Computer Graphics Lab 4

In this lab you will extend your house application in lab 3 to use the openGL lighting model and experiment with a sphere object using specular reflection:

1. You will use GL_POLYGON instead of GL_LINE_LOOP to render the house. What you might find here is that some of the faces don't render properly. This will be most likely to do with the way you specified the vertices originally, remember to do it in a counter clockwise direction when viewing from outside to inside.

2. To make sure only front faces are rendered, we will have to tell openGL not to render back faces. We do this by firstly enabling face culling and then informing openGL that we want back faces to be culled. So add the following lines to the initialise method

   ```
       gl.glEnable(GL.GL_CULL_FACE);
       gl.glCullFace(GL.GL_BACK);
       gl.glEnable(GL.GL_DEPTH_TEST);
   ```

   The last line enables the depth test, this is where openGL figures out which objects are closer as these will obscure objects that are further away. openGL uses a depth buffer to keep track of this depth ordering (more on this later) so we need to clear this depth buffer with the colour buffer in the **display** function, so where you use to have

   ```
   gl.glClear(GL.GL_COLOR_BUFFER_BIT);
   ```

   now you have,

   ```
   gl.glClear(GL.GL_COLOR_BUFFER_BIT | GL.GL_DEPTH_BUFFER_BIT);
   ```

3. The next step is to set up some lights. We're going to be using OpenGL lighting so the first thing to do is tell OpenGL we need it to do the lighting and shading calculations for the geometry we send it. There are a number of steps to achieve this. First we must enable lighting and enable at least one OpenGL light source. Our revised initialise function should also include:

   ```
       gl.glEnable(GL.GL_LIGHTING);
       gl.glEnable(GL.GL_LIGHT0);
       float [] whiteLight = {1, 1, 1, 1};
       float [] ambientLight = {0.1f, 0.1f, 0.1f, 1.0f}; //default
       gl.glLightfv(GL.GL_LIGHT0, GL.GL_DIFFUSE, whiteLight);
       gl.glLightfv(GL.GL_LIGHT0, GL.GL_SPECULAR, whiteLight);
       gl.glLightfv(GL.GL_LIGHT0, GL.GL_AMBIENT, ambientLight);
   ```

   We enable OpenGL lighting and enable the first light (**GL_LIGHT0**). We then set the lights diffuse and specular colour to white (1,1,1,1). We won't use ambient light for the moment. The function glLightfv is used to set various properties of the light source such as position, direction, distance attenuation and diffuse/specular/ambient properties as is done above.

The next step in the lighting process is to position the light source so it illuminates our scene. Lights are subject to the model-view transformation so can be moved around also so we need to be careful about where in our programs we position them. If we want the light position to remain fixed then we should position them in the initialise function also, so we add:

```
float [] lightPosition = {25, 25, 25, 1};
gl.glLightfv(GL_LIGHT0, GL_POSITION, lightPosition);
```

Here we've placed our light at (25, 25, 25) so it should shine on the upper right corner of our house. Notice the one trailing the light position. This is the homogeneous coordinate and tells OpenGL we want a point source. If it was a zero the light would be a directional parallel light source and the (25, 25, 25) would specify the direction instead of a position.

The next step is to set up material properties for your objects. When using OpenGL lighting we no longer set colors for our geometry. We need to set material properties. We can set colour of the material and it’s diffuse/specular reflection properties through the use of the function glMaterial{if}v. Do this in the drawHouse() function before you start rendering the polygons.

```
float [] diffuse_mp = {1.0f,0.0f,0.0f,1.0f};//red
gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL_DIFFUSE, diffuse_mp);
```

So this sets the RGBA diffuse properties of the object. Here we apply it to the front and back faces of the object but we might have applied to one or the other.

The only thing left to do is add normal vectors to each of our polygons that form the house. We need to specify surface normals in openGL so VisibleSurfaceDetection algorithms/lighting algorithms work correctly. To specify a surface normal in openGL we use the gl function **`glNormal3f(a, b, c)`**

So for example let’s say we have a cube face, we would specify the surface normal with the vertices as follows:

```
gl.glBegin(GL_QUADS);
      // front
      gl.glNormal3f(0, 0, 1);
      gl.glVertex3f(-1, 1, 1);
      gl.glVertex3f(-1, -1, 1);
      gl.glVertex3f(1, -1, 1);
      gl.glVertex3f(1, 1, 1);

gl.glEnd();
```

Now position the camera near the light source (or even at the same location as the light source). What you should notice is that some of the polygons or facets are brighter/darker than others based on the lighting model.

4. Try also adding a blue sphere somewhere to the scene. Use the function glut.glutSolidSphere(5,20,20) from the GLUT library to render the sphere (you will have to instantiate this object). This function draws the polygons that make up a sphere and it computes the normals so you do not have to. The first parameter is the size/radius of the sphere and the second two are to do with the number of polygons used in approximating the sphere. Before you call the sphere function define the shading model to be flat shading by calling the function gl.glShadeModel(GL.GL_FLAT). See what the image looks like? Does it look normal? Justify your answer. Try changing the shading model to GL_SMOOTH, now what happens? What do you think is the difference between these two types of shading?

5. Lastly try adding some specular reflection to the sphere. Do this by specifying a specular reflection property for the material in a similar manner to the diffuse reflection property is set. Make the r,g,b values equal when specifying the specular properties of the surface. You will also have to set the width of the cone for the surface. This is done through the function glMaterialf(GL_FRONT_AND_BACK,GL_SHININESS,n) where n determines the width of the cone.