COM3503/4503/6503: 3D Computer Graphics

Dr. Steve Maddock

Exercise sheet 4: Lighting in OpenGL

This exercise sheet will focus on lighting and we will use the GLUT objects to build the scene. The next exercise sheet will look at how to build an object library of our own, which gives us some alternatives to the GLUT objects or the quadric objects.

1. Introduction

OpenGL offers a range of parameters to control lighting calculations. Global lighting parameters can be set, a minimum of 8 individual lights (depending on the particular OpenGL implementation) can have their properties set, and individual objects can have their material properties set. All of these combine, together with the viewing system, to determine how objects will look when displayed on the screen. The theory of lighting is covered in lectures. Here, we will focus on the specifics of OpenGL.

We will cover the lighting calculation contributors in the following order: material properties for objects, then parameters for individual lights, and finally global lighting parameters.

Whilst this is called an exercise sheet, it is more of an explanation sheet and the exercise is to work through it experimenting with the parameters as you go along. There are a couple of exercises near the end.

2. The program

The program makes use of a set of classes to produce a display containing two spheres, some objects representing the world coordinate axes, and a small white sphere representing the light source, the results of which are illustrated in Figure 1. M1.java is the main class containing the program code for the interface and M1Scene is the class where the objects are drawn. M1.java also contains the code to deal with mouse input to control the camera, which is represented as a separate class in Camera.java. The movement of the mouse in x and y controls the movement of the camera around a virtual sphere centred on the world origin. Holding the mouse right button down while moving the mouse up and down changes the radius of the virtual sphere.

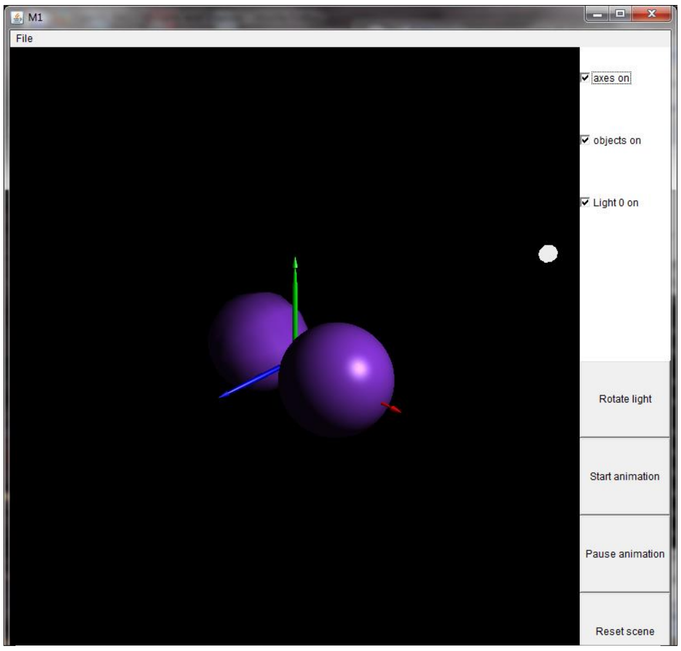

Figure 1. The initial display for M1.java

A separate Light class is used to control lighting parameters, and a separate class is also used to draw the Axes for the world coordinate system. Instances of the Light and Axes classes are created in the M1Scene class constructor. An instance of the Camera class is created in the M1 class and passed as a parameter to the M1Scene class when it is created – see contents of method M1.init().

The method M1Scene.displayObjects() displays two copies of `glutSolidSphere(...)`, each with radius 1.0, at positions (–1,0,0) and (1,0,0) respectively. The second sphere that is drawn has a higher geometric resolution than the first (as it is created with more slices and stacks). This makes it look much smoother than the first sphere. The rotate button rotates a light source around the y axis. The animation buttons start and stop continuous rotation of the light source.

Note that the M1.init() method includes the line gl.glShadeModel(GL2.GL_SMOOTH), which sets the OpenGL state to ensure that when lighting calculations are done at vertices using the normals at the vertices (which are part of the glut objects' definitions), the results are then smoothly (Gouraud) interpolated over the face of a triangle by the GPU – see the lecture notes for a description of this interpolation process. The next exercise sheet will say a little more about data structures and normals.

Over the next few sections we'll focus on the M1Scene and Light classes. We'll start with the *material properties* of the two spheres, which are drawn in M1Scene.displayObjects().

3. Material properties

Once lighting is enabled (we assume this has been done and return to this topic later), the colour of an object is controlled using material properties, *not* glColor. The material properties of an object can be controlled using the parameters shown in Table 1.

Parameter	**Default value**	**Meaning**
GL_AMBIENT	(0.2, 0.2, 0.2, 1.0)	Ambient colour
GL_DIFFUSE	(0.8, 0.8, 0.8, 1.0)	Diffuse colour
GL_AMBIENT_AND_DIFFUSE		Used to set both ambient and diffuse colour to same values
GL_SPECULAR	(0.0, 0.0, 0.0, 1.0)	Specular colour
GL_SHININESS	0.0	Specular exponent
GL_EMISSION	(0.0, 0.0, 0.0, 1.0)	Emissive colour

Table 1: Parameters for the OpenGL function `glMaterial*`

In M1Scene.displayObjects(), the material properties for the first sphere are set using the following program code:

```
float[] matAmbient = {0.2f, 0.08f, 0.32f, 1.0f};
float[] matDiffuse = {0.5f, 0.2f, 0.8f, 1.0f};
float[] matSpecular = {1.0f, 1.0f, 1.0f, 1.0f};
float[] matShininess = {64.0f};
float[] matEmission = {0.0f, 0.0f, 0.0f, 1.0f};

// use glMaterialfv. There is no glMaterialdv
gl.glMaterialfv(GL2.GL_FRONT, GL2.GL_AMBIENT, matAmbient, 0);
gl.glMaterialfv(GL2.GL_FRONT, GL2.GL_DIFFUSE, matDiffuse, 0);
gl.glMaterialfv(GL2.GL_FRONT, GL2.GL_SPECULAR, matSpecular, 0);
gl.glMaterialfv(GL2.GL_FRONT, GL2.GL_SHININESS, matShininess, 0);
gl.glMaterialfv(GL2.GL_FRONT, GL2.GL_EMISSION, matEmission, 0);

// Now draw the first sphere
```

(a) Ambient and diffuse properties

GL_AMBIENT and GL_DIFFUSE affect the colours of the ambient and diffuse light reflected by an object. Ambient reflectance affects overall colour and is most noticeable where an object receives no direct illumination. (If the screen objects appear a little dark, increase the values in matAmbient in the code listed above, so that the overall brightness of objects is increased.) Diffuse reflectance is affected by the angle of incident light relative to the surface normal. The ambient and diffuse parameters can be set to the same value at the same time using GL_AMBIENT_AND_DIFFUSE.

Currently the two spheres have the same material properties, as no new values are set before rendering sphere 2. Uncomment the following lines before the display of the second sphere in M1Scene.displayObjects(…) (where it says UNCOMMENT... in the method). This will change the second sphere to a mustard orange colour.

```
float[] matAmbientDiffuse = new float[4];
matAmbientDiffuse[0] = 0.8f;
matAmbientDiffuse[1] = 0.5f;
matAmbientDiffuse[2] = 0.2f;
matAmbientDiffuse[3] = 1.0f;
gl.glMaterialfv(GL2.GL_FRONT, GL2.GL_AMBIENT_AND_DIFFUSE, matAmbientDiffuse, 0);
```

(Note: glMaterial**f**v is used as there is no glMaterial**d**v.)

(b) Specular properties

The specular reflection produces highlights and depends on the location of the viewpoint. Replace the above lines on ambient and diffuse properties with the following to change the effect the material has on specular light. (By replacing the ambient and diffuse settings above, the two spheres will be the same ambient/diffuse colour again, so that the specular difference is then more noticeable.) You should see that the specular highlight on the second sphere is red in colour (which would be the case if a red light source was used. It may look a little unusual, as we are used to seeing white specular highlights on objects, since indoor lights could be considered as mostly white).

```
matSpecular[0] = 1.0f;
matSpecular[1] = 0.0f;
matSpecular[2] = 0.0f;
matSpecular[3] = 1.0f;
gl.glMaterialfv(GL2.GL_FRONT, GL2.GL_SPECULAR, matSpecular, 0);
```

The size and brightness of the specular highlight can be controlled using the shininess parameter (range [0.0..128.0]). A higher value produces a smaller and brighter highlight. Add the following lines after setting the specular parameters. You should see that the specular highlight on the second sphere becomes larger.

```
matShininess[0]=10.0f;
gl.glMaterialfv(GL2.GL_FRONT, GL2.GL_SHININESS, matShininess, 0);
```

(c) Emission properties

GL_EMISSION can be used to make it appear an object is giving off light of that colour. Replace the above lines on specular properties (in sections (b) and (c)) with the following:

```
matEmission[1] = 0.3f;
gl.glMaterialfv(GL2.GL_FRONT, GL2.GL_EMISSION, matEmission, 0);
```

You should see that the second sphere appears to have a greenish tinge. Most real objects don’t emit light so this will seem a little strange. The parameter is usually used in conjunction with a light source to model, say, a lamp bulb. A light source would be positioned in a scene. Then a geometric object would be positioned at the same location as the light source. The light source would still illuminate the rest of the scene. The geometric object would have it emission properties set and then when it was drawn it would look like it was a real light object emitting light. I have used this technique for the small white sphere that is orbiting the scene when you press the ‘start animation’ button.

Now delete the lines you have added, so that the M1Scene.displayObjects() method is back to its original starting point.

(d) The effect of geometric resolution

A further aspect to note in this section is the poor geometric resolution of the first sphere in contrast to the second sphere. This has a noticeable effect on the shading, as can be seen by the way the specular highlight interacts with the underlying geometry. The specular calculation is only done at vertices and is then interpolated across triangles (see lecture notes for a description of Gouraud interpolative shading, using the Phong lighting model at the vertices). Increasing the geometric resolution of the sphere will thus ameliorate the effect, at the expense of rendering more polygons.

4. Lighting properties

We can have 8 light sources (or more, depending on the hardware) in a scene: GL_LIGHT0, GL_LIGHT1, ..., GL_LIGHT8. The parameters for individual lights are shown in Table 2. Lighting is turned on by enabling GL_LIGHTING and the individual lights that are required. In our program, lighting is enabled in method M1.init() – this init() method is usually where you would initialise the OpenGL state. In addition, we have enabled GL_LIGHT0. In fact this light is enabled by default in OpenGL when lighting is enabled, but I prefer to emphasize this by explicitly stating it. No other lights are enabled. We will use the Light class that I have created to control lighting.

An instance of the Light class is created in the constructor for M1Scene:

```
  light0 = new Light(GL2.GL_LIGHT0);
```

The OpenGL parameters that can be used to control this light are given in Table 2. We will consider two kinds of light, positional lights and directional lights. A positional light is a light at a particular position in the scene, shining in all directions (i.e. omnidirectional). A directional light is, in essence, at infinity and shining in a particular vector direction (the negative of the value specified). Since it is at infinity it is a constant, i.e. all light in the space is shining in the same vector direction.

Parameter	**Default value**	**Meaning**
GL_AMBIENT	(0.0, 0.0, 0.0, 1.0)	Ambient intensity of light
GL_DIFFUSE	(1.0, 1.0, 1.0, 1.0) or (0.0,0.0,0.0,1.0)	Diffuse intensity of light (default for light 0 is white, and black for other lights)
GL_SPECULAR	(1.0, 1.0, 1.0, 1.0) or (0.0,0.0,0.0,1.0)	Specular intensity of light (default for light 0 is white, and black for other lights)
GL_POSITION	(0.0, 0.0, 1.0, 0.0)	(x,y,z,w) position of light. If w is 0, then it is a directional light (at infinite distance in the given vector direction, with all the light shining back to the world in the negative of that direction). If w is 1, then it is a positional light (at the given position in the scene, and emitting in all directions).
GL_SPOT_DIRECTION	(0.0, 0.0, -1.0)	(x,y,z) direction of spotlight
GL_SPOT_EXPONENT	0.0	Spotlight exponent
GL_SPOT_CUTOFF	180.0	Spotlight cutoff angle
GL_CONSTANT_ATTENUATION	1.0	Constant attenuation factor
GL_LINEAR_ATTENUATION	0.0	Linear attenuation factor
GL_QUADRATIC_ATTENUATION	0.0	Quadratic attenuation factor

Table 2: Parameters for the OpenGL function `glLight*`

When the Light instance is created, one of two constructors can be called, as shown in the following code. In the M1Scene constructor, the default Light constructor is called, which uses a default position of (4, 3, 0, 1) for the positional light and sets this to a white light.

```
public class Light implements Cloneable {

  public static final float[] DEFAULT_POSITION = {4.0f,3.0f,0.0f,1.0f};
  // (x,y,z,w) position of light.
  // If w is 0, then it is a directional light (at infinite distance in the given vector direction, shining back to the world origin)
  // If w is 1, then it is a positional light (at the given position in the scene, and emitting in all directions).
  public static final float[] WHITE_LIGHT = {1.0f,1.0f,1.0f};
  public static final float[] DEFAULT_AMBIENT = {0.1f,0.1f,0.1f};

  private int index;
  private float[] position;
  private float[] ambient;
  private float[] diffuse;
  private float[] specular;
  private boolean switchedOn;

  /**
   * Constructors
   */
  public Light(int i) {
    this(i, DEFAULT_POSITION, DEFAULT_AMBIENT, WHITE_LIGHT, WHITE_LIGHT, true);
  }

  public Light(int i, float[] position) {
    this(i, position, DEFAULT_AMBIENT, WHITE_LIGHT, WHITE_LIGHT, true);
  }

  public Light(int i, float[] position, float[] ambient, float[] diffuse,
               float[] specular, boolean on) {
    index = i;
    this.position = position.clone();
    this.ambient = ambient.clone();
    this.diffuse = diffuse.clone();
    this.specular = specular.clone();
    switchedOn = on;
  }

// more program code...
```

As the default is a positional light, the program draws a small sphere at the position of the light source. This sphere receives no lighting from the light since the light is inside it, and thus as far as the light is concerned all the sphere’s faces are pointing away from it. Instead, the emission material properties of the sphere are used to give it a bright appearance, so it looks like it is the light source.

Inside method Light.use(), the relevant OpenGL commands are called to set the lighting properties (stored as state variables on the OpenGL context):

```
// There is no glLightdv, so use glLightfv
  gl.glLightfv(index, GL2.GL_POSITION, position, 0);
  gl.glLightfv(index, GL2.GL_AMBIENT, ambient, 0);
  gl.glLightfv(index, GL2.GL_DIFFUSE, diffuse, 0);
  gl.glLightfv(index, GL2.GL_SPECULAR, specular, 0);
```

Then any object *subsequently* drawn will have these lighting characteristics used when the GPU calculates the lighting for each triangle. (Note: it is important that the lighting is set up before drawing any objects.)

The rotation of the light source around the y axis is controlled by an M1Scene attribute called rotate which is incremented each time the scene is redrawn. This attribute is then used to transform the light position when the light is drawn in the method M1Scene.doLight0():

```
private void doLight0(GL2 gl) {
  gl.glPushMatrix();
    gl.glRotated(rotate,0,1,0);
    light0.use(gl, glut, true);
  gl.glPopMatrix();
}
```

We can transform the light since it is has a position attribute. Thus the transform changes that position.

(a) A second light

Let's add a second light. In M1Scene, add a second light attribute:

```
private Light light0, light1;
```

Then, in the M1Scene constructor add the relevant lines after light0 is created so as to create a second light, which is positioned 3 units up the y axis and 4 units along the z axis:

```
public M1Scene(GL2 gl, Camera camera) {
  light0 = new Light(GL2.GL_LIGHT0);  // Create a default light

  float[] position = {0,3,4,1};
  light1 = new Light(GL2.GL_LIGHT1, position);

  this.camera = camera;
  axes = new Axes(2.2, 1.8, 1.6);
}
```

Then, add the following method M1Scene.doLight1() after M1Scene.doLight0():

```
private void doLight1(GL2 gl) {
  light1.use(gl, glut, true);
}
```

Finally, after the line doLight0(gl); in M1Scene.render(), add the line:

```
doLight1(gl);
```

You should now have two highlights on each sphere, as illustrated in Figure 2. The highlight on the low resolution sphere will be difficult to see because of the low geometric resolution. You could increase the resolution by increasing the number of sphereslices and spherestacks used to draw it.

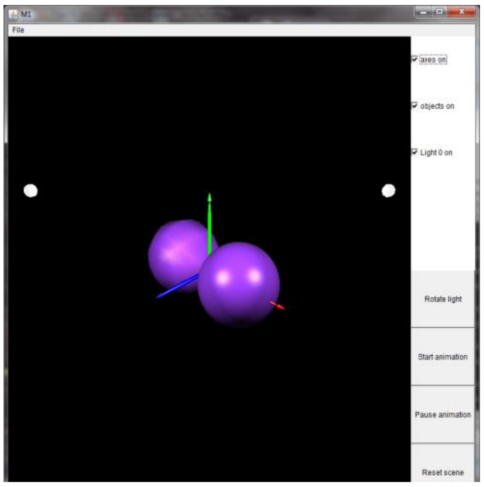

Figure 2. Two position lights

Currently, only light0 is animated. To see both lights animating, change the method M1Scene.doLight1() to the following:

```
  private void doLight1(GL2 gl) {
    gl.glPushMatrix();
      gl.glRotated(rotate,0,1,0);
      light1.use(gl, glut, true);
    gl.glPopMatrix();
  }
```

We can also turn one of the lights into a directional light. In the M1Scene constructor, change the position of light1 to:

```
float[] position = {0,3,4,0};
```

If you now run the program you will see the result in Figure 3. Here I have used a white line to indicate that a positional light is shining from 'infinity' in the stated vector direction back towards the world (i.e. all light directions in the world are constant). In practice we wouldn't want to see this line (nor, probably, the small sphere that is used to represent a positional light). To turn off the display of these geometric objects, the calls to Light.use() need to be changed in M1Scene.doLight0() and M1Scene.doLight1() to the following:

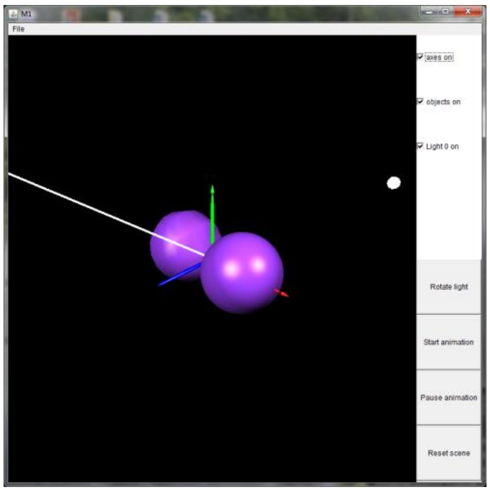

Figure 3. Two lights, one direction light and one position light

```
light0.use(gl, glut, false);
```

and

```
light1.use(gl, glut, false);
```

where the third parameter is now set to false to state that the geometric objects should not be drawn to show where the lights are. Of course this does not affect the functioning of the lights themselves.

5. A spotlight

Now we'll change light1 into a spotlight, as illustrated in Figure 4 (where the geometric resolution of both spheres has been set to a high value to show the spotlight to its best effect – you should increase the geometric resolution of sphere 1 to better see the effects of the spotlight, i.e. change the variables sphereslices and spherestacks in M1Scene.render()).

A new version of the Light class is required for the spotlight effect. Replace the version that you have been using so far with the version in the subfolder called spotlight. Just copy the version of Light.java from the folder spotlight to the folder you are working in to replace the existing Light.java class.

I'll now explain how the spotlight works. To create a spotlight, we must use a positional light, so the position of light1 must have 1 as its last parameter, e.g. {0, 3, 4, 1}. Some extra attributes have been added to the new Light class as well as extra or rewritten methods:

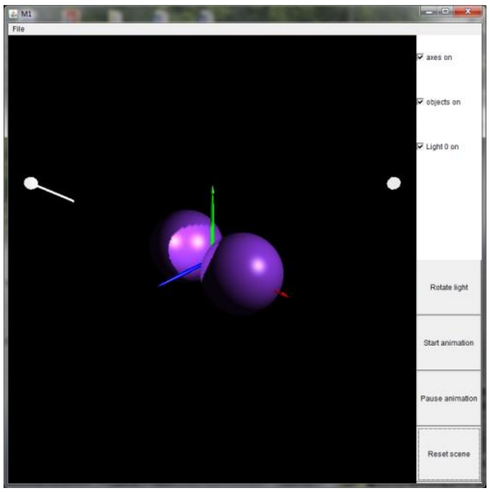

Figure 4. Light 1 is now a spotlight, as indicated by the sphere and attached short line showing the direction it is the spotlight is pointing in

```
private boolean spotlight = false;
private float[] direction;
private float angle;

/// Constructors are here

public void makeSpotlight(float[] direction, float angle) {
  spotlight = true;
  this.direction = direction.clone();
  this.angle = angle;
}
```

The standard constructors are still used in the Light class.

You need to make the following change:

Within the M1Scene constructor, the following lines need to be added:

```
  float[] position = {0,3,4,1};
  light1 = new Light(GL2.GL_LIGHT1, position);
  float[] direction = {0f,-3f,-4f};  // direction from position to origin
  light1.makeSpotlight(direction, 10f);
```

This turns a standard light into a spotlight using the new method Light.makeSpotlight(). This will define a spotlight at the light1 position, pointing in the direction (0,-3,-4), which is back to the world origin, with a dramatic 10 degree cutoff angle. (We could point the spotlight in any direction, but we must be careful to point it in a direction that illuminates part of the scene.)

The following methods are already part of the new Light class. They are given here just so that you can see how the new Light class works to incorporate the spotlight effect.

```
public void use(GL2 gl, GLUT glut, boolean show) {
  if (switchedOn) {
    gl.glEnable(index);
    // There is no glLightdv, so use glLightfv
    gl.glLightfv(index, GL2.GL_POSITION, position, 0);
    gl.glLightfv(index, GL2.GL_AMBIENT, ambient, 0);
    gl.glLightfv(index, GL2.GL_DIFFUSE, diffuse, 0);
    gl.glLightfv(index, GL2.GL_SPECULAR, specular, 0);
    if (spotlight) {      // NEW lines here
      gl.glLightf(index, GL2.GL_SPOT_CUTOFF, angle);
      gl.glLightfv(index, GL2.GL_SPOT_DIRECTION, direction, 0);
    }
    if (show) {
      if (position[3] == 1) {
        displayPosition(gl, glut);
        if (spotlight) displaySpotlight(gl, glut);
      }
      else {
        displayDirection(gl);
      }
    }
  }    else gl.glDisable(index);
}
```

```
private void displaySpotlight(GL2 gl, GLUT glut) {
  gl.glDisable(GL2.GL_LIGHTING);
  gl.glLineWidth(4);
  double x = direction[0];
  double y = direction[1];
  double z = direction[2];
  double mag = Math.sqrt(x*x+y*y+z*z);
  x = 0.5*x/mag;
  y = 0.5*y/mag;
  z = 0.5*z/mag;
  x += position[0];
  y += position[1];
  z += position[2];
  gl.glColor3d(1,1,1);
  gl.glBegin(GL2.GL_LINES);
    gl.glVertex3d(position[0], position[1], position[2]);
    gl.glVertex3d(x,y,z);
  gl.glEnd();
  gl.glLineWidth(1);
  gl.glEnable(GL2.GL_LIGHTING);
}
```

Now run the main program M1.java:

Running the main program will now produce a spotlight effect on the spheres, as illustrated in Figure 4. In order to make this show up better on the low geometric resolution sphere, the values of the variables sphereslices and spherestacks variables in method M1Scene.displayObjects() should be increased. Try experimenting with different values to see the effect on the spotlight profile.

6. Global properties to change the lighting

There are some OpenGL global properties that have an effect on the lighting. Typically, we leave these as the default values. However, for completeness, they are listed in Table 3, which gives the relevant parameters for the glLightModel* function. For further information on these parameters, read chapter 5 of the 'Red Book' (see the module web site for an online version).

Parameter	**Default value**	**Meaning**
GL_LIGHT_MODEL_AMBIENT	(0.2, 0.2, 0.2, 1.0)	Ambient RGBA intensity of the entire scene
GL_LIGHT_MODEL_LOCAL_VIEWER	0.0 or GL_FALSE	How specular reflection angles are computed
GL_LIGHT_MODEL_TWO_SIDE	0.0 or GL_FALSE	Choose between one-sided or two-sided lighting
GL_LIGHT_MODEL_COLOR_CONTROL	GL_SINGLE_COLOR	Whether specular colour is calculated separately from ambient and diffuse

Table 3: Parameters for the OpenGL function `glLightModel*`

7. Exercise 1

Change the display of the two spheres into a stack of three spheres (similar to the stack of cubes on last week's exercise sheet. Make each sphere a different colour, with different material properties. Then add a third light into the scene. You decide where to position it and what kind of light it is. You may need to adjust the parameters of the light sources if the bright areas start to saturate, i.e. the bright white highlight spreads over a larger area.

8. Exercise 2: A simple scene

Create the scene in Figure 5, where a spotlight is emitting a light from the top of a pole towards a ground plane. The GLUT library contains a cone and a sphere to make a model that looks like a spotlight. An elongated sphere could be used for the pole which the spotlight sits on. For the ground plane, you can scale a sphere into a nearly flat shape for a circular floor plane. (Question: why wouldn't a simple scaled cube suffice for the ground plane?)

This exercise will test your understanding of what you have learnt today (as well as your use of transformations from previous exercise sheets) and I'll provide a full worked example in a week or so.

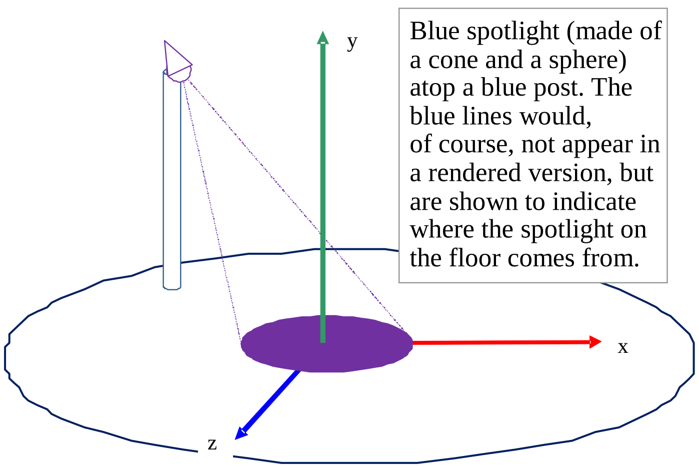
y
x
z
Blue spotlight (made of a cone and a sphere) atop a blue post. The blue lines would, of course, not appear in a rendered version, but are shown to indicate where the spotlight on the floor comes from.

Figure 5: The set-up for the scene.