Exercise 02561-03 **Local Illumination - Phongs Equation**

Readings

ANG: Chapter 6.
PRIM: Chapter 6.

Purpose

The purpose of the exercise is threefold. First, we will get acquainted with the parameters of the Phong Equation/Illumination model for calculating shading at a point. Secondly, we will have a closer look at three approximate shading models of polygons: Flat, Gouraud and Phong shading. Thirdly, we will make experiments with the type and positioning of the light source. OpenGL is used to demonstrate the effects.

Part 1

Run and understand the OpenGL program 02561-03-01.cpp and the program 02561-03-02.cpp from the homepage/Campusnet .

The Phong equation (for a point P) is presented in the textbook in the following version.

$$I = I_a + I_d + I_s = k_a L_a + f_{att}(k_d L_d \cos\theta + k_s L_s \cos^{\alpha}\phi)$$

OpenGL allows specifying both an ambient term that can be attributed to a specific light source as well as a global ambient term. A light source might also behave as a spotlight or distant light source. Furthermore, a surface may emit light. Thus, the Phong equation supported by OpenGL is

$$I = I_e + I_{ag} + (I_a + I_d + I_s) =$$
$$L_e + k_a L_{ag} + f_{att} S_{spot}(k_a L_a + k_d L_d \cos\theta + k_s L_s \cos^{\alpha}\phi)$$
$$L_e + k_a L_{ag} + \frac{1}{a + bd + cd^2}(s \cdot l)^e (k_a L_a + k_d L_d (l \cdot n) + k_s L_s (r \cdot v)^{\alpha})$$

The appendix contains three tables that specify the parameters of the functions glLight, glLightModel, and glMaterial respectively. The tables also specify the default values of the parameters.
Identity the parameters of the tables in accordance to the Phong Equation.

What is the purpose of an ambient term? What are the benefits of operating with two ambient terms in the last equation?

Part 2

Modify 02561-03-02.cpp to include 5 by 5 spheres. The value of a material parameter is changed systematically in a row, while the type of material parameter is changed from row to row. You start from the upper row and work your way downwards successively changing one of the parameters ka, kd, ks, α and Le and keeping the others constant.

Inspect the result of the experiments and answer the following questions:

What is the effect of setting the diffuse material term equal to 0,0,0?

What is the effect of setting the specular term equal to 0,0,0? Characterise the picture.

What is the effect of an increasing shininess? Characterise the change of the picture.

Part 3

Run example 02561-03-03.cpp and answer the following questions. You may modify the program to check the answers.

Which effect will it have for the calculations to place the eye point at infinity? Explain why.

Has the eye position any influence on the shading of the object?

What is the effect of positioning the light source at infinity?

What is the effect of binding the light source to the eyepoint in an animation?

What is the effect of using a directional light source?

Part 4

Explain the differences between the three shading interpolation methods Flat, Gouraud and Phong.

Run example 02561-03-01.cpp with both Flat and Gouraud shading.

Is Gouraud or Phong shading the best method for simulating highlight? Explain.

Does the shading of a Gouraud shaded polygon depend on the final orientation on the screen?

Part 5

Extend example 02561-03-03.cpp and play with the position of the light source and the eye point: Make four animations in which you move the light source, the eye point or both:

- Move the light source and keep the objects and the eye point fixed.
- Move the eye point keeping the objects and the light source fixed.
- Move both the light source and the eye point independently of each other and of the objects.
- Move both the light source and the eye point relative to the object but keeping them fixed relative to each other

Table 5-1 : Default Values for pname Parameter of glLight*()

Parameter Name	Default Value	Meaning
GL_AMBIENT	(0.0, 0.0, 0.0, 1.0)	ambient RGBA intensity of light
GL_DIFFUSE	(1.0, 1.0, 1.0, 1.0)	diffuse RGBA intensity of light
GL_SPECULAR	(1.0, 1.0, 1.0, 1.0)	specular RGBA intensity of light
GL_POSITION	(0.0, 0.0, 1.0, 0.0)	(*x*, *y*, *z*, *w*) position of light
GL_SPOT_DIRECTION	(0.0, 0.0, -1.0)	(*x*, *y*, *z*) direction of spotlight
GL_SPOT_EXPONENT	0.0	spotlight exponent
GL_SPOT_CUTOFF	180.0	spotlight cutoff angle
GL_CONSTANT_ATTENUATION	1.0	constant attenuation factor
GL_LINEAR_ATTENUATION	0.0	linear attenuation factor
GL_QUADRATIC_ATTENUATION	0.0	quadratic attenuation factor

Note: The default values listed for GL_DIFFUSE and GL_SPECULAR in Table 5-1 apply only to GL_LIGHT0. For other lights, the default value is (0.0, 0.0, 0.0, 1.0) for both GL_DIFFUSE and GL_SPECULAR.

Table 5-2 : Default Values for pname Parameter of glLightModel*()

Parameter Name	Default Value	Meaning
GL_LIGHT_MODEL_AMBIENT	(0.2, 0.2, 0.2, 1.0)	ambient RGBA intensity of the entire scene
GL_LIGHT_MODEL_LOCAL_VIEWER	0.0 or GL_FALSE	how specular reflection angles are computed
GL_LIGHT_MODEL_TWO_SIDE	0.0 or GL_FALSE	choose between one-sided or two-sided lighting

Table 5-3 : Default Values for pname Parameter of glMaterial*()

Parameter Name	Default Value	Meaning
GL_AMBIENT	(0.2, 0.2, 0.2, 1.0)	ambient color of material
GL_DIFFUSE	(0.8, 0.8, 0.8, 1.0)	diffuse color of material
GL_AMBIENT_AND_DIFFUSE		ambient and diffuse color of material
GL_SPECULAR	(0.0, 0.0, 0.0, 1.0)	specular color of material
GL_SHININESS	0.0	specular exponent
GL_EMISSION	(0.0, 0.0, 0.0, 1.0)	emissive color of material
GL_COLOR_INDEXES	(0,1,1)	ambient, diffuse, and specular color indices

```
/*02561-03-01: (= OGLPG Example 5-1 : Drawing a Lit Sphere: light.c )*/
#include <GL/glut.h>

void init(void)
{
  GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
  GLfloat mat_shininess[] = { 50.0 };
  GLfloat light_position[] = { 1.0, 1.0, 1.0, 0.0 };
  glClearColor (0.0, 0.0, 0.0, 0.0);
  glShadeModel (GL_SMOOTH);

  glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
  glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
  glLightfv(GL_LIGHT0, GL_POSITION, light_position);

  glEnable(GL_LIGHTING);
  glEnable(GL_LIGHT0);
  glEnable(GL_DEPTH_TEST);
}

void display(void)
{
  glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glutSolidSphere (1.0, 20, 16);
  glFlush ();
}

void reshape (int w, int h)
{
  glViewport (0, 0, (GLsizei) w, (GLsizei) h);
  glMatrixMode (GL_PROJECTION);
  glLoadIdentity();
  if (w <= h)
    glOrtho (-1.5, 1.5, -1.5*(GLfloat)h/(GLfloat)w,
      1.5*(GLfloat)h/(GLfloat)w, -10.0, 10.0);
  else
    glOrtho (-1.5*(GLfloat)w/(GLfloat)h,
      1.5*(GLfloat)w/(GLfloat)h, -1.5, 1.5, -10.0, 10.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
}

int main(int argc, char** argv)
{
  glutInit(&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
  glutInitWindowSize (500, 500);
  glutInitWindowPosition (100, 100);
  glutCreateWindow (argv[0]);
  init ();
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutMainLoop();
  return 0;
}
```

```
/*    02561-03-02.cpp
 * (=OGLPG Example 5.8, material.c)
 * This program demonstrates the use of the GL lighting model.
 * Several objects are drawn using different material characteristics.
 * A single light source illuminates the objects.
 */


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

/*  Initialize z-buffer, projection matrix, light source,
 *  and lighting model.  Do not specify a material property here.
 */
void myinit(void)
{
    GLfloat ambient[] = { 0.0, 0.0, 0.0, 1.0 };
    GLfloat diffuse[] = { 1.0, 1.0, 1.0, 1.0 };
    GLfloat position[] = { 0.0, 3.0, 2.0, 0.0 };
    GLfloat lmodel_ambient[] = { 0.4, 0.4, 0.4, 1.0 };
    GLfloat local_view[] = { 0.0 };

    glEnable(GL_DEPTH_TEST);
    glDepthFunc(GL_LESS);

    glLightfv(GL_LIGHT0, GL_AMBIENT, ambient);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse);
    glLightfv(GL_LIGHT0, GL_POSITION, position);
    glLightModelfv(GL_LIGHT_MODEL_AMBIENT, lmodel_ambient);
    glLightModelfv(GL_LIGHT_MODEL_LOCAL_VIEWER, local_view);

    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);

    glClearColor(0.0, 0.1, 0.1, 0.0);
}

/*  Draw twelve spheres in 3 rows with 4 columns.
 *  The spheres in the first row have materials with no ambient reflection.
 *  The second row has materials with significant ambient reflection.
 *  The third row has materials with colored ambient reflection.
 *
 *  The first column has materials with blue, diffuse reflection only.
 *  The second column has blue diffuse reflection, as well as specular
 *  reflection with a low shininess exponent.
 *  The third column has blue diffuse reflection, as well as specular
 *  reflection with a high shininess exponent (a more concentrated highlight).
 *  The fourth column has materials which also include an emissive component.
 *
 *  glTranslatef() is used to move spheres to their appropriate locations.
 */

void display(void)
{
    GLfloat no_mat[] = { 0.0, 0.0, 0.0, 1.0 };
    GLfloat mat_ambient[] = { 0.7, 0.7, 0.7, 1.0 };
    GLfloat mat_ambient_color[] = { 0.8, 0.8, 0.2, 1.0 };
    GLfloat mat_diffuse[] = { 0.1, 0.5, 0.8, 1.0 };
    GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
    GLfloat no_shininess[] = { 0.0 };
    GLfloat low_shininess[] = { 5.0 };
    GLfloat high_shininess[] = { 100.0 };
    GLfloat mat_emission[] = {0.3, 0.2, 0.2, 0.0};

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

```
/*  draw sphere in first row, first column
 *  diffuse reflection only; no ambient or specular
 */
    glPushMatrix();
    glTranslatef (-3.75, 3.0, 0.0);
    glMaterialfv(GL_FRONT, GL_AMBIENT, no_mat);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
    glMaterialfv(GL_FRONT, GL_SPECULAR, no_mat);
    glMaterialfv(GL_FRONT, GL_SHININESS, no_shininess);
    glMaterialfv(GL_FRONT, GL_EMISSION, no_mat);
    glutSolidSphere(1.0, 16, 16);
    glPopMatrix();

/*  draw sphere in first row, second column
 *  diffuse and specular reflection; low shininess; no ambient
 */
    glPushMatrix();
    glTranslatef (-1.25, 3.0, 0.0);
    glMaterialfv(GL_FRONT, GL_AMBIENT, no_mat);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
    glMaterialfv(GL_FRONT, GL_SHININESS, low_shininess);
    glMaterialfv(GL_FRONT, GL_EMISSION, no_mat);
    glutSolidSphere(1.0, 16, 16);
    glPopMatrix();

/*  draw sphere in first row, third column
 *  diffuse and specular reflection; high shininess; no ambient
 */
    glPushMatrix();
    glTranslatef (1.25, 3.0, 0.0);
    glMaterialfv(GL_FRONT, GL_AMBIENT, no_mat);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
    glMaterialfv(GL_FRONT, GL_SHININESS, high_shininess);
    glMaterialfv(GL_FRONT, GL_EMISSION, no_mat);
    glutSolidSphere(1.0, 16, 16);
    glPopMatrix();

/*  draw sphere in first row, fourth column
 *  diffuse reflection; emission; no ambient or specular reflection
 */
    glPushMatrix();
    glTranslatef (3.75, 3.0, 0.0);
    glMaterialfv(GL_FRONT, GL_AMBIENT, no_mat);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
    glMaterialfv(GL_FRONT, GL_SPECULAR, no_mat);
    glMaterialfv(GL_FRONT, GL_SHININESS, no_shininess);
    glMaterialfv(GL_FRONT, GL_EMISSION, mat_emission);
    glutSolidSphere(1.0, 16, 16);
    glPopMatrix();

/*  draw sphere in second row, first column
 *  ambient and diffuse reflection; no specular
 */
    glPushMatrix();
    glTranslatef (-3.75, 0.0, 0.0);
    glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
    glMaterialfv(GL_FRONT, GL_SPECULAR, no_mat);
    glMaterialfv(GL_FRONT, GL_SHININESS, no_shininess);
    glMaterialfv(GL_FRONT, GL_EMISSION, no_mat);
    glutSolidSphere(1.0, 16, 16);
    glPopMatrix();
```

```
/*  draw sphere in second row, second column
 *  ambient, diffuse and specular reflection; low shininess
 */
    glPushMatrix();
    glTranslatef (-1.25, 0.0, 0.0);
    glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
    glMaterialfv(GL_FRONT, GL_SHININESS, low_shininess);
    glMaterialfv(GL_FRONT, GL_EMISSION, no_mat);
    glutSolidSphere(1.0, 16, 16);
    glPopMatrix();

/*  draw sphere in second row, third column
 *  ambient, diffuse and specular reflection; high shininess
 */
    glPushMatrix();
    glTranslatef (1.25, 0.0, 0.0);
    glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
    glMaterialfv(GL_FRONT, GL_SHININESS, high_shininess);
    glMaterialfv(GL_FRONT, GL_EMISSION, no_mat);
    glutSolidSphere(1.0, 16, 16);
    glPopMatrix();

/*  draw sphere in second row, fourth column
 *  ambient and diffuse reflection; emission; no specular
 */
    glPushMatrix();
    glTranslatef (3.75, 0.0, 0.0);
    glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
    glMaterialfv(GL_FRONT, GL_SPECULAR, no_mat);
    glMaterialfv(GL_FRONT, GL_SHININESS, no_shininess);
    glMaterialfv(GL_FRONT, GL_EMISSION, mat_emission);
    glutSolidSphere(1.0, 16, 16);
    glPopMatrix();

/*  draw sphere in third row, first column
 *  colored ambient and diffuse reflection; no specular
 */
    glPushMatrix();
    glTranslatef (-3.75, -3.0, 0.0);
    glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient_color);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
    glMaterialfv(GL_FRONT, GL_SPECULAR, no_mat);
    glMaterialfv(GL_FRONT, GL_SHININESS, no_shininess);
    glMaterialfv(GL_FRONT, GL_EMISSION, no_mat);
    glutSolidSphere(1.0, 16, 16);
    glPopMatrix();

/*  draw sphere in third row, second column
 *  colored ambient, diffuse and specular reflection; low shininess
 */
    glPushMatrix();
    glTranslatef (-1.25, -3.0, 0.0);
    glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient_color);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
    glMaterialfv(GL_FRONT, GL_SHININESS, low_shininess);
    glMaterialfv(GL_FRONT, GL_EMISSION, no_mat);
    glutSolidSphere(1.0, 16, 16);
    glPopMatrix();
```

```c
/*  draw sphere in third row, third column
 *  colored ambient, diffuse and specular reflection; high shininess
 */
    glPushMatrix();
    glTranslatef (1.25, -3.0, 0.0);
    glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient_color);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
    glMaterialfv(GL_FRONT, GL_SHININESS, high_shininess);
    glMaterialfv(GL_FRONT, GL_EMISSION, no_mat);
    glutSolidSphere(1.0, 16, 16);
    glPopMatrix();

/*  draw sphere in third row, fourth column
 *  colored ambient and diffuse reflection; emission; no specular
 */
    glPushMatrix();
    glTranslatef (3.75, -3.0, 0.0);
    glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient_color);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
    glMaterialfv(GL_FRONT, GL_SPECULAR, no_mat);
    glMaterialfv(GL_FRONT, GL_SHININESS, no_shininess);
    glMaterialfv(GL_FRONT, GL_EMISSION, mat_emission);
    glutSolidSphere(1.0, 16, 16);
    glPopMatrix();

    glFlush();
}

void myReshape(int w, int h)
{
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    if (w <= (h * 2))
      glOrtho (-6.0, 6.0, -3.0*((GLfloat)h*2)/(GLfloat)w,
          3.0*((GLfloat)h*2)/(GLfloat)w, -10.0, 10.0);
    else
      glOrtho (-6.0*(GLfloat)w/((GLfloat)h*2),
          6.0*(GLfloat)w/((GLfloat)h*2), -3.0, 3.0, -10.0, 10.0);
    glMatrixMode(GL_MODELVIEW);
}

/*  Main Loop
 *  Open window with initial window size, title bar,
 *  RGBA display mode, and handle input events.
 */
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize (600, 450);
    glutCreateWindow(argv[0]);
    myinit();
    glutReshapeFunc(myReshape);
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;             /* ANSI C requires main to return int. */
}
```

```
/*02561-03-03  (= OGLPG: Example 5-6 : movelight.c
* Moving a Light with Modeling Transformations
*/

#include <GLglut.h>

static int spin = 0;

void init(void)
{
  glClearColor (0.0, 0.0, 0.0, 0.0);
  glShadeModel (GL_SMOOTH);
  glEnable(GL_LIGHTING);
  glEnable(GL_LIGHT0);
  glEnable(GL_DEPTH_TEST);
}

/*  Here is where the light position is reset after the modeling
 *  transformation (glRotated) is called.  This places the
 *  light at a new position in world coordinates.  The cube
 *  represents the position of the light.
 */
void display(void)
{
  GLfloat position[] = { 0.0, 0.0, 1.5, 1.0 };

  glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glPushMatrix ();
  glTranslatef (0.0, 0.0, -5.0);

  glPushMatrix ();
  glRotated ((GLdouble) spin, 1.0, 0.0, 0.0);
  glLightfv (GL_LIGHT0, GL_POSITION, position);

  glTranslated (0.0, 0.0, 1.5);
  glDisable (GL_LIGHTING);
  glColor3f (0.0, 1.0, 1.0);
  glutWireCube (0.1);
  glEnable (GL_LIGHTING);
  glPopMatrix ();

  glutSolidTorus (0.275, 0.85, 8, 15);
  glPopMatrix ();
  glFlush ();
}

void reshape (int w, int h)
{
  glViewport (0, 0, (GLsizei) w, (GLsizei) h);
  glMatrixMode (GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(40.0, (GLfloat) w/(GLfloat) h, 1.0, 20.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
}
```

```
void mouse(int button, int state, int x, int y)
{
  switch (button) {
    case GLUT_LEFT_BUTTON:
      if (state == GLUT_DOWN) {
        spin = (spin + 30) % 360;
        glutPostRedisplay();
      }
      break;
    default:
      break;
  }
}
int main(int argc, char** argv)
{
  glutInit(&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
  glutInitWindowSize (500, 500);
  glutInitWindowPosition (100, 100);
  glutCreateWindow (argv[0]);
  init ();
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutMouseFunc(mouse);
  glutMainLoop();
  return 0;
}
```