Lights and Materials in OpenGL

This material is not difficult, but the documentation for it is rather scattered and not well-explained. I'll try to gather the most important information here.

Lights. The idea here is that you first enable lighting in general, then you enable specific lights and given them properties. OpenGL numbers the lights as GL_LIGHT0, GL_LIGHT1, and so forth.
There are three kinds of lights in OpenGL:

- directional lights, have direction but no position (or are infinitely far away)
- point lights have a specific location in space
- spot lights have a location and emit light in a cone from this position.

The differences between these depend upon which properties we set for a particular light.

We enable lighting with the single line

```
gl.glEnable(GL2.GL_LIGHTING);
```

Once you have enabled lighting, calls to glColor no longer have any effect and you need to set material properties for each surface.

We turn on light#i with gl.glEnable(GL2.GL_LIGHTi), as in

```
gl.glEnable(GL2.GL_LIGHT0);
```

Light0 has different default properties than the other lights, but I suggest that you ignore the defaults and set any properties you want to use.

The properties of a light are its ambient, diffuse and specular components, its position, direction, and the two spotlight properties: spot_direction, and spot_cutoff. All of these are set with the gl.glLightf( ) and gl.glLightfv( ) functions. Use gl.glLightfv( ) when the parameter being set needs a vector value; glLightf( ) when it needs a single float. They take the forms

```
gl.glLightf(light#, property, value);
gl.glLightfv(light#, property, array, 0);
```

The last argument on glLightfv is an offset for where to start in the array; you will probably alway make that 0. The light# is GL2.GL_LIGHT0, GL2.GL_LIGHT1, and so forth. The properties are GL2.GL_AMBIENT, GL2.GL_DIFFUSE, GL2.GL_SPECULAR, GL2.GL_POSITION, GL2.GL_SPOT_DIRECTION, and GL2.GL_SPOT_CUTOFF. (Note that there is no parameter for directional lights; we'll get to that momentarily.)

For example, we can set the diffuse color of a light #2 to yellow with

```
float [ ] yellow = {1, 1, 0, 1};
gl.glLightfv(GL2.GL_LIGHT2, GL2.GL_DIFFUSE, yellow, 0);
```

You should use rgba format for light colors with the alpha channel set to 1.

For point lights you should set the ambient, diffuse and specular components of the light and the light's position. Positions are specified in homogeneous coordinates (i.e, 4 coordinates with the 4th component 1). For a directional light set the GL_POSITION property, with the 4th component of the position being 0; this is a signal to OpenGL to treat this as a direction rather than a position. For a spot light, set GL_SPOT_DIRECTION to a 3D vector and GL_SPOT_CUTOFF to an angle that represents the aperture of the cone of light; for this use glLightf( ) since you are setting a single float and not a vector.

For example, the following code would create single white point light at position (30, 2, 5):

```
public void init(GLAutoDrawable drawable) {
    <startup for OpenGL omitted?

    float [ ] white = {1, 1, 1, 1};
    float [ ] none = {0, 0, 0, 1};
    float [ ] pos = {30, 2, 5, 1};

    gl.glEnable(GGL2.GL_LIGHTING);
    gl.glEnable(GL2.GL_LIGHT0);
    gl.glLightfv(GL2.GL_LIGHT0, GL2.GL_AMBIENT, none, 0);
    gl.glLightfv(GL2.GL_LIGHT0, GL2.GL_DIFFUSE, white, 0);
    gl.glLightfv(GL2.GL_LIGHT0, GL2.GL_SPECULAR, white, 0);
    gl.glLightfv(GL2.GL_LIGHT0, GL2.GL_POSITION, pos, 0)

    <rest of initialization code omitted>
```

Materials. The current surface material is part of the OpenGL state. We create a material and that is assigned to each piece of geometry we create until a new material is created. We do this by setting the ambient, diffuse and specular properties of the surface. There is also a shininess property that is related to the Phong exponent used in the specular reflection calculation. You can also have surfaces emit light; this is added to the light reflected by the surface, but isn't used in the light reflection model for any other surface.

The glMaterialf( ) and glMaterialfv( ) functions set the material properties. Both of these require you to specify whether you are setting the properties for the front or the back of the surface. There is a story here. The default lighting model only works with the front of a surface, so the back attributes are ignored. It is possible to switch to a 2-sided lighting model, which will allow the front and back surfaces to be different. I'm not going to deal with that here, but you still need to specify that you are working with the front of the surface. A typical parameter is

```
gl.glMaterialfv(GL2.GL_FRONT, GL2.GL_DIFFUSE, color, 0);
```

where color is an array of 4 color components. As with lighting, the 3 color properties are GL_AMBIENT, GL_DIFFUSE, and GL_SPECULAR. The shininess property is GL_SHININESS and you set it with glMaterialf( ), since its value is a single float, as in

```
gl.glMaterialf(GL2.GL_FRONT, GL2.GL_SHININESS, 32.0);
```

The light reflection model requires a normal vector; this needs to be specified explicitly. You might expect OpenGL to calculate a surface normal from a cross-product of edges, but this is not done. The front face of a surface is the face you would see if you were standing at the head of the normal vector looking back at the tail. The default normal vector is <0, 0, 1>, regardless of how your geometry is defined. Like surface materials, the normal is part of the OpenGL state – a normal stays in effect until you specify a new one, and the current normal is attached to geometry when it is created. You set the normal with either

```
gl.glNormal3f(a, b, c)
```

or

```
gl.glNormal3fv( vec, 0 )
```

where vec is an array with 3 components. For example, you make the normal point in the direction of the x-axis with either

```
glNormal3f(1, 0, 0)
```

or

```
float [ ] norm = {1, 0, 0};
gl.glNormal3fv( norm, 0 )
```

Normals are created automatically for quadrics, but you should specify whether you want the outward-pointing normal or the inward-pointing normal. If quad is a pointer to a quadric object, then

```
glu.gluQuadricOrientation(quad, GLU.GLU_OUTSIDE);
```

selects the outward-pointing normal and

```
gu.gluQuadricOrientation(quad, GLUGLU_INSIDE);
```

reverses the normal to make inward-pointing normals. You need this if, for example, you want to have a scene inside a sphere or cylinder.

See the following demo programs:

Gouraud.py
Spot.py
ThreePlanes.py
Sphere.py
Cylinder.py