CSC418 Tutorial 6 – OpenGL Lighting and Shading
Prepared by Kevin Forbes

Types of lighting implemented in OpenGL:

Ambient:
No source point; affects all polys independent of position, orientation and viewing angle; used as a 'fudge' to approximate 2nd order and higher reflections

Diffuse:
Light scattered in all directions after it hits a poly; dependant upon incident angle

Specular:
'Shininess' ; dependant upon incident and viewing angles

Emissive:
Makes a poly appear as though it is glowing; does not actually give off light

Implementation Specifics

- Two kinds of parameters, lighting and material
- Material properties: state variables, can be changed as you draw different polys in a scene
- Light properties: parameters indexed to light numbers; Opengl can use up to 8 lights; light positions are affected by the modelview matrix stack

The Full OpenGL lighting Equation (stolen shamelessly off the internet):

$$
\begin{aligned}
&vertex\_color = \\
&emission_{material} + \\
&ambient_{lightmodel} * ambient_{material} + \\
&\sum_{i=0}^{n-1} \left(\frac{1}{k_c d + k_l d + k_q d^2}_i * \left(\begin{array}{c} ambient_{material} + \\ \max(L \bullet n, 0) * diffuse_{light} * diffuse_{material} + \\ \max(s \bullet n, 0)^{shininess} * specular_{light} * specular_{material} \end{array} \right)_i \right)
\end{aligned}
$$

It might be a good idea to go through each term, specifying which terms reflect the different parts of the model. The original tutorial notes don't cover emmisive light or attenuation, but you might want to mention them briefly.

Shading models:

Shading refers to how the lighting equations are applied to a rasterized poly. OpenGL supports two shading models:

GL_FLAT: Lighting is evaluated once per poly, and the resulting colour value is used for the whole thing.

GL_SMOOTH: Lighting is evaluated at each vertex, and pixel colours are linearly interpolated across polys. This is more expensive, but it looks much better.

OpenGL uses the Phong lighting model at vertices, but has no built-in support for Phong shading. Modern programmable lighting hardware can implement full phong shading (and much more!), but you have to do this yourself.

Normals:

The lighting equations depend upon normals, so we have to provide them. The current normal is specified with a call to a glNormal* function, and will be applied to every subsequent vertex. Normals should be of unit length, or the lighting equations will not work correctly. This can be a problem, because normals are affected by any scaling done in the matrix stack. You must either re-normalize the normals as a pre-processing step, or enable GL_NORMALIZE (which is computationally expensive).

Programming Example:

This code would typically be placed with your opengl initialization code

```
//set the global lighting / shading params
glShadeModel(GL_SMOOTH); // or GL_FLAT
glEnable(GL_NORMALIZE); //or not
glEnable(GL_LIGHTING);

//set the global ambient light
GLfloat ambient = {.2,.2,.2,1};
glLightModelfv(GL_LIGHT_MODEL_AMBIENT, globalAmb);
```

This code sets up a light and enables it

```
GLfloat diffuse[] = {1,0,0,1};
GLfloat ambient[] = {.5,0,0,1};
GLfloat specular[] = {1,1,1,1};

glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse);
glLightfv(GL_LIGHT0, GL_AMBIENT, ambient);
glLightfv(GL_LIGHT0, GL_SPECULAR, specular);
```

```
glEnable(GL_LIGHT0); //enable the light

//set light position (see discussion later for details)

// set last term to 0 for a spotlight (see chp 5 in ogl prog guide)
Glfloat lightpos[] = {1,1,1,1};
glLightfv(GL_LIGHT0,GLPOSITION, lightpos);
```

This code sets a simple material property

```
GLfloat ambient[] = {.5,0,0,1};
GLfloat specular[] = {1,1,1,1};

//can set params for front and back separately (GL_BACK, GL_FRONT_AND_BACK)
glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, ambient);
glMaterialfv(GL_FRONT, GL_SPECULAR, ambient);
```

Shaded polygons can now be drawn using the usual method, so long as normals are specified correctly.

Programming Hints: (if you have time)

Using glColor* to change material properties*:*

This simplifies changing material properties, and can be helpful when drawing using vertex arrays.

```
glColorMaterial( GL_FRONT, GL_AMBIENT_AND_DIFFUSE);
glEnable(GL_COLOR_MATERIAL);
```

Implementing Shading groups

When using smooth shading, the usual way to calculate a vertex normal is to average the normals of all of the faces that it is a part of. This normal is then used for all polygons that use the vertex. This looks good for smooth geometries, but terrible for things like cubes. To make a hard edge in smooth shading mode, specify the normals for each vertex along the edge separately for each polygon, and set them equal to the face normals.

Controlling Light Positions (stolen from OpenGL FAQ at opengl.org)

How can I make my light position stay fixed relative to my eye position? How do I make a headlight?

You need to specify your light in eye coordinate space. To do so, set the ModelView matrix to the identity, then specify your light position. To make a headlight (a light that appears to be positioned at or near the eye and shining along the line of sight), set the ModelView to the identity, set the light position at (or near) the origin, and set the direction to the negative Z axis.

How can I make my light stay fixed relative to my scene?

As your view changes, your ModelView matrix also changes. This means you'll need to respecify the light position, usually at the start of every frame. A typical application will display a frame with the following pseudocode:

Set the view transform.
Set the light position
Send down the scene or model geometry.
Swap buffers.

How can I make a light that moves around in a scene?

Again, you'll need to respecify this light position every time the view changes. Additionally, this light has a dynamic modeling transform that also needs to be in the ModelView matrix before you specify the light position. In pseudocode, you need to do something like:

Set the view transform
Push the matrix stack
Set the model transform to update the light’s position
Set the light position
Pop the matrix stack
Send down the scene or model geometry
Swap buffers.