Lectures
Directional Light Case Study

By
Tom Duff
Pixar Animation Studios
Emeryville, California
and
George Ledin Jr
Sonoma State University
Rohnert Park, California

How does light work?

- **Decide what kind of light(s) you want**

void **glLightfv**(GLenum *light*, GLenum *pname*, const GLfloat **params*)

  - *light* : Specifies the light, which can be GL_LIGHT0, GL_LIGHT1, … or GL_LIGHT7. At least 8 in OpenGL, but may be more in other implementations.
  -
  - *pname*: Specifies a light source characteristic.
  - *param*: Specifies the value to which pname characteristic is set to.

- **Turn on the light(s)**
  - Lighting in general must be enabled.
  - Each individual light must be enabled.

- In the following examples, we will look at how to position the light, using GL_POSITION, and we will look at other light characteristics later:
  - GL_AMBIENT
  - GL_DIFFUSE
  - GL_SPECULAR

Directional Light vs Spot Light

- **Directional Light: The source of light is at infinity. The light is a parallel light from a specified direction.**
  - We use a vector (x,y,z,0) to represent directional light, and we specify this vector by invoking glLight().

    - The “(x,y,z)” is specified at world coordinates.
    - The “(x,y,z)” is to represent the direction of light. This vector always points to the origin.
    - The “0” simply means light source is at infinity.
  - Diffuse and specular lighting calculations take into account the light's direction, but not its actual position, and attenuation is disabled.
- **Spot Light: Light has a position, direction and angle.**
  - We use a vector (x,y,z,1) to represent the spot light, and its location at (x,y,z) by invoking glLight().
    - The “(x,y,z)” is specified at camera coordinates.
    - The “(x,y,z)” is to represent the actual position of the light.
    - The “1” simply means that the light is a spotlight.

  - We use another vector (a,b,c) to represent the direction of the spotlight.
    - The (a,b,c) is specified at world coordinates.
    - The vector (a,b,c) always points to the origin.

Directional light
General Questions

- General questions:
  - How can we specify the direction of each of several lights? (such as LIGHT0, LIGHT1, LIGHT2, and etc)
  - How can we verify that the proportion of the sphere's area that is lit up is the same for: GLfloat light_position[]={0.5,0.5,0,0} or {1,1,0,0} or {-0.5,0.5,0,0} or etc?
  - How do we determine exactly these proportion? For example, is the porportion for {0.5, 0.5,0,0} greater than for {0.5,1.0,0,0}?
- Methods:
  - In the later on slides, we will graphically show how portion of objects are lit by the light specified its location and direction.

Case Study 1
Generic Directional Light Case

Case Study Setup:

- Object's setup:
  - Assume in the world coordinates we have one black sphere of radius 1, centered at (0,0,0).

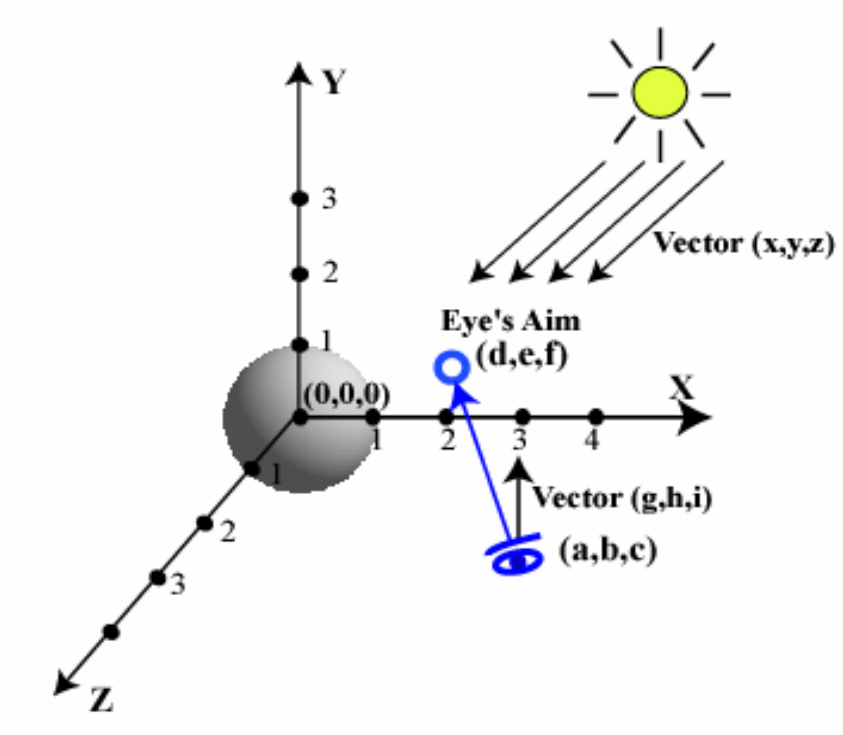
```
glutSolidSphere(1, 100,100);
```

- The camera's setup:

```
gluLookAt(a,b,c, d,e,f, g,h,i);
```

- Light's direction setup:

```
GLfloat light_position[] = (x,y,z,1);
glLightfv(GL_LIGHT0,
          GL_POSITION,
          light_position);
```



Goal:

Find out how will the sphere be lit up by the light?

Case Study 2
Directional Lighting

Case Study Setup:

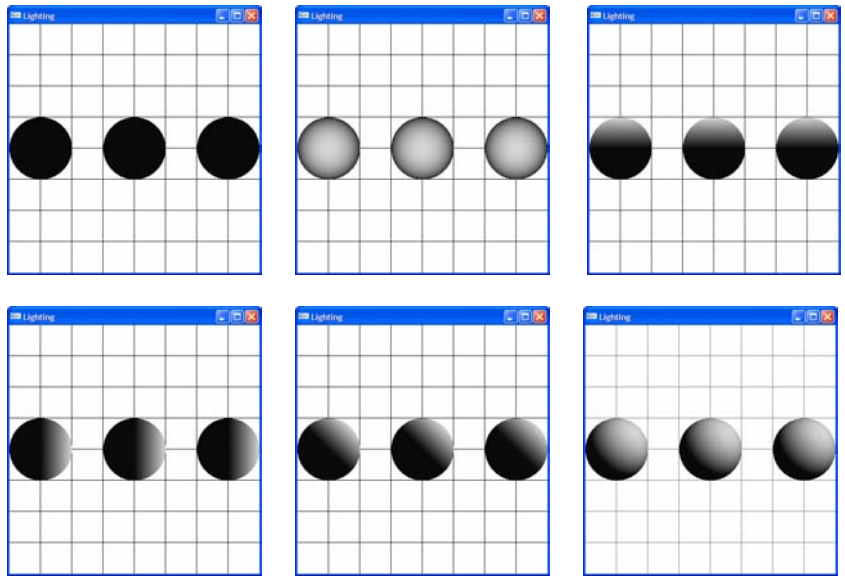
Now assume in world coordinates we have three black spheres of radius 1, centered, respectively, at (0,0,0), (-3,0,0) and (3,0,0).

Code for displaying the three spheres:

```
glutSolidSphere (1, 100,100); //center sphere

glPushMatrix();     //right sphere
glTranslatef(3,0,0);
glutSolidSphere (1, 100,100);
glPopMatrix();


glPushMatrix();     // left sphere
glTranslatef(-3,0,0);
glutSolidSphere (1, 100,100);
glPopMatrix();
```

Goal:

We will experiment with different light directions in world coordinates to see how the three spheres will be lit up.



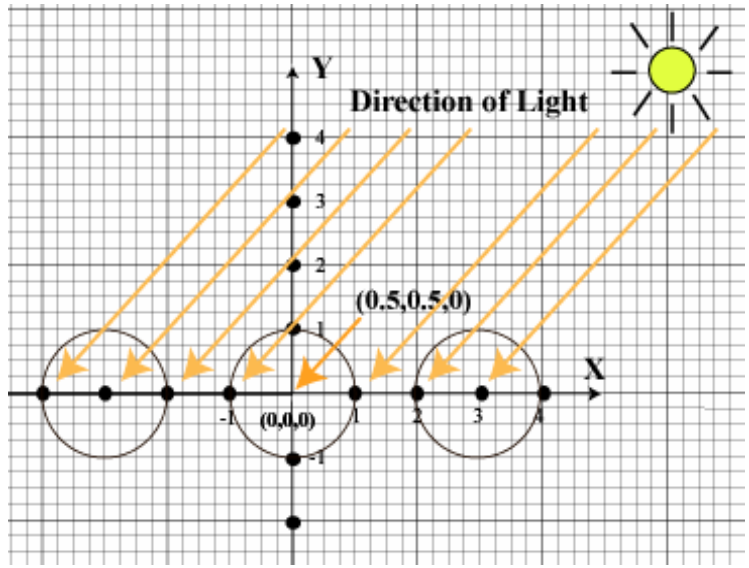
Directional Lighting
Set up the Projection View and Model View

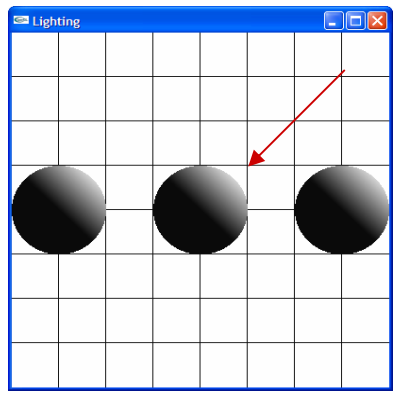
```
void reshape (int w, int h)
{
   glViewport (0, 0, (GLsizei) w, (GLsizei) h);

   glMatrixMode (GL_PROJECTION);
   glLoadIdentity();
   if (w <= h)
      glOrtho (-4, 4, -4*(GLfloat)h/(GLfloat)w,
         4*(GLfloat)h/(GLfloat)w, -4.0, 4.0);
   else
      glOrtho (-4*(GLfloat)w/(GLfloat)h,
         4*(GLfloat)w/(GLfloat)h, -4, 4, -4, 4.0);

   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
   glEnable(GL_DEPTH_TEST);
   gluLookAt(0,0,0,0,0,-1,0,1,0);
}

//main.c
glutReshapeFunc(reshape);
```

- What is achieved by the reshape function?
  - When w = h, the world coordinate bounding box is 8x8x8. If object is outside the bounding box, it cannot be viewed. If part of an object is outside, that part cannot be viewed.
  - When window is resized, and w>h or w<h, the object will maintain its shape, because reshape function adjusts the object's aspect ratio (width/height) according to window size.
  - Camera is explicitly set as located at (0,0,0), looking at (0,0,-1), which means camera is looking in the –z direction.
    - Camera is always at the geometric center of the bounding box whose size is the same as world coordinates' bounding box. The camera cannot see anything outside its box.

Directional Lighting
Set up one light source and the direction of the light
Example 1

```
void init(void)
{
   GLfloat light_position[] = {0.5, 0.5, 0, 0};

   // Enable light capability
   glEnable(GL_LIGHTING);
   // Enable one light, light 0
   glEnable(GL_LIGHT0);

   glLightfv(
     GL_LIGHT0,     // use light0
     GL_POSITION,   // uses "position as light
     parameter
     light_position);  // light's position

}
//in main
Init();
```





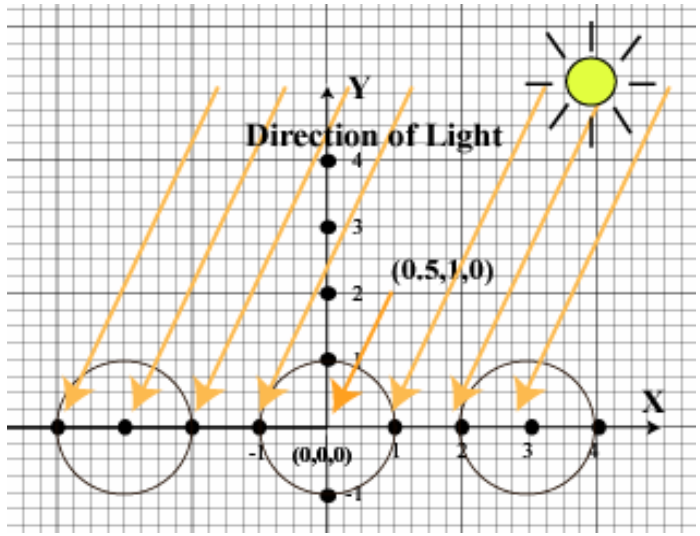
Note on light_position: (x, y, z, w)

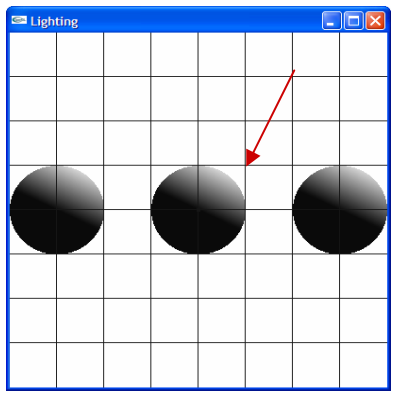
1. When w = 0, the light is treated as directional light source.

2. The direction of light is a vector specified by (x,y,z) of light_position and the origin (0,0,0).
   - (0.5,0.5,0) → (0,0,0).

Directional Lighting
Set up light source and the direction of the light
Example 2

```
void init(void)
{
   GLfloat light_position[] = {0.5, 1, 0, 0};

   // Enable light capability
   glEnable(GL_LIGHTING);
   // Enable one light, light 0
   glEnable(GL_LIGHT0);

   glLightfv(
     GL_LIGHT0,     // use light0
     GL_POSITION,   // uses "position as light
     parameter
     light_position);  // light's position

}
//in main
Init();
```





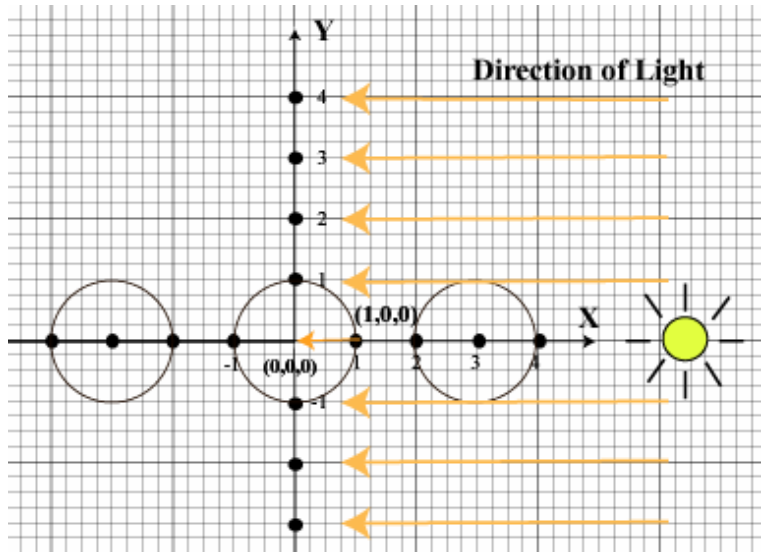
Note on light_position: (x, y, z,w)

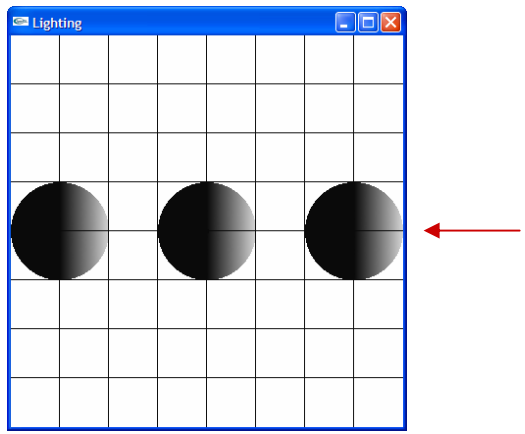
1. When w = 0, the light is treated as directional light source.

2. The direction of light is a vector specified by (x,y,z) of light_position and the origin (0,0,0).
   (0.5,1,0) → (0,0,0).

Directional Lighting
Set up light source and the direction of the light
Example 3

```
void init(void)
{
   GLfloat light_position[] = {1, 0, 0, 0};

   // Enable light capability
   glEnable(GL_LIGHTING);
   // Enable one light, light 0
   glEnable(GL_LIGHT0);

   glLightfv(
     GL_LIGHT0,     // use light0
     GL_POSITION,   // uses "position as light
     parameter
     light_position);  // light's position

}
//in main
Init();
```





Note on light_position: (x, y, z, w)

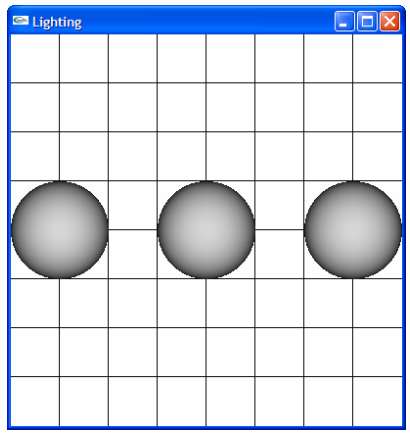
1. **When w = 0, the light is treated as directional light source.**

2. **The direction of light is a vector specified by (x,y,z) of light_position and the origin (0,0,0).**
   (1,0,0) → (0,0,0)

3. **Directional light does not deal with occlusion. The rightmost sphere should block any light shining on the two spheres to the left of it in real life, but in OpenGL directional lighting, it doesn't!**

Directional Lighting
Set up light source and the direction of the light
Example 4 (default)

```
void init(void)
{
   // Enable light capability
   glEnable(GL_LIGHTING);
   // Enable one light, light 0
   glEnable(GL_LIGHT0);

}
//in main
Init();
```



Note on light_position:

1. If we do not specify glLight(), then OpenGL has a default set up.

2. The default setting for light:

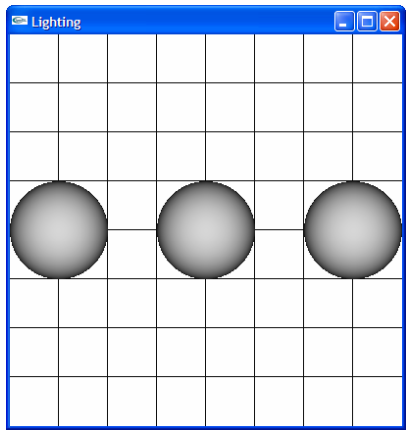
```
GLfloat light_position[] = {0, 0, 1, 0};  //light's position
glLightfv(GL_LIGHT0,    // use light0
          GL_POSITION,  // uses "position as light parameter
          light_position);  // light's position
```

3. Therefore, the default light source is directional, parallel to, and in the direction of, the -*z* axis.
   (0,0,1) → (0,0,0)

Directional Lighting
Set up light source and the direction of the light
Example 5 (a slight light position variation from example 4)

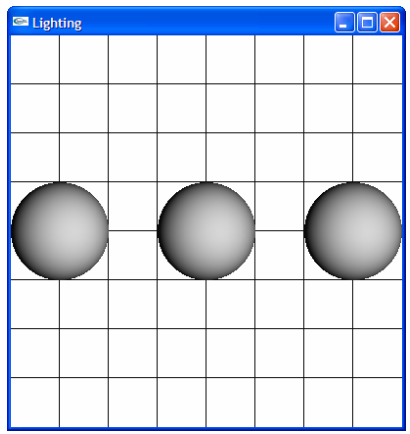
Example 4 (previous case)

```
default setting:
    GLfloat light_position[] = {0, 0, 0, 0};

    glLightfv(
            GL_LIGHT0,    // use light0
            GL_POSITION,  // uses "position as light parameter
            light_position);  // light's position
```



Example 5

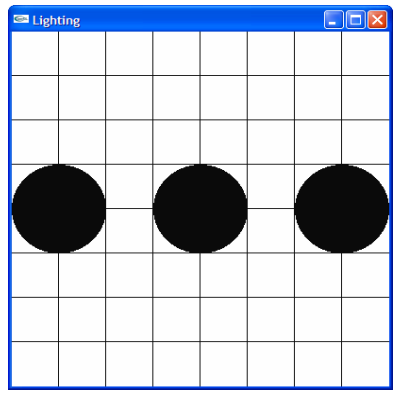
```
   GLfloat light_position[] = {0.3, 0, 0, 0};

   glLightfv(
    GL_LIGHT0,    // use light0
    GL_POSITION,  // uses "position as light parameter
    light_position);  // light's position
```



Note:
A slight variation of light position will slightly change the proportion of light lighting up the objects.

Directional Lighting
Set up light source and the direction of the light
Example 6

```
void init(void)
{
   GLfloat light_position[] = {0, 0, 0, 0};

   // Enable light capability
   glEnable(GL_LIGHTING);
   // Enable one light, light 0
   glEnable(GL_LIGHT0);

   glLightfv(
     GL_LIGHT0,     // use light0
     GL_POSITION,  // uses "position as light
     parameter
     light_position);  // light's position

}
//in main
Init();
```



Note on light_position: (x, y, z, w)

1. When w = 0, the light is treated as directional light source.

2. The direction of light is a vector specified by (x, y, z) of light_position and the origin (0,0,0). In this case the light vector is (0,0,0), which is (0,0,0) → (0,0,0) (undefined). Therefore, none of the three spheres is lit up.

Camera's position, not where the camera is aiming at, affect light's position

- In the previous cases, the camera's position is set to default position, which is (0,0,0);
- Because light's position in OpenGL is relative to camera's position, camera's position has the effect of shifting the relative position of light and the point the light is aiming at on the object.
- However, where the camera is aiming at has no effect of light's position relative to the point the light is aiming at on the object.
  - So: Camera is at (a,b,c), aimed at (d,e,f) is the same as camera as (a,b,c), aimed at (x,y,z) (where (x,y,z) is different from (d,e,f)), as far as the lighting on the objects is concerned.

Case Study 3
Directional Light with Camera at location other than (0,0,0)

Case Study Setup:

Now assume in world coordinates we have three black spheres of radius 1, centered, respectively, at (0,0,0), (-3,0,0) and (3,0,0).

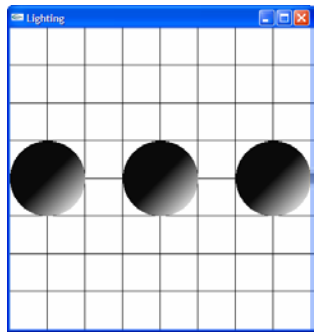
Code for displaying the three spheres:

```
glutSolidSphere (1, 100,100);

glPushMatrix();
glTranslatef(3,0,0);
glutSolidSphere (1, 100,100);
glPopMatrix();


glPushMatrix();
glTranslatef(-3,0,0);
glutSolidSphere (1, 100,100);
glPopMatrix();
```

Goal:

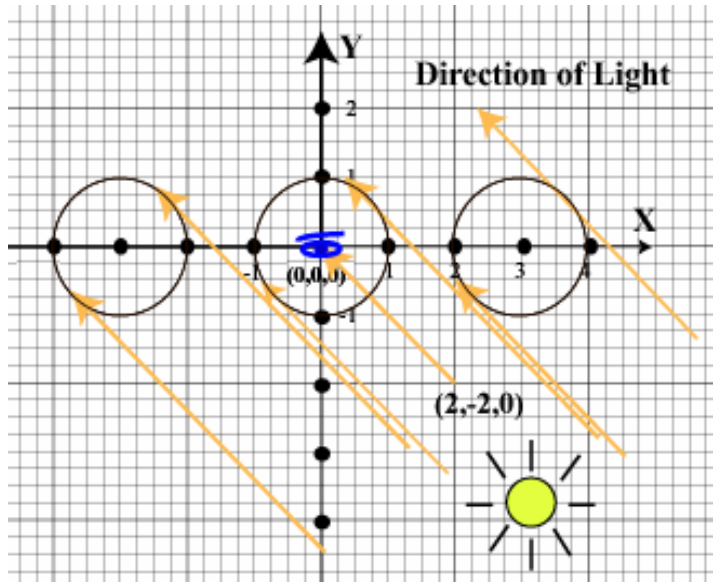
We will experiment with directional light's location in relation to camera's location by setting camera's location other than (0,0,0).

Front View

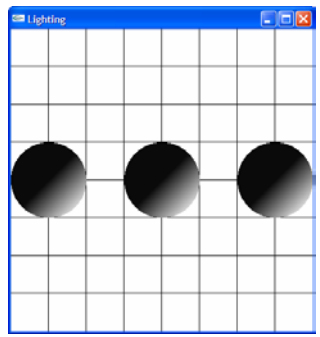


Case Study 3
Directional Light with Camera at location (0,0,1), aiming at (0,0,-1)

```
void init(void)
{
   // spot light's position
   GLfloat light_position[] = {2, -2, 0, 0};

   glEnable(GL_LIGHTING);  // Enable light
   glEnable(GL_LIGHT0);    // Enable one light,
     light 0

 // specify spot light position at (2,-2,0)
   glLightfv(           GL_LIGHT0, GL_POSITION,
     light_position);
}
//in main
Init();
// gluLookAt in reshape function
gluLookAt(1,0,0,1,0,-2,0,1,0);
```



Front View



Note on light_position: (x, y, z, w)

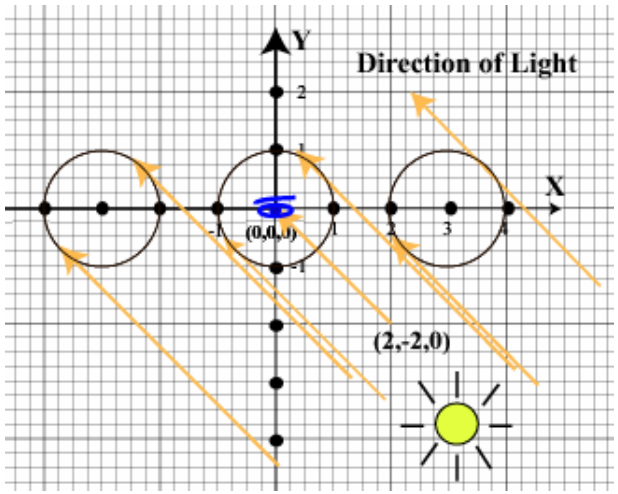
1. When w = 0, the light is treated as directional light source.

2. The direction of light is a vector specified by (x,y,z) of light_position and the origin (0,0,0).
   - (2,-2,0) → (0,0,0).

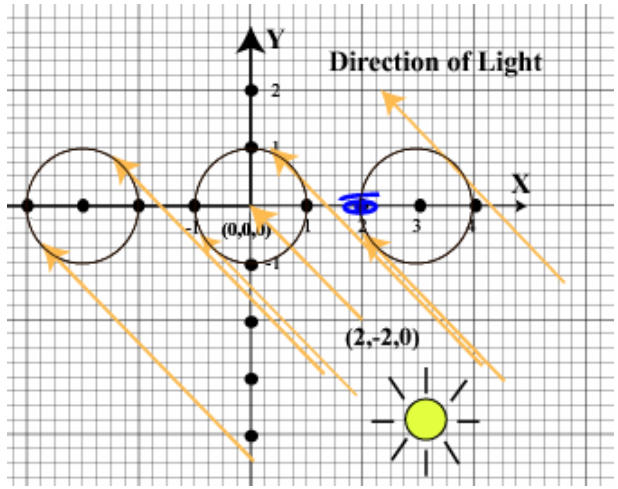
Case Study 3 - Compare Two Cases

1. Camera at (0,0,0), aiming at (0,0,-1). glLight's direction vector (2,-2,0)
2. Camera at location (1,0,0), aiming at (1,0,-2). glLight's direction vector (2,-2,0)

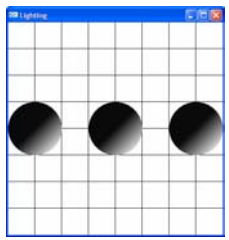
Case 1



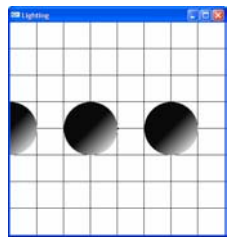
Case 2



Case 1 FrontView:



Case 2 FrontView:



Although in the above two cases, camera's locations are different, since light's actual direction is specified in the world coordinates, and have nothing to do with the camera's coordinates, therefore, the outputs of both cases show that the three spheres are lit by the light from the same direction.

This is the same as the light's direction for spotlight, which is also specified in world coordinates by a vector.

What's ambient light?

- **A*mbient*** illumination is light that is so scattered so much by the environment that its direction is impossible to determine—it seems to come from all directions

  – 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 (and perhaps various other objects in that room).

  – This bounced light (called *ambient*) is assumed to be so scattered that there is no way to tell its original direction, but it disappears if a particular light source is turned off.

  – Sunlight streaming through a window, in combination with light coming from fluorescent light fixtures in the ceiling, produces a room that appears to be lit up everywhere in more or less the same way.

  – A particular characteristic of ambient light is that it lights up a scene without making any discernible shadows.

Case 4
Study of ambient light

Case Study Setup:

- Object Setup:
  Assume in the world coordinates we have one glut sphere of radius 2, centered at (0,0,0).

```
glutSolidSphere (2, 100,100);
```

- Light's position

```
GLfloat light_position[] = {3, 3, 3, 0 };
glLightfv(GL_LIGHT0,
          GL_POSITION,
          light_position);
```

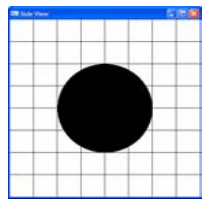
Goal:
We will change the value a (blue color value for light) and b (blue color value for material) to see how the object changes color.

```
// lighting values and code
  GLfloat fLtAmbient[4] = {0,0,a,1};

  // material values and code
  GLfloat fMatAmbient[4] = {0,0,b,1};

glLightfv(GL_LIGHT0, GL_AMBIENT, fLtAmbient);

glMaterialfv(GL_FRONT, GL_AMBIENT, fMatAmbient);
```

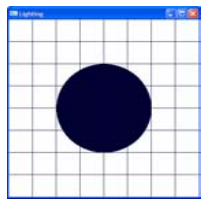
Colors are defined in levels

- Level 0 (blue not lit)
- Level 2 (blue slightly lit)
- …
- Level n (blue strongly lit)

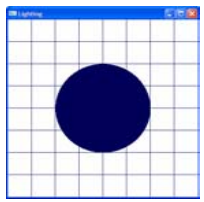
Black



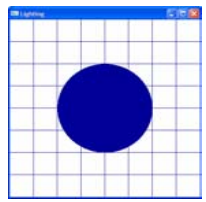
Blue not lit
Level 0



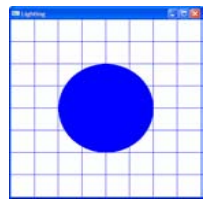
Level 1



Level 3



Level 5



The code for light setting

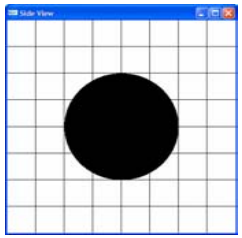
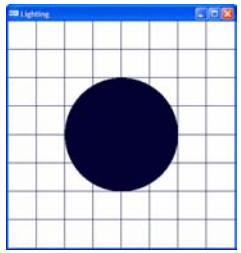
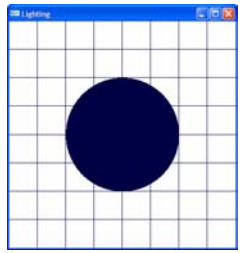
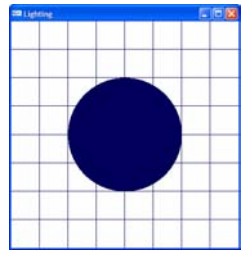
Later on, we will change “a” and “b” to see changes in the color of the sphere.

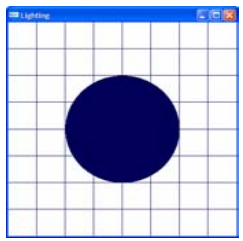
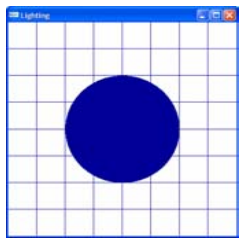
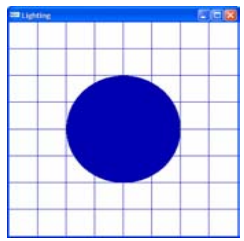
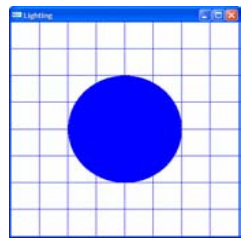
```
// material and lighting property
  // lighting values and code
  GLfloat fLtAmbient[4] = {0,0,a,1};  // red = 0, green = 0, blue = a
  GLfloat fLtDiffuse[4] = {0,0,0,1};
  GLfloat fLtSpecular[4] = {0,0,0,1};
  GLfloat light_position[] = {3, 3, 3, 0 };

  // material values and code
  GLfloat fMatAmbient[4] = {0,0,b,1}; // red = 0, green = 0, blue = b
  GLfloat fMatDiffuse[4] = {0,0,0,1};
  GLfloat fMatSpecular[4] = {0,0,0,1};
  GLfloat fShine = 0;


  glEnable(GL_LIGHTING);
  // enalbe lighting property
  glLightfv(GL_LIGHT0, GL_AMBIENT, fLtAmbient);
  glLightfv(GL_LIGHT0, GL_DIFFUSE, fLtDiffuse);
  glLightfv(GL_LIGHT0, GL_SPECULAR, fLtSpecular);
  glLightfv(GL_LIGHT0, GL_POSITION, light_position);

  glEnable(GL_LIGHT0);


  // enable material property
  glMaterialfv(GL_FRONT, GL_AMBIENT, fMatAmbient);
  glMaterialfv(GL_FRONT, GL_DIFFUSE, fMatDiffuse);
  glMaterialfv(GL_FRONT, GL_SPECULAR, fMatSpecular);
  glMaterialf(GL_FRONT, GL_SHININESS, fShine);
```

Note: the ordering of these lines of code does not matter. You can enable lighting property first, then material or vice versa.

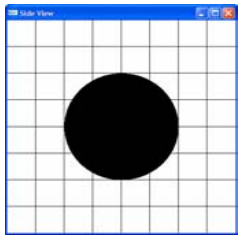
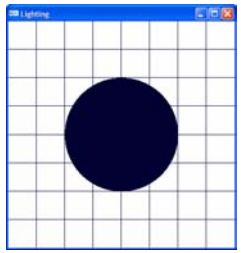
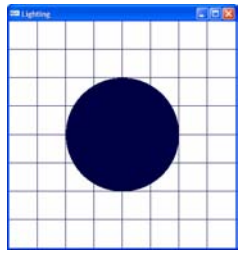
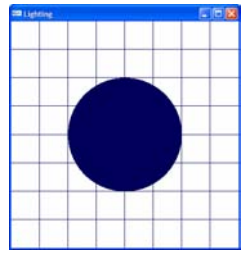
The ordering of the codes presented here is to make the logic of code easy for reader to follow.

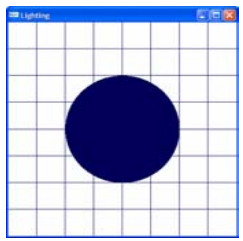
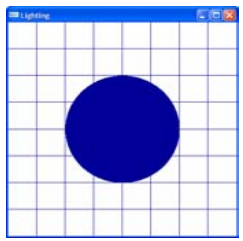
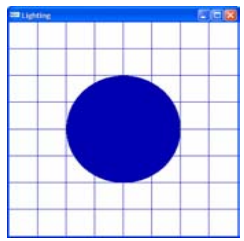
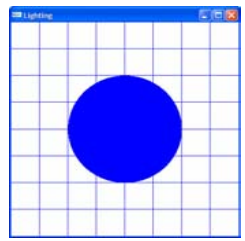
Ambient light does not have direction

	Case 1 Black	Case 2 Level 0 blue	Case 3 Level 1 blue	Case 4 Level 2 blue
				
Ambient Light blue - a:	**0**	**0**	**0.7**	**0.3**
Ambient Material blue -b:	**0**	**1**	**0.3**	**0.7**

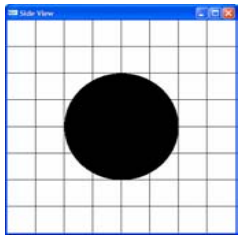
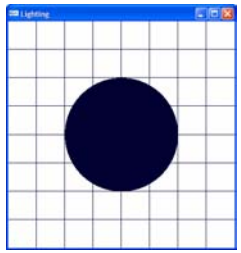
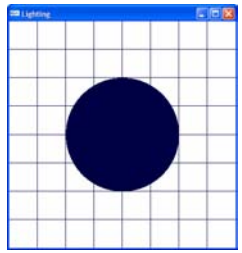
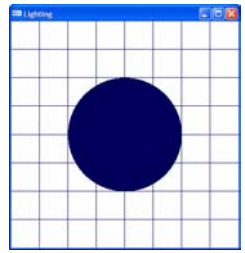
	Case 5 Level 2 blue	Case 6 Level 3 blue	Case 7 Level 4 blue	Case 8 Level 5 blue
				
Ambient Light blue - a:	**0.5**	**1**	**0.5**	**1**
Ambient Material blue - b:	**0.5**	**0.5**	**1**	**1**

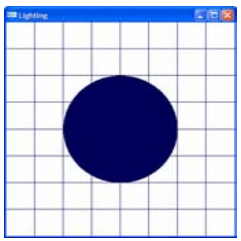
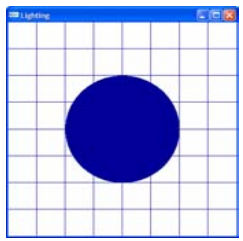
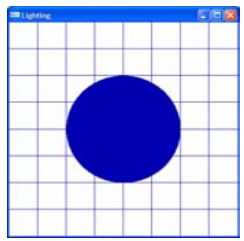
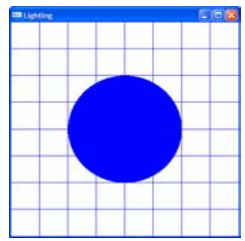
Case 1 shows that the sphere is black, because both ambient material's blue and ambient light's blue is set to 0.

	Case 1 Black	Case 2 Level 0 blue	Case 3 Level 1 blue	Case 4 Level 2 blue
				
Ambient Light blue - a:	0	0	0.7	0.3
Ambient Material blue - b:	0	1	0.3	0.7

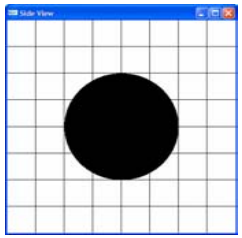
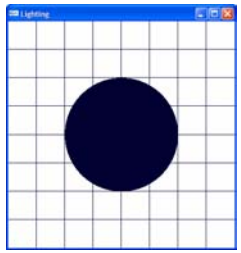
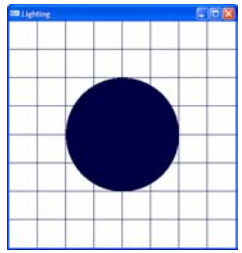
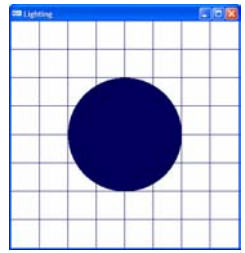
	Case 5 Level 2 blue	Case 6 Level 3 blue	Case 7 Level 4 blue	Case 8 Level 5 blue
				
Ambient Light blue - a:	0.5	1	0.5	1
Ambient Material blue - b:	0.5	0.5	1	1

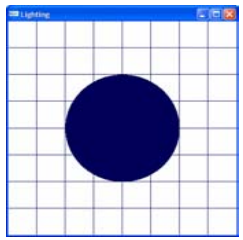
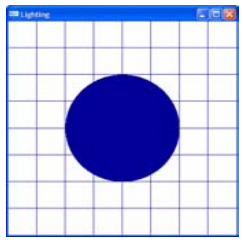
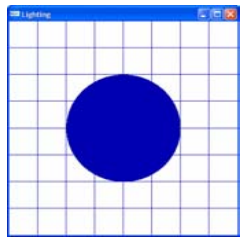
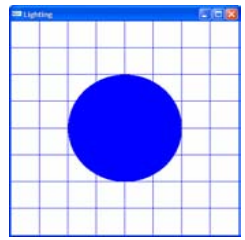
In Case 2, although material's ambient blue is maximum, 1, light's ambient blue = 0, therefore, the object is dark blue.

	Case 1 Black	Case 2 Level 0 blue	Case 3 Level 1 blue	Case 4 Level 2 blue
				
Ambient Light blue - a:	0	0	0.7	0.3
Ambient Material blue - b:	0	1	0.3	0.7

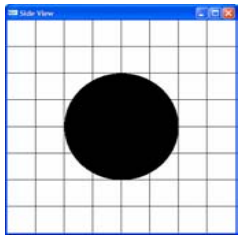
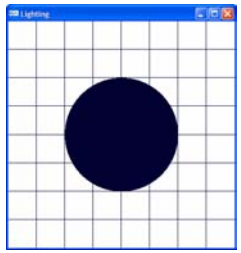
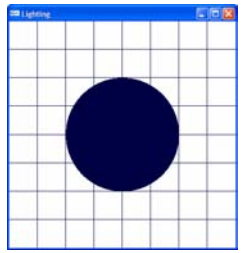
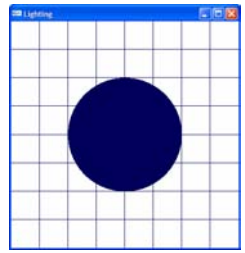
	Case 5 Level 2 blue	Case 6 Level 3 blue	Case 7 Level 4 blue	Case 8 Level 5 blue
				
Ambient Light blue - a:	0.5	1	0.5	1
Ambient Material blue - b:	0.5	0.5	1	1

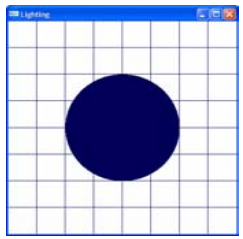
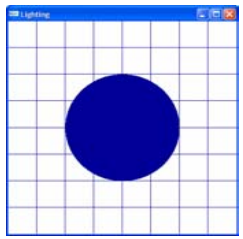
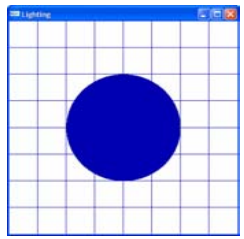
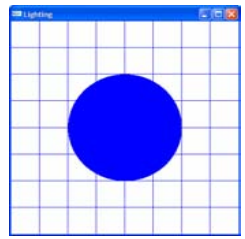
Case 4 is blue lit lighter than in Case 3. Case 7 is blue lit lighter than in Case 6. This is because the material's ambient blue is brighter than the light's ambient blue.

	Case 1 Black	Case 2 Level 0 blue	Case 3 Level 1 blue	Case 4 Level 2 blue
				
Ambient Light blue - a:	0	0	0.7	0.3
Ambient Material blue - b:	0	1	0.3	0.7

	Case 5 Level 2 blue	Case 6 Level 3 blue	Case 7 Level 4 blue	Case 8 Level 5 blue
				
Ambient Light blue - a:	0.5	1	0.5	1
Ambient Material blue - b:	0.5	0.5	1	1

Case 8 shows a blue sphere lit lightest, because both material's ambient blue and light's ambient blue are set to maximum 1.

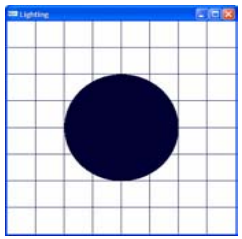
	Case 1 Black	Case 2 Level 0 blue	Case 3 Level 1 blue	Case 4 Level 2 blue
				
Ambient Light blue - a:	0	0	0.7	0.3
Ambient Material blue - b:	0	1	0.3	0.7

	Case 5 Level 2 blue	Case 6 Level 3 blue	Case 7 Level 4 blue	Case 8 Level 5 blue
				
Ambient Light blue - a:	0.5	1	0.5	1
Ambient Material blue - b:	0.5	0.5	1	1

Case 5

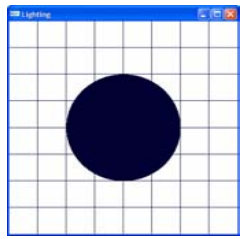
The following three spheres display level 0, blue not lit, because as long as the light's blue component is "0", although the material is set blue "1", the object displays the same shade of blue color.

Case 1
Level 0 blue



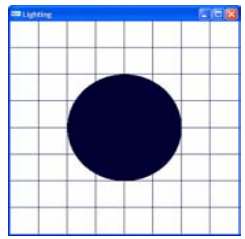
Ambient Light's color
red = 0, green = 0, blue = 0:

Case 2
Level 0 blue

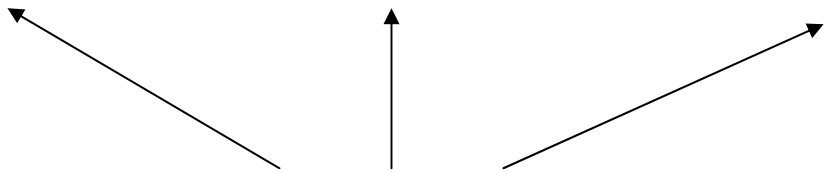


Ambient Light's color
red = 0, green = 1, blue = 0:

Case 3
Level 0 blue



Ambient Light's color
red = 1, green = 0, blue = 0:

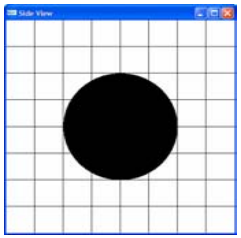


Ambient material's
red = 0, green = 0, blue = 1:

Case 6

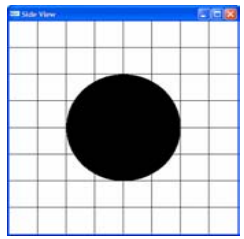
All the following three display black ball, because the material has no color.

Case 1
Black



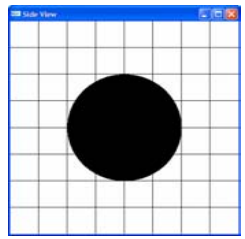
Ambient Light's color
red = 0, green = 0, blue = 1:

Case 2
Black

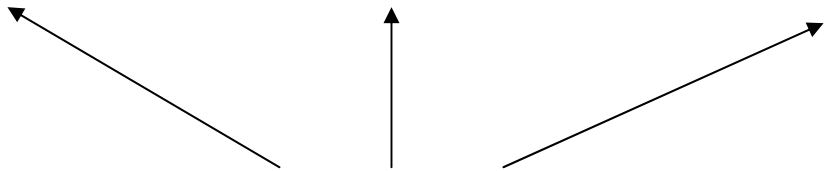


Ambient Light's color
red = 0, green = 1, blue = 0:

Case 3
Black



Ambient Light's color
red = 1, green = 0, blue = 0:



Ambient material's
red = 0, green = 0, blue = 0:

Conclusions about ambient components of light and material

- Ambient light does not have direction:
  - Although the light's position is (3,3,3) and the object is located at (0,0,0), because the diffuse and the specular components are all zeros, and ambient light does not have direction, therefore, the whole object (a sphere) gets light and appears to be of the same color. (In our example, from dark blue to light blue)
- When the material's ambient component is X (e.g. blue), then the object shows that color X. The larger the X (0<X<1), the deeper the color of the object. In the following case, objects drawn afterwards show blue color.

```
GLfloat fMatAmbient[4] = {0,0,X,1); // blue = X amount
glMaterialf(GL_FRONT, GL_AMBIENT, fMatAmbient);
```

- When the material's ambient component is X (e.g. blue), but the color of light's ambient component for the X is 0, then the object is extremely dark. The larger the Y (0<Y<1), then the brighter the object of X color (in this case, blue).

```
GLfloat fLtAmbient[4] = {0,0,Y,1); //blue=Y amount.When Y is larger, the color X is brighter.
glLightfv(GL_LIGHT0, GL_AMBIENT, fLtAmbient);

GLfloat fMatAmbient[4] = {0,0,X,1); // blue = X amount
glMaterialf(GL_FRONT, GL_AMBIENT, fMatAmbient);
```

Conclusion about ambient components of light and material, Continued …

- Our experiment shows that the material ambient component X (e.g. blue) has larger effect on the brightness of an object of color X than the corresponding lgihting ambient component Y (same color as X) For example, Case A shows a brighter blue object than Case B.

  – Y2 – Y1 = a (Y1 > Y2) X1 – X2 = a (X1 > X2)

- Case A

```
GLfloat fLtAmbient[4] = {0,0,Y1,1); // blue = Y1 amount
glLightfv(GL_LIGHT0, GL_AMBIENT, fLtAmbient);

GLfloat fMatAmbient[4] = {0,0,X1,1); // blue = X1 amount
glMaterialf(GL_FRONT, GL_AMBIENT, fMatAmbient);
```

- Case B

```
GLfloat fLtAmbient[4] = {0,0,Y2,1); // blue = Y2 amount
glLightfv(GL_LIGHT0, GL_AMBIENT, fLtAmbient);

GLfloat fMatAmbient[4] = {0,0,X2,1); // blue = X2 amount
glMaterialf(GL_FRONT, GL_AMBIENT, fMatAmbient);
```

What's diffuse light?

- In a *diffuse* interaction, the reflected light is scattered equally in all directions.

- Unlike ambient light which will light the entirely object equally, the diffuse light interaction only happen on the side of object facing the light.

  – Any light coming from a particular position or direction probably has a diffuse component. The effect of diffuse light is that it lights up the side of object facing the light.

Case 7
Study of diffuse light

Case Study Setup:

- Object Setup:
  Assume in the world coordinates we have one glut sphere of radius 2, centered at (0,0,0).

```
glutSolidSphere (2, 100,100);
```

- Light's position

```
GLfloat light_position[] = {3, 3, 3, 0 };
glLightfv(GL_LIGHT0,
          GL_POSITION,
          light_position);
```

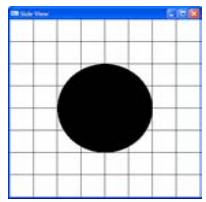
Goal:
We will change the value a (green color value for light) and b (green color value for material to see how the object changes color.

```
// lighting values and code
  GLfloat fLtDiffuse[4] = {0,0,a,1};

  // material values and code
  GLfloat fMatDiffuse[4] = {0,b,0,1};

glLightfv(GL_LIGHT0, GL_AMBIENT, fLtDiffuse);

glMaterialfv(GL_FRONT, GL_AMBIENT, fMatDiffuse);
```

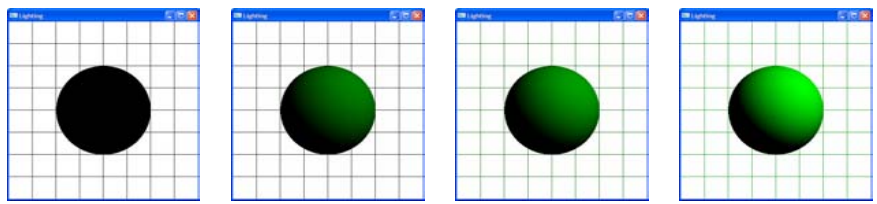
Colors are defined in levels

- Level 0 (green not lit)
- Level 2 (green slightly lit)
- …
- Level n (green strongly lit)

Black



Level 0 **Level 1** **Level 3** **Level 5**

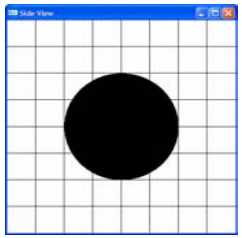
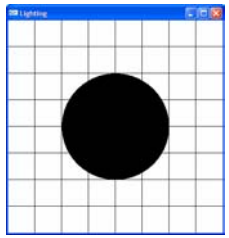
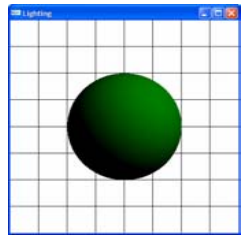
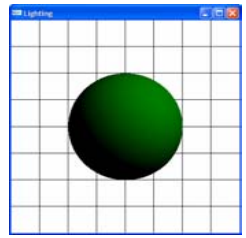


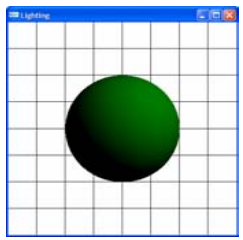
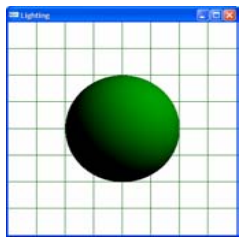
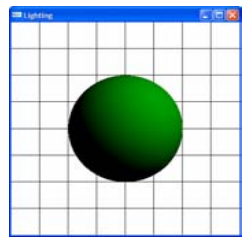
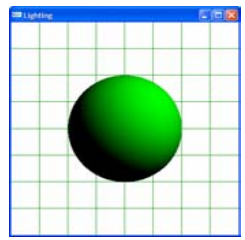
The code for light setting

Later on, we will change “a” and “b” to see what color will the sphere be displayed.

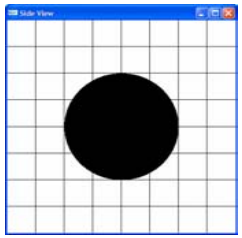
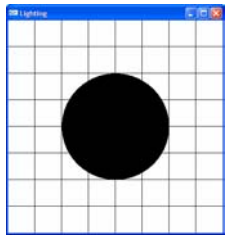
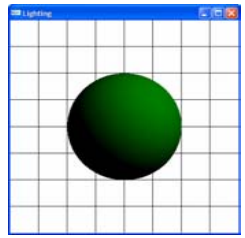
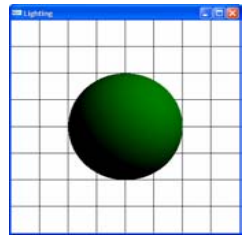
```
// material and lighting property
  // lighting values and code
  GLfloat fLtAmbient[4] = {0,0,0,1};
  GLfloat fLtDiffuse[4] = {0,a,0,1}; // red = 0,  green = a, blue = 0

  GLfloat fLtSpecular[4] = {0,0,0,1};
  GLfloat light_position[] = {3, 3, 3, 0 };

  // material values and code
  GLfloat fMatAmbient[4] = {0,0,0,1};
  GLfloat fMatDiffuse[4] = {0,b,0,1}; // red = 0, green = b, blue = 0
  GLfloat fMatSpecular[4] = {0,0,0,1};
  GLfloat fShine = 0;


  glEnable(GL_LIGHTING);
  // enalbe lighting property
  glLightfv(GL_LIGHT0, GL_AMBIENT, fLtAmbient);
  glLightfv(GL_LIGHT0, GL_DIFFUSE, fLtDiffuse);
  glLightfv(GL_LIGHT0, GL_SPECULAR, fLtSpecular);
  glLightfv(GL_LIGHT0, GL_POSITION, light_position);

  glEnable(GL_LIGHT0);


  // enable material property
  glMaterialfv(GL_FRONT, GL_AMBIENT, fMatAmbient);
  glMaterialfv(GL_FRONT, GL_DIFFUSE, fMatDiffuse);
  glMaterialfv(GL_FRONT, GL_SPECULAR, fMatSpecular);
  glMaterialf(GL_FRONT, GL_SHININESS, fShine);
```

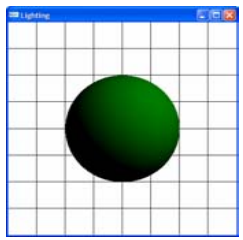
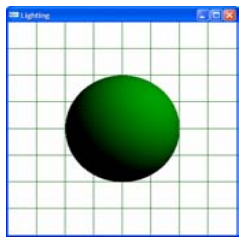
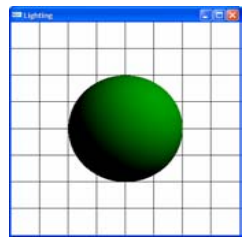
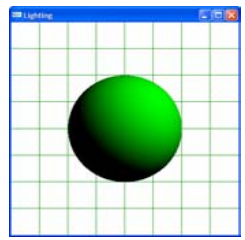
Diffuse takes the direction of light.

	Case 1 Black	Case 2 Level 0 green	Case 3 Level 1 green	Case 4 Level 1 green
				
Ambient Light green - a:	**0**	**0**	**1**	**0.5**
Ambient Material green -b:	**0**	**1**	**0.5**	**1**

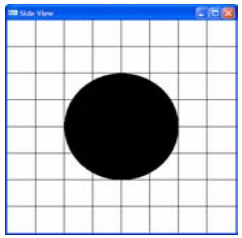
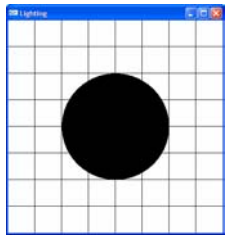
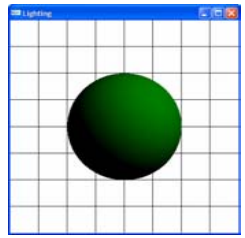
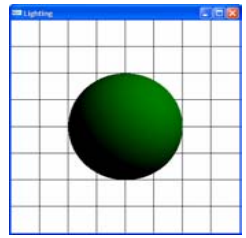
	Case 5 Level 2 green	Case 6 Level 3 green	Case 7 Level 4 green	Case 8 Level 5 green
				
Ambient Light green - a:	**0.7**	**0.9**	**0.7**	**1**
Ambient Material green - b:	**0.7**	**0.7**	**0.9**	**1**

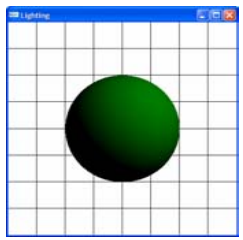
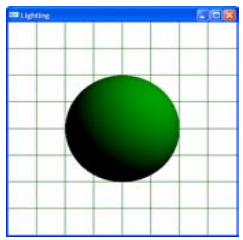
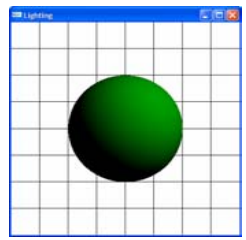
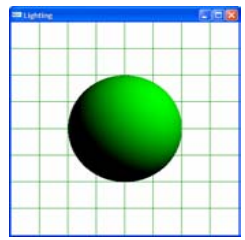
Case 1 shows that the sphere is black, because both material's diffuse green and light's diffuse green is set to 0.

	Case 1 Black	Case 2 Level 0 green	Case 3 Level 1 green	Case 4 Level 1 green
				
Ambient Light green - a:	0	0	1	0.5
Ambient Material green -b:	0	1	0.5	1

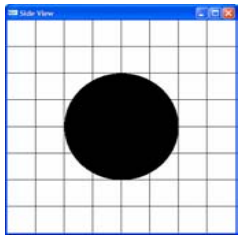
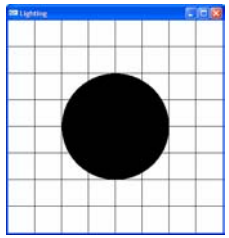
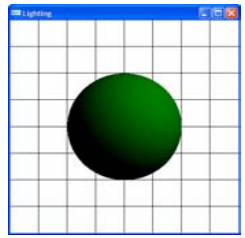
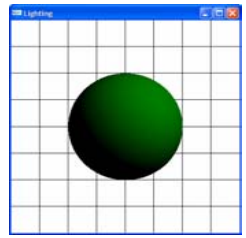
	Case 5 Level 2 green	Case 6 Level 3 green	Case 7 Level 4 green	Case 8 Level 5 green
				
Ambient Light green - a:	0.7	0.9	0.7	1
Ambient Material green - b:	0.7	0.7	0.9	1

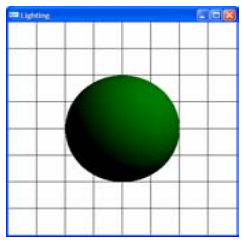
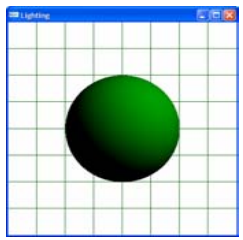
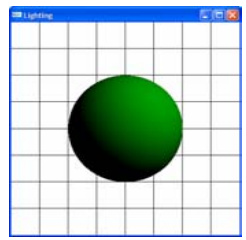
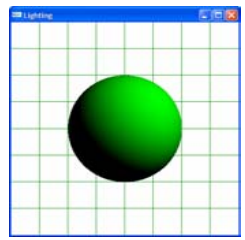
In Case 2, although material's diffuse green is maximum, 1, but light's diffuse blue = 0, therefore, the object is green hardly lit.

	Case 1 Black	Case 2 Level 0 green	Case 3 Level 1 green	Case 4 Level 1 green
				
Ambient Light green - a:	0	0	1	0.5
Ambient Material green -b:	0	1	0.5	1

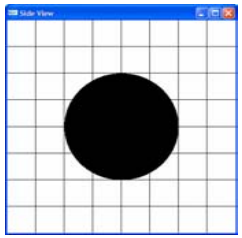
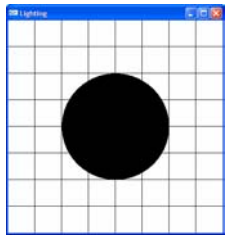
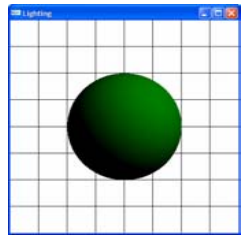
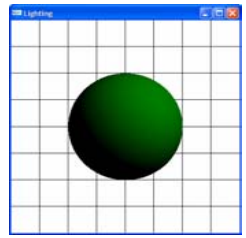
	Case 5 Level 2 green	Case 6 Level 3 green	Case 7 Level 4 green	Case 8 Level 5 green
				
Ambient Light green - a:	0.7	0.9	0.7	1
Ambient Material green - b:	0.7	0.7	0.9	1

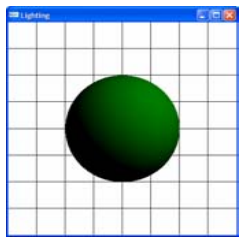
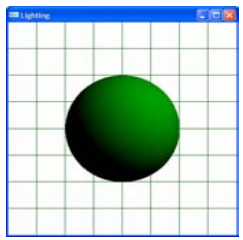
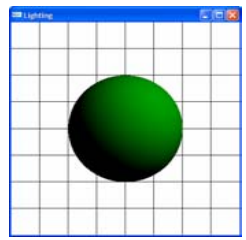
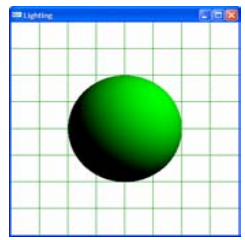
Case 3 is green lighter lit than case 4. Case 6 is green lighter lit than case 7. This is because the material's diffuse green is less bright than the light's diffuse green.

	Case 1 Black	Case 2 Level 0 green	Case 3 Level 1 green	Case 4 Level 1 green
				
Ambient Light green - a:	0	0	1	0.5
Ambient Material green -b:	0	1	0.5	1

	Case 5 Level 2 green	Case 6 Level 3 green	Case 7 Level 4 green	Case 8 Level 5 green
				
Ambient Light green - a:	0.7	0.9	0.7	1
Ambient Material green - b:	0.7	0.7	0.9	1

Case 4 shows that the sphere is green lit strongest, because both material's diffuse green and light's diffuse green is set to maximum 1.

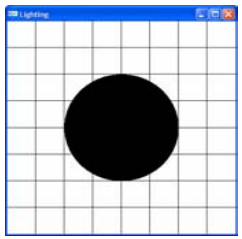
	Case 1 Black	Case 2 Level 0 green	Case 3 Level 1 green	Case 4 Level 1 green
				
Ambient Light green - a:	0	0	1	0.5
Ambient Material green -b:	0	1	0.5	1

	Case 5 Level 2 green	Case 6 Level 3 green	Case 7 Level 4 green	Case 8 Level 5 green
				
Ambient Light green - a:	0.7	0.9	0.7	1
Ambient Material green - b:	0.7	0.7	0.9	1

Case 8

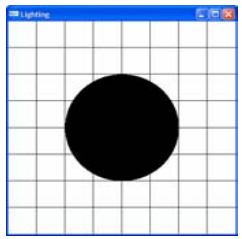
The following three display level 0, green hardly lit, because as long as the light's diffuse green component is "0" (althought the material is set to diffuse green "1"), the object displays the same shade of green color.

Case 1
Level 0 green



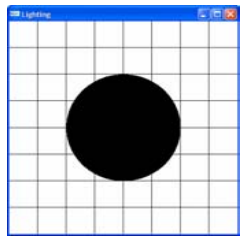
diffuse Light's color
red = 0, green = 0, blue = 0:

Case 2
Level 0 green



diffuse Light's color
red = 0, green = 0, blue = 1:

Case 3
Level 0 green



diffuse Light's color
red = 1, green = 0, blue = 0:

diffuse material's
red = 0, green = 1, blue = 0:

Conclusion about diffuse components of light and material

- Diffuse light takes consideration of light's direction.
- We can graphically draw the objects and light's direction in world coordinate on paper, and where the light hits the object will be lit up by the light.
- The larger the value of diffuse light and diffuse material of color X, the more lit up the color X on the object towards the light's source.
- When the material's ambient component is X (e.g. green), then the object shows that color X. The larger the X (0<X<1), the deeper the color of the object. In the following case, objects drawn afterwards show blue color.

```
GLfloat fMatDiffuse[4] = {0,X,0,1); // green = X amount
glMaterialf(GL_FRONT, GL_AMBIENT, fMatDiffuse);
```

- When the material's diffuse component is X (e.g. green), but the light's diffuse component for the X is 0, then the object is extremely dark. The larger the Y (0<Y<1), then the brighter the object of X color (in this case, green).

```
GLfloat fLtDiffuse[4] = {0,Y,0,1); // green = Y amount. The larger the Y, the brighter the X color
glLightfv(GL_LIGHT0, GL_AMBIENT, fLtDiffuse);

GLfloat fMatDiffuse[4] = {0,X,0,1); // green = X amount
glMaterialf(GL_FRONT, GL_AMBIENT, fMatDiffuse);
```

Conclusion about diffuse components of light and material, Continued …

- Our experiment shows that the light's diffuse component X (e.g. green) has larger effect on the brightness of object of color X than the corresponding material's diffuse component Y (same color as X) For example, Case B shows object has more green lit up than Case A.
  - This is opposite of ambient characteristic, where under the same situation, case A will be lit brighter than case B.

    - `Y2 – Y1 = a  (Y1 > Y2)` `X1 - X2 = a  (X1 > X2)`

- Case A

```
GLfloat fLtAmbient[4] = {0,0,Y1,1); // blue = Y1 amount
glLightfv(GL_LIGHT0, GL_AMBIENT, fLtAmbient);

GLfloat fMatAmbient[4] = {0,0,X1,1); // blue = X1 amount
glMaterialf(GL_FRONT, GL_AMBIENT, fMatAmbient);
```

- Case B

```
GLfloat fLtAmbient[4] = {0,0,Y2,1); // blue = Y2 amount
glLightfv(GL_LIGHT0, GL_AMBIENT, fLtAmbient);

GLfloat fMatAmbient[4] = {0,0,X2,1); // blue = X2 amount
glMaterialf(GL_FRONT, GL_AMBIENT, fMatAmbient);
```

Case 9
Combining ambient and diffuse light

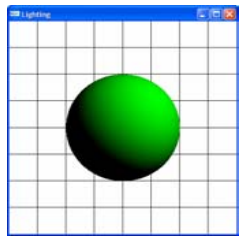
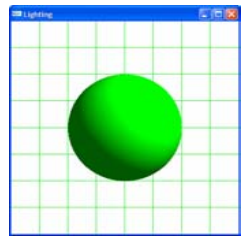
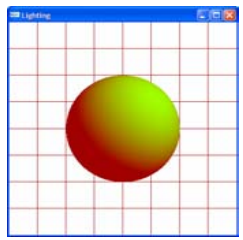
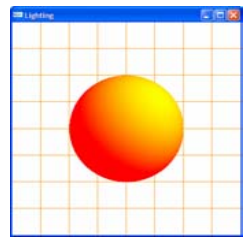
Case Study Setup:

- Object Setup:
  Assume in the world coordinates we have one glut sphere of radius 2, centered at (0,0,0).

  glutSolidSphere (2, 100,100);

- Light's position

  GLfloat light_position[] = {3, 3, 3, 0 };
  glLightfv(GL_LIGHT0,
  GL_POSITION,
  light_position);

Goal:
We will change the value of both ambient and diffuse components of light source and object's material to see how the object change color.

```
// lighting values and code
  GLfloat fLtAmbient[4] = {a1,a2,a3,1};
  GLfloat fLtDiffuse[4] = {b1,b2,b3,1};


  // material values and code
  GLfloat fMatDiffuse[4] = {c1,c2,c3,1};
  GLfloat fMatDiffuse[4] = {d1,d2,d3,1};

  glLightfv(GL_LIGHT0, GL_AMBIENT, fLtAmbient);
  glMaterialfv(GL_FRONT, GL_AMBIENT, fMatDiffuse);
```

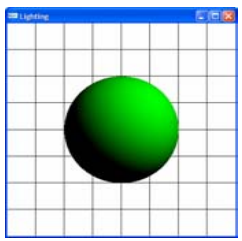
Using ambient green combined with diffuse green

1. **In choice B, some ambient green lights up the whole ball, but not in choice A.**
   - **Ambient light of material and light source light up the whole object. Diffuse light of material and light source lit up the object's side facing the light source.**

2. **In choice B, some ambient green adds diffuse green to make the side of object facing the light brighter, but not in choice A.**
   - **When using both ambient and diffuse light with the light source and material of object, the ambient and diffuse color are added.**

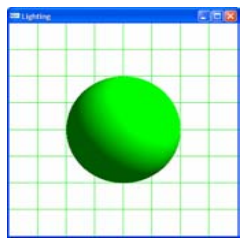
Choice A
Without ambient green

```
// material and lighting property
  // lighting values and code
  GLfloat fLtAmbient[4] = {0,0,0,1};//green=0
  GLfloat fLtDiffuse[4] = {0,1,0,1};//green=1
  GLfloat fLtSpecular[4] = {0,0,0,1};
  GLfloat light_position[] = {3, 3, 3, 0 };

  // material values and code
  GLfloat fMatAmbient[4] = {0,0,0,1};//green=0
  GLfloat fMatDiffuse[4] = {0,1,0,1};//green=1
  GLfloat fMatSpecular[4] = {0,0,0,1};
```



Choice B
With ambient green

```
// material and lighting property
// lighting values and code
GLfloat fLtAmbient[4] = {0,0.5,0,1};//green=0.5
GLfloat fLtDiffuse[4] = {0,1,0,1};//green=1
GLfloat fLtSpecular[4] = {0,0,0,1};
 GLfloat light_position[] = {3, 3, 3, 0 };

// material values and code
GLfloat fMatAmbient[4] = {0,0.5,0,1};//green=0.5
GLfloat fMatDiffuse[4] = {0,1,1,1};//green=1
GLfloat fMatSpecular[4] = {0,0,0,1};
```



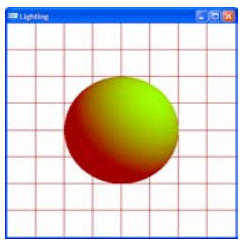
Using ambient red combined with diffuse green

1. **In choice A, reddish color lights up the whole ball because of ambient red in both light source and material. The side of object facing light is concentrated green color because of diffuse green in both light source and material.**

2. **In choice B, reddish color lights up the whole ball brightly because of ambient red in both light source and material is set to maximum, 1. The side of object facing light is concentrated yellow color because of diffuse green adds ambient red in both light source and material makes the side of object facing light look yellow.**

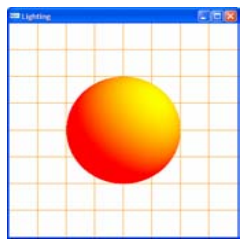
Choice A
Without ambient red

```
// material and lighting property
  // lighting values and code
  GLfloat fLtAmbient[4] = {0.7,0,0,1};//red=0.7
  GLfloat fLtDiffuse[4] = {0,1,0,1};//green=1
  GLfloat fLtSpecular[4] = {0,0,0,1};
  GLfloat light_position[] = {3, 3, 3, 0 };

  // material values and code
  GLfloat fMatAmbient[4] = {0.7,0,0,1};//red=0.7
  GLfloat fMatDiffuse[4] = {0,1,0,1};//green=1
  GLfloat fMatSpecular[4] = {0,0,0,1};
```



Choice B
With ambient red

```
// material and lighting property
  // lighting values and code
  GLfloat fLtAmbient[4] = {1,0,0,1};//red=1
  GLfloat fLtDiffuse[4] = {0,1,0,1};//green=1
  GLfloat fLtSpecular[4] = {0,0,0,1};
  GLfloat light_position[] = {3, 3, 3, 0 };

  // material values and code
  GLfloat fMatAmbient[4] = {1,0,0,1};//red=1
  GLfloat fMatDiffuse[4] = {0,1,1,1};//green=1
  GLfloat fMatSpecular[4] = {0,0,0,1};
```



What's Specular Light?

- *specular* light comes from a particular direction, and it tends to bounce off the surface in a preferred direction.

- Example:
  - 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.

Case 10
Study of specular light

Case Study Setup:

- Object Setup:
  Assume in the world coordinates we have one glut sphere of radius 2, centered at (0,0,0).

  glutSolidSphere (2, 100,100);

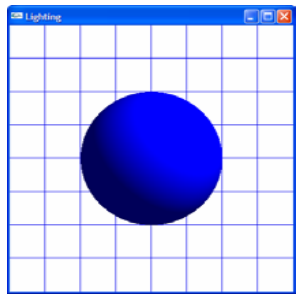
- Light's position

  GLfloat light_position[] = {3, 3, 3, 0 };
  glLightfv(GL_LIGHT0,
  GL_POSITION,
  light_position);

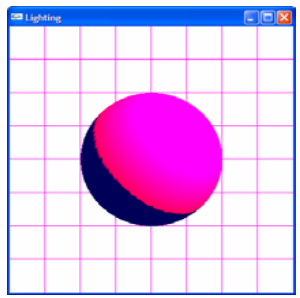
- The ambient and diffuse set up:
  - blue. (So that we can see the specular color better.

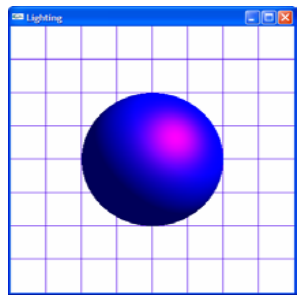
Goal:
We will change the value of specular light and the shininess of object to see how the object changes color.

Ambient, diffuse = blue



With reddish specular color and different shininess

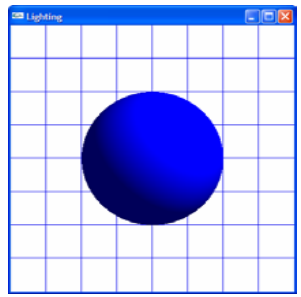




Setup a blue ball with ambient and diffuse blue before adding any specular color.

Ambient blue of both light source and material lights up the sphere with blue color.
Diffuse blue of both light source and material adds more blue to the side of object facing the light.

```
// material and lighting property
  // lighting values and code
  GLfloat fLtAmbient[4] = {0,0,0.5,1};//blue=0.5
  GLfloat fLtDiffuse[4] = {0,0,1,1};//blue=1
  GLfloat fLtSpecular[4] = {0,0,0,1};
  GLfloat light_position[] = {3, 3, 3, 0 };

  // material values and code
  GLfloat fMatAmbient[4] = {0,0,0.5,1};//blue=0.5
  GLfloat fMatDiffuse[4] = {0,0,1,1};//blue=1
  GLfloat fMatSpecular[4] = {0,0,0,1};
```



Create brightest reddish specular color
Light's specular red = 1, material's specular red = 1

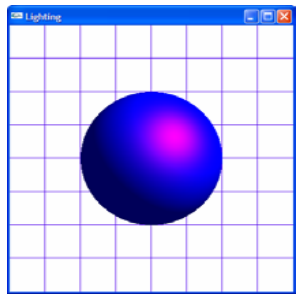
```
// material and lighting property
  // lighting values and code
  GLfloat fLtAmbient[4] = {0,0,0.5,1};
  GLfloat fLtDiffuse[4] = {0,0,1,1};
  GLfloat fLtSpecular[4] = {1,0,0,1}; //red = 1, green = 0, blue = 0
  GLfloat light_position[] = {3, 3, 3, 0 };


  glLightfv(GL_LIGHT0, GL_SPECULAR, fLtSpecular);

  // material values and code
  GLfloat fMatAmbient[4] = {0,0,0.5,1};
  GLfloat fMatDiffuse[4] = {0,0,1,1};
  GLfloat fMatSpecular[4] = {1,0,0,1}; // red = 1, green = 0, blue = 0
  GLfloat fShine = 0;     // when shiny = 0, we get maximum shininess

  glMaterialfv(GL_LIGHT0, GL_SPECULAR, fMatSpecular);
  glMaterialf(GL_FRONT, GL_SHININESS, fShine);
```

Note: red specular light on blue diffuse and ambient light, produces purple specular color on the object.

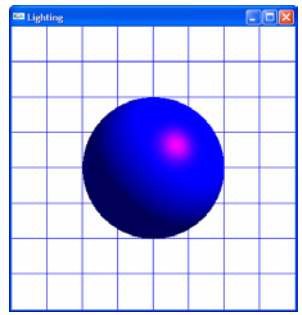
How does shininess affect specular?
A value of “0” is maximum shininess; “128” is minimum shininess.
If shininess is < 0 or > 128, it takes the default value of “0”.

```
GLfloat fShine = 0;
```

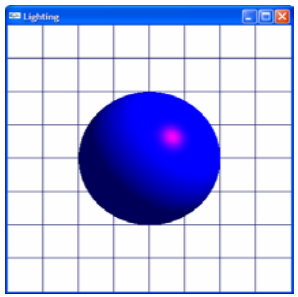
```
GLfloat fShine = 10;
```



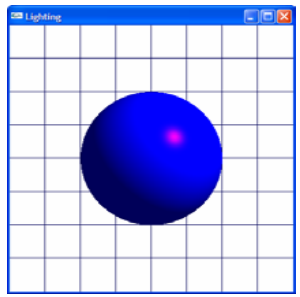
```
GLfloat fShine = 40;
```



```
GLfloat fShine = 100;
```



```
GLfloat fShine = 128;
```



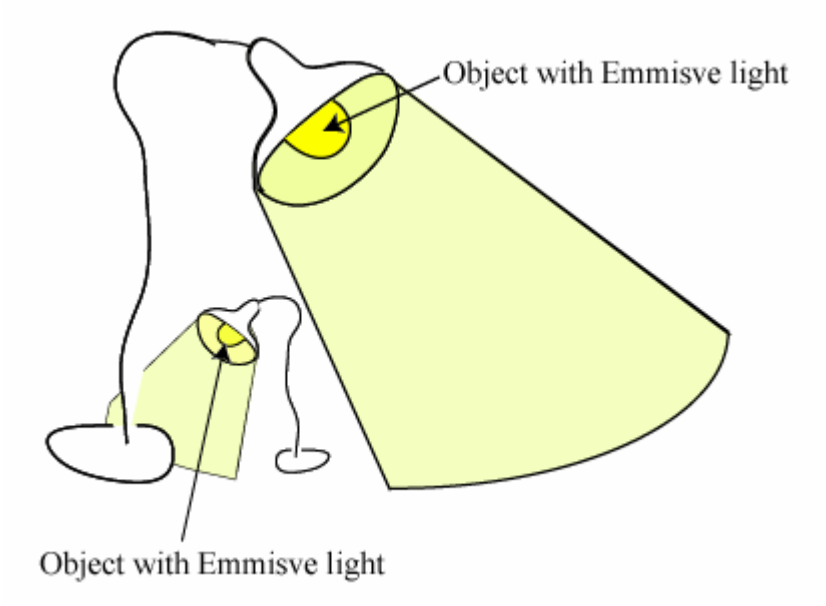
```
GLfloat fShine = 129;
```

What's Emissive Light?

- In real world, In additon to reflecting light that strikes it, a surface may also emit light.
- In OpenGL, we can add an emissive term that is not affected by incoming light and can help model visible light sources or glowing objects.
  - The emissive color of a material is applied to all vertices of an object regardless of their direction to the light.
  - This blends with the ambient light color of a scene, so if you had a powerful emissive color and low ambient lighting on everything else it might produce a slight glowing effect, although objects of different materials around it would not be affected by it.
- Special characteristic of Emissive light:
  - The emissive contribution from a surface is not used to calculate shading. Thus, a purely emissive surface appears the same regardless of any other sources or materials.
  - Emissive light is unaffected by the position of the viewer.

Remember Luxo Jr from Pixar…

- In Luxo Jr short from Pixar, there are two lamps emitting light, which produced two cones of light (spot light). This is of course an exaggerated effect.



Case 11 - Study of emissive light

Case Study Setup:

- Object Setup:
  Assume in the world coordinates we have one glut sphere of radius 2, centered at (0,0,0).

  glutSolidSphere (2, 100,100);

- Light's position

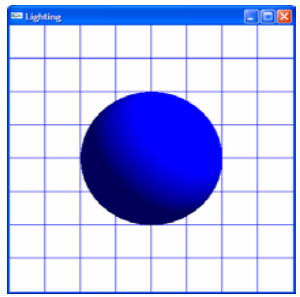
  GLfloat light_position[] = {3, 3, 3, 0 };
  glLightfv(GL_LIGHT0,
  GL_POSITION,
  light_position);

- The ambient and diffuse set up:
  - blue. (So that we can see the specular color better.)

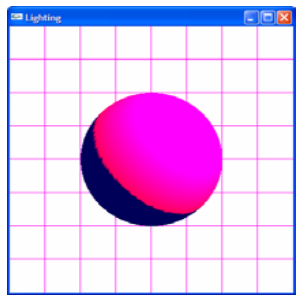
Goal:

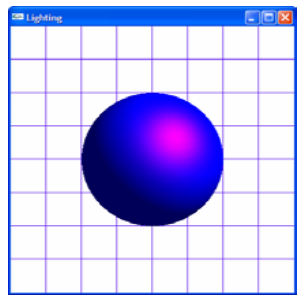
We will change the emissive value of the material to change the color of the sphere.

Ambient, diffuse = blue



With specular and different shininess





Create an object emitting green color

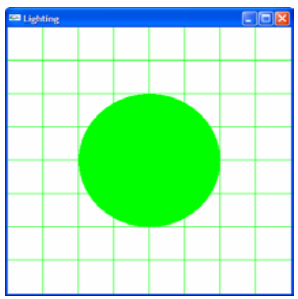
```
// material and lighting property
  // lighting values and code
  GLfloat fLtAmbient[4] = {0,0,0,1};
  GLfloat fLtDiffuse[4] = {0,0,0,1};
  GLfloat fLtSpecular[4] = {0,0,0,1};
  GLfloat light_position[] = {3, 3, 3, 0 };
     …

  // material values and code
  GLfloat fMatAmbient[4] = {0,0,0,1};
  GLfloat fMatDiffuse[4] = {0,0,0,1};
  GLfloat fMatSpecular[4] = {0,0,0,1};
  GLfloat fMatEmissive[4] = {0,a,0,1}; //green = a
  GLfloat fShine = 0;

   …
  glMaterialfv(GL_FRONT, GL_EMISSION, fMatEmissive);
```

Note:
In this example, both ambient and diffuse Light is specifed as (0,0,0,1), which means there is no ambient and diffuse light.

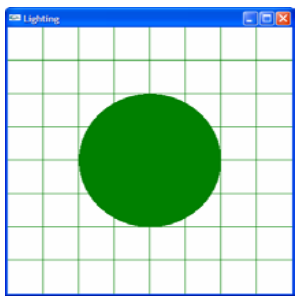
Yet, with the green emissive light defined along, the object displays as a green light Source.

This means emissive light does not need a light source.

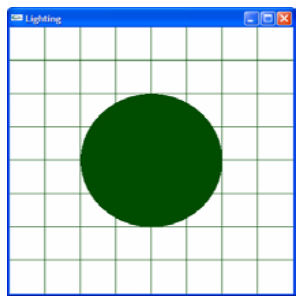
a = 1
Object is very light green



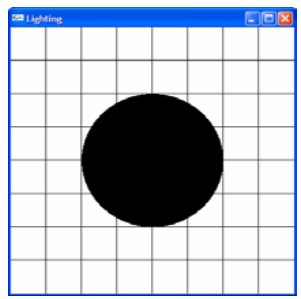
a = 0.5
Object is light green



a = 0.2
Object is dark green



a = 0
Object is black



Case 12 - Another way to define material color
Let material track ambient color of light

```
// set the material foundation color
glColor4f(1,1,1,1); // material is set to white

// set the light's color
GLfloat fLtAmbient[4] = { a, b, c, 1 };
                        //red=a,green=b,blue=c
GLfloat fLtDiffuse[4] = {1,1,1,1};
GLfloat fLtSpecular[4] = {1,1,1,1};

glEnable(GL_LIGHTING);

// Let the material track light's ambient color
glColorMaterial(GL_FRONT,GL_AMBIENT_AND_DIFFUSE);
glColorMaterial(GL_FRONT,GL_SPECULAR);
```

a=1, b=0, c=0
red

a=0, b=1, c=0
green

a=0, b=0, c=1
blue

a=1, b=1, c=0
yellow

a=1, b=0, c=1
purple

a=0, b=1, c=1
cyan

Case 13 - Another way to define material color
Let material track both ambient and diffuse color of light

```
// set the light's color
GLfloat fLtAmbient[4] = { a, b, c, 1.0f };
                        //red=a,green=b,blue=c
GLfloat fLtDiffuse[4] = { e, f, g, 1.0f };
                        //red=a,green=b,blue=c
GLfloat specref[] = {0,0,0,1};  // no specular

glEnable(GL_LIGHTING);

// Let the material track light's
// ambient and diffuse color
glColorMaterial(GL_FRONT,GL_AMBIENT_AND_DIFFUSE);

glMaterialfv(GL_FRONT, GL_SPECULAR, specref);
// because no specular, shininess does not matter
glMateriali(GL_FRONT, GL_SHININESS, 0);
```

1

```
// set material base color to white
glColor4f(1,1,1,1);
```

a=1, b=0, c=0
red
e=0, f=1, g=0
Green+red=yellow

a=0.5, b=0, c=0
red
e=0, f=0.7, g=0
Green + some red

a=0, b=0.5, c=0
green
e=0.7, f=0, g=0
red + some green

2

```
// set material base color to grey
glColor4f(0.5,0.5,0.5,1);
```

a=1, b=0, c=0
red
e=0, f=1, g=0
Green+red=yellow

a=0.5, b=0, c=0
red
e=0, f=0.7, g=0
Green + some red

a=0, b=0.5, c=0
green
e=0.7, f=0, g=0
red + some green

Note:
Output in 1 has lighter hue than output in 2, because output in 1 has a white sphere in its base color, while output in 2 has a grey sphere In its base color.

Summary of Material Colors

- The OpenGL lighting model makes the approximation that a material's color depends on the percentages of the incoming red, green, and blue light it reflects.
  - Example: a perfectly red ball reflects all the incoming red light and absorbs all the green and blue light that strike it.
- The previous case studies show in OpenGL that the larger the value of color X defined in light's or material's ambient, diffuse, and specular components (default: (red,green,blue) = (0,0,0), which is minimum. The maximum is: (red,green,blue) = (1,1,1))
- the more light of color X the object displays. This is how OpenGL simulates lights shining on the objects in general.

How material color is combined with lighting?

- Like lights, materials have different ambient, diffuse, and specular colors, which determine the ambient, diffuse, and specular reflectance of the material.
  - A material's ambient reflectance is combined with the ambient component of each incoming light source
  - The diffuse reflectance is combined with the light's diffuse component.
  - The specular reflectance is combined with light’s specular component .
- Ambient and diffuse reflectance define the color of the material and are typically similar but not identical.
- Specular reflectance is usually white or gray, so that specular highlights end up being the color of the light source's specular intensity. If you think of a white light shining on a shiny red plastic sphere, most of the sphere appears red, but the shiny highlight is white or whitish.
- And material by itself can emit light.

Same light source can create different lighting effect

- Although a light source delivers a single distribution of frequencies, the ambient, diffuse, and specular components might be different.
- Example:
  - If you have a white light in a room with red walls, the scattered light tends to be red, although the objects are actually struck by white light.

Summary of Default Material Properties

- You can define the material properties of the objects in the scene: the ambient, diffuse, and specular colors, the shininess, and the color of any emitted light.
  - void glMaterial{if}(GLenum *face*, GLenum *pname*, *TYPE param*);
  - void glMaterial{if}v(GLenum *face*, GLenum *pname*, *TYPE *param*);
    - *face* can be GL_FRONT, GL_BACK, or GL_FRONT_AND_BACK to indicate which face of the object the material should be applied to.

Default Values for pname Parameters 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

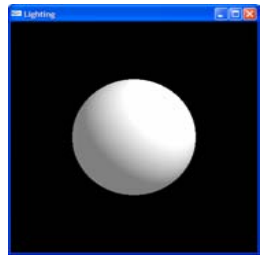
Summary of Default Values for pname Parameter of glLight*() which create light sources

- You can define the light location and properties in the scene.
  - void glLight{if}(GLenum *light*, GLenum *pname*, *TYPE param*);
  - void glMaterial{if}v(GLenum *light*, GLenum *pname*, *TYPE *param*);
    - The “light” can be LIGHT0, LIGHT1, and etc. Maximum 8 lights in OpenGL.

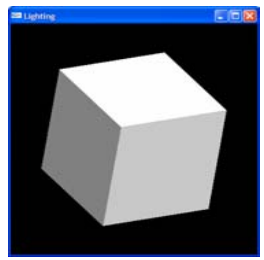
Parameter Name	Default Value	Meaning
GL_AMBIENT	(0.0, 0.0, 0.0, 1.0)	ambient RGBA intensity of light
GL_DIFFUSE	(1.0, 1.0, 1.0, 1.0)	diffuse RGBA intensity of light
GL_SPECULAR	(1.0, 1.0, 1.0, 1.0)	specular RGBA intensity of light
GL_POSITION	(0.0, 0.0, 1.0, 0.0)	(*x*, *y*, *z*, *w*) position of light
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

Why does the sphere show unevenly distributed ambient and diffuse light, but the cube displays evenly distributed light?

- Example 1 - Sphere:
  - Some area of the sphere receive more light than other areas.



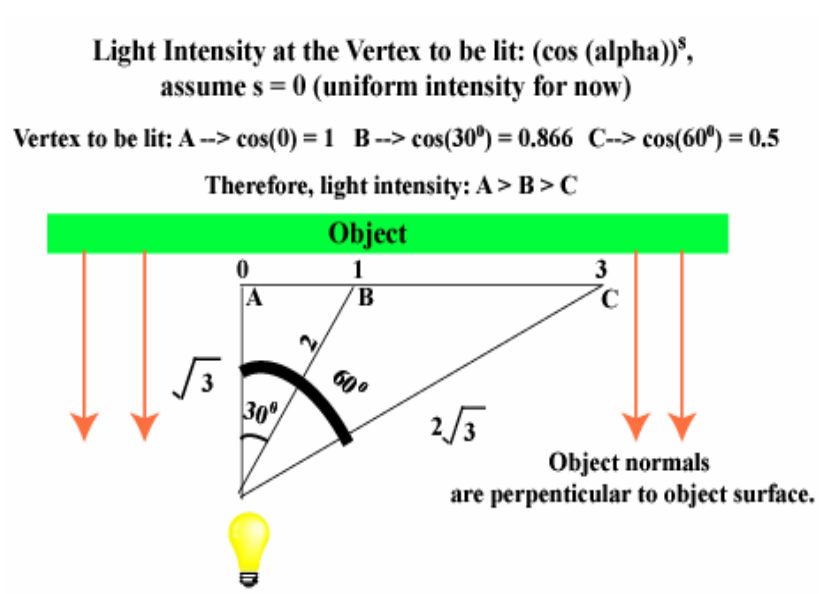
- Example 2 - Cube:
  - Each side of the cube shows evenly distributed diffuse light.



Case 14 – Surface Normals

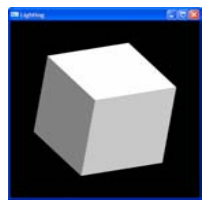
Example 1
Why does the sphere show unevenly distributed ambient and diffuse light, but the cube displays evenly distributed light?

- **Reason:**
  - An object's normals are perpendicular to the object surface.
  - An object's normals determine its orientation relative to the light sources.
  - For each vertex, OpenGL uses the assigned normal to determine how much light that particular vertex receives from each light source.
  - How much light a vertex on the object gets is determined by the $\cos(\text{alpha})^s$, where alpha is the angle between the normal of the vertex and light, and s is the light intensity. We will discuss "s" later.
    - When alpha is 0, cos(0)=1, the vertex gets the most light. With higher alpha, the vertex gets less light.



Example 1 - Sphere

Example 2 - Cube

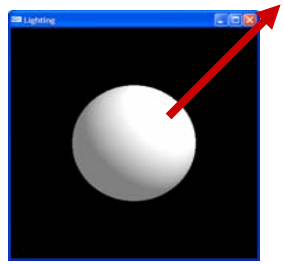


Case 14 – Surface Normals

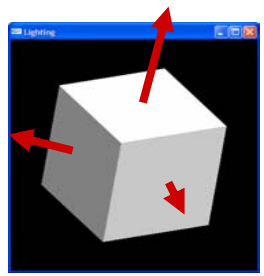
Example 1, continued
OpenGL defines its glut objects's normals perpendicular to their surface

- Glut object such as glutSolid Sphere and glutSolidCube have defined normals in their routines.
  - Example 1 uses glutSolidSphere to define a sphere.
  - Example 2 uses glutSolidCube to define a cube.
- When we create our own objects, we need to define normals.

Example 1 - Sphere



Example 2 - Cube

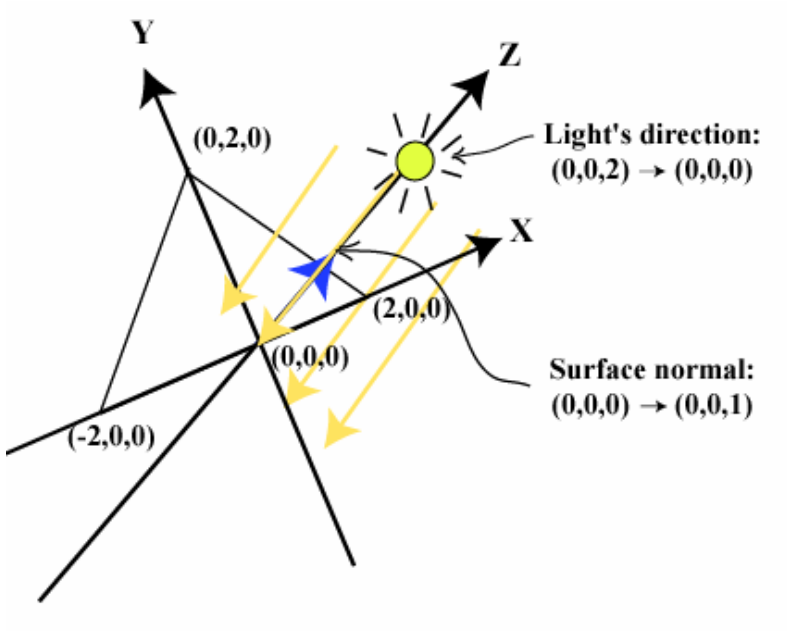


How to define our own object surface normals?

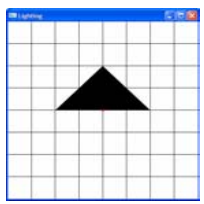
- All surface normals must eventually be converted to unit normals in OpenGL.
- A unit normal is just a normal vector of length of 1.
- You can tell OpenGL to convert your normals to unit normals automatically by enabling normalization with glEnable(GL_NORMALIZE).
  - However, this has performance penalties. It's far better to calculate your normals ahead of time instead of relying on OpenGL to do this for you.
  - Remember that when you use glScale transformation function, it also scales the lengths of your normals.

Case 14 – Example 2
Define unit normal

```
// define a triangle object to be drawn
void triangle()
     {
     // define unit normal
            glNormal3f(0,0,1);
            // draw a triangle with three vertexes
            glVertex3f(2,0,0);
            glVertex3f(-2,0,0);
            glVertex3f(0,2,0);
     glEnd();
     }

// lighting effect
     GLfloat light_position[] = {0,0,2,0};
     glEnable(GL_LIGHTING);
     glEnable(GL_LIGHT0);
     glLightfv(GL_LIGHT0, GL_POSITION,
     light_position);
```



Before Lighting

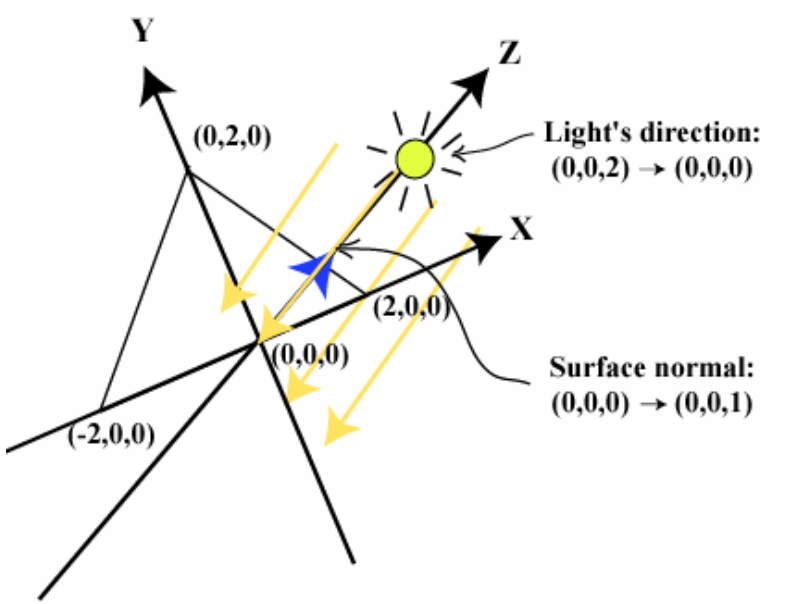


After applying lighting

Case 14 – Example 3

Convert non-unit normal to unit normal using glEnalbe(GL_NORMALIZE)

```
// define a triangle object to be drawn
void triangle()
     {
     glEnable(GL_NORMALIZE);
     glBegin(GL_TRIANGLES);
             // define a non-unit normal
             glNormal3f(0,0,2);
             // draw a triangle with three vertexes
             glVertex3f(2,0,0);
             glVertex3f(-2,0,0);
             glVertex3f(0,2,0);
     glEnd();
     }

// lighting effect
     GLfloat light_position[] = {0,0,2,0};
     glEnable(GL_LIGHTING);
     glEnable(GL_LIGHT0);
     glLightfv(GL_LIGHT0, GL_POSITION,
     light_position);
```



Without glEnable(NORMALIZE)
unable to see object

With glEnable(NORMALIZE)
able to see object

Note:
Using glEnable(GL_NROMALIZE) has performance penalties.

Case 14 – Example 4
Write a function that can reduce any normal vector to unit normal vector

```
// reduce any nromal vector to unit normal vector
void ReduceToUnitVector(float vector[3])
    {
    float length;

    // Calculate the length of the vector
    length = (float)sqrt((vector[0]*vector[0]) +
                         (vector[1]*vector[1]) +
                         (vector[2]*vector[2]));


    // avoid deviding by zero by providing an
    // acceptable value for vectors too close to zero.

    // Dividing each vector by the length will result
    // in a unit vector.
    vector[0] /= length;
    vector[1] /= length;
    vector[2] /= length;
    }
```

```
// An example of using ReduceToUnitVector to convert
// normal vector
// {0,0,2} to unit vector
void triangle()
    {
    float v[3] = {0,0,2};

    ReduceToUnitVector(v);
    glBegin(GL_TRIANGLES);

        glNormal3fv(v);
        glVertex3f(2,0,0);
        glVertex3f(-2,0,0);
        glVertex3f(0,2,0);

    glEnd();
    }
```

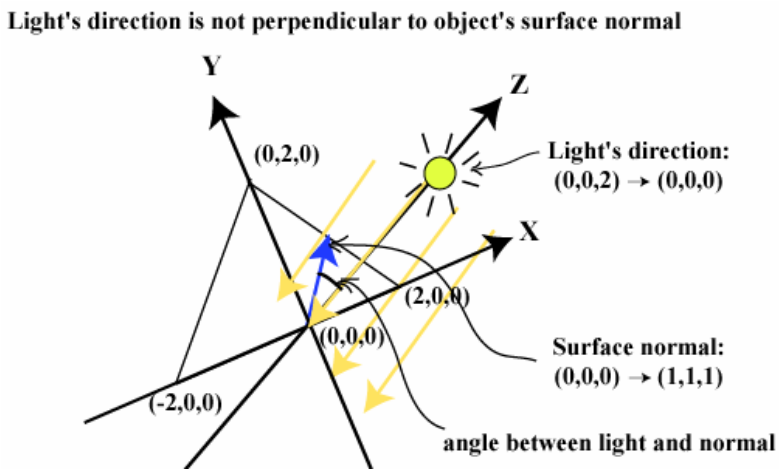
output

After converting (0,0,2) to unit vector, we get unit vector (0,0,1)

Case 14 – Example 5

The larger the angle, alpha, between normal and light the darker the object.

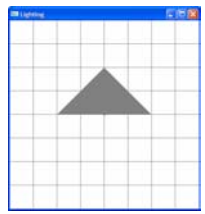
```
void triangle()
    {
    float v[3] = {1,1,1}; //define a normal vector

    ReduceToUnitVector(v); // normalize normal
    glBegin(GL_TRIANGLES);

        glNormal3fv(v);  // define surface normal
        glVertex3f(2,0,0);
        glVertex3f(-2,0,0);
        glVertex3f(0,2,0);

    glEnd();
    }
```




```
// lighting effect
    GLfloat light_position[] = {0,0,2,0};
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glLightfv(GL_LIGHT0, GL_POSITION,
    light_position);
```

Surface normal (0,0,1)
Light direction (0,0,2)
Angle between normal and light = 0

Surface normal (1,1,1)
Light direction (0,0,2)
Angle between normal and light > 0



Case 14 – Example 6

A function to calculate normal from any three vertices

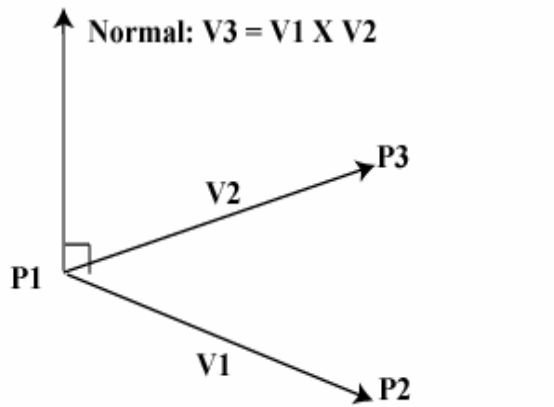
```
// A generic function which calcuate normal from
// any three vertices from a surface
// Points p1, p2, & p3 specified in counter clock-wise order
void CalcNormal(float v[3][3], float out[3])
     {
     float v1[3],v2[3];
     static const int x = 0;
     static const int y = 1;
     static const int z = 2;

     // Calculate two vectors from the three points
     v1[x] = v[0][x] - v[1][x];
     v1[y] = v[0][y] - v[1][y];
     v1[z] = v[0][z] - v[1][z];

     v2[x] = v[1][x] - v[2][x];
     v2[y] = v[1][y] - v[2][y];
     v2[z] = v[1][z] - v[2][z];

     // Take the cross product of the two vectors to get
     // the normal vector which will be stored in out
     out[x] = v1[y]*v2[z] - v1[z]*v2[y];
     out[y] = v1[z]*v2[x] - v1[x]*v2[z];
     out[z] = v1[x]*v2[y] - v1[y]*v2[x];

     // Normalize the vector (shorten length to one)
     ReduceToUnitVector(out);
     }
```

A normal vector is a produce of two vectors.

To caculate a normal for a surface, we need to:

(1) pick three points (P1,P2,P3) on the surface

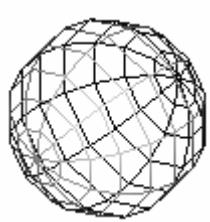
(2) use the two vectors (V1 and V2) defined by the three points to produce a cross product. (V3)

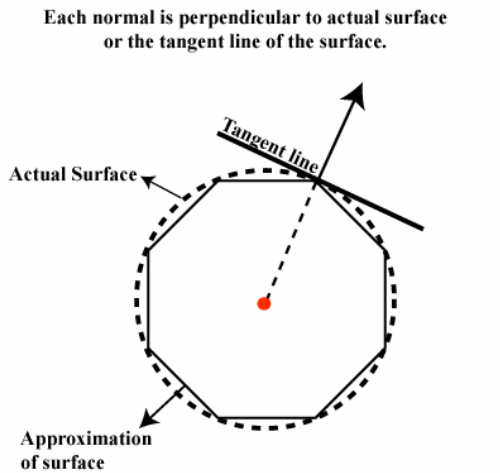


Case 14 – Example 7 -Normal Averaging

- Normal Averaging
  - For smooth surface, we can approximate normals to each face of the surface in order to produce apparently smooth surfaces with flat polygons.

- A sphere made of quads and triangles:





- The polygonal representation of a sphere is only an approximation of the true surface. Theoretically, if we used enough polygons, the surface will appear smooth.

- For the above sphere, the normals would point directly out from the center of the sphere through each vertex.

- Each normal will be perpendicular to the tangent line to the surface. (See picture on the right)

For a sphere, the calculation of normals is very simple, because the normal has the same value as the vertex.

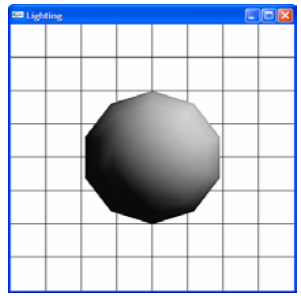
Case 14 – example 8
flat shading vs smooth shading

// draw an inadequently tessellated sphere,
// a sphere made up of few polygons
glutSolidSphere (2, 10,10);

// shine light diagonally

```
GLfloat light_position[] = {2,2,2,0};
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);

glLightfv(GL_LIGHT0,
          GL_POSITION,
          light_position);
```

```
// smooth shading
glShadeModel (GL_SMOOTH);
```



```
// flat shading
glShadeModel (GL_FLAT);
```

glutSolidSphere automatically calculates normal of each polygon of the sphere.

- **With smooth shading, there is a color/shade interpolation between different vertices of the polygons of the sphere.**
- **But with flat shading, there is no color/shade interpolation.**

Case 15 – multiple light source

- Case Study Setup:
- Object Setup:
- Assume in the world coordinates we have one glut sphere of radius 2, centered at (0,0,0).
- glutSolidSphere (2, 100,100);
- Light 0's specifications

```
GLfloat light_position1[] = {2,2,2,1};
GLfloat spot_light_direction[] ={-1,-1,-1,1};
GLfloat fLtAmbient1[4] = { 0, 1, 0, 1 };
GLfloat fLtDiffuse1[4] = { 0, 1, 0, 1 };
GLfloat fLtSpecular1[4] = {1,1,0,1};
```

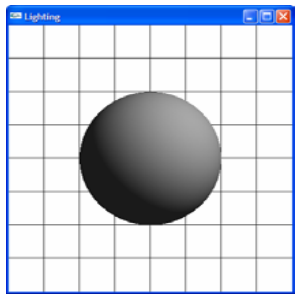
- Light 1's specification

```
GLfloat light_position2[] = {-2,-2,2,0};
GLfloat fLtAmbient2[4] = { 1, 0, 0, 1 };
GLfloat fLtDiffuse2[4] = { 1, 0, 0, 1 };
GLfloat fLtSpecular2[4] = {0,0,1,1};
```

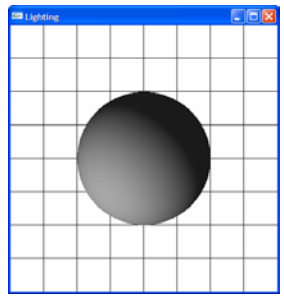
Goal:

We will specify two different light sources (at different locations) to shine on one sphere to find out how mutiple lighting works in OpenGL.

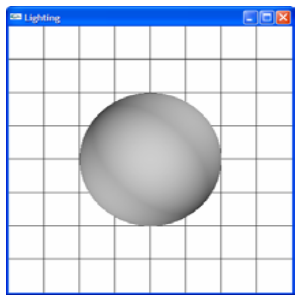
LIGHT0 only



LIGHT1 only



LIGHT0 + LIGHT1



Case 15 – multiple light source – example 1

- GL_LIGHT0 shines diffuse white and specular white by default; all the other lights (LIGHT1, LIGHT2 and others) shine diffuse black and specular black.

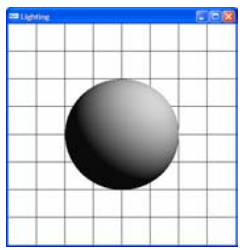
- Therefore, to see any light effect of LIGHT1, LIGHT2 and others, we need to enable ambient, diffuse or specular light for them.

OpenGL Commands

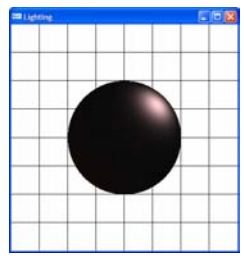
- `glMaterial{if}(face, parameter, value)`
  - Changes one of the coefficients for the front or back side of a face (or both sides)
- `glLight{if}(light, property, value)`
  - Changes one of the properties of a light (intensities, positions, directions, etc)
  - There are 8 lights: `GL_LIGHT0, GL_LIGHT1, …`
- `glLightModel{if}(property, value)`
  - Changes one of the global light model properties (global ambient light, for instance)
- `glEnable(GL_LIGHT0)` enables `GL_LIGHT0`
  - You must enable lights before they contribute to the image
  - You can enable and disable lights at any time

Direction light vs spot light
Spotlight can specify GL_SPOT_EXPONENT, but direction light can't

```
// direction light direction
  GLfloat light_position[] = {3, 3, 3, 0};

  glEnable(GL_LIGHTING);  // Enable light

  glEnable(GL_LIGHT0);    // Enable one
    light, light0

  // specify spotlight position at (3,3,3)
  glLightfv(GL_LIGHT0, GL_POSITION,
    light_position);

  // spotlight intensity
  glLightf(GL_LIGHT0, GL_SPOT_EXPONENT,50);
```



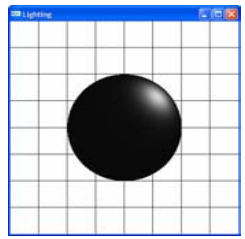
```
// spotlight position
  GLfloat light_position[] = {3, 3, 3, 1};
  GLfloat spot_light_direction[] = {-1, -1, -1 };

  glEnable(GL_LIGHTING);  // Enable light
  glEnable(GL_LIGHT0);    // Enable one light, light0

  // light cut-off is 45, cone angle is 90 degree
  glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, 45.0);

  // specify spotlight position at (3,3,3)
  glLightfv(GL_LIGHT0, GL_POSITION, light_position);

  // specify the spotlight direction
  glLightfv(GL_LIGHT0,
            GL_SPOT_DIRECTION,
            spot_light_direction);

  // spotlight intensity
  glLightf(GL_LIGHT0, GL_SPOT_EXPONENT,50);
```



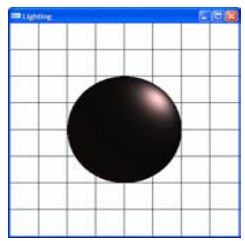
Adds color into spot light's ambient component

```
// spotlight position
  GLfloat light_position[] = {3, 3, 3, 1};
  GLfloat spot_light_direction[] = {-1, -1, -1 };
  GLfloat fLtAmbient1[4] = { 1, 0, 0, 1 }; //red

  glEnable(GL_LIGHTING);  // Enable light
  glEnable(GL_LIGHT0);    // Enable one light, light0

  // light cut-off is 45, cone angle is 90 degree
  glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, 45.0);

  // specify spotlight position at (3,3,3)
  glLightfv(GL_LIGHT0, GL_POSITION, light_position);
  glLightfv(GL_LIGHT0, GL_AMBIENT, fLtAmbient1);

  // specify the spotlight direction
  glLightfv(GL_LIGHT0,
            GL_SPOT_DIRECTION,
            spot_light_direction);

  // spotlight intensity
  glLightf(GL_LIGHT0, GL_SPOT_EXPONENT,50);
```

Before adding red ambient to spotlight
See default white colored spot



After adding ambient red to spotlight
See reddish spot



Adds ambient color into spotlight's and enable material color

```
// sphere's material color
glColor4f(a,b,c,1); //red=a,green=b,blue=c
```

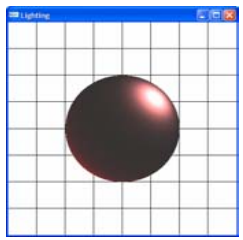
```
// spotlight position
  GLfloat light_position[] = {3, 3, 3, 1};
  GLfloat spot_light_direction[] = {-1, -1, -1 };
  GLfloat fLtAmbient1[4] = { 1, 0, 0, 1 }; //red

  glEnable(GL_LIGHTING);  // Enable light
  glEnable(GL_LIGHT0);    // Enable one light,
    light0

  // light cut-off is 45, cone angle is 90 degree
  glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, 45.0);

  // specify spotlight position at (3,3,3)
  glLightfv(GL_LIGHT0, GL_POSITION,
    light_position);
  glLightfv(GL_LIGHT0, GL_AMBIENT, fLtAmbient1);

  // specify the spotlight direction
  glLightfv(GL_LIGHT0,
            GL_SPOT_DIRECTION,
            spot_light_direction);

  // spotlight intensity
  glLightf(GL_LIGHT0, GL_SPOT_EXPONENT,50);
```

```
// enable material color
glEnable(GL_COLOR_MATERIAL);
```

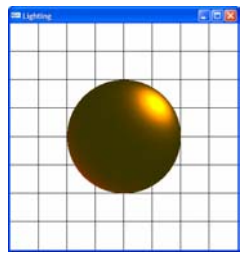
Before turn on material-track-ambient-light
See reddish spot (material color has no effect)

After turnong material-tracking-ambient-light

a=1,b=1,c=1 (white)
Material white shows up
in the spotlight



a=1,b=1,c=0 (yellow)
Material yellow shows up
in the spotlight
Red+yellow = pink

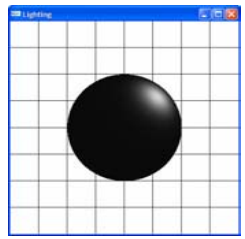


When enable material color,
material color will be mixed with light's color

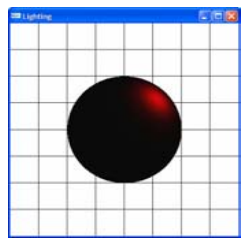
Adds color into spotlight's diffuse component

```
// spotlight position
  GLfloat light_position[] = {3, 3, 3, 1};
  GLfloat spot_light_direction[] = {-1, -1, -1 };
  GLfloat fLtDiffuse[4] = { 1, 0, 0, 1 }; //red

  glEnable(GL_LIGHTING);  // Enable light
  glEnable(GL_LIGHT0);    // Enable one light, light0

  // light cut-off is 45, cone angle is 90 degree
  glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, 45.0);

  // specify spotlight position at (3,3,3)
  glLightfv(GL_LIGHT0, GL_POSITION, light_position);
  glLightfv(GL_LIGHT0, GL_DIFFUSE, fLtDiffuse);

  // specify the spotlight direction
  glLightfv(GL_LIGHT0,
            GL_SPOT_DIRECTION,
            spot_light_direction);

  // spotlight intensity
  glLightf(GL_LIGHT0, GL_SPOT_EXPONENT,50);
```

Before adding red ambient to spotlight
See default white colored spot



After adding diffuse green to spotlight
See green spot



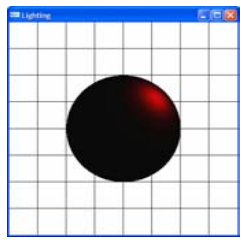
Adds ambient color into spotlight's and enable material-track-ambient-light

```
// sphere's material color
glColor4f(a,b,c,1); //red=a,green=b,blue=c
```

```
// spotlight position
  GLfloat light_position[] = {3, 3, 3, 1};
  GLfloat spot_light_direction[] = {-1, -1, -1 };
  GLfloat fLtAmbient1[4] = { 1, 0, 0, 1 }; //red

  glEnable(GL_LIGHTING);  // Enable light
  glEnable(GL_LIGHT0);    // Enable one light,
    light0

  // light cut-off is 45, cone angle is 90 degree
  glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, 45.0);

  // specify spotlight position at (3,3,3)
  glLightfv(GL_LIGHT0, GL_POSITION,
    light_position);
  glLightfv(GL_LIGHT0, GL_AMBIENT, fLtAmbient1);

  // specify the spotlight direction
  glLightfv(GL_LIGHT0,
            GL_SPOT_DIRECTION,
            spot_light_direction);

  // spotlight intensity
  glLightf(GL_LIGHT0, GL_SPOT_EXPONENT,128);
```

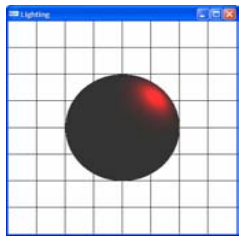
```
// enable materialcolor
glEnable(GL_COLOR_MATERIAL);
```

Before turn on material-track-ambient-light
See reddish spot (material color has no effect)

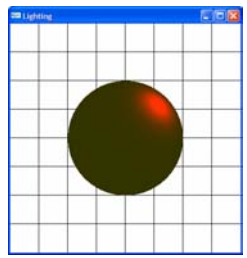


After turnong material-tracking-ambient-light

a=1,b=1,c=1 (white)
Material white shows up in the spotlight, make red spot brighter



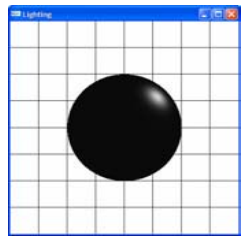
a=1,b=1,c=0
Material blue shows up in the spotlight



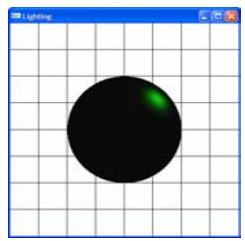
Adds color into spot light's diffuse component

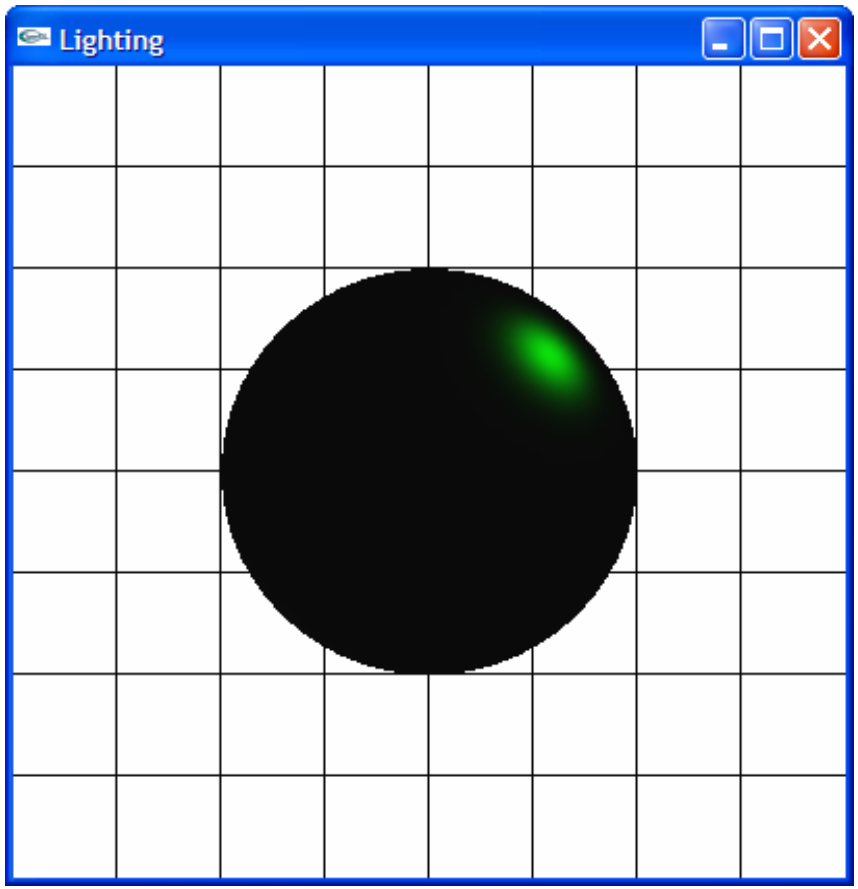
```
// spotlight position
  GLfloat light_position[] = {3, 3, 3, 1};
  GLfloat spot_light_direction[] = {-1, -1, -1 };
  GLfloat fLtDiffuse[4] = { 0, 1, 0, 1 };

  glEnable(GL_LIGHTING);  // Enable light
  glEnable(GL_LIGHT0);    // Enable one light,
    light0

  // light cut-off is 45, cone angle is 90 degree
  glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, 45.0);

  // specify spotlight position at (3,3,3)
  glLightfv(GL_LIGHT0, GL_POSITION,
    light_position);
  glLightfv(GL_LIGHT0, GL_DIFFUSE, fLtDiffuse1);

  // specify the spotlight direction
  glLightfv(GL_LIGHT0,
            GL_SPOT_DIRECTION,
            spot_light_direction);

  // spotlight intensity
  glLightf(GL_LIGHT0, GL_SPOT_EXPONENT,128);
```

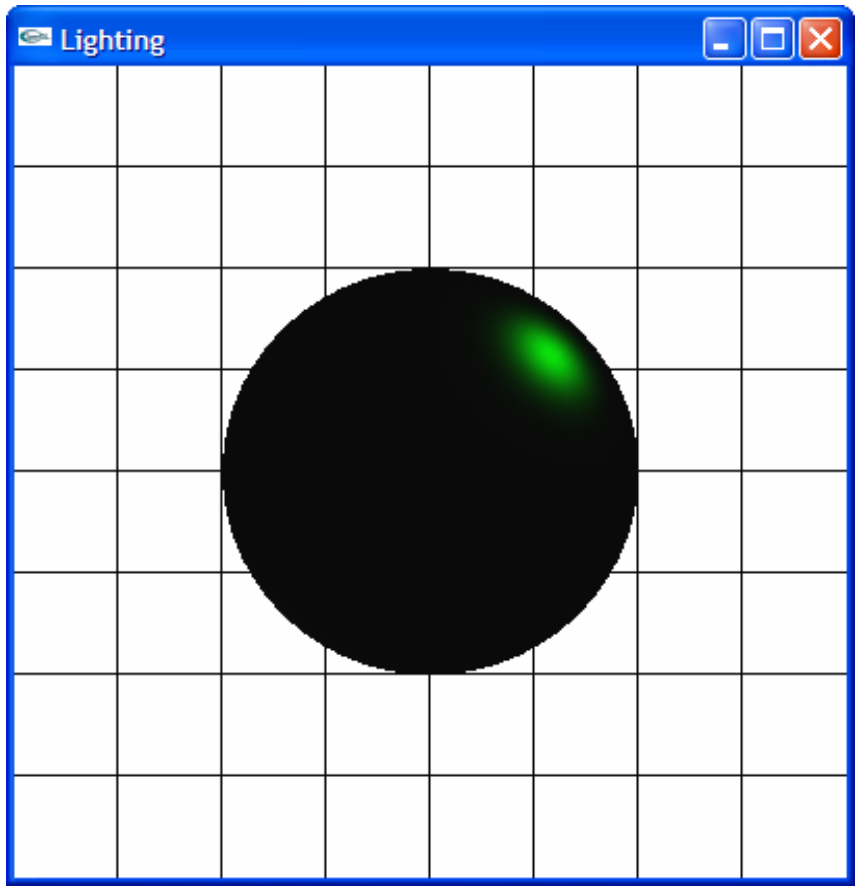
After adding green ambient and diffuse to spot light
Default is white



Before adding green ambient to spot light



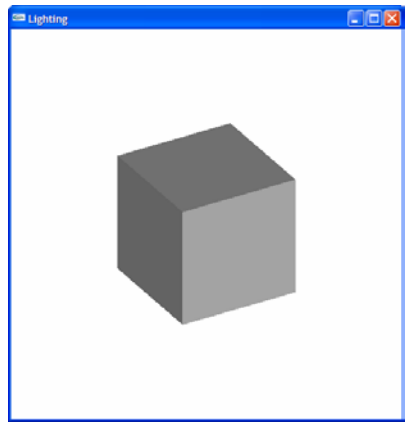




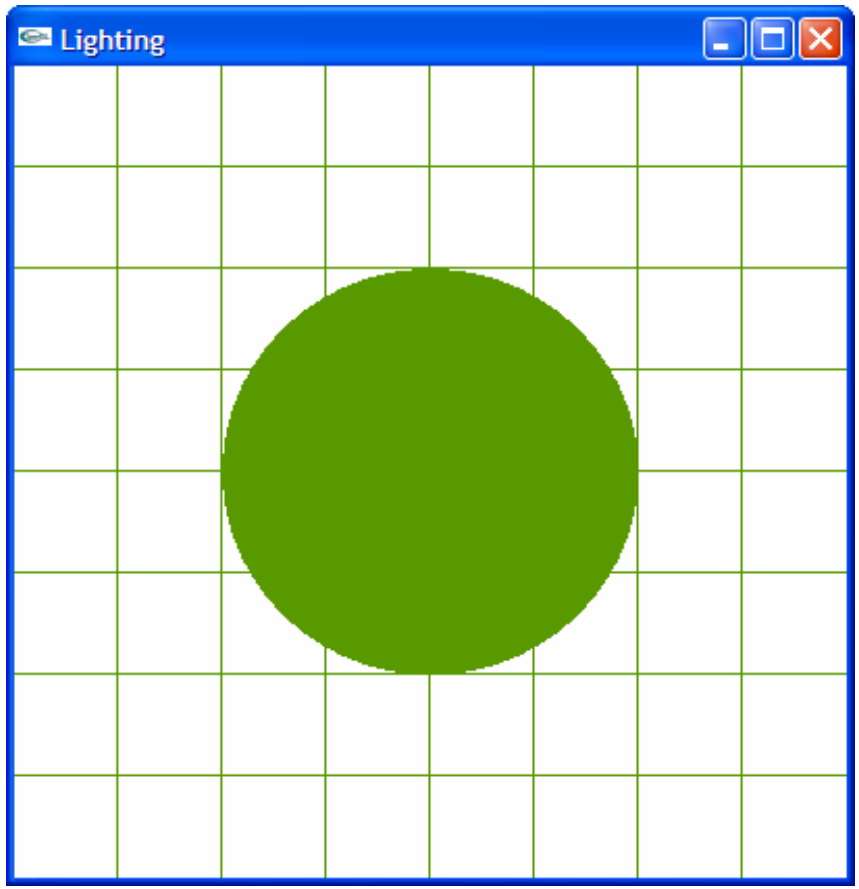
- New to OpenGL 1.2 is GL_RESCALE_NORMALS .
  glEnable(GL_SCALE_NORMALS) allows you to scale all the normals by the same amount to make them unit length.
  - OpenGL figures this out by examing the modelview matrix. This results fewer mathematical operations per vertex than are otherwise required.

Define Normal Vectors for Each Vertex of Every Object

- An object's normals determine its orientation relative to the light sources.
- For each vertex, OpenGL uses the assigned normal to determine how much light that particular vertex receives from each light source.
- In this example, we use glut object glutSolidCube, which has defined normal as part of its routine.
- glutSolidSphere (2.0)



How to create light growing effect?



Overview of how to create lighting in OpenGL

- Before Start:

  0. Determine the world coordinates

- Main Steps:

  1. Define Normal Vectors for Each Vertex of Every Object
  2. An object's normals determine its orientation relative to the light sources.
  3. Create, Position, and Enable One or More Light Sources
  4. Select a Lighting Model
  5. Define Material Properties for the Objects in the Scene

How to keep the light Stationary?

- **Keeping the Light Stationary**

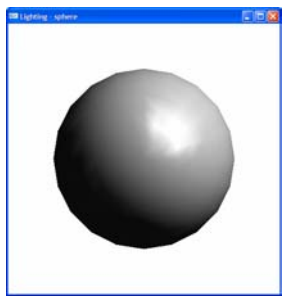
```
glViewport (0, 0, (GLsizei) w, (GLsizei) h);
glMatrixMode (GL_PROJECTION);
glLoadIdentity();
if (w <= h)
    glOrtho (-1.5, 1.5, -1.5*h/w, 1.5*h/w, -
    10.0, 10.0);
else
    glOrtho (-1.5*w/h, 1.5*w/h, -1.5, 1.5, -
    10.0, 10.0);

glMatrixMode (GL_MODELVIEW);
glLoadIdentity();
…
/* later in init() */
GLfloat light_position[] = { 1.0, 1.0, 1.0,
    1.0 };
glLightfv(GL_LIGHT0, GL_POSITION, position);
/* NO other MODELVIEW transformation is set…*/
```

- Answer:
  - After setting the light position

(3) Select a Lighting Model

```
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 };

   SetBackground(1,1,1,0);
   glShadeModel (GL_SMOOTH);

   glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
   glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
   glLightfv(GL_LIGHT0, // default color is white
     GL_POSITION,       // use light position for
     lighting
     light_position);      // light's position

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

- This example uses the default settings for these two aspects of the light model.
  - an infinite viewer
    - Using 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.
  - One sided lighting
    - Lighting calculations on the front facing only.
- If you need to specify Light Model, use glLightModel*() function.



Selecting a Lighting Model

- The OpenGL notion of a lighting model has three components:
  - The global ambient light intensity
  - Whether the viewpoint position is local to the scene or considered to be an infinite distance away
  - Whether lighting calculations should be performed differently for both the front and back faces of objects
- void glLightModel{if}(GLenum *pname*, *TYPE param*);
- void glLightModel{if}v(GLenum *pname*, *TYPE *param*);

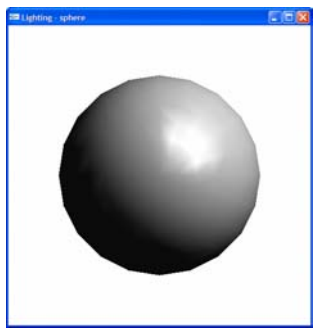
Default values for Lighting Model

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

Define Material Properties for the objects in the scene

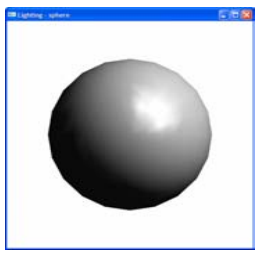
```
void init(void)
{
   GLfloat mat_specular[] = { 1.0, 1.0, 1.0,
    1.0 };
   GLfloat mat_shininess[] = { 50.0 };
   GLfloat light_position[] = { 1.0, 1.0, 1.0,
    0.0 };

   SetBackground(1,1,1,0);
   glShadeModel (GL_SMOOTH);

   glMaterialfv(GL_FRONT, GL_SPECULAR,
    mat_specular);
   glMaterialfv(GL_FRONT, GL_SHININESS,
    mat_shininess);

   glLightfv(GL_LIGHT0, // default color is
    white
    GL_POSITION,       // use light position
    for lighting
    light_position);      // light's position

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

- You can specify a material's ambient, diffuse, and specular colors and how shiny it is.
- In this example, only these last two material properties—the specular material color and shininess—are explicitly specified (with the glMaterialfv() calls).



Position and Attenuation

- **There are two kinds of light source.**
  - Directional light source: the effect of an infinite location is that the rays of light can be considered parallel by the time they reach an object.
  - Positional light source: Its exact position within the scene determines the effect it has on a scene and, especially, the direction from which the light rays come.
- **We use directional light source for the sphere.**



```
GLfloat light_position[] = { 1.0, 1.0,
    1.0, 0.0 };
glLightfv(GL_LIGHT0, GL_POSITION,
    light_position);
```

- You supply a vector of four values (*x, y, z, w*) for the GL_POSITION parameter. If the last value, *w*, is zero, the corresponding light source is a directional one, and the (*x, y, z*) values describe its direction.
- Otherwise, diffuse and specular lighting calculations are based on the actual location of the light in eye coordinates, and attenuation is enabled.

Light Attenuation Formula

- For a directional light where light is infinitely far away, attenuation is disabled for a directional light.
- However, you might want to attenuate the light from a positional light.
- OpenGL attenuates a light source by multiplying the contribution of that source by an attenuation factor:

$$\text{attenuation factor} = \frac{1}{k_c + k_l d + k_q d^2}$$

where

d = distance between the light's position and the vertex

k_c = GL_CONSTANT_ATTENUATION

k_l = GL_LINEAR_ATTENUATION

k_q = GL_QUADRATIC_ATTENUATION

By default, k_c is 1.0 and both k_l and k_q are zero, but you can give these parameters different values:

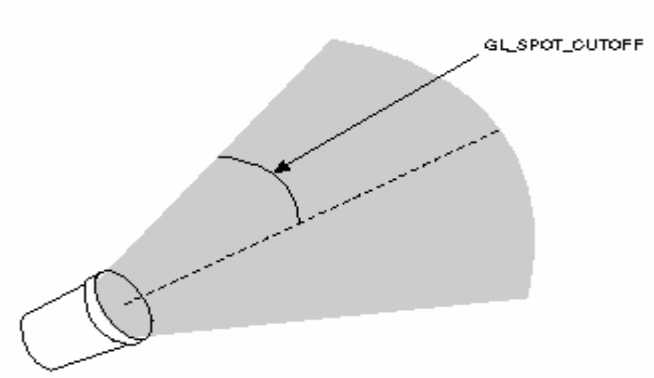
Example of attenuation constance specification:

```
glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, 2.0);
glLightf(GL_LIGHT0, GL_LINEAR_ATTENUATION, 1.0);
glLightf(GL_LIGHT0, GL_QUADRATIC_ATTENUATION, 0.5);
```

Spotlights

- 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, which is called **GL_SPOT_CUTOFF Parameter**.
- Note that no light is emitted beyond the edges of the cone. By default, the spotlight feature is disabled because the GL_SPOT_CUTOFF parameter is 180.0. This value means that light is emitted in all directions (the angle at the cone's apex is 360 degrees)
- The value for GL_SPOT_CUTOFF is restricted to being within the range [0.0,90.0]

  (unless it has the special value 180.0).



Example of Spot light

- `// sets the cutoff parameter to 45 degrees`
  - `glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, 45.0);`
- `// 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);`

Creating Light Sources

- **Default Values for pname Parameter of glLight*()**
- GL_AMBIENT (0.0, 0.0, 0.0, 1.0)
  - ambient RGBA intensity of lightGL_
- DIFFUSE (1.0, 1.0, 1.0, 1.0)
  - diffuse RGBA intensity of lightGL_
- 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

- *pname*
- Specifies a light source parameter for *light*.
  - **GL_AMBIENT**, **GL_DIFFUSE**, **GL_SPECULAR**, **GL_POSITION**, **GL_SPOT_DIRECTION**, **GL_SPOT_EXPONENT**, **GL_SPOT_CUTOFF**, **GL_CONSTANT_ATTENUATION**, **GL_LINEAR_ATTENUATION**, **GL_QUADRATIC_ATTENUATION**