Computer Graphics

Rendering Faces for Visual Realism
Ch8 Sec1-2

Visual Realism Requirements

- Light Sources
- Materials (e.g., plastic, metal)
- Shading Models
- Depth Buffer Hidden Surface Removal
- Textures
- Reflections
- Shadows

Rendering Objects

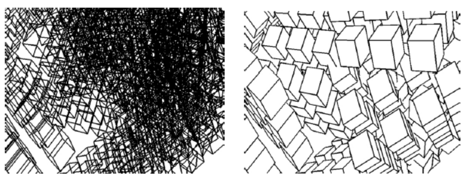
- We know how to **model** mesh objects, manipulate a jib camera, view objects, and make pictures.
- Now we want to make these objects look visually interesting, realistic, or both.
- We want to develop methods of **rendering** a picture of the objects of interest: *computing* how each pixel of a picture should look.

Rendering Objects (2)

- Much of rendering is based on different **shading models**, which describe how light from light sources interacts with objects in a scene.
  - It is impractical to simulate all of the physical principles of light scattering and reflection.
  - A number of approximate models have been invented that do a good job and produce various levels of realism.

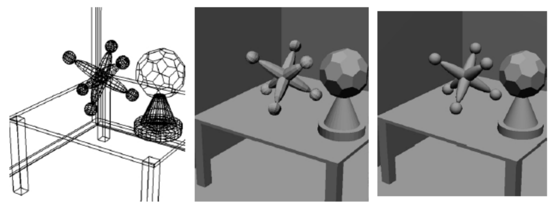
Rendering

- Rendering: deciding how a pixel should look
- Example: compare wireframe (left) to wire-frame with hidden surface removal (right)



Rendering (2)

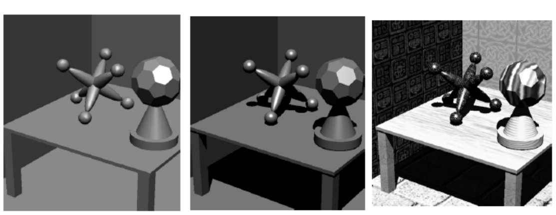
- Example: Compare mesh objects drawn using wire-frame, flat shading, smooth (Gouraud) shading



a) b)

Rendering (3)

- Compare to images with specular highlights, shadows, textures



Visual Realism Requirements

- Light Sources
- Materials (e.g., plastic, metal)
- Shading Models
- Depth Buffer Hidden Surface Removal
- Textures
- Reflections
- Shadows

Hidden Surface

- Hidden Surface Removal is very important in 3D scenes
- Only surfaces closest to the eye should be seen and objects that are hidden by others should be eliminated.
- The use of a depth buffer facilitates hidden surface removal.

Hidden Surface Removal

- To use depth-buffering, you need to enable it:

```
glutInitDisplayMode (GLUT_DEPTH| ..);
glEnable (GL_DEPTH_TEST);
```

- Initialize the depth buffer and color buffer by using:

```
glClear(GL_DEPTH_BUFFER_BIT |
 GL_COLOR_BUFFER_BIT);
```


Hidden Surface Removal

- Put them together:

```
glutInitDisplayMode(GLUT_DEPTH|..);
glEnable(GL_DEPTH_TEST);
...
// in display function
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
draw3DObjectA();
draw3DObjectB();
```


Visual Realism Requirements

- Light Sources
- Materials (e.g., plastic, metal)
- Shading Models
- Depth Buffer Hidden Surface Removal
- Textures
- Reflections
- Shadows

What is light?

- Light is the most important idea behind visual representation of anything a human can visually perceive.
- What you see isn't based on the objects that you are viewing but on the rays of light reflected from those objects.
- Your eyes don't directly see objects.
- There is no physical correlation between your eyes and those objects.

What is light?

- Light rays originate from an energy source (sun or lamp)
- Theoretically, a ray of light travels in a straight line.
- Your perception of an object comes from the light reflected or scattered off of an object that your eyes absorb.

Middle Tennessee State University

What is light?

- Two rules:
  - Your eyes are mechanisms that perceive or absorb photons of light and not objects. Your as a programmer, must simulate this functionality on the computer screen.
  - A ray of light travels in a straight line (not exactly true but we can think of it this way).

Middle Tennessee State University

What is light?

- Albert Einstein in 1905 developed the theory of light:
  - He described the "photoelectric effect".
  - Described the activity of the ultraviolet light hitting a surface and emitting electrons off that surface.
  - This behavior was supported by an explanation that light was made up of a stream of energy packets called photons.

Middle Tennessee State University

Color of light

- Light seen by human eye is a mixture of lights scattered and reflected against the surroundings of different material property.
- All physical matter is made up of atoms.
- Reflection of photons off of physical matter depends on
  - Kind of atoms
  - Amount of each kind
  - Arrangement of atoms in the object

Color of light

- Some photons are absorbed
- Some photons are reflected
- Color that the material reflects is observed as that material's color
- The more light the material reflects, the more shiny it will appear to the viewer.
- Each color is simply energy that can be represented by a wavelength.

Color of light

- Color is only a wavelength visible to the eye.
- A wavelength is measured by the distance between the peaks of the energy wave:

- Visible light is contained within the wavelengths ranging from 390 nanometers to 720 nanometers in length.

Color of light

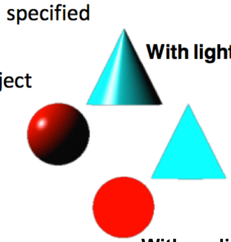
- 390 nanometers is the color violet
- 720 nanometers is the color red

- Everything in between is visible light and the range from 390 to 720 is called the color spectrum.

Lighting Principles

Lighting simulates how objects reflect light

– Lighting properties must be specified
– Light's color and position
– Material composition of object
– Global lighting parameters

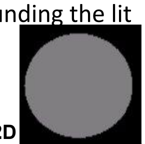


With light

With no light

Lighting Properties:

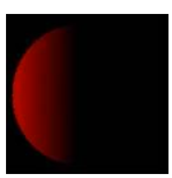
- Light comes from light sources that can be turned on/off
- ***Ambient*** lighting comes from light that is so scattered there is no way to tell its original location
- Ambient light is the average volume of light created from all light sources surrounding the lit area.
- Example: Backlighting in a room



3D sphere looks 2D

Lighting Properties:

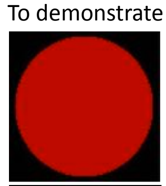
- The ***diffuse*** component of lighting is light that comes from one direction.
- Diffuse light has a position in space and comes from a single direction.
- Once it hits a surface, it is scattered equally in all directions.
- Example: A flashlight



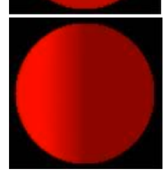
A diffuse red light cast onto a black object

Lighting Properties:

- To demonstrate how Ambient& Diffuse light works together:



Dark red ambient light only



Add diffuse light on the right side
Note sphere now looks3D

Lighting Properties:

- Specular light comes from a particular direction like a diffuse light.
- It bounces off the surface in a particular direction
- It relies on the angle between the viewer and the light source.
- Creates a highlighted area

Lighting Properties:

- Emissive light is responsible for the object's material's property to reflect or absorb light.
- When applied to an object's material, emissive light simulates the light reflected off the object.

RGB Values for Lights

- A light source is characterized by the amount of red, green, & blue light it emits.
- Examples: If R=G=B=1.0, the light is the brightest possible white.
- If R=G=B=.5, the color is still white, but only at half intensity, so it appears gray
- If R=G=1.0 and B=0.0, the light appears yellow.

How OpenGL Simulates Lights

- Gourad lighting model
  - Computed at vertices
- Lighting contributors
  - Lighting properties
  - Lighting model properties
  - Surface material properties

Setting Lighting Properties

glLightfv(*light, property, value*);

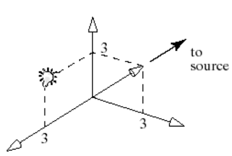
- **light specifies which light**
  - multiple lights (at least 8), starting with GL_LIGHT0
- **Properties**
  - Colors for ambient, diffuse, & specular component
  - position and type
  - attenuation

Types of Lights

- OpenGL supports two types of Lights
  - Local (Point) light sourcess of Light
    - Local light positioned at (x, y, z, 1)
  - Infinite (Directional) light sources
    - Infinite light directed along (x, y, z, 0)

Point and Vector Light Locations

- The figure shows a local source at (0, 3, 3, 1) and a remote source "located" along vector (3, 3, 0, 0).
- Infinitely remote light sources are often called "**directional**". There are computational advantages to using directional light sources, since direction **s** in the calculations of diffuse and specular reflections is *constant* for all vertices in the scene.
- But directional light sources are not always the correct choice: some visual effects are properly achieved only when a light source is close to an object.



Creating and Using Light Sources in Open-GL

- Each light has a position specified in homogeneous coordinates using a GLfloat array named, for example, litePos.

  Glfloat litePos[4]={3, 3, 1, 1};
- The light is created using

  glLightfv (GL_LIGHT0, GL_POSITION, litePos);
- If the position is a vector (4th component = 0), the source is infinitely remote (like the sun).

Creating and Using Light Sources in OpenGL (2)

- The light color is specified by a 4-component array [R, G, B, A] of GLfloat, named (e.g.) amb0. The A value can be set to 1.0 for now:

  Glfloat amb0={0.2, 0.8, 0.0, 1.0};
- The light color is specified by

  glLightfv (GL_LIGHT_0, GL_AMBIENT, amb0);
- Similar statements specify GL_DIFFUSE and GL_SPECULAR.

Creating and Using Light Sources in OpenGL (3)

- Lights do not work unless you turn them on.
  - In your main program, add the statements

    glEnable (GL_LIGHTING);

    glEnable (GL_LIGHT0);
  - If you are using other lights, you will need to enable them also.
- To turn off a light

  glDisable (GL_LIGHT0);
- To turn them all off,

  glDisable (GL_LIGHTING);

Creating an Entire Light

```
GLfloat amb0[ ] = {0.2, 0.4, 0.6, 1.0};

// define some colors
GLfloat diff0[ ] = {0.8, 0.9, 0.5, 1.0};
GLfloat spec0[ ] = { 1.0, 0.8, 1.0, 1.0};
glLightfv(GL_LIGHT0, GL_AMBIENT, amb0);

// attach them to LIGHT0
glLightfv(GL_LIGHT0, GL_DIFFUSE, diff0);
glLightfv(GL_LIGHT0, GL_SPECULAR, spec0);
```

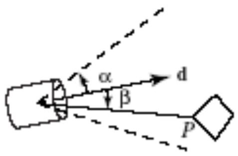

Creating and Using Light Sources in OpenGL (4)

- Global ambient light is present even if no lights are created. Its default color is {0.2, 0.2, 0.2, 1.0}.
- To change this value, create a GLfloat array of values newambient and use the statement

  glLightModelfv (GL_LIGHT_MODEL_AMBIENT, newambient);
- Default light values are:
  - {0, 0, 0, 1} for ambient for all lights,
  - {1, 1, 1, 1} for diffuse and specular light for LIGHT_0, and
  - {0, 0, 0, 1} for diffuse and specular light for lights LIGHT_1 to LIGHT_7

Spotlights in Open-GL

- A spotlight emits light only in a cone of directions; there is no light outside the cone. Inside the cone, $I = I_s(\cos \beta)^\varepsilon$, where $\cos \beta$ uses the angle between **d** and a line from the source to *P*.



Spotlights in OpenGL (2)

- To create the spotlight, create a GLfloat array for **d**. Default values are **d** = {0, 0, 0, 1}, $\alpha = 180^o$, $\varepsilon = 0$: a point source.
- Then add the statements:

```
Glfloat d={2, 2, 2, 1};
glLightf (GL_LIGHT0, GL_SPOT_CUTOFF, 45.0); //45.0 is α in
    degrees)
glLightf (GL_LIGHT0, GL_SPOT_EXPONENT, 4.0); //4.0 is ε)
glLightfv (GL_LIGHT0, GL_SPOT_DIRECTION, d);
```


Attenuation of Light with Distance

- OpenGL also allows you to specify how rapidly light diminishes with distance from a source.
- OpenGL attenuates the strength of a positional light source by the following attenuation factor:

$$atten = \frac{1}{k_c + k_l D + k_q D^2}$$

where k_c, k_l, and k_q are coefficients and *D* is the distance between the light's position and the vertex in question.

Attenuation of Light with Distance (2)

$$atten = \frac{1}{k_c + k_l D + k_q D^2}$$

- These parameters are controlled by calling:

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

The default values are $k_c = 1$, $k_l = 0$, and $k_q = 0$ (no attenuation).

Moving Light Sources in OpenGL

- To move a light source independently of the camera:
  - set its position array,
  - clear the color and depth buffers,
  - set up the ModelView matrix to use for everything except the light source and push the matrix
  - move the light source and set its position
  - pop the matrix
  - set up the camera, and draw the objects.

How OpenGL Simulates Lights

- Gourad lighting model
  - Computed at vertices
- Lighting contributors
  - Lighting properties
  - Lighting model properties
  - Surface material properties

Changing the OpenGL Light Model

- **The color of global ambient light**: specify its color using:
  GLfloat amb[ ] = {0.2, 0.3, 0.1, 1.0};
  glLightModelfv(GL_LIGHT_MODEL_AMBIENT, amb);

- **Is the viewpoint local or remote?**
  - OpenGL computes specular reflections using the "halfway vector" **h** = **s** + **v** . The true directions **s** and **v** are normally different at each vertex in a mesh.
  - OpenGL uses **v** = (0, 0, 1), along the positive z-axis, to increase rendering speed. To use the true value of **v** for each vertex, execute

  glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, GL_TRUE);

Changing the OpenGL Light Model (2)

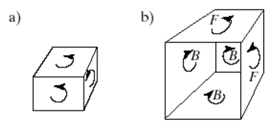
- **Are both sides of a polygon shaded properly?**
  Each polygonal face in a model has two sides. When modeling, we tend to think of them as the "inside" and "outside" surfaces. The convention is to list the vertices of a face in counter-clockwise (CCW) order as seen from outside the object.
- OpenGL has no notion of inside and outside. It can only distinguish between "front faces" and "back faces". A face is a **front face** if its vertices are listed in counter-clockwise (CCW) order as seen by the eye.

Changing the OpenGL Light Model (3)

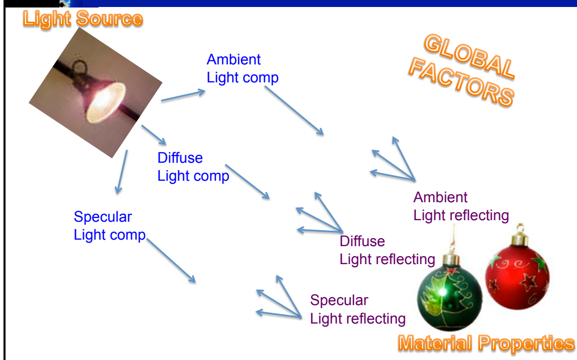
- For a space-enclosing object (a) all visible faces are front faces; OpenGL draws them properly with the correct shading.
- If a box has a face removed (b), OpenGL does not shade back faces properly. To force OpenGL to shade back faces, use:

```
glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE);
```

- OpenGL reverses the normal vectors of any back-face and performs shading properly.
- Replace GL_TRUE with GL_FALSE (the default) to turn off this facility.



Steps in Adding Lighting in OpenGL



Steps in Adding Lighting in OpenGL (1)

Step 1: Establish vertex normal

1. Make sure the normal for each vertex has been defined
2. Draw with :

```
glNormal3f(0.0, 0.0, 1.0);
glVertex3f(-50.0, 50.0, 10.0);
```

3. One normal for one vertex, or one normal for a group of vertices

Steps in Adding Lighting in OpenGL (2)

Step 2: Enable/Define the necessaries:

- glEnable(GL_LIGHTING);
- glEnable(GL_DEPTH_TEST); // perform hidden depth removal
- glEnable(GL_NORMALIZE); // !!! important to get the right shading !!!
- glShadeModel (GL_SMOOTH);
- Optional:
  - glFrontFace(GL_CCW);

Steps in Adding Lighting in OpenGL (3)

Step 3. Setup the Light source

```
GLfloat  sourceLight[] = { 0.25, 0.25, 0.25, 1.0 };
Glfloat specularLight[] = {0.8, 0.8, 0.8, 1.0};
GLfloat lightPos[] = { -50.0, 25.0, 250.0, 0.0 };
Glfloat shininess[] = {50.0};

glLightfv(GL_LIGHT0,GL_AMBIENT,sourceLight);
glLightfv(GL_LIGHT0,GL_DIFFUSE,sourceLight);
glLightfv(GL_LIGHT0,GL_DIFFUSE,specularLight);
glLightfv(GL_LIGHT0,GL_POSITION,lightPos);

glEnable(GL_LIGHT0);
```


Steps in Adding Lighting in OpenGL (4)

Step 3. Setup the Light source continued …

- Optional:
  - Setup multiple light sources (at different locations)
  - Specify certain light source as spot light (α and ε)

Steps in Adding Lighting in OpenGL (5)

Step 4. Setup the material properties of each object

```
GLfloat mat_color[] = {0.23, 0.23, 0.23, 1.0 };
GLfloat mat_specular[] = {0.5, 0.5, 0.5, 1.0 };
GLfloat mat_shininess[] = {100.0 };

glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
glMaterialfv(GL_FRONT, GL_AMBIENT, mat_color);
glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_color);
```


Steps in Adding Lighting in OpenGL (6)

Step 5. Define various options in the global shading model

```
GLfloat amb[ ] = {0.2, 0.3, 0.1, 1.0};
glLightModelfv(GL_LIGHT_MODEL_AMBIENT, amb);

glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE);

glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, 2.0);
glLightf(GL_LIGHT0, GL_LINEAR_ATTENUATION, 2.0);
glLightf(GL_LIGHT0, GL_QUADRATIC_ATTENUATION, 2.0);

…
```

Steps in Adding Lighting in OpenGL (7)

Useful Optional Step for setting up Color Tracking

```
// Enable color tracking
glEnable(GL_COLOR_MATERIAL);

// Set Material properties to follow glColor values
glColorMaterial(GL_FRONT, GL_AMBIENT_AND_DIFFUSE);

// use glColor to specify color directly
glColor3f(1.0, 0.0, 0.0);
glNormal3f(0.0, 0.0, 1.0);
glVertex3f(-50.0, 50.0, 20.0);
```


Code: Independent Motion of Light

```
void display()
{
    GLfloat position[ ] = {2, 1, 3, 1}; //initial light position
                                       // clear color and depth buffers
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    glPushMatrix();
    glTranslated(...);            // move the light
    glLightfv(GL_LIGHT0, GL_POSITION, position);
    glPopMatrix();

    gluLookAt(...);           // set the camera position

    <.. draw the object ..>

    glutSwapBuffers();
}
```


Code: Light Moves with Camera

```
GLfloat pos[ ] = {0,0,0,1};
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();

glLightfv(GL_LIGHT0, GL_POSITION, position); // light at (0,0,0)
gluLookAt(...);               // move light and camera
                              // draw the object
<.. draw the object ..>
```


Visual Realism Requirements

- Light Sources
- Materials (e.g., plastic, metal)
- Shading Models
- Depth Buffer Hidden Surface Removal
- Textures
- Reflections
- Shadows

Shading Models: Introduction

- Assume to start with that light has no color, only brightness: R = G = B
- Assume we also have a point source of light (sun or lamp) and general ambient light

Shading Models: Introduction (4)

- In the simplest model, specular reflected light has the same color as the incident light. This tends to make the material look like plastic.
- In a more complex model, the color of the specular light varies over the highlight, providing a better approximation to the shininess of metal surfaces.

Shading Models: Introduction (5)

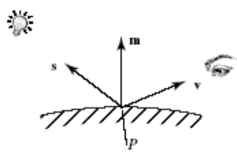
- Most surfaces produce some combination of diffuse and specular reflection, depending on surface characteristics such as roughness and type of material.
- The total light reflected from the surface in a certain direction is the sum of the diffuse component and the specular component.
  - For each surface point of interest, we compute the size of each component that reaches the eye.

Reflected Light Model

- Finding Reflected Light: a model
  - Model is not completely physically correct, but it provides fast and relatively good results on the screen.
  - Intensity of a light is related to its brightness. We will use I_s for intensity, where s is R or G or B.

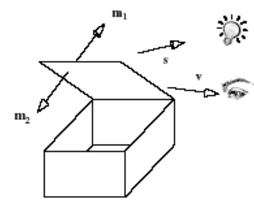
Calculating Reflected Light

- To compute reflected light at point P, we need 3 vectors: normal **m** to the surface at P and vectors **s** from P to the source and **v** from P to the eye. We use world coordinates.



Calculating Reflected Light (2)

- Each face of a mesh object has an inside and an outside.
- Normally the eye sees only the outside (front, in Open-GL), and we calculate only light reflected from the outside.



Calculating Reflected Light (3)

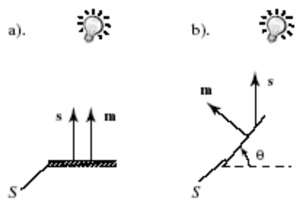
- If the eye can see inside, we must also compute reflections from the inside (back, in OpenGL).
  - If **v** · **m** > 0, the eye can see the face and lighting must be calculated.

Calculating Diffuse Light

- Diffuse scattering is assumed to be independent of the direction from the point, *P*, to the location of the viewer's eye.
- Because the scattering is uniform in all directions, the orientation of the facet *F* relative to the eye is not significant, and I_d is independent of the angle between **m** and **v** (unless **v** · **m** < 0, making I_d =0.)
- The amount of light that illuminates the facet *does* depend on the orientation of the facet relative to the point source: the amount of light is proportional to the area of the facet that it sees: the area *subtended* by a facet.

Calculating Diffuse Light (2)

- The intensity depends on the projection of the face perpendicular to **s** (Lambert's law): $I_d \cos\theta$



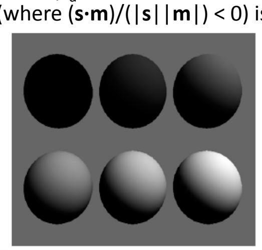
Calculating Diffuse Light (3)

- For ϑ near 0°, brightness varies only slightly with angle, because the cosine changes slowly there. As ϑ approaches 90°, the brightness falls rapidly to 0.
- We know $\cos \vartheta = (\mathbf{s} \cdot \mathbf{m})/(|\mathbf{s}||\mathbf{m}|)$.
- $I_d = I_s \rho_d (\mathbf{s} \cdot \mathbf{m})/(|\mathbf{s}||\mathbf{m}|)$.
  - I_s is the intensity of the source.
  - ρ_d is the diffuse reflection coefficient and depends on the material the object is made of.
- $\mathbf{s} \cdot \mathbf{m} < 0$ implies $I_d = 0$.
- So to take all cases into account, we use $I_d = I_s \rho_d \max[(\mathbf{s} \cdot \mathbf{m})/(|\mathbf{s}||\mathbf{m}|), 0]$.

Example: Spheres Illuminated with Diffuse Light.

- Spheres with reflection coefficients from 0 to 1 by 0.2.
- The source intensity is 1.0. Background intensity is 0.4.
- The sphere is totally black when $\rho_d = 0.0$, and the shadow in its bottom half (where $(\mathbf{s} \cdot \mathbf{m})/(|\mathbf{s}||\mathbf{m}|) < 0$) is also black.

I_d=1.0,
I_a=0.4
ρ_d=0, 0.2, 0.4, 0.6, 0.8, 1.0



Calculating the Specular Component

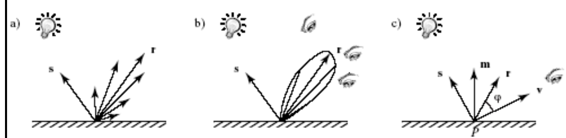
- Real objects do not scatter light uniformly in all directions; a specular component is added to the shading model.
- Specular reflection causes highlights, which can add significantly to realism of a picture when objects are shiny.

Calculating the Specular Component (2)

- A simple model for specular light was developed by Phong. It is easy to apply.
  - The highlights generated by the Phong model give an object a plastic-like or glass-like appearance.
  - The Phong model is less successful with objects that are supposed to have a shiny metallic surface, although you can roughly approximate them with OpenGL by careful choices of certain color parameters.

Calculating the Specular Component (3)

- Most of the light reflects at equal angles from the (smooth and/or shiny) surface, along direction **r,** the reflected direction.

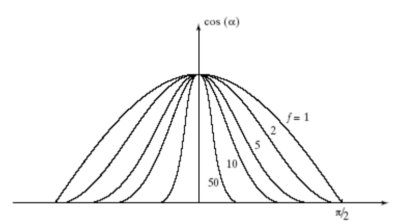


Calculating the Specular Component (2)

- In Ch. 4, we found that **r** = -**s** + 2 **m** (**s**·**m**)/(|**m**|^2) (mirror reflection direction).
- For surfaces that are not mirrors, the amount of reflected light decreases as the angle ϕ between **r** and **v** (vector from reflection point to eye) increases.
- For a simplified model, we say the intensity decreases as $\cos^f \phi$, where f is chosen experimentally between 1 and 200.

Calculating the Specular Component (3)

- The effect of f : large f values give concentrated highlights; smaller ones give larger dimmer highlights.

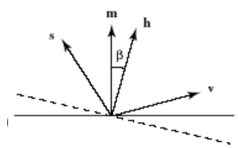


Calculating the Specular Component (4)

- $\cos \phi = \mathbf{r}\cdot\mathbf{v}/(|\mathbf{r}||\mathbf{v}|)$
- $I_{sp} = I_s \rho_s (\mathbf{r}\cdot\mathbf{v}/(|\mathbf{r}||\mathbf{v}|))^f$.
  - ρ_s is the specular reflection coefficient, which depends on the material.
- If $\mathbf{r}\cdot\mathbf{v} < 0$, there is no reflected specular light.
- $I_{sp} = I_s \rho_s \max[(\mathbf{r}\cdot\mathbf{v}/(|\mathbf{r}||\mathbf{v}|))^f, 0]$.

Speeding up Calculations for Specular Light

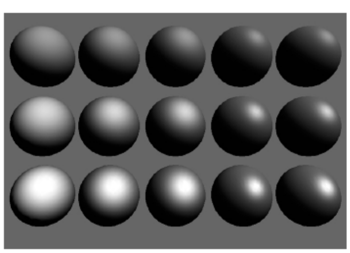
- Find the halfway vector $\mathbf{h} = \mathbf{s} + \mathbf{v}$.
- Then the angle β between **h** and **m** approximately measures the falloff of intensity. To take care of errors, we use a different f value, and write



$I_{sp} = I_s \rho_s \max[(\mathbf{h}\cdot\mathbf{m}/(|\mathbf{h}||\mathbf{m}|))^f,$

Calculating the Specular Component (5)

- From bottom to top, ρ_s = 0.75, 0.5, 0.25. From left to right, f = 3, 6, 9, 25, 200.
- ρ_a= 0.1
- ρ_d= 0.4



Ambient Light

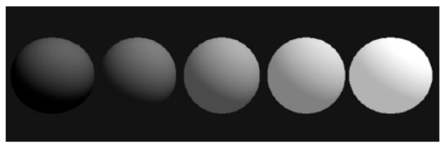
- Our desire for a simple reflection model leaves us with far from perfect renderings of a scene.
  - E.g., shadows appear to be unrealistically deep and harsh.
- To soften these shadows, we can add a third light component called *ambient light*.

Calculating Ambient Light

- The source is assigned an intensity, I_a.
- Each face in the model is assigned a value for its **ambient reflection coefficient**, ρ_a (often this is the same as the diffuse reflection coefficient, ρ_d), and the term $I_a \rho_a$ is simply added to whatever diffuse and specular light is reaching the eye from each point *P* on that face.
- I_a and ρ_a are usually arrived at experimentally, by trying various values and seeing what looks best.

Adding Ambient Light to Diffuse Reflection

- The diffuse and ambient sources have intensity 1.0, and $\rho_d = 0.4$. $\rho_a = 0, 0.1, 0.3, 0.5, 0.7$ (left to right).
- Modest ambient light softens shadows; too much ambient light washes out shadows.



Combining Light Contributions and Adding Color

- $I = I_a \rho_a + I_s \rho_d$ lambert $+ I_{sp} \rho_s$ x phong^f

  where Lambert = max[(**s**·**m**)/(|**s**||**m**|), 0] and Phong = max[(**h**·**m**/(|**h**||**m**|), 0]
- To add color, we use 3 separate total intensities like that above, one each for Red, Green, and Blue, which combine to give any desired color of light.
- We say the light sources have three types of color: ambient = (I_{ar}, I_{ag}, I_{ab}), diffuse = (I_{dr}, I_{dg}, I_{db}), and specular = (I_{spr}, I_{spg}, I_{spb}).

Combining Light Contributions and Adding Color (2)

- Generally the diffuse light and the specular light have the same intensity.
- ρ_s is the same for R, G, and B, so specular light is the color of the light source.
- An object's color (in white light) is specified by 9 coefficients (ambient and diffuse are color of object):
- ambient reflection coefficients: ρ_{ar}, ρ_{ag}, and $\rho_{ab;}$
- diffuse reflection coefficients: ρ_{dr}, ρ_{dg}, and ρ_{db}
- specular reflection coefficients: ρ_{sr}, ρ_{sg}, and ρ_{sb}

Example

- If the color of a sphere is 30% red, 45% green, and 25% blue, it makes sense to set its ambient and diffuse reflection coefficients to (0.3*K*, 0.45*K*, 0.25*K)* respectively, where *K* is some scaling value that determines the overall fraction of incident light that is reflected from the sphere.
- Now if it is bathed in white light having equal amounts of red, green, and blue ($I_{sr} = I_{sg} = I_{sb} = I$) the individual diffuse components have intensities $I_r = 0.3\ K\ I$, $I_g = 0.45\ K\ I$, $I_b = 0.25\ K\ I$, so as expected we see a color that is 30% red, 45% green, and 25% blue.

Example (2)

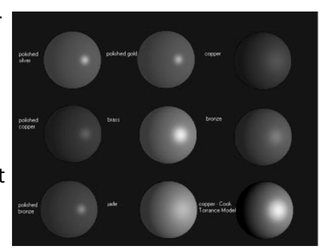
- Suppose a sphere has ambient and diffuse reflection coefficients (0.8, 0.2, 0.1), so it appears mostly red when bathed in white light.
- We illuminate it with a greenish light I_s = (0.15, 0.7, 0.15).
- The reflected light is then given by (0.12, 0.14, 0.015), which is a fairly even mix of red and green, and would appear yellowish.
  - 0.12 = 0.8 x 0.15, 0.14 = 0.2 x 0.7, 0.015 = 0.1 x 0.15

Example

- Because specular light is mirror-like, the color of the specular component is often the same as that of the light source.
  - E.g., the specular highlight seen on a glossy red apple when illuminated by a yellow light is yellow rather than red.
- To create specular highlights for a plastic-like surface, set the specular reflection coefficients $\rho_{sr} = \rho_{sg} = \rho_{sb} = \rho_s$, so that the reflection coefficients are 'gray' in nature and do not alter the color of the incident light.
- The designer might choose $\rho_s = 0.5$ for a slightly shiny plastic surface, or $\rho_s = 0.9$ for a highly shiny surface.

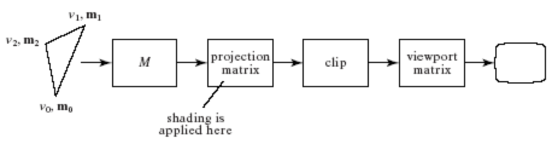
Combining Light Contributions and Adding Color (3)

- A list of ρ and f values for various materials for ambient, diffuse, and specular light is given in Fig. 8.17.
- Spheres of different materials (mostly metallic, but one jade at bottom center) are shown at right (Fig. 8.18).



Shading and the Graphics Pipeline

- Shading is applied to a vertex at the point in the pipeline where the projection matrix is applied. We specify a normal and a position for each vertex.



Shading and the Graphics Pipeline (2)

- glNormal3f (norm[i].x, norm[i].y, norm[i].z) specifies a normal for each vertex that follows it.
- The modelview matrix transforms both vertices and normals (**m**), the latter by $M^{-T}\mathbf{m}$. M^{-T} is the transpose of the inverse matrix M^{-1}.
- The positions of lights are also transformed.

Shading and the Graphics Pipeline (3)

- Then a color is applied to each vertex, the perspective transformation is applied, and clipping is done.
- Clipping may create new vertices which need to have colors attached, usually by linear interpolation of initial vertex colors.
- If the new point a is 40% of the way from v_0 to v_1, the color associated with a is a blend of 60% of (r_0, g_0, b_0) and 40% of (r_1, g_1, b_1):

  color at point a = (*lerp*(r_0, r_1, 0.4), *lerp*(g_0, g_1, 0.4), *lerp* (b_0, b_1, 0.4))

Shading and the Graphics Pipeline (4)

- The vertices are finally passed through the viewport transformation where they are mapped into screen coordinates (along with pseudodepth, which now varies between 0 and 1).
- The quadrilateral is then rendered (with hidden surface removal).