ICS 186A Winter 2003: Assignment 5

Due: Feb 14, 2003

WARNING:

The assignment is long, not necessarily difficult. So start early. Do not wait till next week.

Goal of the project:

Implement your own lighting and compare it with OpenGL lighting.

What is the lighting equation?

How OpenGL models the light is slightly different from what we discussed in the class. Again, the physical parameters are the same. But the OpenGL has more parameters to tune the lighting so that the people who model can have more freedom to get what they *want* (rather than what *is*).

Following are the changes: OpenGL has a global ambient light description, as we discussed in the class. But in addition, it also allows "ambient" component for every light.
Further, material can also have an "emission" component (such that it can behave as a light source). The ambient (k_a), diffuse (k_d) and specular (k_s) constants are not available in OpenGL. Instead, they are combined with the material properties. Instead of one (RGB) color for the light, OpenGL allows for the same light different (RGB) colors for ambient, diffuse, and specular components. You can probably model diffuse light like fluorescence light, or a specular light like laser rays using this modeling parameters. OpenGL uses the "half-vector" approach for specular component computation, instead of finding the angle of the viewer from the reflection vector.

The OpenGL equation for lighting is:
Vertex color = (material emission) + (global ambient)*(material ambient) +
□ $(1/(k_1 + k_2 d + k_3 d^2))$ * [(light ambient)*(material ambient) + (max{L.N, 0})*(light diffuse)*(material diffuse) + $(\max\{H.N,0\})^n$ *(light specular)*(material specular)]

H is the unit half vector between light and viewing directions from the vertex.
N is the normal vector. The "max" functions will make sure that the light is not shining in the back-side of the face.

How do you let OpenGL know of the parameters required in the lighting equation?

The material parameters are given using the glMaterialfv command. Check man pages of this command. Material's and light's ambient, diffuse and specular components are given as an array of three floats (R, G, B).
For example,
Glfloat model_ambient[]={0.2, 0.3, 0.8, 1.0);
glMaterialfv(GL_FRONT, GL_AMBIENT, model_ambient);
would set the front facing polygon's ambient component as RGB={0.2,0.3,0.8}. The last parameter is alpha=1.0. Ignore this parameter for this assignment and set it to 1.0 by default. We will discuss alpha when we discuss transparency. Instead of GL_AMBIENT, you can use GL_DIFFUSE, GL_AMBIENT_AND_DIFFUSE, GL_SPECULAR, GL_EMISSION and GL_SHININESS (for specular exponent n) to set other material properties. Specular exponent is just one float, and not a three-float array.

The light parameters are given using the glLightfv command. The above constants like GL_AMBIENT etc. (except shininess) can be used to set various properties of the light. Additional properties of light include position, and the attenuation constants k_1, k_2, and k_3. These properties are set using the parameters GL_POSITION, GL_CONSTANT_ATTENUATION, GL_LINEAR_ATTENUATION, and GL_QUADRATIC_ATTENUATION respectively.

For example,
Glfloat light_position[]={0.2, 0.3, 0.8, 1.0);
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
sets the position of light "GL_LIGHT0" to light_position. Note the homogenous coordinate. You can use at most eight lights: GL_LIGHT0 to GL_LIGHT7. You can enable (switch on) lights using glEnable(GL_LIGHT0) command. But any lighting would work only if you had used glEnable(GL_LIGHTING) command.

For further details, read the red book, Chapter 5 on lighting.

What is to be done for this assignment?

Implement the lighting equation and compute the color at every vertex. Also use OpenGL lighting. Use a key input, 'I' and/or 'i' to use your implementation of lighting and 'O' and/or 'o' to switch to OpenGl's implementation of lighting. Observe if there is any difference in the image. In both cases, use the same parameters. Remember you need the normal vector at each vertex to compute the lighting. Compute this normal vector using your earlier assignment. Use glShadeModel(GL_SMOOTH) to render it as smooth shading (Gouraud shading).

Use this lighting implementation on the ‘spinner’ assignment (two spheres). Use your previous assignment to move the viewing direction. Another interaction you have to provide, in addition to the user view point, is the interaction with the position of the light. Use a key input, ‘V’ and/or ‘v’ to move the viewpoint, ‘L’ and/or ‘l’ to move the light.

How do I implement?

You have to implement this assignment as follows. Implement a class for “Light” and a class for “Material”. In both these classes, have the following members: Glfloat ambient[], diffuse[], specular[]. In the “Light” class have Glfloat position[], k1, k2, k3. Further, in the “Light” class, have “int IsEnabled” to indicate if that light is switched on or not. In the “Material” class have Glfloat specular_exponent, normal[]. Ignore the global ambient component for this section of the assignment (see extra credit section for more details). In addition to these members, you can have any other additional member that you deem fit.

Implement “my_glLightfv” and “my_glMaterialfv” to set the members of above two classes. You can have multiple instances of the “Light” class to initialize multiple lights in the scene. For example, the function call to set the “Light” class member values is:
my_glLightfv(GL_LIGHT0, GL_AMBIENT, white_light);
The first parameter of “my_glLightfv” tells you which instance of the “Light” class you are modifying, and the second parameter tells you which member of that instance you are modifying, the third parameter tells you the value of that member.

You should be able to do something like

```
GLfloat light1_pos[] = {0.0, 10.0, 10.0, 0.0};
GLfloat white_light[] = {1.0, 1.0, 1.0, 1.0};
my_glLightfv(GL_LIGHT0, GL_AMBIENT, white_light );
my_glLightfv(GL_LIGHT0, GL_DIFFUSE, white_light );
my_glLightfv(GL_LIGHT0, GL_SPECULAR, white_light );
my_glLightfv(GL_LIGHT0, GL_POSITION, light1_pos );
my_glEnable(GL_LIGHTING );
my_glEnable(GL_LIGHT0 );
// Render objects
```

You can implement my_glLight and my_glMaterial the way you want such that it changes the values of the appropriate variables correctly. I would like it to be as modular as possible.

IMPORTANT:

You should use the above implementation in conjunction with your previous assignment. In other words, you should use your implementation of glTranslate, glRotate, etc.

Every vertex position, normal vector, and the position of the light have to be multiplied with the model-view matrix. (Only with the model view matrix, and not with the projection matrix.) These transformations (especially the scale and shear, if any) *may change the length of the normal vector. Remember to normalize this vector before using for lighting calculation.*

Homogenous representation of a point has the last coordinate w=1. Homogenous coordinate of a vector has the last coordinate w=0. (So for the normal vector…) Now, would a translation affect a vector?

Extra Credit:

1. Implement other features of OpenGL lights such as spot light effects. The associated parameters of glLight are GL_SPOT_DIRECTION, GL_SPOT_EXPONENT, GL_SPOT_CUTOFF. Refer the red book for a slightly modified lighting equation to take into account this light property.
2. Implement other features of OpenGL materials like GL_BACK, GL_FRONT_AND_BACK. To implement this, you have to instantiate the "Material" class, one for Front and another for Back. Now you can set different set of properties like ambient, diffuse etc. for the "Back" face. The given normal vector direction has to be reversed to get the normal vector for the "Back" face and the same lighting equation can be used with these new properties and normal vector.
3. Implement other lighting models like GL_LIGHT_MODEL_AMBIENT to account for global ambient light, GL_LIGHT_MODEL_LOCAL_VIEWER to take into account viewer in infinite distance, GL_LIGHT_MODEL_TWO_SIDE to light both sides. You should have implemented item 2 above to implement GL_LIGHT_MODEL_TWO_SIDE.

You should clearly document your implementation, extra cool features, user interface etc. in your README file. Do not leave anything to be "discovered" by the grader.