CPSC/ECE 4780/6780

General-Purpose Computation on Graphical Processing Units (GPGPU)

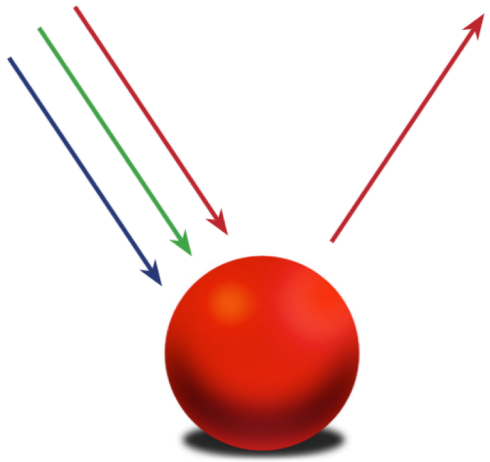
Lecture 18: (OpenGL) Lights and Materials

Recap of Last Lecture

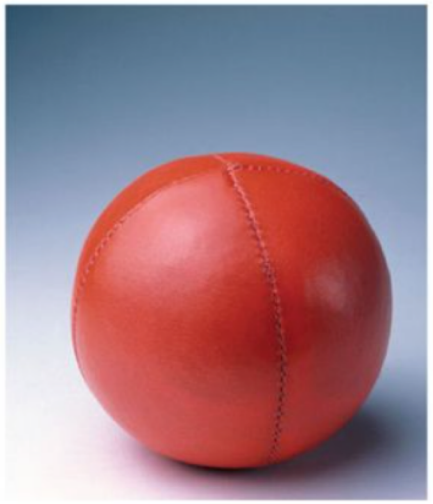
- What are modeling and viewing transformations?
- What is projection transformations?
- What is viewport transformation?
- What are matrix stacks? How to manipulate the matrix stacks?

Colors in Our World

- Colors we see in the real world are based on the interaction between **lights** coming from sources and the **materials** of which the objects are composed

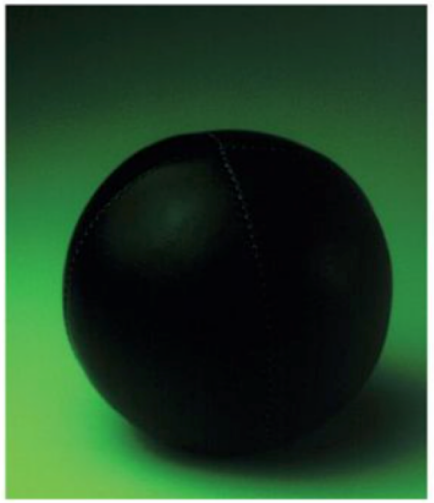


A perfectly red ball reflects all the incoming red light and absorbs all the green and blue light that strikes it



A red ball seen under white light

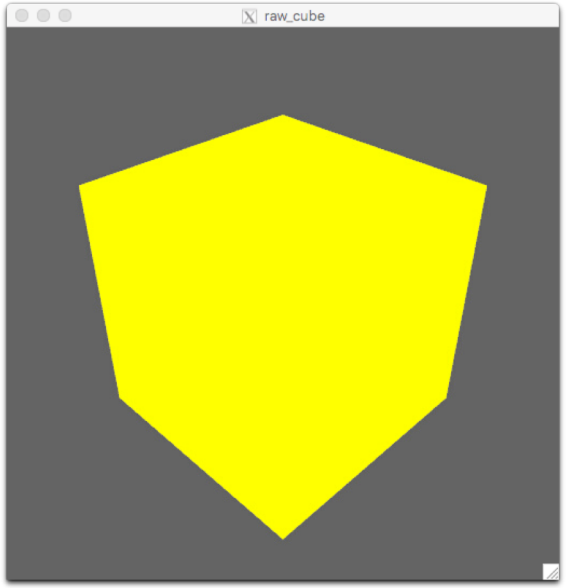
A red ball seen under red light

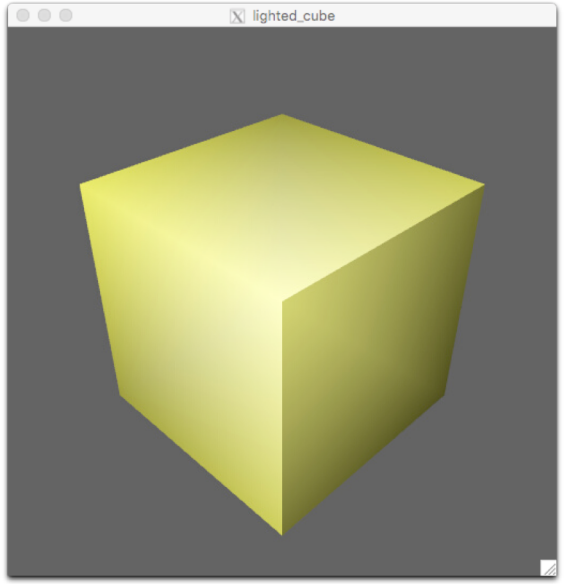


A red ball seen under green light

Lights

- The real light is made of photons (hard to simulate)
- Modeling lights adds a lot to the visual appeal of the rendered scene
- Light models are developed to capture the core effect that light has when it falls on objects and makes them visible



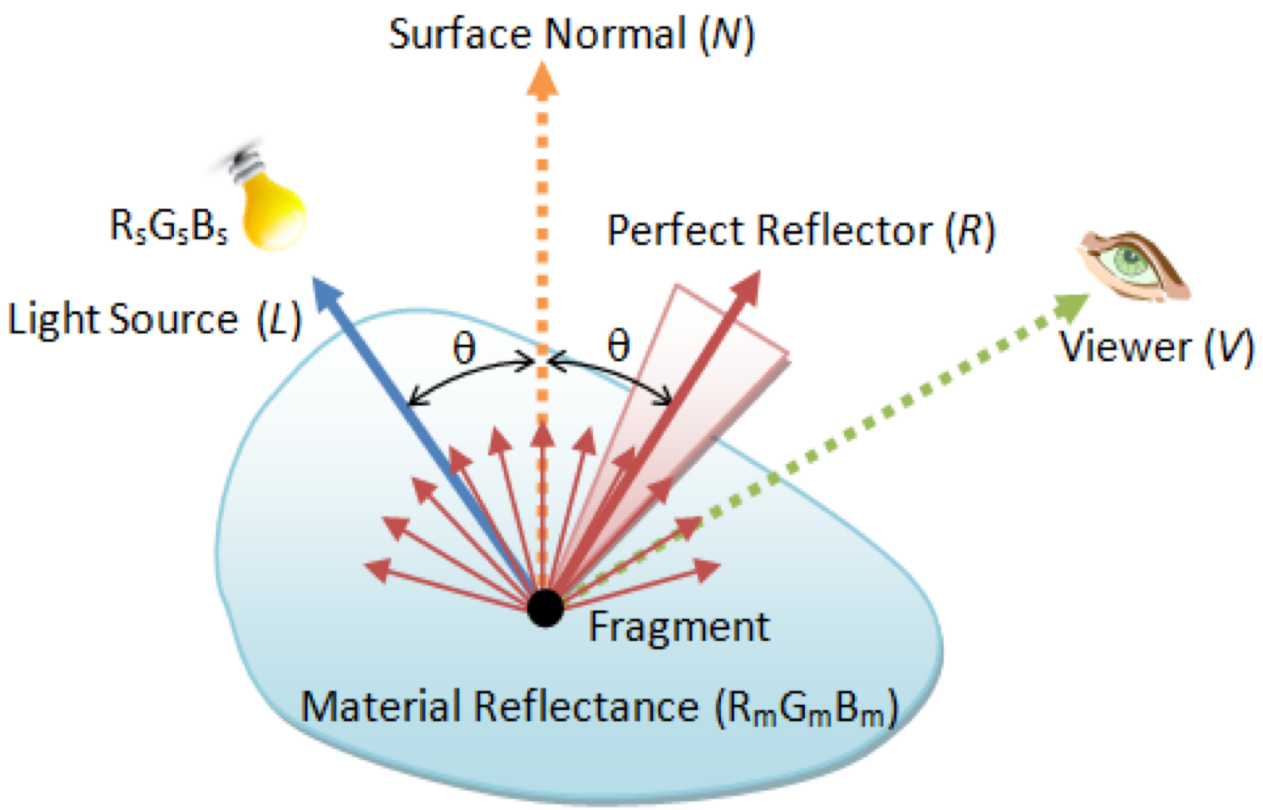


OpenGL Lighting Model

- OpenGL approximates light and lighting as if light can be broken into red, green, and blue components
- The light in a scene comes from several light sources that can be individually turned on and off
- The light sources have an effect only when there are surfaces that absorb and reflect light

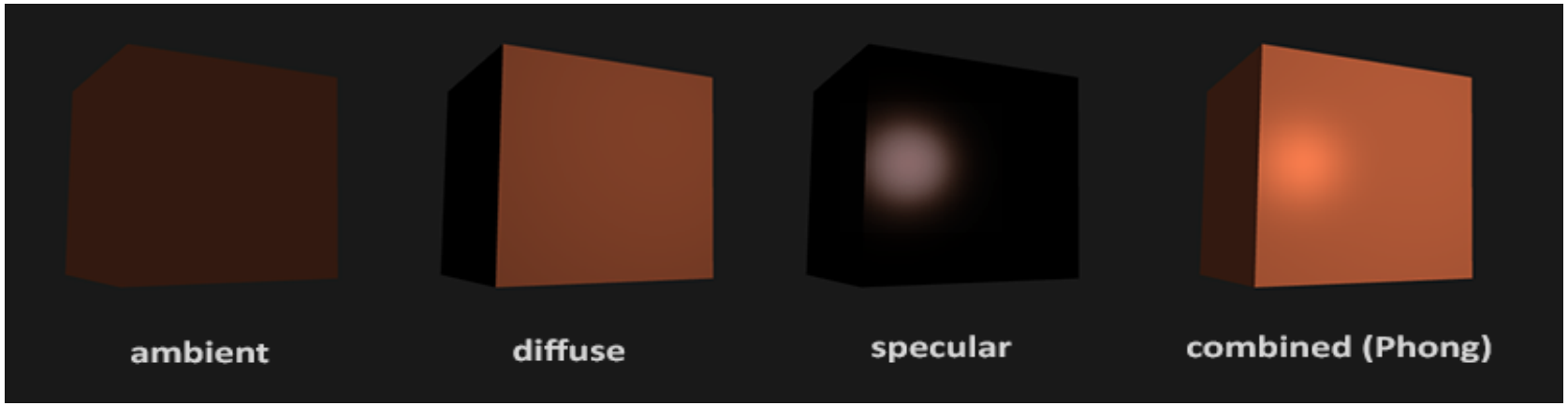
Phong Lighting Model

- A local illumination model
- Compute inexpensive
- Four vectors:
  - The light source *L*
  - The viewer *V*
  - The fragment normal *N*
  - The perfect reflector *R*



Components of Phong Lighting Model

- Ambient lighting, diffuse lighting, and specular lighting



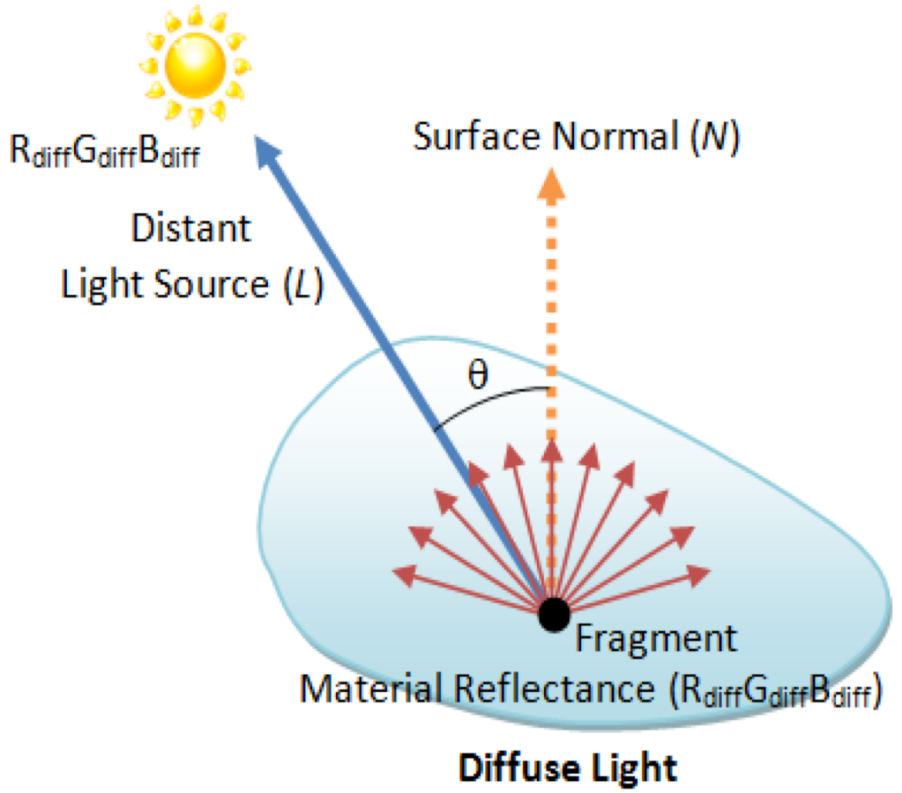
- In 3D applications, you usually don't create ambient, diffuse or specular lights directly. Instead, you use light sources
  - E.g., sun (outdoor), a light bulb (indoors), a flashlight (in a cave)
- These light source types can have different combinations of ambient, diffuse and specular intensities as well as specialized properties

Ambient Light

- One of the properties of light is that it can scatter and bounce in many directions reaching spots that aren't in its direct vicinity; thus reflect on other surfaces and have an indirect impact on the lighting of an object => **Global illumination** (expensive and complicated)
- A very simplistic model of global illumination, is ambient lighting
- Ambient light is a constant amount of light applied to every point of the scene with no origin and no direction
  - The resultant color of the fragment $c_{amb} = s_{amb}\, m_{amb}$
  - s_{amb} is the light ambient color, m_{amb} is the fragment's ambient reflectance
- The ambient light looks somewhat artificial

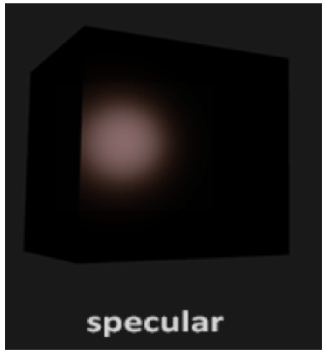
Diffuse Light

- Diffuse light models distant **directional** light source
- The reflected light is scattered equally in all directions, and appears the same to all viewers regardless of their positions
- The resultant color:
  - $c_{diff} = max(L \cdot N, 0)\, s_{diff}\, m_{diff}$
  - s_{diff} is the light diffuse color, m_{diff} is the fragment's diffuse reflectance, $max(L \cdot N, 0)$ is the strength of the incident light
- The most important property of diffuse light is its direction
  - Makes the parts of objects that face the light brighter than the parts that are opposite from it



In order to have any effect on the brightness of a surface the light must hit the surface such that the angle between it and the surface normal will be greater or equal to zero and up to (but not including!) 90 degrees.

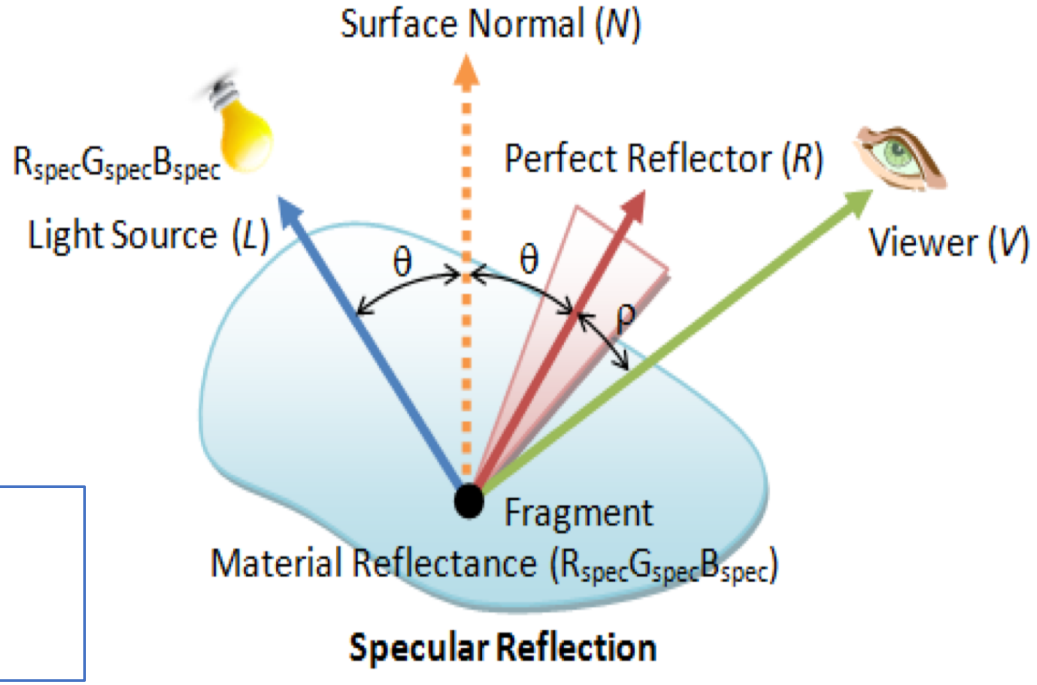
Specular Light



- Specular lighting adds the position of the viewer
  - Light strikes a surface at some angle it is also reflected away at the same angle
  - If the viewer is located exactly somewhere along the way of the reflected light ray it receives a larger amount of light than a viewer who is located further away
  - The end result of specular lighting is that objects will look brighter from certain angles and this brightness will diminish as you move away
- Specular lighting is more a property of the object, rather than the light itself

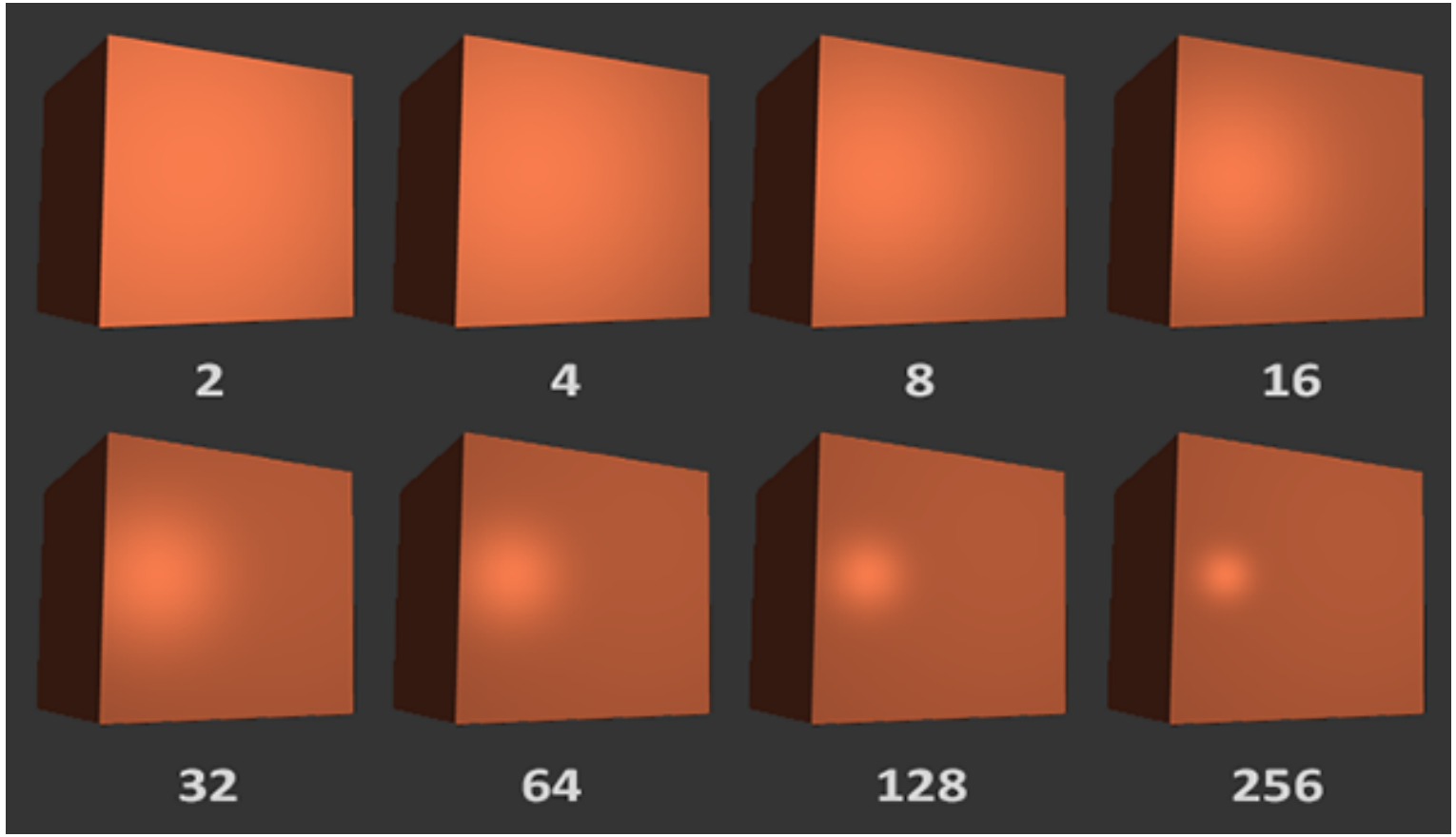
Calculating Specular Light

- Calculating specular lighting must take into consideration both the direction the light hits (and bounces off) as well as the position of the viewer
  - ρ is the angle which is created by the vector R and V
  - The resultant color:
  - $c_{spec} = max(R \cdot V, 0)^{sh} \, s_{spec} \, m_{spec}$
    - sh is the shininess factor ()



As *sh* increases, the light cone becomes narrower (because R·V ≤ 1), the highlighted spot becomes smaller

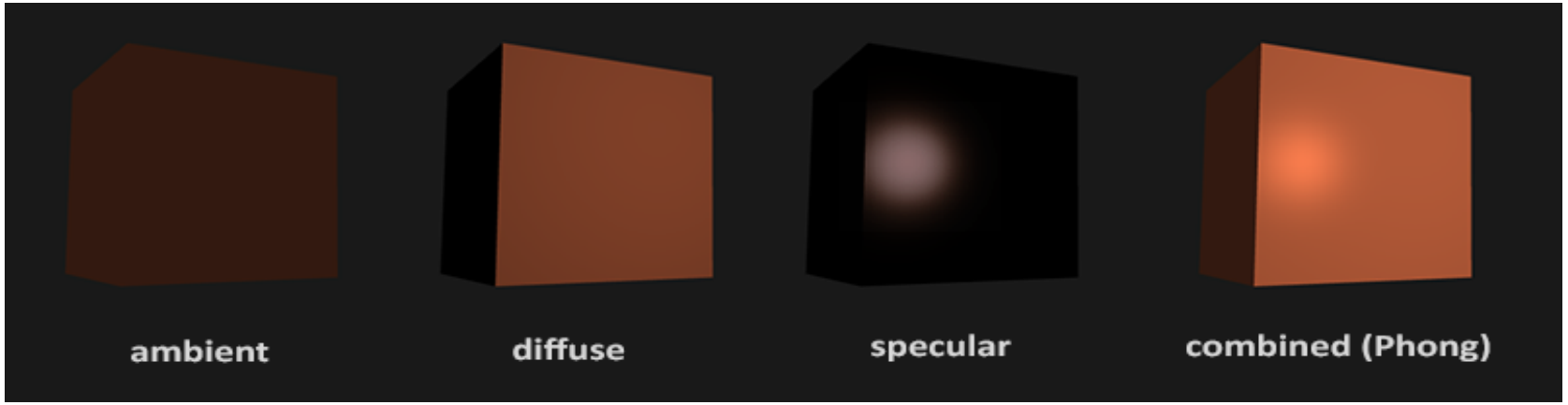
Impact of Shininess Values on Specular Light



The higher the shininess value of an object, the more it properly reflects the light instead of scattering it all around, and thus the smaller the highlight becomes

Combine Ambient, Diffuse, and Specular Light Together

- The resultant color is the sum of the contribution in all the three components:
  - $c_{final} = c_{amb} + c_{diff} + c_{spec}$



Enable OpenGL Lighting

- In OpenGL, you need to enable the lighting state, and each of the light sources, identified via GL_LIGHT*0* to GL_LIGHT*n*.
  - **glEnable**(GL_LIGHTING); // Enable lighting
  - **glEnable**(GL_LIGHT0); // Enable light source 0
  - **glEnable**(GL_LIGHT1); // Enable light source 1
- Once lighting is enabled, color assigned by glColor() are no longer used. Instead, the color depends on the light-material interaction and the viewer's position

Define Light Sources

- void **glLight**{if}(GLenum light, GLenum pname, TYPE param);
- void **glLight**{if}**v**(GLenum light, GLenum pname, TYPE *param);
  - *light*: specifies the light to be created
  - *pname*: specifies the characteristic of the light being set
  - *param*: indicates the values to which the pname characteristic is set

```
GLfloat light_ambient[] = { 0.0, 0.0, 0.0, 1.0 };
GLfloat light_diffuse[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat light_specular[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat light_position[] = { 1.0, 1.0, 1.0, 0.0 };

glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
```

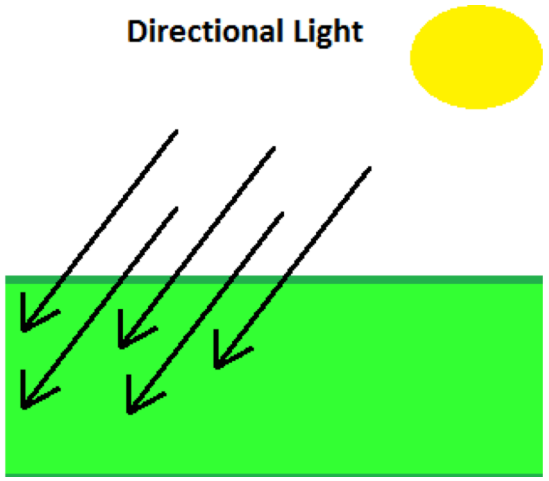
Parameter Name	Default Value	Meaning
GL_AMBIENT	(0.0, 0.0, 0.0, 1.0)	ambient RGBA intensity of light
GL_DIFFUSE	(1.0, 1.0, 1.0, 1.0)	diffuse RGBA intensity of light
GL_SPECULAR	(1.0, 1.0, 1.0, 1.0)	specular RGBA intensity of light
GL_POSITION	(0.0, 0.0, 1.0, 0.0)	(x, y, z, w) position of light
GL_SPOT_DIRECTION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

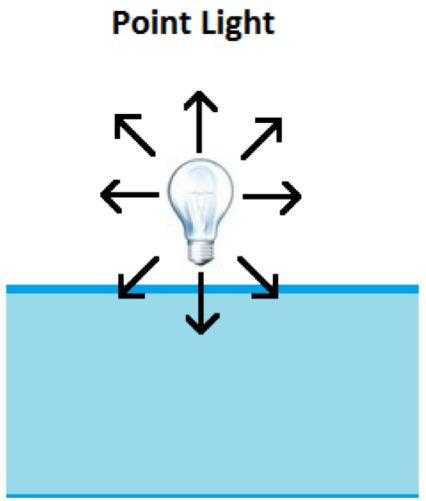
Specially set for GL_LIGHT0

Default values for pname parameters

Lights Position

- OpenGL supports two types of lights
  - Infinite light (directional light)
  - Local light (point light)
- Determined by the light positions you provide
  - GLfloat light_position[] = { x, y, z, w};
  - glLightfv(GL_LIGHT0, GL_POSITION, light_position);
    - w = 0: infinite light source (faster)
    - w != 0: positional light at position (x/w, y/w, z/w)



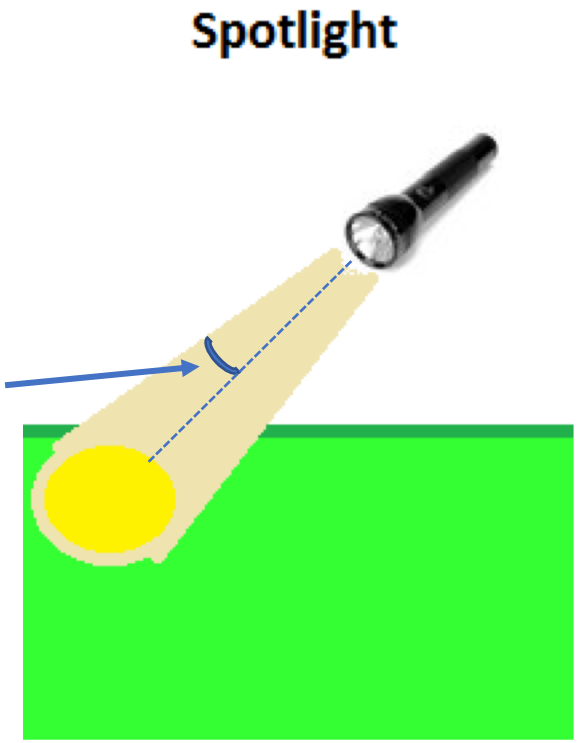


Lights Attenuation

- The intensity of light decreases as distance from the light increases
- OpenGL attenuates a light source by multiplying the contribution of that source by an attenuation factor:
  - Attenuation factor = $\frac{1}{k_c + k_l d + k_q d^2}$
  - d: distance between the light's position and the vertex
  - k_c= GL_CONSTANT_ATTENUATION, default to 1
  - k_l= GL_LINEAR_ATTENUATION , default to 0
  - k_q = GL_QUADRATIC_ATTENUATION , default to 0
  - Change k_c, k_l, and k_q, e.g.,
    - 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

- Restricting the shape of light a positional light emits to a cone
  - Specify the cutoff angle ([0.0, 90.0])
    - glLightf(GL_LIGHT0, **GL_SPOT_CUTOFF**, 45.0);
  - Specify the direction by GL_SPOT_DIRECTION
    - GLfloat spot_direction[] = { -1.0, -1.0, 0.0 };
    - glLightfv(GL_LIGHT0, **GL_SPOT_DIRECTION**, spot_direction);
- Control the intensity distribution of light
  - Multiply the attenuation factor with light's intensity
  - Or set GL_SPOT_EXPONENT
    - Higher spot exponents result in a more focused light source



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 whether it should be 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);
  - Sets properties of the lighting model
  - *pname*: the characteristic of the lighting model to be set
  - *param*: the values to which the pname characteristic is set

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

Default values for pname parameters

Materials

- Objects with different material properties react differently to light
- Like lights, materials have different ambient, diffuse, and specular colors, which determine the ambient, diffuse, and specular reflectance of the material
  - Ambient and diffuse reflectance define the color of the material
  - Specular reflectance is usually white or gray
- Materials also have an **emissive** color to simulate light originating from an object
- A surface has a shininess parameter (GL_SHININESS)
- A surface has two faces: front and back

OpenGL Material

- Use glMaterial() function to specify the ambient, diffuse, and specular component for different faces of surface
  - void **glMaterial**{if}(GLenum face, GLenum pname, TYPE param);
  - void **glMaterial**{if}**v**(GLenum face, GLenum pname, TYPE *param);
    - Specifies a current material property for use in lighting calculations
    - *face*: GL_FRONT, GL_BACK, or GL_FRONT_AND_BACK
    - *pname*: the particular material property to be set
    - *param*: the desired values for the specified property

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

Default values for pname parameters

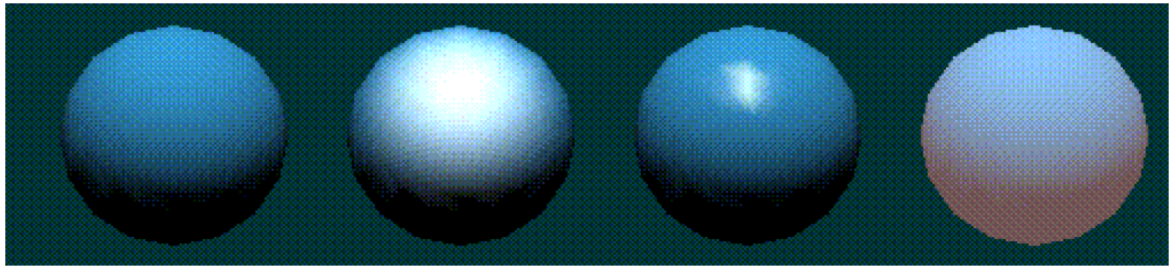
OpenGL Material Examples

- Set diffuse and ambient reflection
  - gLfloat mat_amb_diff[] = { 0.1, 0.5, 0.8, 1.0 };
  - glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, mat_amb_diff);
- Set specular reflection
  - GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
  - GLfloat low_shininess[] = { 5.0 };
  - glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
  - glMaterialfv(GL_FRONT, GL_SHININESS, low_shininess);
- Set emission
  - GLfloat mat_emission[] = {0.3, 0.2, 0.2, 0.0};
  - glMaterialfv(GL_FRONT, GL_EMISSION, mat_emission);

Example: Spheres with Different Materials

```
1    GLfloat no_mat[] = { 0.0, 0.0, 0.0, 1.0 };
2    GLfloat mat_ambient[] = { 0.7, 0.7, 0.7, 1.0 };
3    GLfloat mat_ambient_color[] = { 0.8, 0.8, 0.2, 1.0 };
4    GLfloat mat_diffuse[] = { 0.1, 0.5, 0.8, 1.0 };
5    GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
6    GLfloat no_shininess[] = { 0.0 };
7    GLfloat low_shininess[] = { 5.0 };
8    GLfloat high_shininess[] = { 100.0 };
9    GLfloat mat_emission[] = {0.3, 0.2, 0.2, 0.0};
10
11   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
12
13 /*  draw sphere in first row, first column
14  *  diffuse reflection only; no ambient or specular
15  */
16     glPushMatrix();
17     glTranslatef (-3.75, 3.0, 0.0);
18     glMaterialfv(GL_FRONT, GL_AMBIENT, no_mat);
19     glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
20     glMaterialfv(GL_FRONT, GL_SPECULAR, no_mat);
21     glMaterialfv(GL_FRONT, GL_SHININESS, no_shininess);
22     glMaterialfv(GL_FRONT, GL_EMISSION, no_mat);
23     glutSolidSphere(1.0, 16, 16);
24     glPopMatrix();
25
26 /*  draw sphere in first row, second column
27  *  diffuse and specular reflection; low shininess; no ambient
28  */
29     glPushMatrix();
30     glTranslatef (-1.25, 3.0, 0.0);
31     glMaterialfv(GL_FRONT, GL_AMBIENT, no_mat);
32     glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
33     glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
34     glMaterialfv(GL_FRONT, GL_SHININESS, low_shininess);
35     glMaterialfv(GL_FRONT, GL_EMISSION, no_mat);
36     glutSolidSphere(1.0, 16, 16);
37     glPopMatrix();
```

```
38
39 /*  draw sphere in first row, third column
40  *  diffuse and specular reflection; high shininess; no ambient
41  */
42     glPushMatrix();
43     glTranslatef (1.25, 3.0, 0.0);
44     glMaterialfv(GL_FRONT, GL_AMBIENT, no_mat);
45     glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
46     glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
47     glMaterialfv(GL_FRONT, GL_SHININESS, high_shininess);
48     glMaterialfv(GL_FRONT, GL_EMISSION, no_mat);
49     glutSolidSphere(1.0, 16, 16);
50     glPopMatrix();
51
52 /*  draw sphere in first row, fourth column
53  *  diffuse reflection; emission; no ambient or specular refl.
54  */
55     glPushMatrix();
56     glTranslatef (3.75, 3.0, 0.0);
57     glMaterialfv(GL_FRONT, GL_AMBIENT, no_mat);
58     glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
59     glMaterialfv(GL_FRONT, GL_SPECULAR, no_mat);
60     glMaterialfv(GL_FRONT, GL_SHININESS, no_shininess);
61     glMaterialfv(GL_FRONT, GL_EMISSION, mat_emission);
62     glutSolidSphere(1.0, 16, 16);
63     glPopMatrix();
```



Minimize Material-Property Changes with glColorMaterial()

- void **glColorMaterial**(GLenum face, GLenum mode);
  - Causes the material property specified by *mode* and the material face specified by *face* to track the value of the current color at all times
  - *face*: GL_FRONT, GL_BACK, or GL_FRONT_AND_BACK (the default)
  - *mode*: GL_AMBIENT, GL_DIFFUSE, GL_AMBIENT_AND_DIFFUSE (the default), GL_SPECULAR, or GL_EMISSION
  - At any given time, only one mode is active

```
glEnable(GL_COLOR_MATERIAL);
glColorMaterial(GL_FRONT, GL_DIFFUSE);
// now glColor* changes diffuse reflection
glColor3f(0.2, 0.5, 0.8);
/* draw some objects here */
glColorMaterial(GL_FRONT, GL_SPECULAR);
// glColor* no longer changes diffuse reflection
// now glColor* changes specular reflection
glColor3f(0.9, 0.0, 0.2);
/* draw other objects here */
glDisable(GL_COLOR_MATERIAL);
```

An example

- Use glColorMaterial() whenever you need to change a single material parameter for most vertices in your scene
  - If you need to change more than one material parameter, use glMaterial*()

RGB Values for Lights and Materials

- The color of light sources is characterized by the **amount** of red, green, and blue light they emit
  - For a light, the RGB numbers correspond to a percentage of full intensity for each color
    - (1, 1, 1): brightest white; (0.5, 0.5, 0.5): white with half intensity (gray); (1, 1, 0): yellow
- The material of surfaces is characterized by the **percentage** of the incoming red, green, and blue components that is reflected in various directions
  - For materials, the numbers correspond to the reflected proportions of those colors
    - (1, 0.5, 0): reflects all the incoming red light, half the incoming green, and none of the incoming blue light
- Add two lights (R1, G1, B1) and (R2, G2, B2) gives (R1+R2, G1+G2, B1+B2)
- Combine the effects of a light (LR, LG, LB) with a material (MR, MG, MB) gives (LR*MR, LG*MG, LB*MB)

Steps to Add Lighting to Your Scene

- Define normal vectors for each vertex of all the objects. These normals determine the orientation of the object relative to the light sources
- Create, select, and position one or more light sources
- Create and select a lighting model, which defines the level of global ambient light and the effective location of the viewpoint (for the purposes of lighting calculations)
- Define material properties for the objects in the scene

A Simple Example: Drawing a Lit Sphere

```
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>

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

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

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

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

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

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

A Simple Example: Drawing a Lit Sphere