Lighting in OpenGL – HandsOn Session 1

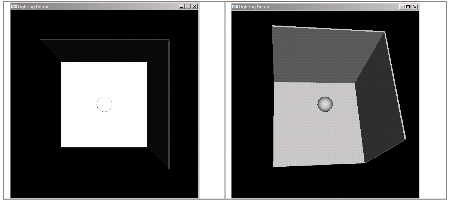
In this in-class session, we explore the OpenGL implementation of lighting, and some of the geometry behind illumination. Start by downloading the OpenGL source code **spotlight.cpp** from the class syllabus. This program renders a scene composed of three walls of a room, and allows the user to interactively rotate and translate the scene. We will extend this piece of code to light up the room.

**Activity 1 [Enable Lighting and Light Source]**

1. Compile and run the program and observe the output. Then add two lines of code to the EnableLighting function, to enable lighting and the light source GL\_LIGHT0. Recompile and run the code. Note that light cancels the effect of glColor3f.

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT0);

The default shading model is GL\_FLAT. This means that OpenGL calculates the intensity of the reflected light at the *first vertex* of each polygon, and uses it as the intensity of every point inside the polygon. Specify a smooth shading model using

glShadeModel(GL\_SMOOTH);

This enables OpenGL to calculate the intensity of the reflected light at *each polygon vertex*, and to *interpolate* across the polygon to get the intensity at each point on the polygon.

The lighting still doesn’t work properly for us. This is because OpenGL uses unit normals at each vertex to compute the vetex color. GLUT shapes already have normals set. We need to tell OpenGL to preserve the unit normals (and not let them get affected by transformations):

glEnable(GL\_NORMALIZE);

Notice the difference in appearance between the two scenes above.

**Activity 2 [Set Light Properties]**

a)  OpenGL supports up to eight lights. Light 0 is unique in having a default diffuse and specular setting of fully bright white (1,1,1,1). All other lights have a default settings of totally dark (0,0,0,1). Each of the eight lights has a default position of (0,0,1,0), which would effectively place it as a directional light, facing forward, from just behind the camera.  Comment out the line that enables light 0 and set the following light properties for light 1:

 GLfloat light1\_diffuse[] = {1.0, 1.0, 1.0, 1.0}; /\* bright white \*/

GLfloat light1\_specular[] = {1.0, 1.0, 1.0, 1.0};

glLightfv(GL\_LIGHT1, GL\_DIFFUSE, light1\_diffuse);

glLightfv(GL\_LIGHT1, GL\_SPECULAR, light1\_specular);

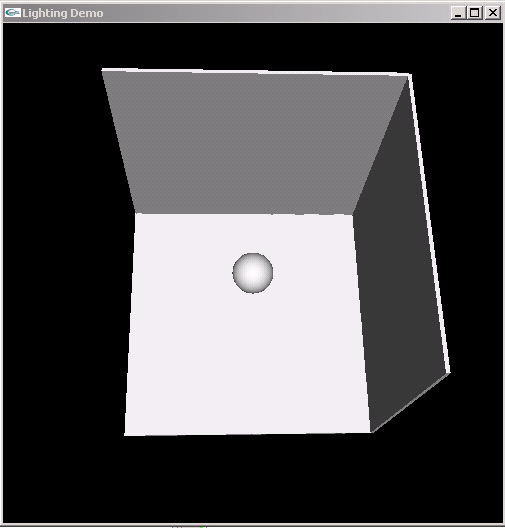
These lines of code should go in the EnableLighting function. Do not forget to enable light 1. The scene should appear as if light 0 were enabled.

b)  Add a soft white ambient component to light 1 from the previous activity:

 GLfloat light1\_ambient[] = {0.8, 0.8, 0.8, 1.0}; /\* soft white \*/

glLightfv(GL\_LIGHT1, GL\_AMBIENT, light1\_ambient);

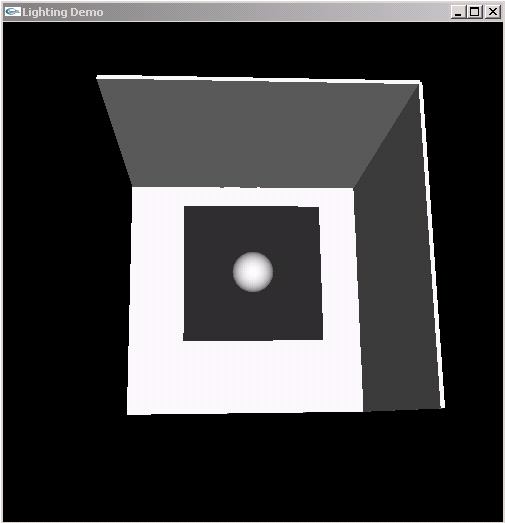
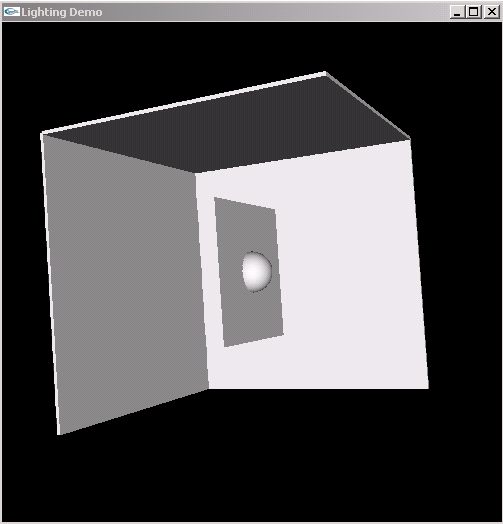
This component adds some level of brightness to the scene.



**Activity 3 [Specify Normals]**

1. Add a vertical unit square to the scene, centered at the origin and facing the light:



b) The square shape is not lit up. Explain why.

Specify a unit normal for each vertex:

glNormal3f(0, 0, 1);

This enables OpenGL to calculate the color intensity at vertices of the square, and interpolate across the square. The square has the same color as the back wall (see right image above).

For predefined OpenGL shapes (cube, sphere, etc.), vertex normals are already set.

**Activity 4 [Specify Material Color and Shineness]**

When light is enabled, glColor3f has no effect. Typically, material properties indirectly define the color of an object. Add the following lines of code to the EnableLighting function:

GLfloat mat\_specular[] = {1.0, 1.0, 1.0, 1.0};

GLfloat mat\_diffuse[] = {1.0, 0.5, 0.0, 1.0};

GLfloat mat\_ambient[]= {1.0, 0.5, 0.0, 1.0};

GLfloat mat\_shininess = 5.0;

glMaterialfv(GL\_FRONT, GL\_SPECULAR, mat\_specular);

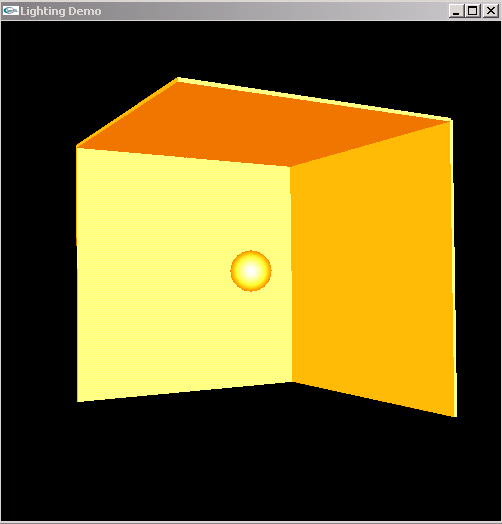
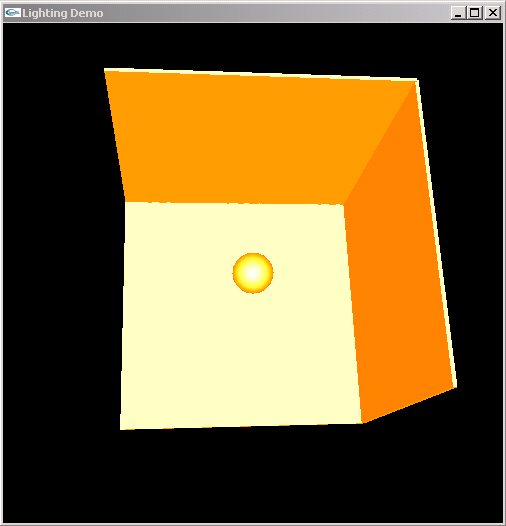
glMaterialfv(GL\_FRONT, GL\_AMBIENT, mat\_ambient);

glMaterialfv(GL\_FRONT, GL\_DIFFUSE, mat\_diffuse);

glMaterialf(GL\_FRONT, GL\_SHININESS, mat\_shininess);

Eliminate the center square tested in the previous activity, recompile and run.

The scene should appear as above (same scene, different orientations).

**Activity 5 [Positional Light Source]**

All eight OpenGL lights have a default position of (0,0,1,0). The fourth argument distinguishes between a *directional* light source (0, light at infinity) and *positional* light source (1). So far we have worked with the default directional parameter.

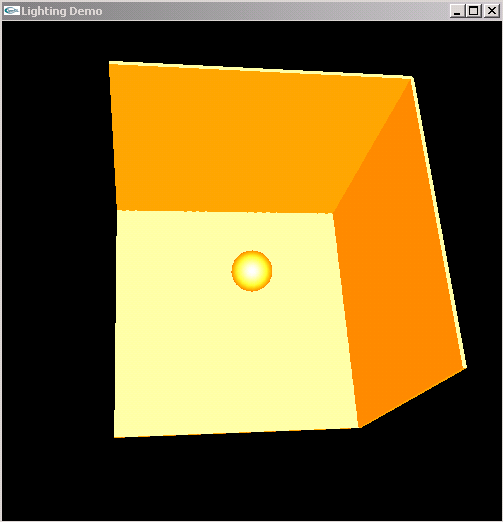
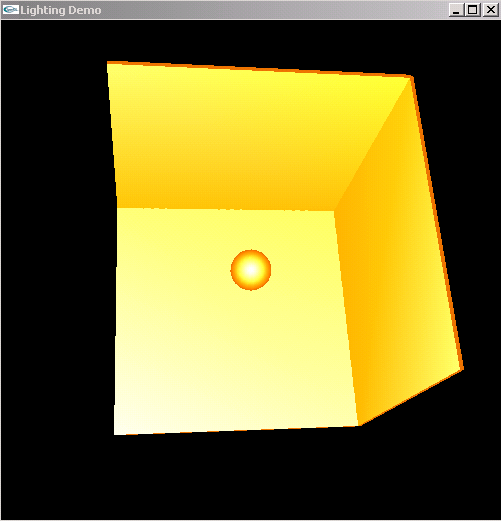


a) Identify the type of light (positional, directional) in the two images below.

How can you tell the difference? Which lighting method is more expensive? b) Turn light 1 into a positional light using

GLfloat light1\_position[] = {0.0, 0.0, 1.0, 1.0};

glLightfv(GL\_LIGHT1, GL\_POSITION, light1\_position);

c) The light position gets multiplied by the MODELVIEW matrix to determine the actual position in the scene (think of the light as just another object in the scene).

**Try this out**: Eliminate (comment out) the ambient and specular components of light 1, leaving just the diffuse component in place. Reposition the light source (by calling glLightfv(GL\_LIGHT1, GL\_POSITION, ...) with each rotation/translation of the scene. The light should move with the scene as shown below.