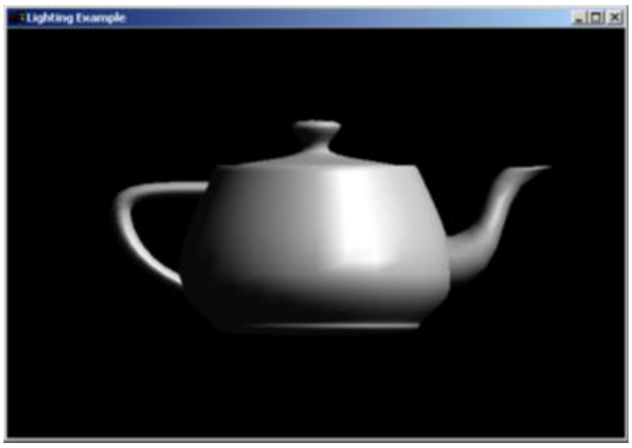
# Lab09 for CPT205 Computer Graphics (Lighting and Materials)

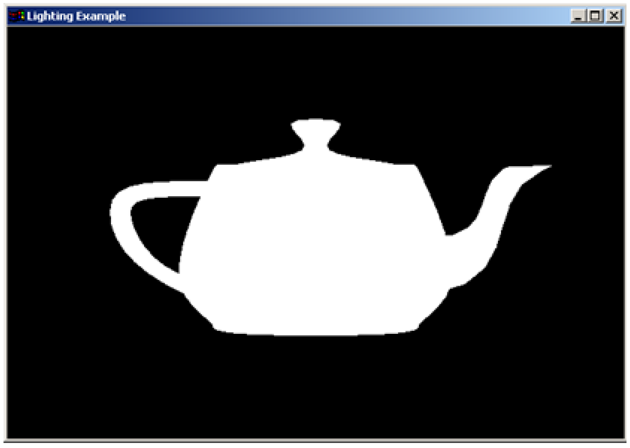
There are two sample programs for this lab: one for lighting and the other for materials. You can run these two programs first and try to understand what is done and what functions are used. Then you can further practise with the functions and see the effects.

## Sample Program 1

```
// Lab09 - Lighting
#define FREEGLUT_STATIC
#include <GL/freeglut.h>
#include <stdlib.h>

GLfloat light_x = 1.0;
GLfloat light_y = 1.0;
GLfloat light_z = 1.0;

//Initialise a light
void myinit(GLfloat light_x, GLfloat light_y, GLfloat light_z) {
    GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
    GLfloat mat_shininess[] = { 50.0 };
    GLfloat light_position[] = { light_x, light_y, light_z, 0.0 };  // Directional light

    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);

    glDepthFunc(GL_LEQUAL);
    glEnable(GL_DEPTH_TEST);
}

// Define keyboard event for incrementally changing the lighting direction
void keyboard_input(unsigned char key, int x, int y) {
    if (key == 'r' || key == 'R') {
        light_x = light_x + 0.2;
        if (light_x >= 2) {
            light_x = 0;
        }
        myinit(light_x, light_y, light_z);
        glutPostRedisplay();
    }
}

// Draw a solid teapot
void display(void) {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glutSolidTeapot(1.0);
    glFlush();
}

//This will be called when the viewport is changed
void myReshape(GLsizei w, GLsizei h) {
    glViewport(0, 0, w, 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_RGBA | GLUT_DEPTH);
    glutInitWindowPosition(0, 0);
    glutInitWindowSize(600, 400);
    if (glutCreateWindow("Lighting Example") == GL_FALSE)
        exit(0);
    myinit(light_x, light_y, light_z);
```

```
        glutKeyboardFunc(keyboard_input);
        glutReshapeFunc(myReshape);
        glutDisplayFunc(display);
        glutMainLoop();
    }
```





Teapot with lighting                Teapot without lighting

**Steps for defining a lighting model in OpenGL**

1) Defining normal vector

- The normal vector is defined for each vertex of the object to determine the orientation of the object relative to the light source, and hence the amount of light each vertex receives from the light source.
- This is accomplished as part of the glutSolidTeapot() routine in the example.

2) Creating position and enabling light sources

- The example uses one light source and its location (actually direction) is defined by glLightfv().
- It uses the default colour for GL_LIGHT0, which is white.
- The glEnable() routine is called to turn the light on, and to prepare OpenGL for lighting calculations.

3) Selecting a lighting model

- This defines the level of global ambient light and the effective location of the viewpoint for lighting calculations.
- This is done with glLightModel*(), which describes the lighting model parameters.
- In the example, the global ambient light is the only element of the lighting model, which is defined explicitly.
- The lighting model also defines whether the viewer of the scene is considered to be at an infinite distance away or local to the scene, and whether lighting calculations should be performed differently for the front and back surfaces of objects in the scene.
- The example uses the default settings for these two aspects of the model - an infinite viewer and one-sided lighting. The use of a local viewer adds significantly to the complexity of the calculations that must be performed because OpenGL must calculate the angle between the viewpoint and each object. With an infinite viewer, however, the angle is ignored, and the results are slightly less realistic. Moreover, since in this example, the back surface of the object is never seen (i.e. the inside of the object), one-sided lighting is sufficient.

4) Defining material properties

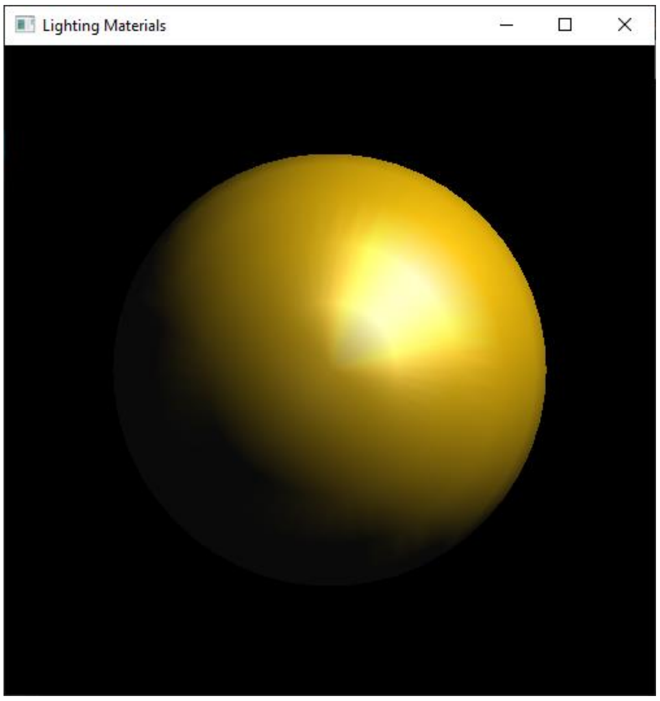
- The material properties of an object determine how the object reflects light, and therefore, what material it seems to be made of.
- Because the interaction between the object material surface and incident light is complex, specifying material properties to give the object a desired appearance is an art.

- You can specify the ambient, diffuse and specular colours of a material and how shiny the material is.
- The example specifies explicitly only the last two material properties - the specular material colour and shininess with glMaterialfv().

**Sample Program 2**

```
// Lab09 - Material properties
#define FREEGLUT_STATIC
#include <GL/freeglut.h>
#include <stdlib.h>

GLfloat diffuseMaterial[4] = { 0.5, 0.5, 0.5, 1.0 };

// Define light and material properties
void myinit(void) {
    GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
    GLfloat light_position[] = { 1.0, 1.0, 1.0, 0.0 };  // Directional light

    glMaterialfv(GL_FRONT, GL_DIFFUSE, diffuseMaterial);
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
    glMaterialf(GL_FRONT, GL_SHININESS, 25.0);
    glLightfv(GL_LIGHT0, GL_POSITION, light_position);

    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glDepthFunc(GL_LEQUAL);
    glEnable(GL_DEPTH_TEST);

    // Specify which material parameters track the current color
    glColorMaterial(GL_FRONT, GL_DIFFUSE); // GL_EMISSION, GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR, and
GL_AMBIENT_AND_DIFFUSE - add to ppt
    glEnable(GL_COLOR_MATERIAL);
}

// Define keyboard event for incrementally changing the RGB component values separately
void keyboard_input(unsigned char key, int x, int y) {
    if (key == 'r' || key == 'R') {
        diffuseMaterial[0] += 0.1;
        if (diffuseMaterial[0] > 1.0)
            diffuseMaterial[0] = 0.0;
    }
    else if (key == 'g' || key == 'G') {
        diffuseMaterial[1] += 0.1;
        if (diffuseMaterial[1] > 1.0)
            diffuseMaterial[1] = 0.0;
    }
    else if (key == 'b' || key == 'B') {
        diffuseMaterial[2] += 0.1;
        if (diffuseMaterial[2] > 1.0)
            diffuseMaterial[2] = 0.0;
    }
    glColor4fv(diffuseMaterial);
    glutPostRedisplay();
}

// Draw a solid sphere
void display(void) {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glutSolidSphere(1.0, 50, 10);  // Radius, Longitude subdivisions and Latitude subdivisions
    glFlush();
}

//This will be called when the viewport is changed
void myReshape(GLsizei w, GLsizei h) {
    glViewport(0, 0, w, 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_RGBA | GLUT_DEPTH);
    glutInitWindowPosition(0, 0);
    glutInitWindowSize(500, 500);
    glutCreateWindow("Lighting Materials");

    myinit();
    glutKeyboardFunc(keyboard_input);
    glutReshapeFunc(myReshape);
    glutDisplayFunc(display);
    glutMainLoop();
}
```


Lighting Materials