CIS 636 Interactive Computer Graphics
CIS 736 Computer Graphics
Spring 2011

Lab 1a of 7
OpenGL Setup and Basics

Fri 28 Jan 2011
Part 1a (#1 – 5) due: Thu 03 Feb 2011 (before midnight)

The purpose of this lab exercise is to help you get up and running with Mesa (Linux OpenGL) in the CIS Department's Linux environment and over XWindows, and to show you some basic rendering.

This lab assignment is worth a total of 10 points (1%).

Upload an electronic copy of the assignment in PDF form (converted from your word processor, or scanned) to your K-State Online (KSOL) drop box before the due date and time.

References

NeonHelium tutorials: http://nehe.gamedev.net
Mesa home page: http://www.mesa3d.org
OpenGL FAQ: http://www.opengl.org/resources/faq/
OpenGL viewing docs: http://www.opengl.org/resources/faq/technical/viewing.htm

Problems 4-7 of this lab are adapted from:

Dunham, D. (2008). Lab 5, *CS 5721 (Computer Graphics)*, fall, 2008. Duluth, USA: University of Minnesota. Retrieved from: http://bit.ly/9dLERE

1. **(20%) Modelview transformation: 3-D Rotation of 2-D objects.** Follow Lesson 03 to rotate the flat polygons and then render them. Turn in `lab1_1.c` and `lab1_1.jpg`.

2. **(20%) Modelview transformation: 3-D Rotation of 3-D objects.** Follow Lesson 04 to draw 3-D polyhedra and rotate them. Turn in `lab1_2.c` and `lab1_2.jpg`.

3. **(20%) XWindows.** Repeat Lesson 04 from a notebook computer or PC running Mac OS X, Windows XP, Windows Vista, or Windows 7. Turn in `lab1_3.jpg`.

4. **(10%) Step 1. Perspective Viewing with OpenGL**

   In addition to orthographic viewing (discussed in last lab), you can also choose to use perspective viewing. This is done very similarly to orthographic viewing in that you still need to tell OpenGL which matrix stack you want to modify. To specify that you want to modify the PROJECTION matrix, you use the following lines of code, which generally happen after the glViewport function call:

```
glMatrixMode( GL_PROJECTION );

glLoadIdentity();
```

These two functions tell the graphics card to change the active matrix to the projection matrix and then set it to the identity matrix, which essentially clears the matrix. Next, we want to set the actual projection. The very general function that sets up the perspective matrix is call glFrustum. For instance,

```
glFrustum( -width/2.0, width/2.0, -height/2.0, height/2.0, -1.0,
-20.0 );
```

The arguments to `glFrustum(...)` specify the left, right, bottom, top, near, and far values for the perspective matrix.

However, a utility function (from the GL Utility library) is also available to give you a different (and maybe easier) way for creating the perspective matrix using a field of view (how wide of a view) and aspect ratio parameter (these two concepts are described in the course book):

```
gluPerspective( field_of_view, aspect, near, far );
```

The arguments to this function represent the field of view (in degrees) in the Y (up) direction, the aspect ratio (width over height), the near plane, and the far plane.

Note that in class we learned to have the near and far planes be negative. OpenGL flips the values to negative internally, and thus, *requires POSITIVE* values for near and far.

After we set the projection, we must tell OpenGL to make the current matrix stack be the `MODELVIEW` matrix. Thus, these two calls must follow:

```
glMatrixMode( GL_MODELVIEW );

glLoadIdentity();
```

5. **(10%) Step 2: Making things move in Perspective**

Using the code you developed for the last lab, in which we made a small triangle move around in a circle on the screen, make the following changes to get a feel for perspective viewing.

Set your projection matrix to a perspective matrix using the example `glFrustum` call provided above.

What do you expect to see with these changes? Compile your code and run. Is this what you expected? Put a screenshot of the perspective projection into your archive and check that it is the same as your equivalent `glFrustum` call.

6. **(10%) Step 3: Specifying the Viewing matrix**

You specify the viewing matrix with the function `gluLookAt`(...). This function constructs the viewing matrix discussed in class and places it on the OpenGL Matrix stack. This allows you to specify the "camera's" viewpoint, with the location of the eye, where the eye is looking, and the up vector. Remember that the default camera that you get with OpenGL has the eye at (0,0,0), the center or where the camera is looking is (0,0,-1), and up is (0,1,0).

```
gluLookAt(  eyeX, eyeY, eyeZ,
            centerX, centerY, centerZ,
            upX, upY, upZ                 );
```

The eye values are where the camera is located. The center values represent what the camera is centered on, or rather where it's looking, and the up value is the vector pointing directly up. The three vectors help to form an ORTHONORMAL BASIS for the camera. Modify your code to change the view of your moving triangle. The gluLookAt function should be called after the matrix stack has been switched back to the GL_MODELVIEW matrix and the Identity matrix has been loaded on that stack.

First, set your gluLookAt variables to the following:

eye = (0, 0, 0), center = (0, 0, -1), up = (0, 1, 0)

this is essentially the default viewing condition and should give you the same view as you had before. Now modify it to change the eye point (for instance to *eye=(0.1,0,1)*), or other parameters a bit. This function is how you will change viewpoints (or camera) locations in your OpenGL applications.

7. **(10%) Step 4: Instancing Objects**

In your wireframe drawing and solid object drawing assignments (HW4 & HW5), you will instance objects like Triangle and Cube to create scenes with multiple objects. We will now do the same with OpenGL using OpenGL's matrices. You can create your own objects out of triangles (as we've done so far in these labs). However, you can also use utility functions like:

```
glutWireTorus(innerRadius, outerRadius, nsides, rings)
glutSolidTorus(innerRadius, outerRadius, nsides, rings)
glutSolidSphere(radius, slices, stacks)
glutWireSphere(radius, slices, stacks)
glutSolidCone(base, height, slices, stacks)
glutWireCone(base, height, slices, stacks)
```

Unfortunately, when you use these functions, you don't have access to the vertices in these objects, so how do you translate or rotate them? In your own code, you could apply a transform to scale, rotate, or translate the vertices of an object. With OpenGL, you will change the vertices' locations using OpenGL Matrix stacks.

The following functions create matrices (like the ones you've developed in your classes) to rotate, translate, and scale objects and then places the matrix on the OpenGL Matrix stack. For instance,

```
glTranslatef(x, y, z);
```

will create a translation matrix with x, y, and z values and multiply it to the current matrix stack. Similarly,

```
glRotatef(angle_in_degress, x, y, z);
```

will create a rotation matrix that rotates objects by the angle specified about the axis defined in the arguments. So, if you want to rotate by 45 degrees about the Z-axis, you'd have glRotatef(45.0, 0,0,1). Likewise,

```
glScalef(x, y, z)
```

will create a scale matrix on the matrix stack.

As in your matrix classes, order of operation matters! You can make these calls over and over to manipulate an object, and the order in which they will be executed is opposite of the order in which they appear.

The other functions you need to use are `glPushMatrix()` and `glPopMatrix()`. These functions allow you to work on objects independent of other matrix transforms that have occurred (or will soon occur). The "push" matrix function pushes the matrices in the matrix stack down one level so you can work on the current matrix, making any transformations you want to make. When you're done making transformations, you use the "pop" function to restore the matrix stack to what it was before you pushed the matrix stack down a level. You will use the functions quite a bit, so let's try an example:

First, change your perspective matrix to use the `gluPerspective` function with an field of view of 45.0 degrees and an aspect ration of (float)width/(float)height. Set the near plane to 1 and the far plane to 250. Also, restore your eye, center, and up vectors to the defaults.

Next, create a red wireframe cone and translate it, by -5 in X, 20 in Y, and -100 in Z.

```
// makes the current color red
glColor3f(1.0, 0.0, 0.0);
// pushes the current matrix stack down and makes a copy on top
glPushMatrix();
// multiplies the top of stack with trans matrix
glTranslatef(-5, 20, -100);
// creates a wire cone using current matrix
glutWireCone(20, 30, 20, 20);
glPopMatrix();
```

Now, rotate it by 45 degrees about the Y-axis. However, we want the rotation to occur first. Otherwise, the cone will definitely be rotated out of view. Add the following line after the translate function call:

```
glRotatef(45.0, 0,1,0);
```

If you had other objects that were subject to the same transformations, the code to draw them (whether glut utility function or glVertex calls) could all be placed between the glPushMatrix and glPopMatrix calls.

If you have time, add a green torus and blue sphere to your scene and place them off in the distance using these parameters. Note you will use separate glPushMatrix and glPopMatrix code blocks for the three objects:

```
glutWireSphere (blue): translate=(20, -20, -200)
```

`glutWireTorus (green): translate=(-20,-40,-150), rotate1=(23.6, 0,1,0), rotate2=(68, 1,0,0)`

note: rotations should occur first and in the order given by their names.

Turn in a view of a red `glutSolidCone`. **Note**: you may combine your screenshot and source code for parts 4-7 into a single one.

A look ahead to MP2: Watch OpenGL Tutorial 1 – 2 of 3.

Class Participation (required):

Select a term project topic by Fri 18 Feb 2011. Upload an **editable** (text, RTF, Word, *etc.*) version of your draft project proposal to your File Dropbox no later than Thu 10 Feb 2011, when Lab 1b is due. Respond to and follow the instructor and teaching assistant's comments in your revised proposal. You may post discussion threads, draft material, sample citations, *etc.* in the class discussion board on KSOL, to CIS636-L or CIS736-L (remember, these generate actual e-mails to all students in the class as well as the instructor and TA), or CIS736TA-L (which goes only to the instructor and TA).