Computer Graphics and Applications
Lecture 7: Lighting

What we want to understand here is:

- How real-world lighting conditions are approximated by OpenGL
- Render illuminated objects by defining the desired light sources
- Define the material properties of the objects being illuminated
- Control the position of light sources

OpenGL approximates light and lighting as if light can be broken into red, green, and blue components. Thus, the color of light sources is characterized by the amount of red, green, and blue light they emit, and the material of surfaces is characterized by the percentage of the incoming red, green, and blue components that are reflected in various directions.

In the OpenGL lighting model, the light in a scene comes from several light sources that can individually be turned on and off. Some light comes from a particular direction or position, and some light is generally scattered about the scene. For example, when you turn on a light bulb in a room, most of the light comes from the bulb, but some light comes after bouncing off one, two, three, or more walls. Moreover, there might be a general ambient light in the scene that comes from no particular source, as if it had been scattered so many times that its original source is impossible to determine.

The OpenGL lighting model considers the lighting to be divided into four independent components: **emitted, ambient, diffuse,** and **specular**. All four components are computed independently, and then added together.

Emitted, Ambient, Diffuse, and Specular Light

Emitted light is the simplest - it originates from an object and is unaffected by any light sources.

Ambient light originates from a source that's been scattered so much by the environment that its direction is impossible to determine - it seems to come from all directions. Backlighting in a room has a large ambient component, since most of the light that reaches your eye has bounced off many surfaces first. A spotlight outdoors has a tiny ambient component; most of the light travels in the same direction, and since you're outdoors, very little of the light reaches your eye after bouncing off other objects. When ambient light strikes a surface, it's scattered equally in all directions.

Ambient is average volume of light created from all the light sources

Diffuse light comes from one direction, so it's brighter if it comes squarely down on a surface than if it barely glances off the surface. Once it hits a surface, however, it's scattered equally in all directions, so it appears equally bright, no matter where the eye is located. Any light coming from a particular position or direction probably has a diffuse component.

Diffuse is directional light cast by a light source

Specular light comes from a particular direction, and it tends to bounce off the surface in a preferred direction. A well-collimated laser beam bouncing off a high-quality mirror produces almost 100 percent specular reflection. Shiny metal or plastic has a high specular component, and chalk or carpet has almost none. You can think of specularity as shininess.

Specular: produces reflections and highlights

Material Colors

The OpenGL lighting model makes calculates material's colour depending on the percentages of the incoming red, green, and blue light it reflects. For example, a perfectly red ball reflects all the incoming red light and absorbs all the green and blue light that strikes it. Like lights, materials have different ambient, diffuse, and specular colors, which determine the ambient, diffuse, and specular reflectances of the material.

Creating Light Sources

Light sources have a number of properties, such as color, position, and direction. The command used to specify all properties of lights is **glLight*()**; Here's an example of using **glLight*()**:

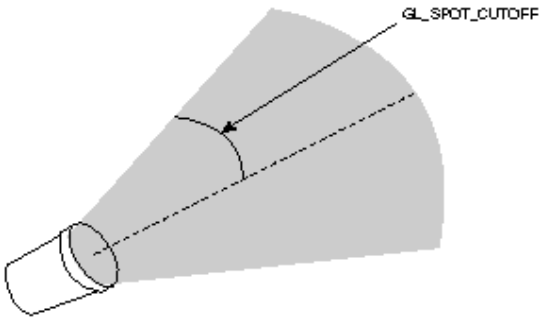
```
GLfloat light_ambient[] = { 0.0, 0.0, 0.0, 1.0 };
// the above defines the ambient component takes four parameters, (the
//RBGA). Each parameter ranges from 0.0 to 1.0
// similarly diffuse and specular as below
GLfloat light_diffuse[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat light_specular[] = { 1.0, 1.0, 1.0, 1.0 };


GLfloat light_position[] = { 1.0, 1.0, 1.0, 0.0 };
// this defines the position of light. It has four coordinates namely x, y,
//z and w. x, y, z are the usual 3D coordinates. w is special and usually
//taken to be 0.0

glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
```

```
// here we assign the previously defined ambient light to the light
//GL_LIGHT0. Note you can have up to 8 lights in a given window. They are
// GL_LIGHT0, GL_LIGHT1, …, GL_LIGHT8 and this naming convention has to be
// followed when declaring lights.

glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
// now we have defined the diffuse specular and the position of light for
// the light GL_LIGHT0

glEnable(GL_LIGHT0);
glEnable(GL_LIGTING);
glEnable(GL_DEPTH_TEST):
// and finally ask OpenGL to turn on the light.
```

Download lighting01.cpp and compile it. Note you may see some warnings but ignore the warnings and run the program. Explore the init () function where the lighting is defined by change values of the colour components and light position.

Spotlights

As previously mentioned, you can have a positional light source act as a spotlight - that is, by restricting the shape of the light it emits to a cone. To create a spotlight, you need to determine the spread of the cone of light you desire. To specify the angle between the axis of the cone and a ray along the edge of the cone, use the GL_SPOT_CUTOFF parameter. The angle of the cone at the apex is then twice this value, as shown in below.



Note that no light is emitted beyond the edges of the cone. The value for GL_SPOT_CUTOFF is restricted to being within the range 0.0 to 90.0 (The following line sets the cutoff parameter to 45 degrees:

```
glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, 45.0);
```

You also need to specify a spotlight's direction, which determines the axis of the cone of light:

```
GLfloat spot_direction[] = { -1.0, -1.0, 0.0 };
glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, spot_direction);
```

Download lighting02.cpp and compile it and run to explore a spot light.

Multiple Lights

As mentioned, you can have up to eight lights in your scene. Since OpenGL needs to perform calculations to determine how much light each vertex receives from each light source, increasing the number of lights adversely affects performance. The constants used to refer to the eight lights are GL_LIGHT0, GL_LIGHT1, GL_LIGHT2, GL_LIGHT3, and so on.

Download lighting03.cpp and compile it and run to explore the use of multiple lights.

Two-sided Lighting

Lighting calculations are performed for all polygons, whether they're front-facing or back-facing. Since you usually set up lighting conditions with the front-facing polygons in mind, however, the back-facing ones typically aren't correctly illuminated. Also you might want to have the inside surface be fully lit according to the lighting conditions you've defined; you might also want to supply a different material description for the back faces. When you turn on two-sided lighting, as follows

glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE);

To turn two-sided lighting off, pass in GL_FALSE as the argument in the preceding call.

Shading Model

A line or a filled polygon primitive can be drawn with a single colour (flat shading) or with many different colours (smooth shading, also called Gouraud shading). You specify the desired shading technique with **glShadeModel()**.void **glShadeModel** (GLenum *mode*);

Sets the shading model. The mode parameter can be either GL_SMOOTH (the default) or GL_FLAT.

With flat shading, the color of one vertex of a primitive is duplicated across all the primitive's vertices. With smooth shading, the color at each vertex is treated individually. For a line primitive, the colors along the line segment are interpolated between the vertex colors. For a polygon primitive, the colors for the interior of the polygon are interpolated between the vertex colors.

Download smooth.cpp, compile and run to see an example of shading.
Change glShadeModel (GL_SMOOTH); to glShadeModel (GL_FLAT); to see the difference between smooth and flat shading.

Defining Material Properties

You've seen how to create light sources with certain characteristics. The material properties of the objects in your scene are conceptually similar to ones you've already used to create light sources. The mechanism for setting them is similar, except that the command used is called **glMaterial*()**.

void **glMaterial**{if}[v](*face*, *name*, *TYPE*);

Specifies a current material property for use in lighting calculations. The face parameter can be GL_FRONT, GL_BACK, or GL_FRONT_AND_BACK to indicate which face of the object the material should be applied to. The particular material property being set is identified by *name.*

As mentioned previously the material properties are worked out based on the light reflected by the object based on the components: diffuse, ambient, specular.

Diffuse and Ambient Reflection

The GL_DIFFUSE and GL_AMBIENT parameters set with **glMaterial*()** affect the color of the diffuse and ambient light reflected by an object. Diffuse reflectance plays the most important role in determining what you perceive the color of an object to be. It's affected by the color of the incident diffuse light and the angle of the incident light relative to the normal direction.

Ambient reflectance affects the overall color of the object. Because diffuse reflectance is brightest where an object is directly illuminated, ambient reflectance is most noticeable where an object receives no direct illumination. An object's total ambient reflectance is affected by the global ambient light and ambient light from individual light sources.

Example:

```
GLfloat mat_amb_diff[] = { 0.1, 0.5, 0.8, 1.0 };
glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE,
            mat_amb_diff);
```

Specular Reflection

Specular reflection from an object produces highlights. OpenGL allows you to set the RGBA color of a specular highlight (with GL_SPECULAR) and to control the size and brightness of the highlight (with GL_SHININESS). You can assign a number in the range of [0.0, 128.0] to GL_SHININESS - the higher the value, the smaller and brighter (more focused) the highlight.

Example:

```
GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat low_shininess[] = { 5.0 };
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
glMaterialfv(GL_FRONT, GL_SHININESS, low_shininess);
```

Emission

By specifying an RGBA color for GL_EMISSION, you can make an object appear to be giving off light of that color. Since most real-world objects (except lights) don't emit light, you'll probably use this feature mostly to simulate lamps and other light sources in a scene.

Example:

```
GLfloat mat_emission[] = {0.3, 0.2, 0.2, 0.0};
glMaterialfv(GL_FRONT, GL_EMISSION, mat_emission);
```

Download materil.cpp, compile and run to see an example of assigning material properties to our teapot example.

Note: A series of further examples are also available within Blackboard.