f(0.0,0.5,0.0);
    glVertex3f(0.0,0.0,0.5);
glEnd();
```

The above snippet basically draws two triangles with the specified three dimensional vertices. If we run the above snippet in our existing OpenGL program (the one where we draw basic shapes in 2D, as in the previous lab) we would see nothing new. This is because we have not properly set up the camera. What we need to do now is to set up the camera so that it gives us a three dimensional view of the shape being drawn.

Setting up the camera
In order to set up the camera, we need to establish the location and direction of the camera. This is done by using the function gluLookAt which is provided the following parameters:

1. Location of the Camera (Eye).
2. Location of the point in world coordinates that appears in the center of the camera.

3. The direction of the up vector, since only specifying a direction of sight is not sufficient.

The function is called using the arguments as follows:

```c
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt( -0.5, 1.0, 5.0, //Camera -0.5, 0.0,-1.0, //Centre 0.0, 1.0, 0.0); //Up
```

But before this can be done, it is necessary to set up a view. Here we will be using a perspective projection, which we will be discussing in more detail later. For now it is sufficient to say that a perspective projection specified by four arguments creates a pyramid within the 3D world that is drawn onto the 2D screen. This is quite the way humans see the world. In order to create a perspective projection in OpenGL, we use the following sequence of commands:

```c
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(45,aspectRatio,1,1000);
```

For now we have set up the camera to look at a point from a given direction. Now we would start by moving the camera about and redrawing the object where it was.

### Moving the camera about

Once setup, the camera is easy to move about. All you have to do is to change the parameters to the `gluLookAt` function. Remember that the three parameters are needed to correctly produce a view. Also, there are two ways to produce the effect of relative motion. We can move the camera or we can move the object. If we are to move the camera, we would just change the location of the camera as specified in the `gluLookAt` function.

### Using basic transforms

As mentioned in the text, OpenGL maintains the list of all transformations to be applied to a drawing in a `MODELVIEW` matrix. This matrix can be saved by pushing it onto the stack so it can be later retrieved by popping the stack. However each matrix operation needs to be informed as to which matrix is to be manipulated. We have seen two matrices so far; `MODELVIEW` and `PROJECTION`. In order to select a matrix for manipulation, we would select the matrix using `glMatrixMode` function. Similarly the `glLoadIdentity` transform is used to initialize any selected matrix to identity matrix so that it is properly initialized.
Now for instance, if we wanted to draw a shape rotated by theta about the origin, all we would do is:

```c
glMatrixMode(GL_MODELVIEW);
glPushMatrix();
glRotatef(theta, 0.0, 0.0, 1.0);
// Draw shape using glBegin/glEnd
.......... glPopMatrix();
```

Similarly we can use any sequence of glScalef, glTranslatef and glRotatef as we like building up composite transforms. Just remember to pop the matrix back after you have done drawing using the transformations.

**General Outline**

The general outline for the program are as follows;

- Initialize OpenGL in the main function, install the appropriate event handlers (display, keyboard, resize).

- Set up the initial camera position in the resize function, this will help avoid any problem when a window resize. This includes setting up a viewport using the best aspect ratio. Also set up the initial camera position and direction.

- In the display function, use static variable to store information that will change for each frame, such as the angle of rotation in an animation. This is far better than using global variables, however sometimes less flexible.

- I find it useful to draw the three axis in different colors before I draw anything else just to identify any problems. This can be done by drawing a line from the origin to a point on each axis in different colors (red, green and blue for X, Y and Z axis seems a natural choice).

- Change the camera direction by using modified values to gluLookAt function.

- If you are going to be drawing other shapes that need the MODELVIEW matrix to be modified, save the current MODELVIEW matrix using glPushMatrix. Perform the transformation, draw the shape and pop the matrix back using glPopMatrix.

- Refer to the text (FS Hill) chapter 5 for a detailed look at the working or to solve any ambiguities I might have left in this manual.

**Evaluation**

1. Draw a 3D sphere and a pyramid suitably separated near the origin. The shapes should be parameterized to be drawn in any size, they are however drawn at the origin.
2. Write a function that moves the camera in a circle (x and y plane) around a shape. The camera should always be looking at the object.

3. Modify the above function so that it moves up and down (in the z-axis) as well as circulate. Also have the path drawn before hand so we can see what the path looks like.

4. The Earth and its moon. Draw two spheres. One is rotating at a distance about the y-axis, whereas the other smaller sphere is rotating about the first sphere.

5. Use the keyboard to navigate around the above simulation. Use the arrow keys to move forward, backward, turn left or right and use the + and = keys to look up or down respectively. The escape key or ALT+F4 exits the application.