Assignment 3. Lighting and Shading

Total of Points of the Assignment: 20

Your software implementation of the graphics pipeline can transform and project triangular models. For the final rendering, however, it still contains no lighting and shading. The goal of this assignment is to extend your previous program to lighting and shading implementation.

You should extend your program to shade a triangle model in two separate windows. One of the windows will show the rendering produced by a straight OpenGL code. The other window will show an equivalent lighting effect produced by shaders. Your program should provide at least the following features:

a) Support for at least one light source.
b) Turn light(s) on and off.
c) Implement Phong Illumination Model.
d) Support flat and smooth shading models.
e) Support for interactive changes of the (RGBA) values associated with the global ambient light.
f) Support for interactive changes of the (RGBA) values associated with the ambient, diffuse and specular component of the light sources.
g) Handling of visibility z-buffering

Evaluation Criteria

1) (6 points) **OpenGL** part supports:

   (1 point) Support for at least one light source
   (1 point) Turn light(s) on and off
   (1 point) Support flat and smooth shading models
   (1 point) Handling of visibility z-buffering
   (1 point) Support for interactive changes of the (RGBA) values associated with the global ambient light
   (1 point) Support for interactive changes of the (RGBA) values associated with the ambient, diffuse and specular component of the light sources

2) (12 points) **GLSL** part supports:

   (1 point) Support for at least one light source
   (2 points) Turn light(s) on and off
   (6 points) Implement Phong Illumination Model
   (1 point) Support flat and smooth shading rendered using OpenGL commands
   (1 point) Support for interactive changes of the (RGBA) values associated with the global ambient light

(1 point) Support for interactive changes of the (RGBA) values associated with the ambient, diffuse and specular component of the light sources

3) (2 points) Renderings in both windows are synchronized.

Tips on How to Complete the Assignment

OpenGL part:

1) Study chapter 5 of "Interactive Computer Graphics" (or chapter 5 of the OpenGL red book). You will definitely need them to complete this assignment. They will improve your understanding of the subject and will prove to be extremely useful.
2) According to the section *Specular Term* on page 244 of the OpenGL red book, which vectors OpenGL uses to compute power term that modulates the intensity of the specular reflection? Answer this question before you proceed.
3) Use the equation shown at the bottom of page 244 (*Putting It All Together* – OpenGL red book) to implement the illumination (Phong) model. Ignore the spotlight effect. Also no attenuation needs to be implemented (OpenGL default is no attenuation) as part of the basic assignment.
4) In order to render shaded images of the models you will need to allow depth comparison. Why didn't you need it before? Thus, you should add GLU_DEPTH to the list of attributes when you create the window:

```
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_ALPHA | GLUT_DEPTH);
```

   In order for the depth comparison become effective, you need to enable it. Include glEnable(GL_DEPTH_TEST) in the beginning of your display function and glDisable(GL_DEPTH_TEST) at its end.
5) Define a light source GL_LIGHT0 at (0.0, 0.0, 1.0, 1.0) (I will use this as the standard light position to evaluate the rendering results of the cube model in your assignment). Use the following initial values for the ambient, diffuse and specular light components respectively, (0.2f, 0.2f, 0.2f, 1.0), (1.0, 1.0, 1.0, 1.0), (1.0, 1.0, 1.0, 1.0). Use (0.1f, 0.1f, 0.1f, 1.0f) as the global ambient intensity. In order to set these values, you will need to use the commands glLightfv and glLightModelfv. Make sure you understand them. For rendering cow and phone models, you can set the light source at the camera's position.

   Define the material properties of the object using the information (`ambient`, `diffuse`, `specular` and `material shine`) that appears in the beginning of the model file. For this, you will need the command glMaterialfv.

   Also use glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, GL_TRUE). Again, make sure you understand this command and its implications. Why are you using this?

Important: both light and material properties are associated with a window. Thus, I suggest you put all these commands in single function and call it right after you have defined the properties for the current window.

6) Dealing with interface issues is probably going to take a significant amount of time. The interface will have to accommodate (and update) lots of parameters. Make sure you use a flexible interface generator.
7) While light is disabled, render the model using glColor3f(r, g, b), where r = (md_r), g = (md_g), b = (md_b), and md_r, md_g, md_b are the material diffuse reflection coefficients for r, g and b, respectively.

You have to pay special attention to the fact that as you rotate an object, you are rotating its normals.

GLSL part:

1) Implement the vertex shader to compute shading for a given vertex. Your shading function needs to get the light vector (L, from the light source to the surface point), the view vector (V, from the COP of the camera to the point), and either the reflected vector (R) or the halfway vector (H) (which one does OpenGL uses?). Remember that before you compute dot products using these vectors, you should normalize them.
2) There are a lot of built-in variables that you can take advantage of: gl_Normal, gl_Vertex, gl_LightSource, etc. Using these features would save you a lot of time for passing values from OpenGL to vertex shader. But, you cannot rely on gl_FrontMaterial to access material property for models with multiple materials, why? (The difference between uniform and attribute variables)
3) Since the phone model has multiples materials, you need some attribute variables in vertex shader to calculate the right color for each vertex. How to pass values from OpenGL application to attribute variables in vertex shader? (glVertexAttrib() or glVertexAttribPointer())
4) When implementing Phong model, not all parts can be illuminated by the diffuse and specular light. You need to care about the angle between normal of the vertex and light direction. After rotation, normal of each vertex should be recalculated. Also, light direction would change depending on the relative position of light and vertex.
5) In vertex shader, there is no connectivity information among vertexes. How would you implement flat shading and smooth shanding? (Passing value would be too laboring, so you'd better finish that in OpenGL application).

Other Tips

How to Render to Multiple Windows using OpenGL and Synchronize the Renderings in all Windows

You can create as many windows as you want using the command *glutCreateWindow*. This function returns a unique integer identifier for each created window. Before rendering to a window, you should explicitly select a specific one using the command *glutSetWindow*. It is a good idea to save the identifiers returned by *glutCreateWindow* in an array and use some defined constants in order to improve readability. For instance, you can define the following:

```
#include <GL/glut.h>

#define OPENGL_WINDOW 0
#define CLOSE2GL_WINDOW 1

int win_id[2];

float Znear = 0.1,          // near clipping plane
Zfar = 10.0,                // far clipping plane
Hfov = 60.0,                // horizontal and
Vfov = 60.0;                // vertical field of view
```

and use it with the following template of main function. Notice that I will be using C++ notation.

```
//*******************************************************************************
//
// main function for controlling the implementation of your assignment
//
//*******************************************************************************
int main(int argc, char *argv[])
{
//
// load the triangle model by calling your function that reads the triangle model description file
// passed as the first argument to your program
//
<tri model>.<your function for reading an input file>(argv[1]);
//
// Initialize two windows, one for rendering OpenGL and another one for rendering s
// Make sure you understand the meaning of the parameters used with each command. In particular,
// make sure you understand the meaning of the parameters used with glutInitDisplayMode and understand
// its relationship to the glutSwapBuffers command. You can find detailed explanations about them in the
// OpenGL red book
//
// First, initialize OpenGL window
//
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
glutInitWindowPosition(0, 0);
glutInitWindowSize(<win width>, <win height>);
win_id[OPENGL_WINDOW] = glutCreateWindow("OpenGL");
glutDisplayFunc(<name of your function for rendering using OpenGL >);
glutReshapeFunc(openglReshape);
glutMouseFunc(<mouse control function>);
glutMotionFunc(<mouse motion control function>);
glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
glClearColor(0.0, 0.0, 0.0, 0);
glEnable(GL_DEPTH_TEST);
//
```

```
// Now, initialize Close2GL window
//
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
glutInitWindowPosition(450, 0);
glutInitWindowSize((<win width>, <win height>);
win_id[CLOSE2GL_WINDOW] = glutCreateWindow("Close2GL");
glutDisplayFunc(<name of your function for rendering using Close2GL >);
glutReshapeFunc(close2glReshape)
glutMouseFunc(<mouse control function>);
glutMotionFunc(<mouse motion control function>);
glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH | GLUT_ALPHA);
glClearColor(0.0, 0.0, 0.0, 0);
glEnable(GL_DEPTH_TEST);
//
// call your function for initializing your user interface
//
<your function for initializing your user interface>;
//
// call glutMainLoop()
//
glutMainLoop();
return 0;
}
```

A simple way to guarantee synchronized renderings in all windows is to loop through all windows and call *glutPostRedisplay* for each one of them.