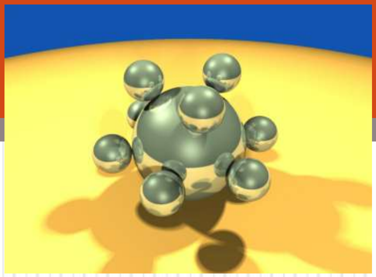


Introduction to Shading Models

Visual Realism Requirements

- Light Sources
- Materials (e.g., plastic, metal)
- Shading Models
- Textures
- Reflections
- Shadows

Rendering Objects

- we want to make the objects look visually interesting, realistic, or both.

- Develop methods of **rendering** for the objects of interest.

- Rendering: ***computes*** **how each pixel of a picture should look using different shading models.**

Rendering Objects (2)

- Much of rendering is based on different shading models,
  - describes 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.

Shading Models: Introduction

- A shading model dictates how light is scattered or reflected from a surface.
- Simple shading models focuses on achromatic light
  - It has brightness but no color
  - Only shade of gray
  - Described by single intensity value.
- Graphics uses two types of light sources
  - Ambient light - doesn't come directly from a source, but through windows or scattered by the air, comes equally from all directions.
  - Point-source light comes from a single point.

Shading Models: Introduction (2)

- When light hits an object,
  - some light is absorbed (and turns into heat),
  - some is reflected,
  - some may penetrate the interior (e.g., of a clear glass object).
- If all the light is absorbed, the object appears black and is called a blackbody.
- If all the light is transmitted, the object is visible only through reflection

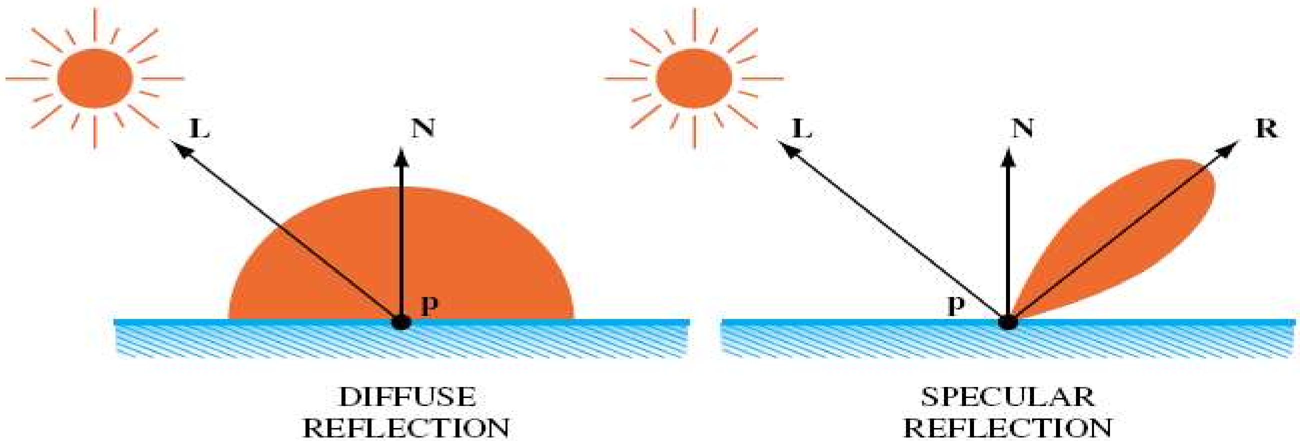
Shading Models: Introduction (3)

- When light is reflected from an object, some of the reflected light reaches our eyes, and we see the object.

- The amount of light that reaches the eye depends on the
  - Orientation of the surface
  - Light sources
  - Observer

- There are two types of reflection of incident light:
  - **Diffuse Scattering**
  - **Specular Reflections**

Shading Models: Introduction (4)



- **Diffuse Scattering**:
  - some of the incident light slightly penetrates the surface
  - re-radiated uniformly in all directions.
  - The light takes on some fraction of the color of the surface.
- **Specular reflection**:
  - more mirror-like and highly directional.
  - Incident light does not penetrate.
  - Light is reflected directly from the object's outer surface, giving rise to highlights of approximately the same color as the source.
  - The surface looks shiny.

Shading Models: Introduction (5)

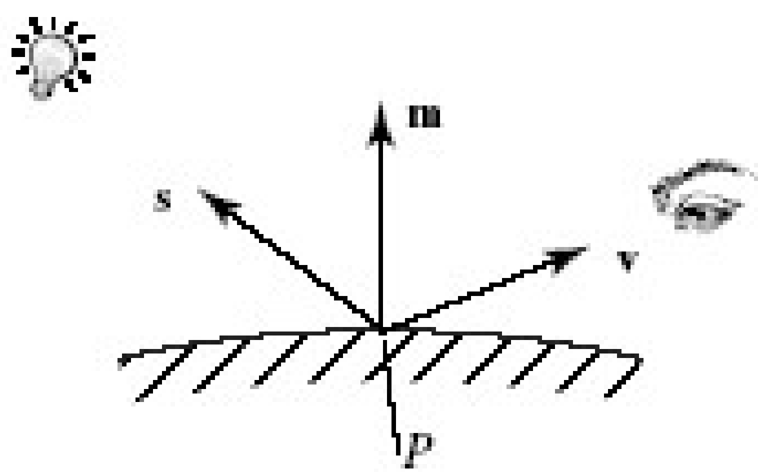
- In the simplest model, specular reflected light has the same color as the incident light. This tends to make the material look like plastic.
- In a more complex model, the color of the specular light varies over the highlight, providing a better approximation to the shininess of metal surfaces.
- Most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

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
  - vectors **s** from P to the source
  - **v** from P to the eye.
  - the angles between these three vectors form the basis for computing light intensities



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.

- This light arrives by multiple reflections from various objects in the surroundings and from light sources that populate the environment, such as light coming through a window, fluorescent lamps, etc.

- We assume a uniform background glow called **ambient light** exists in the environment.

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),
- 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.

Calculating Diffuse Light

- A fraction of incident light is reradiated in all directions

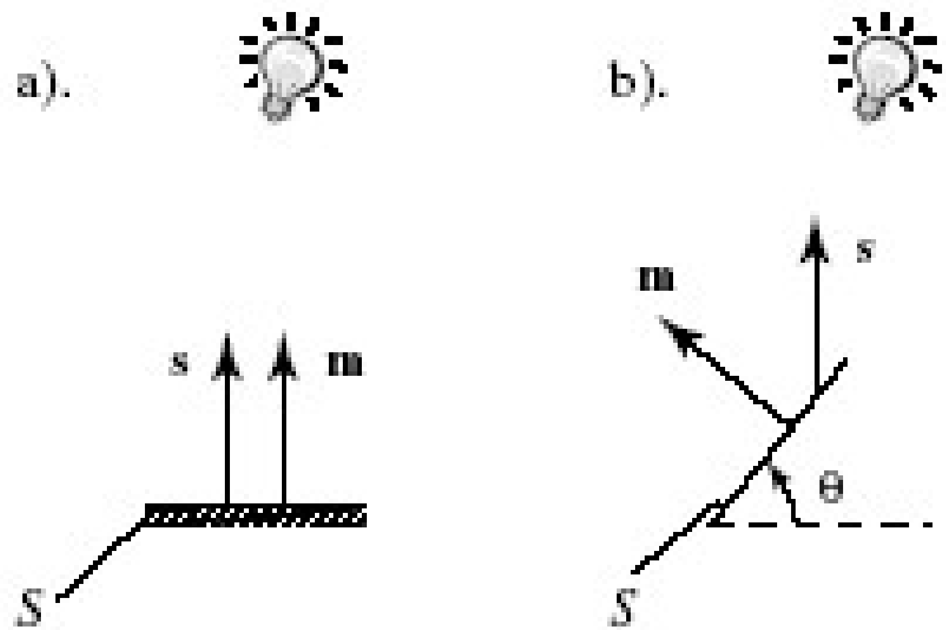
- Diffuse scattering is assumed to be independent of the direction from the point, *P*, to the location of the viewer's eye. (omindirectional scattering)

- Because the scattering is uniform in all directions, the orientation of the face *F* relative to the eye is not significant,
  - I_d is independent of the angle between $\mathbf{m}$ and $\mathbf{v}$ (unless $\mathbf{v} \cdot \mathbf{m} < 0$, making $I_d = 0$.)

- The amount of light that illuminates the face *does* depend on the orientation of the face relative to the point source:
  - the amount of light is proportional to the area of the face that it sees: the area *subtended* by a face.

Calculating Diffuse Light (2)

- The relationship between brightness and surface orientation is called as Lambert's law.
- Left :I_s (normal vector m is aligned with s)
- Right: $I_s \cos\theta$ (face is turned partially away from light source)



Calculating Diffuse Light (3)

- For θ near 0^o, brightness varies only slightly with angle, because the cosine changes slowly there.
- As θ approaches 90^o, the brightness falls rapidly to 0.

- We know $\cos\theta = (\mathbf{s}\cdot\mathbf{m})/(|\mathbf{s}|\,|\mathbf{m}|)$.

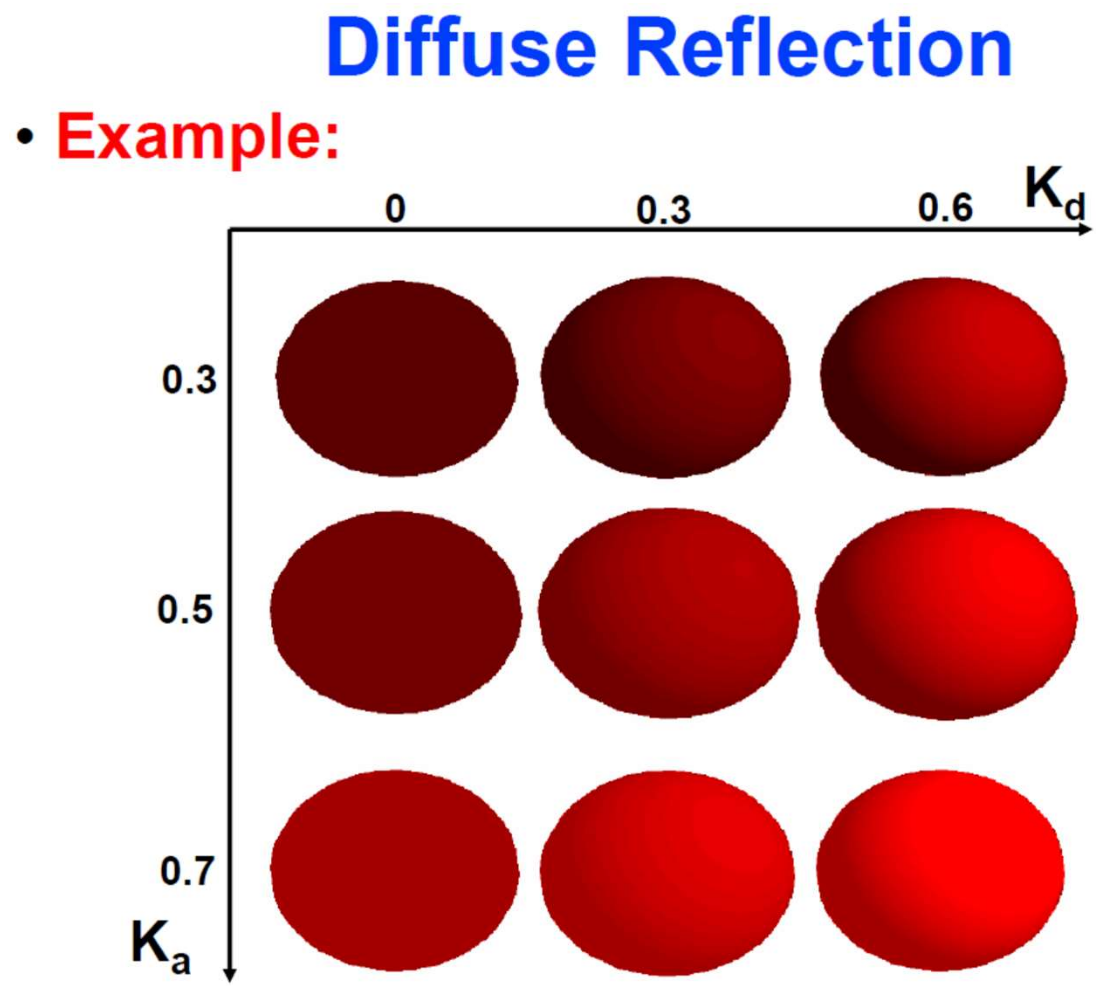
- $I_d = I_s\, \rho_d\, (\mathbf{s}\cdot\mathbf{m})\, /\, (|\mathbf{s}|\,|\mathbf{m}|)$
  - I_d – Intensity of the reradiated light that reaches eye.
  - I_s is the intensity of the source.
  - ρ_d is the diffuse reflection coefficient and depends on the material the object is made of.

Calculating Diffuse Light (4)

- If facet is aimed away from the eye this dot product is negative and we want I_d to evaluate to zero
- **If $\mathbf{s} \cdot \mathbf{m} < 0$** we want $I_d = 0$.
- So to take all cases into account, we use

$$\boxed{I_d = I_s \rho_d \max\left[(\mathbf{s} \cdot \mathbf{m})/(|\mathbf{s}|\,|\mathbf{m}|),\ 0\right]}$$

Example: Spheres Illuminated with Diffuse Light.

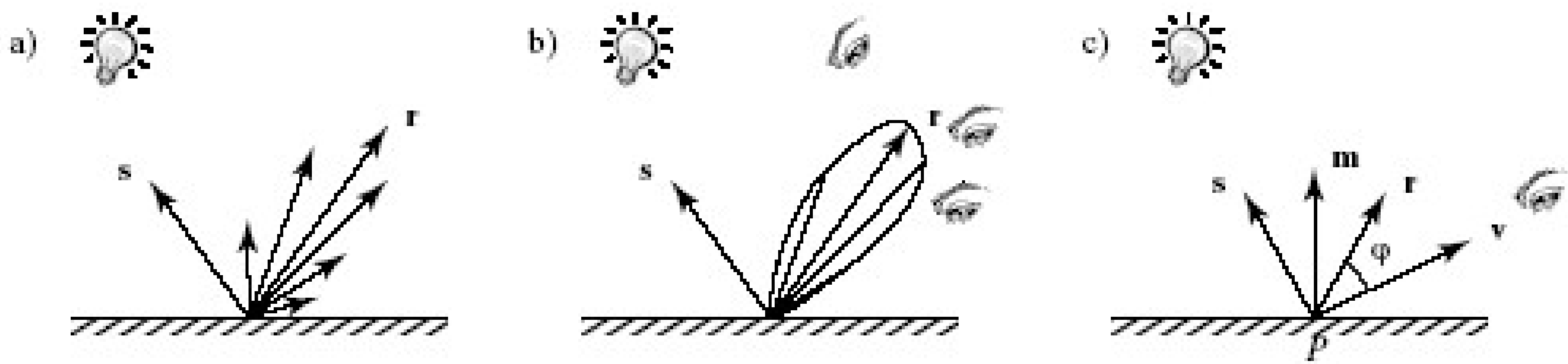


Calculating the Specular Component

- 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.
- 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,
  - In this model, the amount of light reflected is the greatest in the direction of perfect mirror reflection,r where the angle of incidence equals the angle of reflection.

Calculating the Specular Component (2)

- 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)

- The direction r of perfect reflection depends on both s and normal vector m
  - compute $\mathbf{r} = -\mathbf{s} + 2\,\mathbf{m}\,(\mathbf{s}\cdot\mathbf{m})/(|\mathbf{m}|^2)$ (mirror reflection direction).
- For surfaces that are not mirrors, the amount of reflected light decreases as the angle $\boldsymbol{\varphi}$ between $\mathbf{r}$ and $\mathbf{v}$ increases.
- For a simplified model, we say the intensity decreases as $\cos^f \varphi$,
  - where f (amount of falloff) is chosen experimentally between 1 and 200.

Calculating the Specular Component (3)

- $\cos \varphi = \mathbf{r} \cdot \mathbf{v} / (|\mathbf{r}|\,|\mathbf{v}|)$

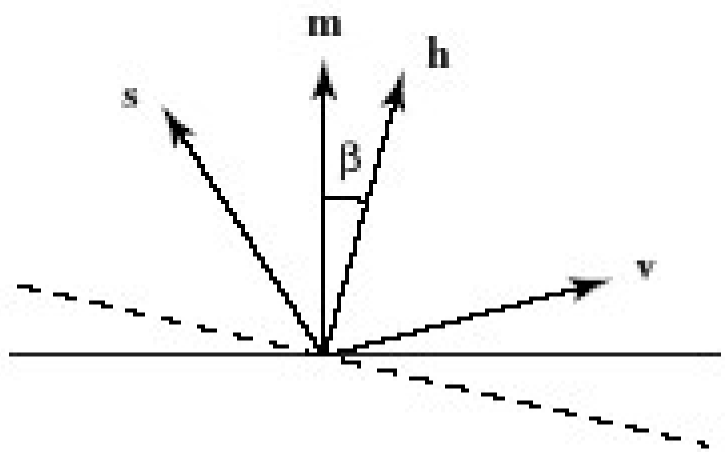
- $I_{sp} = I_s\, \rho_s\, (\mathbf{r} \cdot \mathbf{v} / (|\mathbf{r}|\,|\mathbf{v}|))^f$.
  - ρ_s is the specular reflection coefficient, which depends on the material.

- If $\mathbf{r} \cdot \mathbf{v} < 0$, there is no reflected specular light, the set $I_{sp} = 0$

Specular Component, $I_{sp} = I_s\, \rho_s \max[(\mathbf{r} \cdot \mathbf{v} / (|\mathbf{r}|\,|\mathbf{v}|))^f, 0]$

Speeding up Calculations for Specular Light

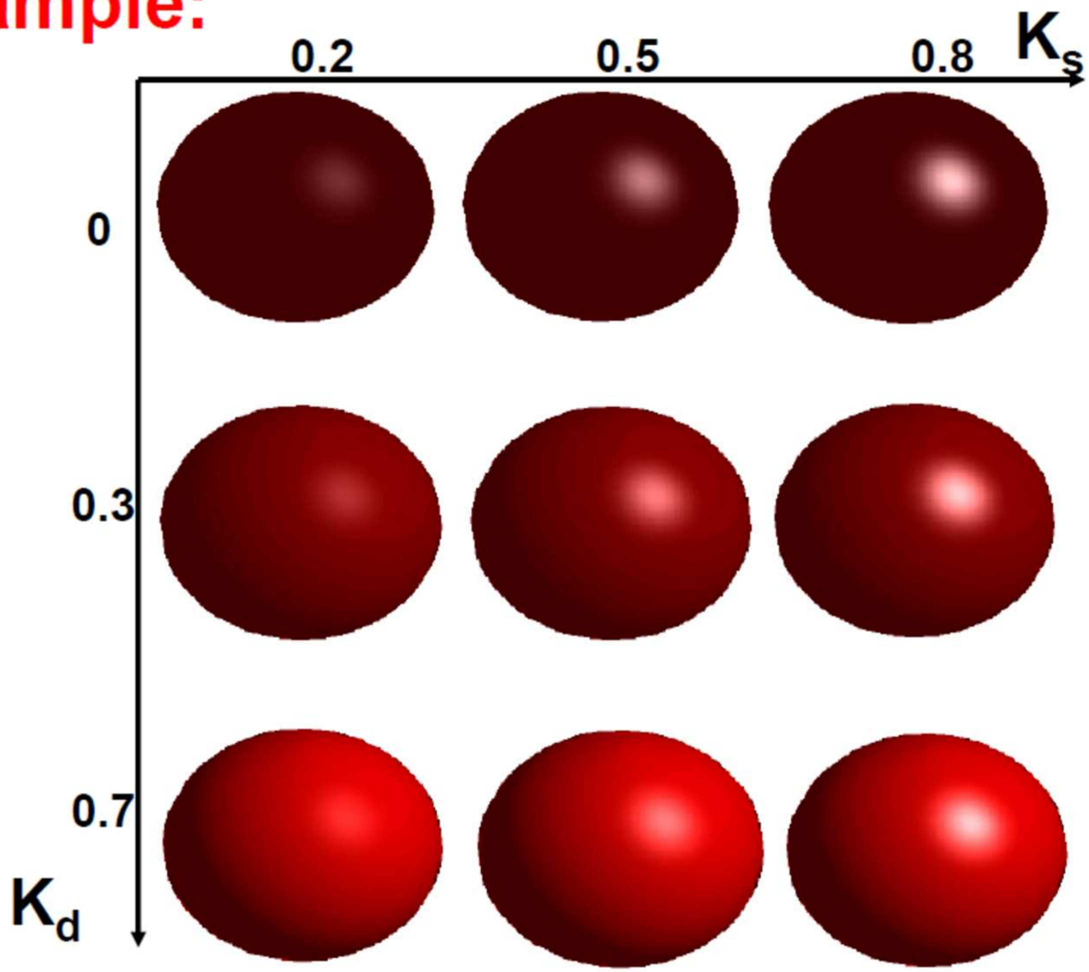
- Find the halfway vector $\mathbf{h} = \mathbf{s} + \mathbf{v}$.
- Then the angle β between **h** and **m** approximately measures the falloff intensity.
- To take care of errors, we use a different f value, and write



$$I_{sp} = I_s \rho_s \max[(\mathbf{h}\cdot\mathbf{m}/(|\mathbf{h}|\,|\mathbf{m}|))^f, 0]$$

Specular Reflection

- **Example:**



Combining Light Contributions and Adding Color

- I=amibent+diffuse+specular

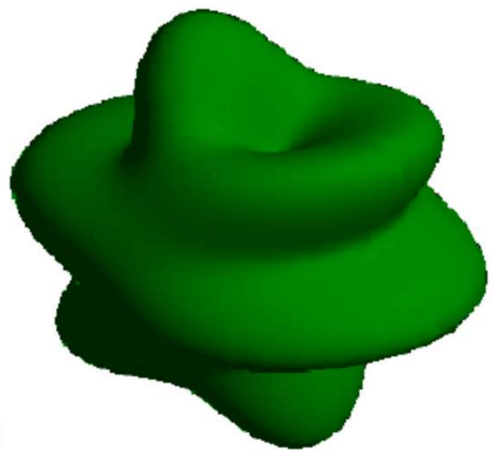
- $\mathbf{I = I_a \rho_a + I_s \rho_d \text{ lambert} + I_s \rho_s \text{ x phong}^f}$
  - **Lambert = max[(s·m)/(|s||m|), 0]**
  - **Phong = max[(h·m/(|h||m|), 0]**

- To add color, we use 3 separate total intensities 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:
- $\mathbf{I_r = I_{ar} \rho_{ar} + I_{sr} \rho_{dr} \text{ lambert} + I_{sr} \rho_{sr} \text{ x phong}^f}$ **(similarly for** $\mathbf{I_{g,} I_{d)}}$
  - ambient = (I_{ar}, I_{ag}, I_{ab})
  - diffuse = (I_{dr}, I_{dg}, I_{db})
  - specular = $(I_{spr}, I_{spg}, I_{spb})$.

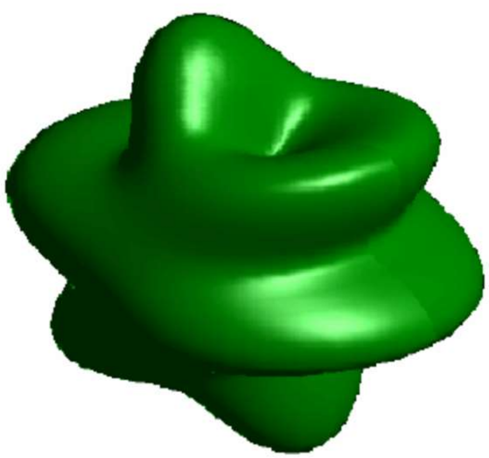
Light model : Simple to Complex

- **Example:**

Ambient Illumination



Ambient + Diffuse



Ambient + Diffuse + Specular

Ambient

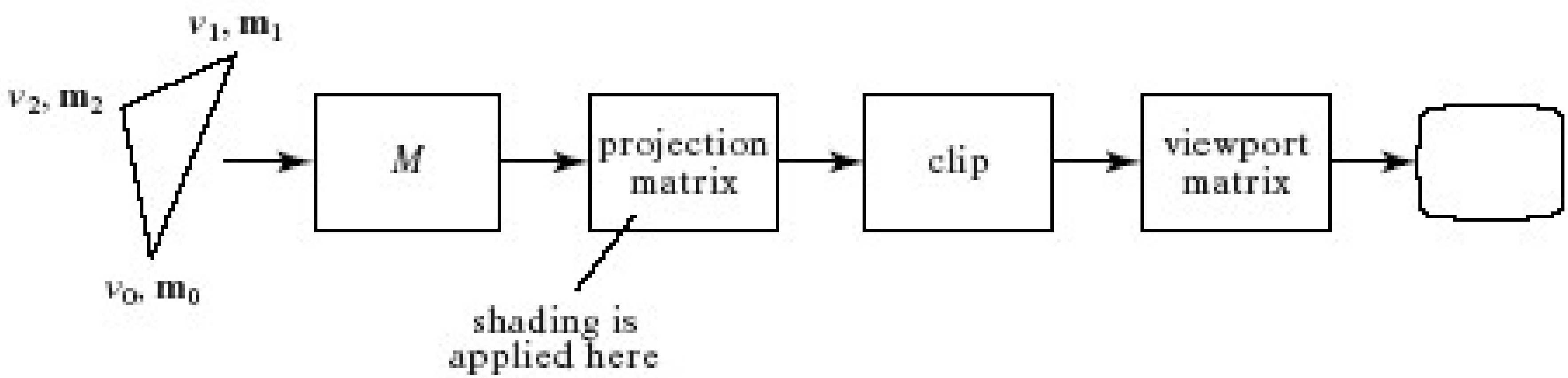
Ambient + Diffuse

Ambient + Diffuse + Specular

Shading and Graphics Pipeline

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 M transforms both vertices and normals (**m**), the latter by $M^{-T}\mathbf{m}$.
- M^{-T} is the transpose of the inverse matrix M.
- The positions of lights are also transformed.
- OpenGL allows to specify various light sources and their locations.

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.
- Suppose color at v_0 (r_0, g_0, b_0) and v_1 (r_1, g_1, b_1):
- 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)

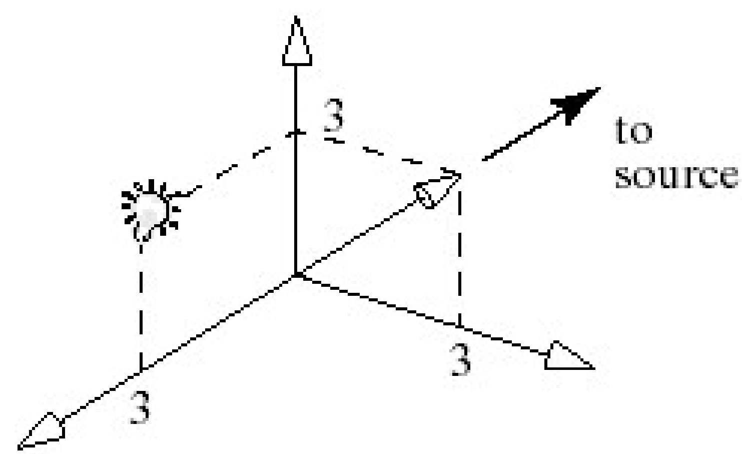
- The vertices are finally passed through the viewport transformation where they are mapped into screen coordinates (along with pseudodepth, which now varies between 0 and 1).
- The quadrilateral is then rendered (with hidden surface removal).

Creating and Using Light Sources in Open-GL

- OpenGL allows to define up to eight sources
- Light sources are through number in [0, 7]: GL_LIGHT_0, GL_LIGHT_1, etc.
  - Each light has a position specified in homogeneous coordinates using a GLfloat array named litepos, for example,
    - GLfloat litePos[]={3.0,6.0,5.0,1.0}
- The light is created using
  - glLightfv (GL_LIGHT_0, GL_POSITION, litePos);
- If the position is a vector (4th component = 0), the source is infinitely remote (like the sun).

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 use 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 OpenGL(2

- Arrays are defined to hold the colors emitted by light sources and are passed to glLightfv
- The light color is specified by a 4-component array [R, G, B, A] of GLfloat, named (e.g.) amb0.
- The A (alpha: used to blend two colors on the screen) value can be set to 1.0 for now.
  - Glfloat amb0[]={0.2,0.4,0.6,1.0}; similar for diff0[],spec0[];
- 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

Default values:

- For all sources: default ambient = (0,0,0,1) – dimmest possible: black.
- For light source LIGHT0:
- default diffuse = (1,1,1,1) – brightest possible : white.
- Default specular = (1,1,1,1) – brightest possible: white.
- For all other light sources, diffuse and specular values have default black

Creating and Using Light Sources in OpenGL (4)

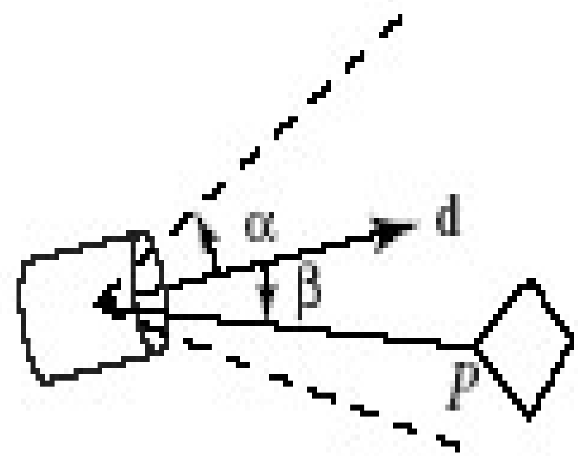
- Lights do not work unless you turn them on.
  - In your main program, add the statements
    - **glEnable (GL_LIGHTING);**
    - **glEnable (GL_LIGHT_0);**
  - If you are using other lights, you will need to enable them also.
- To turn off a light,
    - **glDisable (GL_LIGHT_0);**
- To turn them all 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 Spotlights in Open-GL

- Light sources are point sources emit light uniformly in all directions.
- OpenGL allows to make into spotlights
- A spotlight emits light only in a restricted set of directions;
- The spotlight is aimed in **direction d**, with a cut-off angle alpha.



- There is no light outside the cone. Inside the cone,

$$I = I_s(\cos \beta)^{\varepsilon}, \text{ where}$$

β is the angle between **d** and a line from the source to *P* and

ε is chosen by the user to give the desired falloff of light with angle.

$(I_s(\cos \beta)^{\varepsilon})$ - attenuation of light when reaching P.

Creating and Using Spotlights in OpenGL (2)

- To create the spotlight, create a GLfloat array for **d**.
- Default values are **d** = {0, 0,-1}, $\alpha = 180^o$, $\varepsilon = 0$: a point source.
- Spotlight parameters can be set by adding the statements
  - glLightf (GL_LIGHT_0, GL_SPOT_CUTOFF, 45.0); (45.0 is α in degrees)
  - glLightf (GL_LIGHT_0, GL_SPOT_EXPONENT, 4.0); (4.0 is ε)
  - GLfloat d[ ]={2.0,1.0,-4.0};
  - glLightfv (GL_LIGHT_0, GL_SPOT_DIRECTION, d);
  - glLightf->to set single value, glLightfv->to set a vector

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.
- The expression helps to model constant, linear and quadratic (inverse square law) dependence on distance from a source.

Attenuation of Light with Distance (2)

- These parameters are controlled by function calls:
- glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, 2.0);
- and similarly for GL_LINEAR_ATTENUATION, and GL_QUADRATIC_ATTENUATION.
- The default values are $k_c = 1$, $k_l = 0$, and $k_q = 0$ (no attenuation). Which eliminate any attenuation

Changing the OpenGL Light Model

- **3 parameters** to be set that specify general rules for applying the lighting model.
- **The color of global ambient light**:
  - We can establish a gobal ambient light source in a scene that is independent of any sorce :

    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" $\mathbf{h} = \mathbf{s} + \mathbf{v}$.
  - The true directions **s** and **v** are normally different at each vertex in a mesh.
  - If light source is directional s is constant. But v still varies from vertex to vertex
  - To use the true value of **v** for each vertex, execute

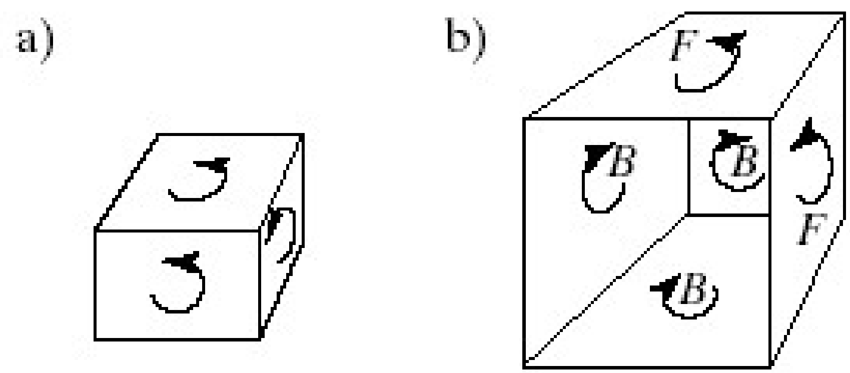
glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, GL_TRUE);

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);



Lightning :Light Source

- Control light position and direction
  - OpenGL treats the position and direction of light source just as it treats the position of geometric primitives
  - MODELVIEW transformation is applied.
- Three types of control
  - A light position that remains fixed
  - A light that moves around the stationary object
  - A light that moves along the view point

Controlling the Light's Position (Light stationary:)

- **glModelMatrixMode(GL_MODELVIEW)**
- **glLoadIdentity();**

- modeling and viewing here
  - GLfloat position[]={3.0,6.0,5.0,1.0}
  - glLightfv(GL_LIGHT0, GL_POSITION, position)

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 it
  - move the light source and set its position
  - pop the matrix
  - set up the camera, and draw the objects.

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

gluLookAt(...); // set the camera position
<.. draw the object ..>
glutSwapBuffers(); }
```

Controlling the Light's Position
Move light with viewpoint:

key: specify light position in eye coordinates before viewing transf.

```
GLfloat position[] = {0, 0, 0, 1}
 glModelMatrixMode(GL_MODELVIEW)
glLoadIdentity();
glLightfv(GL_LIGHT0, GL_POSITION, position)
gluLookAt( …)
    draw object()
```