Winter 2016, CS 112
Programming Assignment 3
Submission Due: Feb 17, 2016, 11:59 PM PST

PROJECT GOAL: Implement your own lighting and compare it with OpenGL lighting.

1 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 OpenGL has more parameters to tune the lighting so that people who use it have more freedom to get what they *want* (rather than what real objects *are*).
Following are the changes: OpenGL has a global ambient light description, as we discussed in class. In addition, it also allows "ambient" component for every light. Further, material can also have an "emission" component (such that an object can be lit even when there is no other light source). The ambient (k_a), diffuse (k_d) and specular (k_s) constants are actually arrays of 4 constants each (for the RGBA channels). Instead of one color per light, OpenGL defines different colors, for the ambient, diffuse, and specular components. You can probably model diffuse light like fluorescence light, or a specular light like laser rays using these modeling parameters.
The OpenGL equation for lighting is:

Vertex color = (material emission) + (global ambient)*(material ambient) +
$\sum$ (1/(k_1+ k_2d + k_3d $_2$)) * [(light ambient)*(material ambient) + (max{L.N, 0})*(light diffuse)*(material diffuse) + (max{H.N,0})^n *(light specular)*(material specular)]

H=(L+V)/|L+V|, where L is the unit vector towards the light source, and V is the unit vector towards the eye. This is called the half-vector between L and V.
N is the normal vector. n defines the shininess. The "max" functions will make sure that the light is not shining in the back-side of the face. If this value goes beyond 1, it is clamped to 1 (lighting has been saturated at that point).

2 Passing Light Equation Parameters to OpenGL

The material parameters are given using the **glMaterialfv** command. Check manual pages of this command. Material's and light's ambient, diffuse and specular components are given as an array of four floats (R, G, B, A).
For example,
Glfloat model_ambient[]={0.2, 0.3, 0.8, 1.0);
glMaterialfv(GL_FRONT, GL_AMBIENT, model_ambient);
This will 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_SPECULAR, GL_EMISSION, GL_SHININESS (for specular exponent n) and GL_AMBIENT_AND_DIFFUSE 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. Pay attention to the difference between positional light source and directional light source, see red book chapter 5. Also pay attention to light direction and position transformation. Model view matrix will have effect on light positions and directions, see red book chapter 5 for more details. In most OpenGL configurations 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.
http://www.glprogramming.com/red/chapter05.html

3 Assignment Tasks

Implement the lighting equation and compute the color at every vertex. Also use OpenGL lighting.

- Use a key input, 'l/L' to toggle between your and OpenGL's lighting. Observe if there is any difference in the image. In both cases, use the same lighting parameters. When you're using OpenGL's lighting, simply enable lighting with the proper parameters. When you're using your own lighting, you have to disable lighting and give the appropriate color to each vertex.
- 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 't/T' to toggle between moving the viewpoint and moving the light position. The viewpoint interaction is the same as you have been using in the previous assignments. The lighting interaction works the same way, except now you will be transforming the light position instead of transforming an object. For this, you need to create a variable that tells whether the mouse if moving the point of view or the light. You will also want to keep different

angles for the light than those for the camera (i.e. create variables lightAngle1 and lightAngle2).

This will allow you to both move around in your scene as well as change the position of the light.

4 How to Do It

Download the package from the EEE dropbox and create a project. To run the program, copy the bunny.ply file to the project folder (The one your project file exists in). For Mac users, since relative path is not well supported by xcode, you may want to modify the path to absolute path or set command line argument to do it. Bunny.ply is a polygon file that defines vertices and faces for an object. You can try inverting normals, turn on/off light, modify the bunny by pressing on the keyboard. Refer the inputModule.cpp file for key controls.

As you may notice, the structure of this package is a little different from the previous two. The ply.cpp replaces the sceneModule.cpp to serve as both model definer/importer and renderer, except the setUserView() function is now in the inputModule.cpp for convenient reason. Geometry.cpp is a library includes linear algebra functions. Check if there is a useful function in it before you implement your own whenever you need to do geometry/linear algebra calculation.

The main task is to implement the Phong illumination model and apply the result to each vertex, in order to compare the result with that of OpenGL. You will be modifying the draw() function from the ply.cpp file to include your own per-vertex lighting. The code for retrieving all material and lighting parameters is already there. Use the transformed vertex and normal instead of the one specified in the parameters, because illumination should be done after model view transformation. The retrieved light positions and directions have already been transformed. All you need to do is to apply the lighting equation and put the result in output parameter GLfloat *color, which will be the color assigned to corresponding vertex. You should NOT be using any global variable for calculating the illumination, those local variables are sufficient. Spot light illumination effect and local viewer model is optional. See red book chapter 5 if you are interested. You are not required to modify other parts of the ply.cpp file.

In order to move light position, first apply view transformation in the same way you did for objects, and then apply rotation and/or translation to move the light, and then re-specify the new light position as the original light position. The light position will be transformed using current model view matrix. You NEED to go through the code (especially main.cpp and ply.cpp) to understand how light position works.

Project Submission

Please put all of your project files in a folder. Then compress this folder (.zip or .rar) and upload it to EEE Dropbox before the deadline. Then you should come to one of the following lab sessions to run the code and show your work.

Monday 2/22/2016 8pm-8:50pm location: ICS364B
Tuesday 2/23/2016 9am-9:50am location: ICS364B
Wednesday 2/24/2016 8pm-8:50pm location: ICS364B
Thursday 2/25/2016 9am-9:50am location: ICS364B