Illumination Models and Surface-Rendering Methods

CEng 477
Introduction to Computer Graphics
Fall 2007-2008

Illumination Models and Surface Rendering Methods

- In order to achieve realism in computer generated images, we need to apply natural lighting effects to the surfaces of objects.
- Illumination models are used to calculate the amount of light reflected from a certain position on a surface
  - In other words, they provide ways to compute the color on a pixel of a surface.
- Using the illumination model for computing the color of every pixel is costly. Therefore surface rendering methods are used to interpolate illumination values computed for certain special pixels (like vertices of a triangle).

What determines the color of a position at a surface?

- Object's material properties
  - Is it a shiny object?
  - What make wood look different than a metalic object?
- Object's position with respect to the light sources
  - And also with respect to other objects (for example shadows)
- The features of light sources
  - Color of light, strength of light, etc.
- The orientation of the viewing plane

Illumination Models

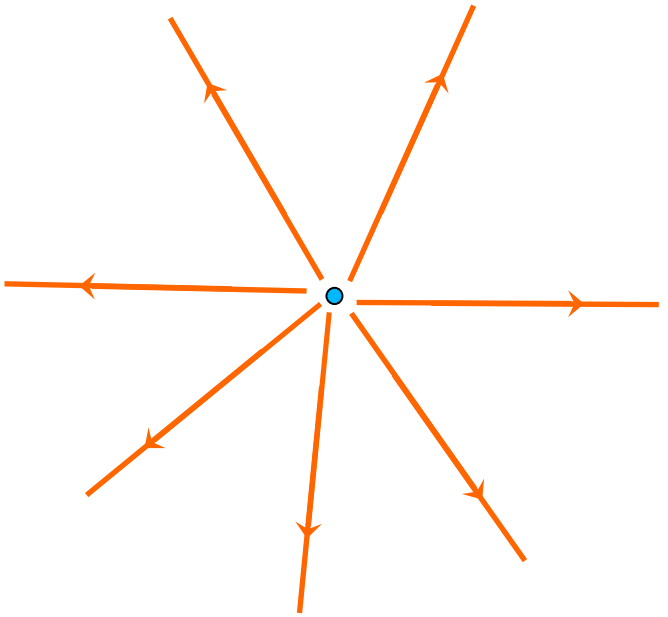
- For computational efficiency computer graphics applications usually employ approximation models of the real physical illumination process.
- To get more realistic pictures, models that are closer to the actual physical lighting model should be developed. But they are costly and rendering a single frame may take a lot of time.
  - Examples: ray tracing, radiosity

Light Sources

- A light source can be defined with the following parameters:
  - Position
  - Color of the emitted light
  - The emission direction
  - The shape of the light source

Point Light Sources

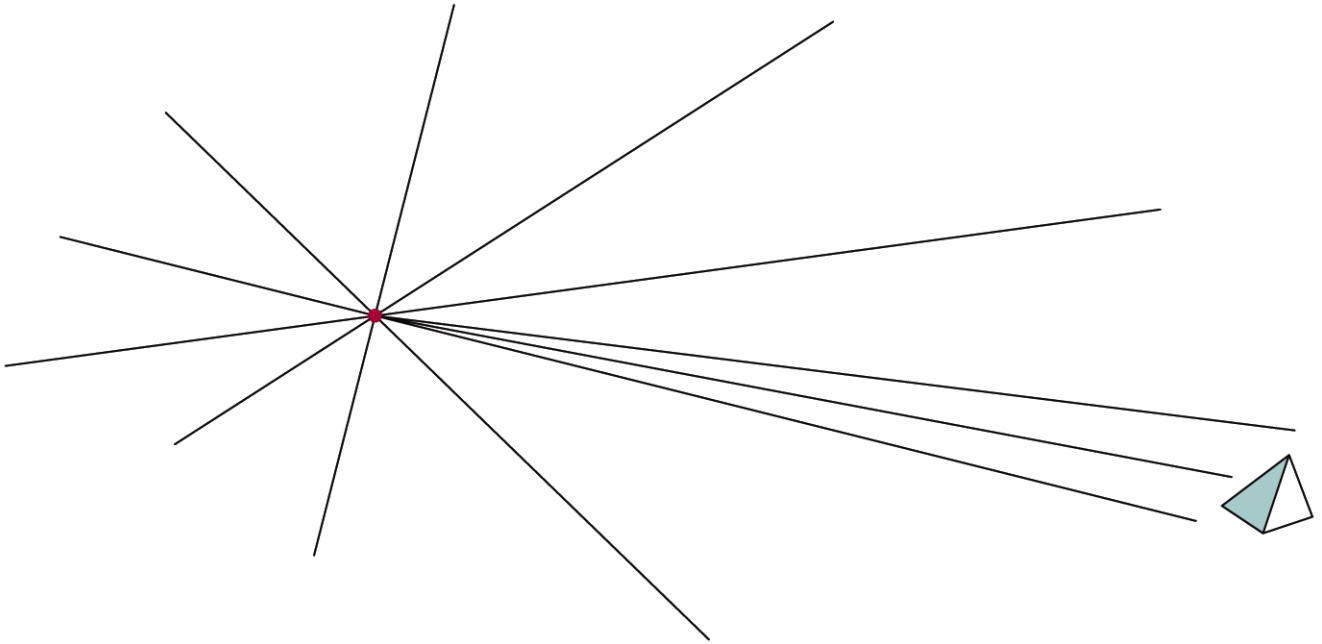
- Described by a position and a light color
- Light rays are emitted from the point light source in all directions



- The position information is used to identify the objects illuminated by the point source.

Infinitely Distant Light Sources

- They can be simulated by a single emission direction vector. This direction is used to calculate the amount of light received at a position.



Light Intensity Attenuation

- As the light emitted from a light source travels through space its intensity decreases. This is called attenuation.
  - Example: a traffic light will not be able to illuminate an object 2km away.
- The decrease in the light source is inversely proportional to the squared distance from the object.
  - It decreases by $1/d_l^2$

Problem with $1/d_l^2$

- Too much intensity variation for objects that are close to the light source
- Too little intensity variation for object that are far away
- Solution: Use the following function instead

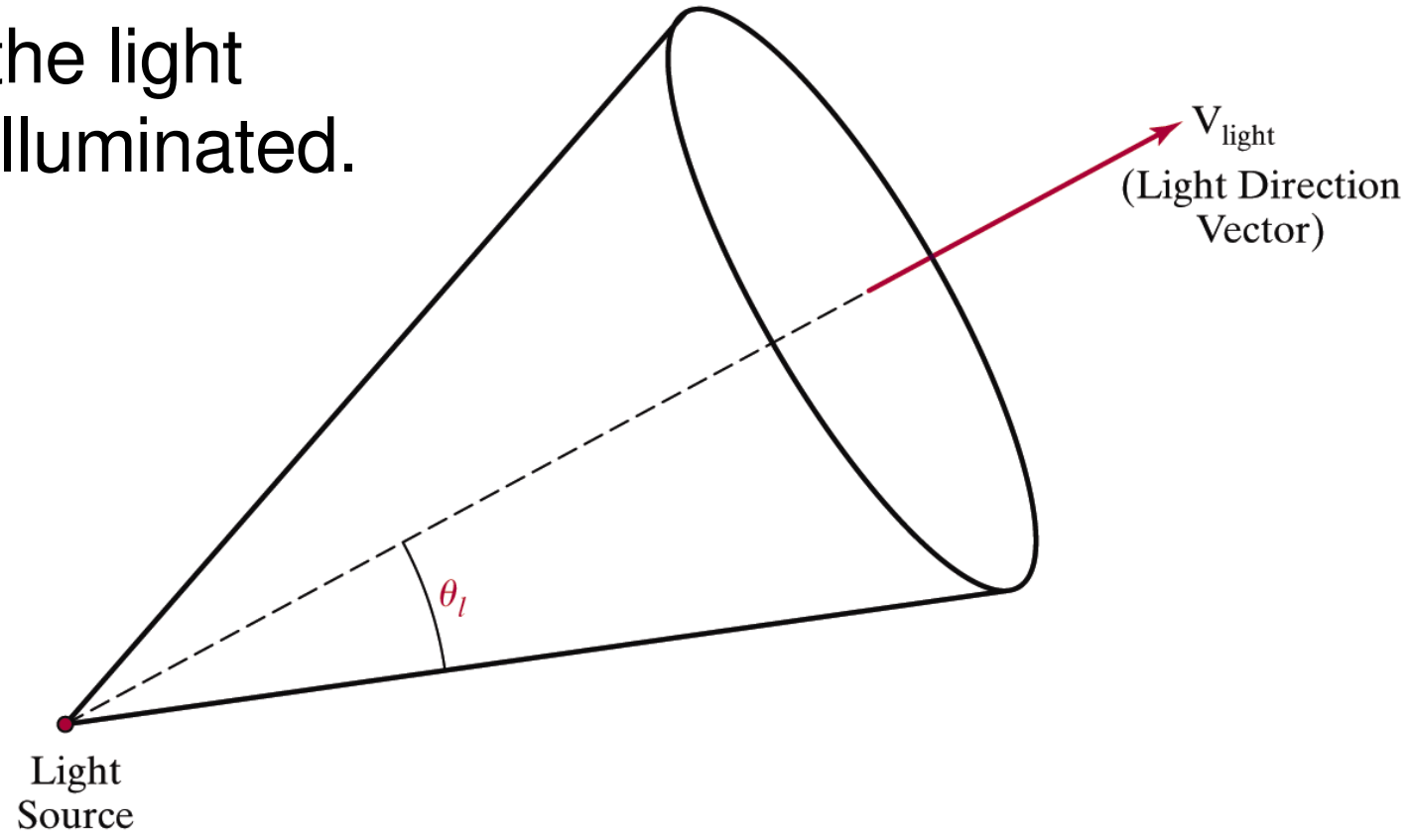
$$f_{atten}(d_l) = \frac{1}{a_0 + a_1 d_l + a_2 d_l^2}$$

- The constants can be adjusted to produce better attenuation effects.
- For a light source that is at infinity, we do not need to use attenuation. We can assume all the objects are equidistant from the light source and receive same amount of light.

Directional Light Sources and Spotlight Effects

- A directional light source can be defined by a light direction (V_{light}) and an angular limit (θ_l)

Objects outside the light cone will not be illuminated.

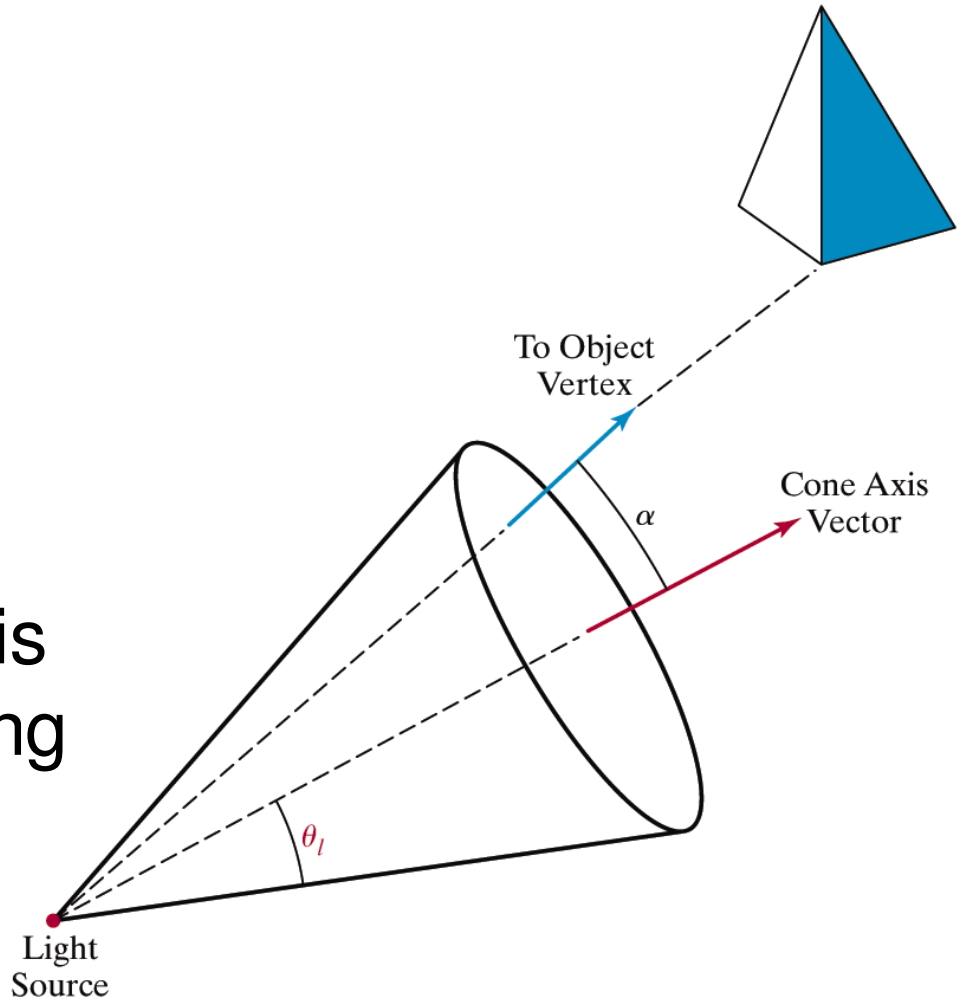


Determination of objects inside the light cone

- The cosine of the angle between the light direction vector and object direction is given by:

$$\cos\alpha = V_{\mathbf{obj}} \cdot V_{\mathbf{light}}$$

- If $\cos\alpha \geq \cos\theta_l$ then the object is within the light cone (assuming θ_l is between 0 and 90 degrees)



We can also apply attenuation based on the angular distance α

Angular Attenuation

- Given the two unit vectors $\mathbf{V}_{\text{light}}$ and $\mathbf{V}_{\text{obj}}$, we can define angular attenuation as

$$f_{l.angatten} = \begin{cases} 1.0, & \text{if source is not a spotlight} \\ 0.0, & \text{if } V_{\mathbf{obj}} \cdot V_{\mathbf{light}} = \cos\alpha < \cos\theta_l \\ & (\text{object is outside the spotlight cone}) \\ (V_{\mathbf{obj}} \cdot V_{\mathbf{light}})^{a_l}, & \text{otherwise} \end{cases}$$

Surface Lighting Effects

- The light reflected from the surface of an object depends on several factors
  - Degree of reflectivity of the object (shiny vs. dull surfaces)
  - Surface texture properties
  - Color reflectance coefficients (e.g. color of the object)

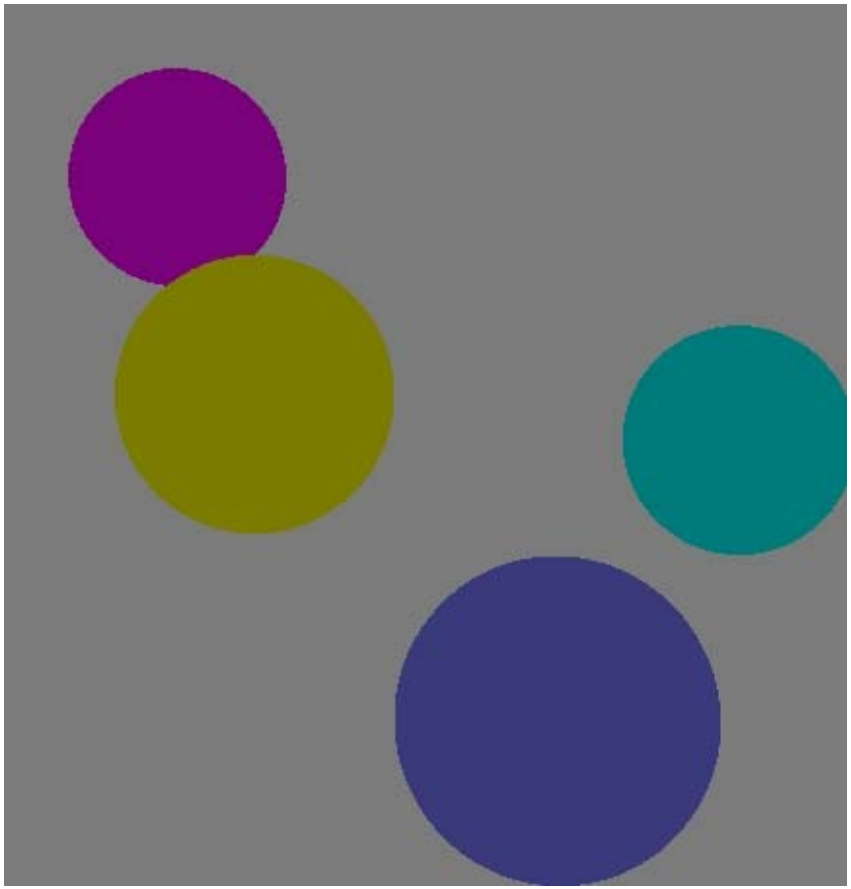
Basic Illumination Model

- The basic illumination model we will learn in this course includes three components
  - Ambient Light
  - Diffuse Reflection
  - Specular Reflection

Ambient Light

- Reflected light from the environment and the nearby objects cause other objects to illuminate

- Also called background light
- Constant color reflected from all points on the surface.
- Tells us how bright the surface will look like when no light source can directly reach the surface
- When used alone, does not produce very interesting pictures



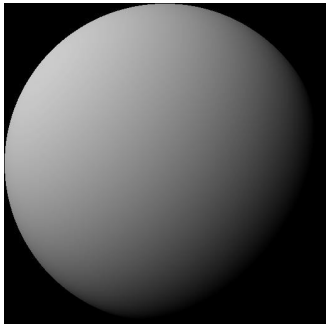
Ambient lighting only

Modeling Ambient Light

- We can use an ambient intensity parameter I_a that describes the level of ambient light in a scene.
- Every object in the scene will be illuminated by this amount independent of the surface orientation and viewer location.
- But different surfaces may reflect different amount of ambient light based on their absorbance/reflectance properties. We can model this by a constant factor for each surface:

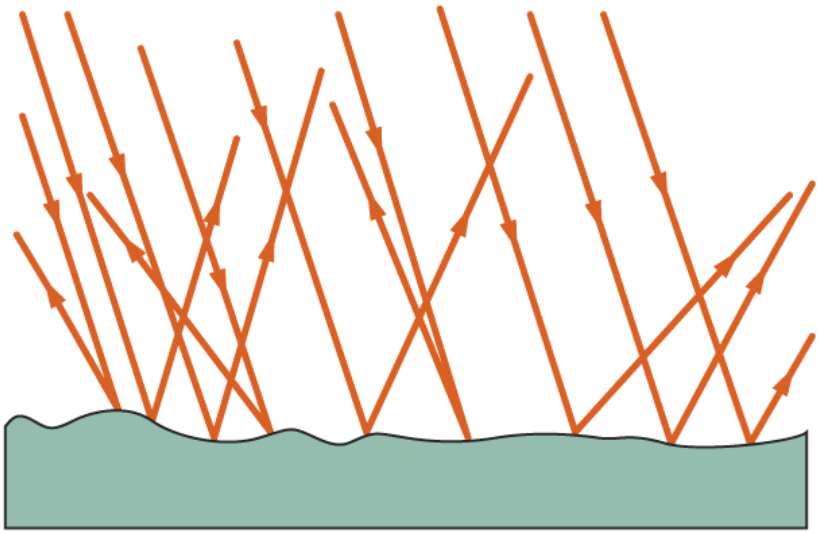
$$k_a I_a$$

Diffuse reflection



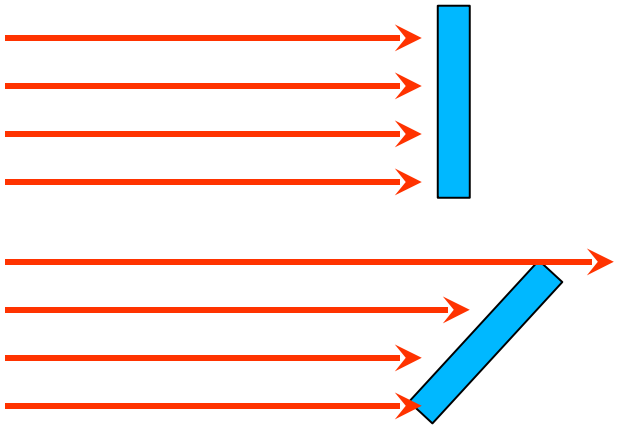
- The light that is reflected in all directions is called diffuse reflection.
- The reflected light is independent of the viewing position (equally bright from all viewing directions)
- But the light position with respect to the surface orientation is important to determine the light reflected from the surface.

- When an object is illuminated with white light the original color of the object is what we see as the diffuse reflection.
- If a blue object is illuminated with red light, it will appear black.



Modeling Diffuse Reflection

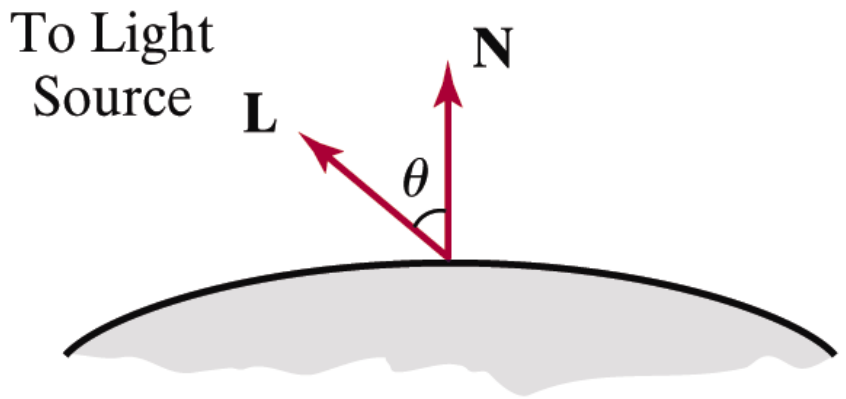
- Orientation of the surface determines the amount of light incident on the surface



Given the angle θ between the surface normal and the incident light direction, we can write the diffuse reflection equation as:

$$I_{l,\,diff} = k_d I_{l,\,incident}$$
$$= k_d I_l \cos\theta$$

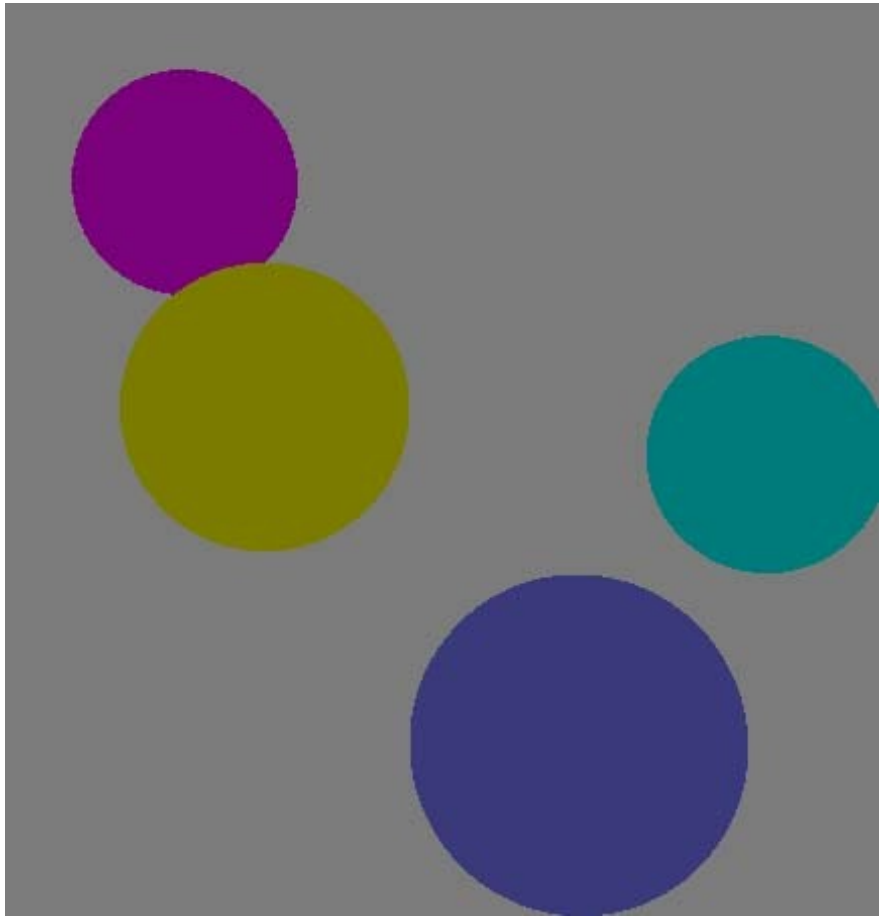
Combining Ambient and Diffuse



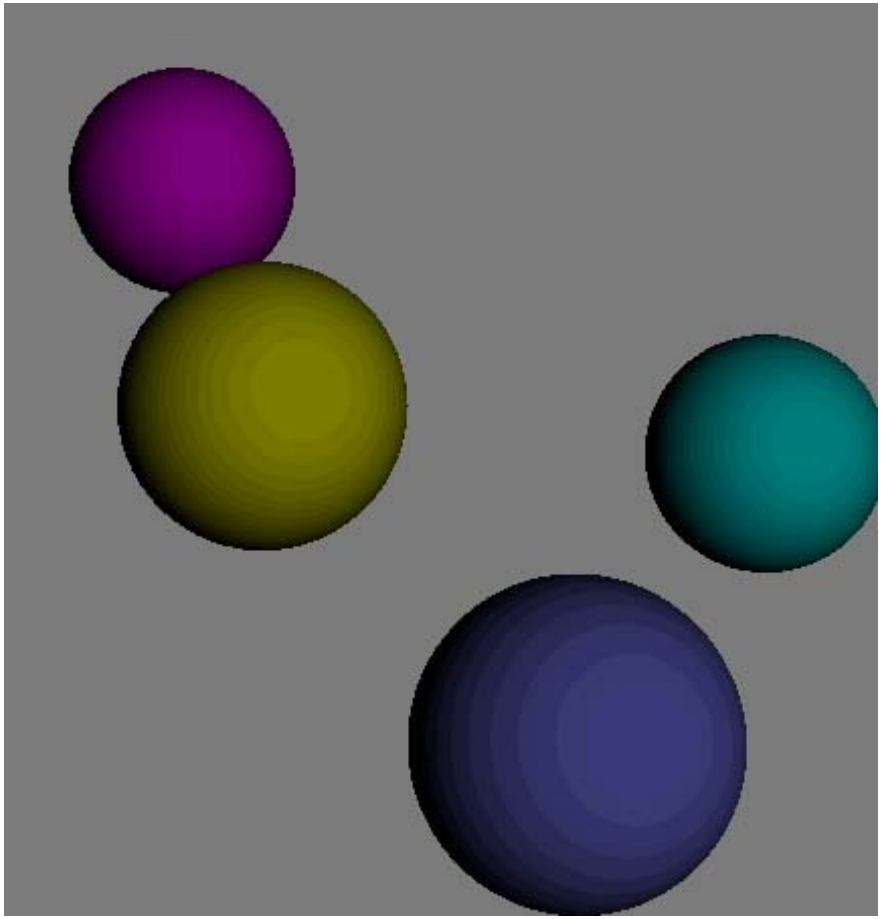
$$I_{l,\,diff} = \begin{cases} k_a I_a + k_d I_l (N \cdot L), & \text{if } N \cdot L > 0 \\ k_a I_a, & \text{if } N \cdot L \leq 0 \end{cases}$$

$$L = \frac{P_{source} - P_{surface}}{|P_{source} - P_{surface}|}$$

Ambient Light vs. Diffuse Reflection

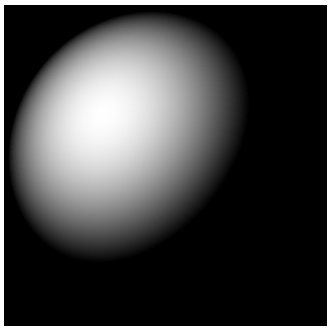


Ambient lighting only



With diffuse lighting

Specular reflection



- Light reflected from a certain spot on the object is concentrated and appears as a lot brighter compared to other spots. This is due to specular reflection and is an important property of shiny objects.

- Specular reflection is both dependent on the light direction, surface orientation, and ***viewer position***.

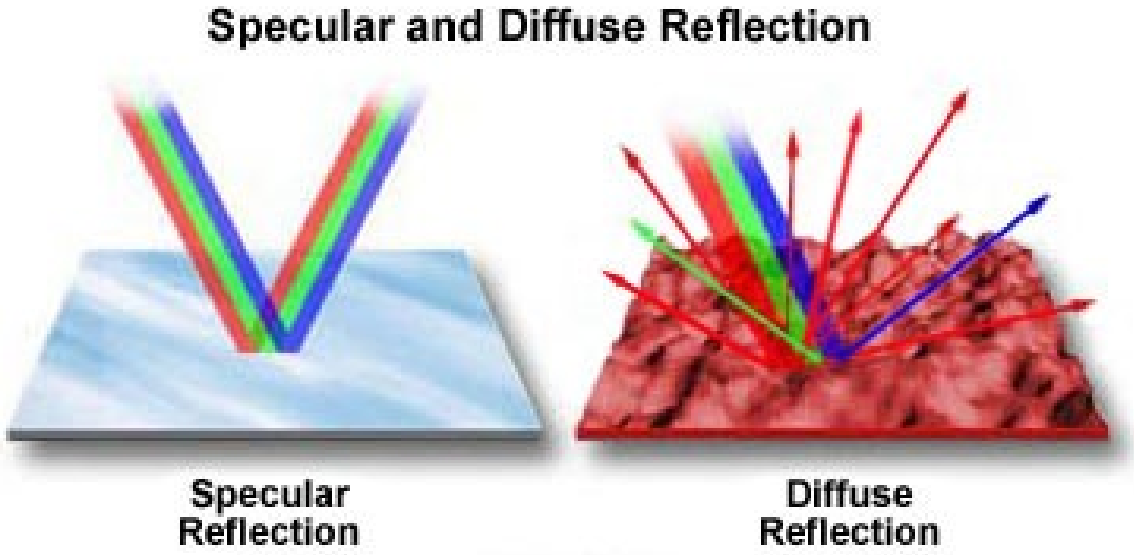
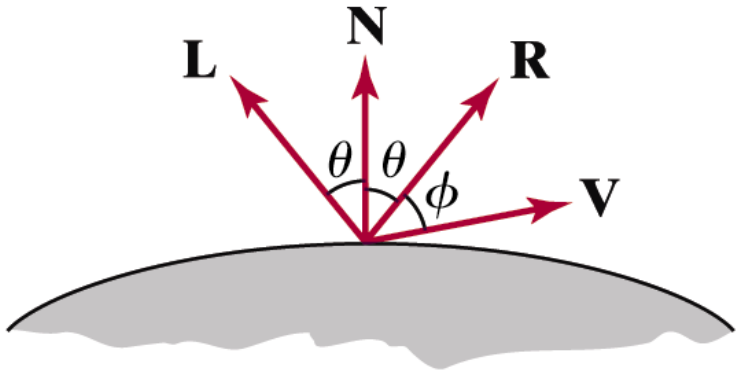
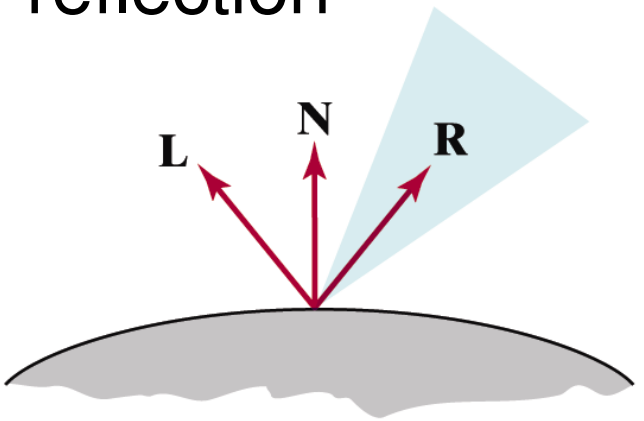


Figure 2

Modeling Specular Reflection

- Let us try to picture specular reflection



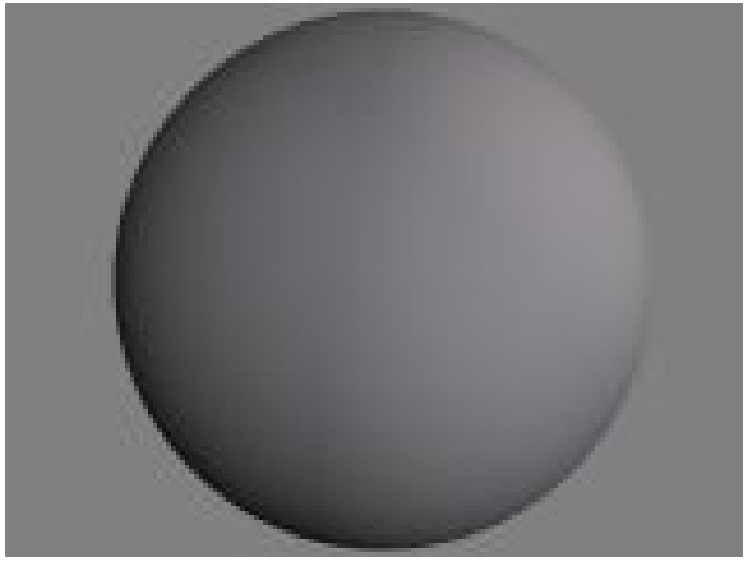


- The specular reflection angle equals angle of incidence
- The specular reflection is visible only at directions close to **R**. Shiny surfaces like mirrors have a narrow specular reflection range (given as parameter n_s, specular-reflection exponent.)

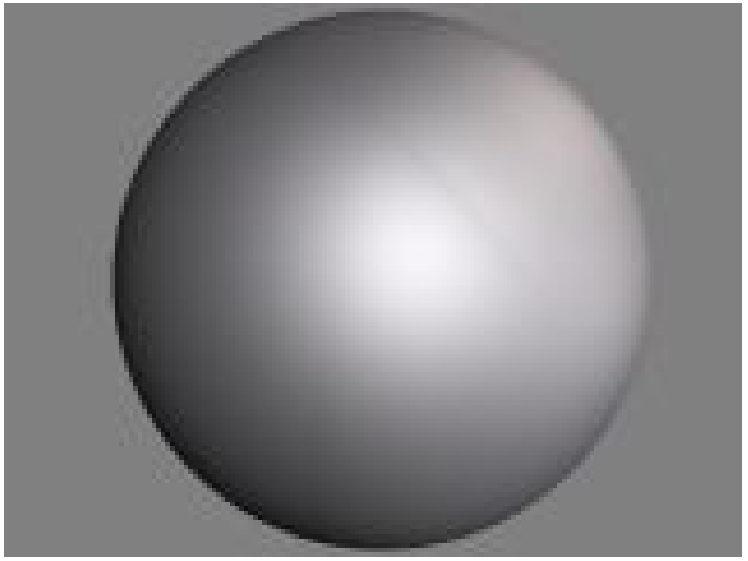
Modeling Specular Reflection

$$I_{l,\,spec} = \begin{cases} k_s\, I_l (V \cdot R)^{n_s}, & \text{if} \quad V \cdot R > 0 \quad \text{and} \quad N \cdot L > 0 \\ 0.0, & \text{if} \quad V \cdot R < 0 \quad \text{and} \quad N \cdot L \leq 0 \end{cases}$$

Diffuse vs. Specular Reflection



Diffuse Reflection



Diffuse and Specular Reflection

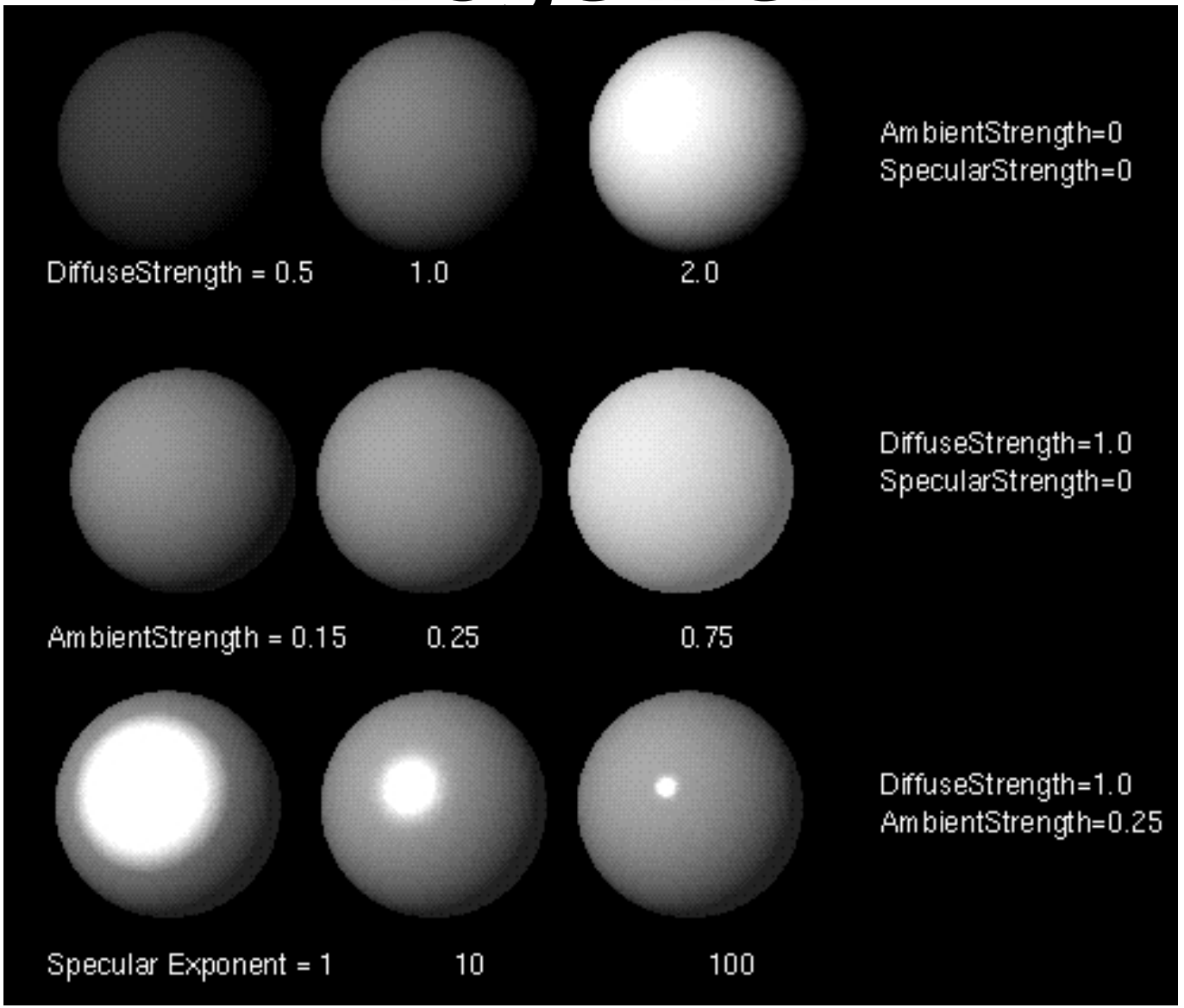
Combining all together

- Considering objects that can emit light and multiple light sources in the scene

$$I = I_{surfemission} + I_{ambdiff} + \sum_{l=1}^{n} f_{l,radaten} f_{l,angaten} \left(I_{l,diff} + I_{l,spec} \right)$$

- Also, note that, if you consider different colors of light and surfaces, you should use similar equations (possibly with different parameters) and apply it to each color component (R,G,B).

Ambient, Diffuse, and Specular Reflections Together

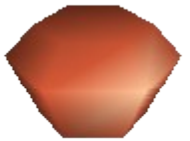


Summary

- The light reflected off the surface of an object is composed ambient, diffuse and specular components.
- Ambient component is independent of the surface orientation and viewer position
- Diffuse component is independent of the viewer position but depends on the relative orientation of the light source and the surface
- In specular reflection the position of the viewer is also important

Polygon Rendering Methods

- Surface representations:
  Usually polygon mesh approximations.
  - Simpler calculations
  - HW support
- Curved surfaces versus polyhedra:
  - If smooth surface is desired use interpolation



  - If you want to display polyhedra use constant-intensity surface rendering

Polygon Rendering Methods

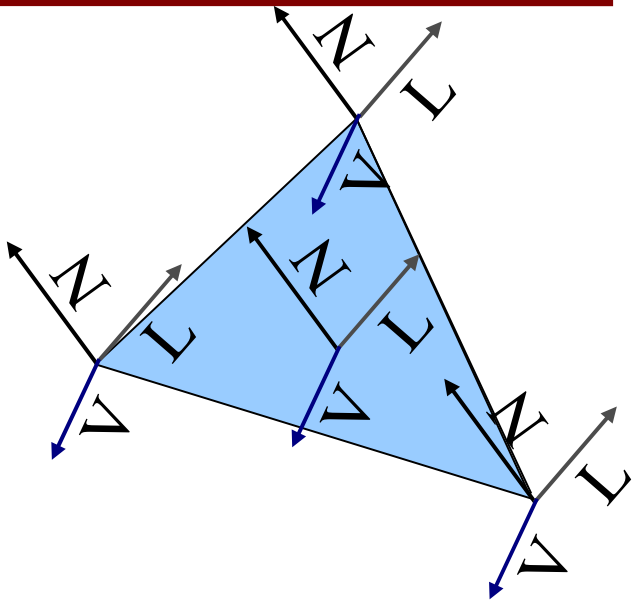
- Constant intensity rendering:
  - No interpolation.
  - Also called “flat shading”

- Intensity interpolation rendering:
  - “Gouraud shading”
  - Based on intensity interpolation of vertices.

- Normal vector interpolation rendering:
  - “Phong shading”
  - Based on interpolation of normal vectors of vertices.

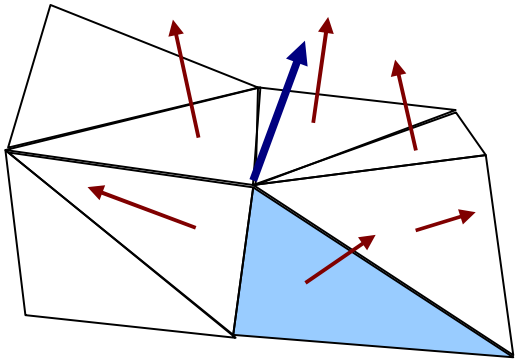
Constant Intensity Rendering

- Assumes:
  - Light source is at infinity
  - Viewer is at infinity
- So that: All illumination parameters are constant through the polygon surface.



- Cheapest method:
  Calculate the surface normal and other illumination parameters for any chosen point on the surface and fill the polygon with that intensity.
- Usefull especially in fast previews.

Intensity interpolation (Gouraud Shading)

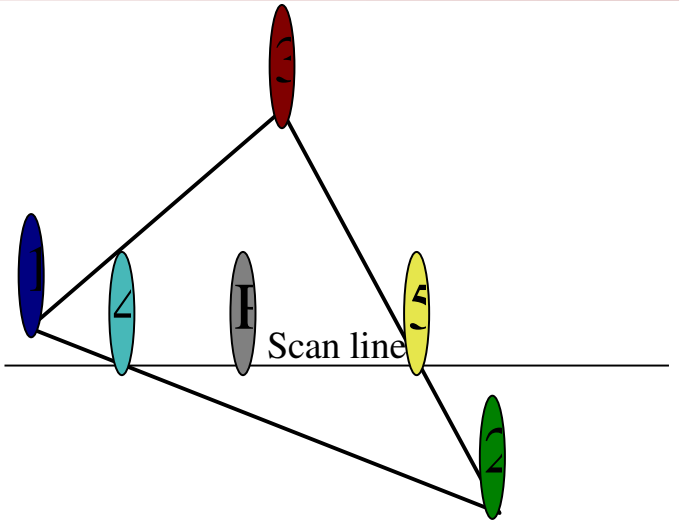


- Approximation of smooth surface by polygon rendered with interpolated intensities. For each vertex of polygon:
  - Unit Normal at the vertex is taken as average of all adjacent surface normals.
  - Calculate the intensity at the vertex by using an illumination model.
  - Interpolate the intensity values at vertices to calculate the interior pixel intensities of the polygon.

$$N_{Vert} = \frac{\sum_{k=1}^{n} N_k}{\left|\sum_{k=1}^{n} N_k\right|}$$

Intensity interpolation (Gouraud Shading)

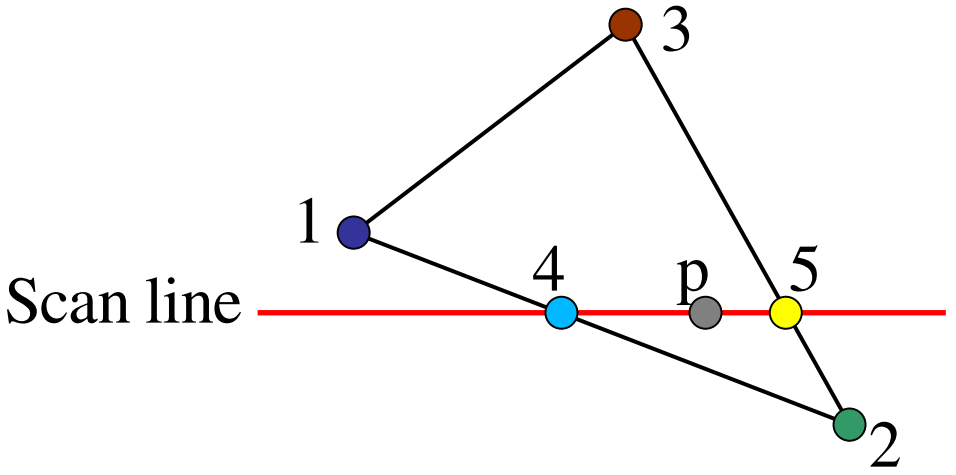
- Polygon rendering can be combined with scan conversion.
- Two-stage interpolation:
  - First, interpolate border pixels,
  - Then interpolate interior pixels from borders pixels.



$$I_4 = \frac{y_4 - y_2}{y_1 - y_2} I_1 + \frac{y_1 - y_4}{y_1 - y_2} I_2$$

$$I_p = \frac{x_5 - x_p}{x_5 - x_4} I_4 + \frac{x_p - x_4}{x_5 - x_4} I_5$$

Example



$P_1=(1,2),\quad P_2=(5,0),\quad P_3=(3,4),\quad P_4=(3,1),\quad P_5=(4,1),\quad P=(3.5,1)$

$N_1=(0,1,0),\quad N_2=(0,0,1),\quad N_3=(0,0.7,0.7)$

$V=(0,0,1),\qquad L=(0,0.7,0.7)$

$K_a=0.1,\quad I_a=0.5,\quad K_d=0.5,\quad K_s=0.2,\quad n_s=1,\quad I_L=0.6$

Find : $I_4,\quad I_5,\quad I_P$ with Gouraud shading

Example

First Step: Find the illumination amount at the vertices: I_1, I_2, and I_3

$$I = K_a I_a + K_d (N \cdot L) I_L + K_s (V \cdot R)^{n_s} I_L,$$

$$R = 2(N \cdot L) N - L$$

$$R_1 = 2 \cdot 0.7 N_1 - L = (0, 1.4, 0) - (0, 0.7, 0.7)$$

$$= (0, 0.7, -0.7)$$

$$I_1 = 0.1 \cdot 0.5 + 0.5 \cdot 0.7 \cdot 0.6 + 0 \cdot 2(-0.7) 0.6 = 0.26$$

$$R_2 = 2 \cdot 0.7 N_2 - L = (0, 0, 1.4) - (0, 0.7, 0.7)$$

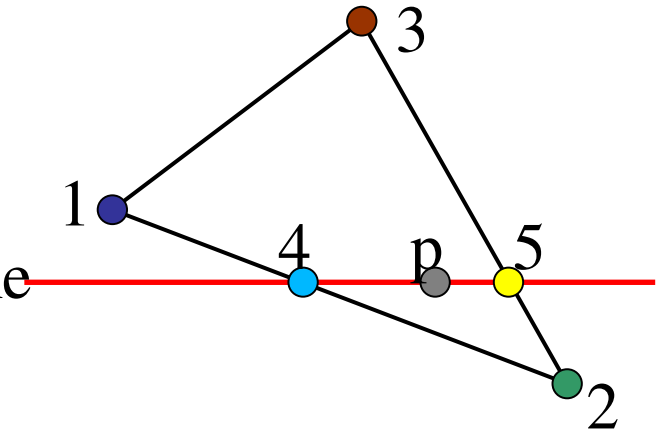
$$= (0, -0.7, 0.7)$$

$$I_2 = 0.1 \cdot 0.5 + 0.5 \cdot 0.7 \cdot 0.6 + 0.2(0.7) 0.6 = 0.34$$

$$R_3 = 2 \cdot 1 N_3 - L = (0, 1.4, 1.4) - (0, 0.7, 0.7)$$

$$= (0, 0.7, 0.7)$$

$$I_3 = 0.1 \cdot 0.5 + 0.5 \cdot 1 \cdot 0.6 + 0.2(0.7) 0.6 = 0.43$$



$$P_1 = (1, 2), \quad P_2 = (5, 0), \quad P_3 = (3, 4),$$

$$P_4 = (3, 1), \quad P_5 = (4, 1), \quad P = (3.5, 1)$$

$$N_1 = (0, 1, 0), \quad N_2 = (0, 0, 1),$$

$$N_3 = (0, 0.7, 0.7)$$

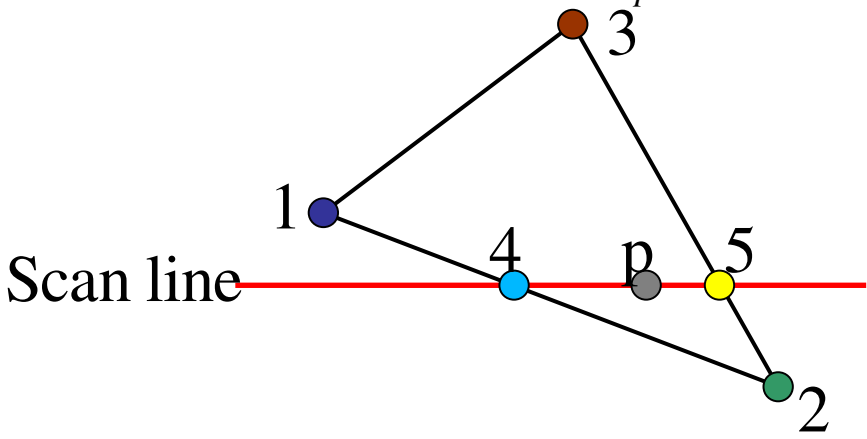
$$V = (0, 0, 1), \quad L = (0, 0.7, 0.7)$$

$$K_a = 0.1, \quad I_a = 0.5, \quad K_d = 0.5,$$

$$K_s = 0.2, \quad n_s = 1, \quad I_L = 0.6$$

Example

Second step: Find the illumination amount at the vertices: I_4, I_5, and I_p using I_1, I_2, and I_3.



$$I_4 = \frac{y_4 - y_2}{y_1 - y_2} I_1 + \frac{y_1 - y_4}{y_1 - y_2} I_2$$

$$= \frac{(1-0)}{(2-0)} 0.26 + \frac{(2-1)}{(2-0)} 0.34 = 0.13 + 0.17 = 0.3$$

$$I_5 = \frac{y_5 - y_2}{y_3 - y_2} I_3 + \frac{y_3 - y_5}{y_3 - y_2} I_2$$

$$= \frac{(1-0)}{(4-0)} 0.43 + \frac{(4-1)}{(4-0)} 0.34 = 0.11 + 0.25 = 0.36$$

$$I_P = \frac{x_5 - x_P}{x_5 - x_4} I_4 + \frac{x_P - x_4}{x_5 - x_4} I_5$$

$$I_P = \frac{(4-3.5)}{(4-3)} 0.3 + \frac{(3.5-3)}{(4-3)} 0.36 = 0.15 + 0.18 = 0.33$$

$$I_1 = 0.26$$

$$I_2 = 0.34$$

$$I_3 = 0.43$$

A more efficient approach

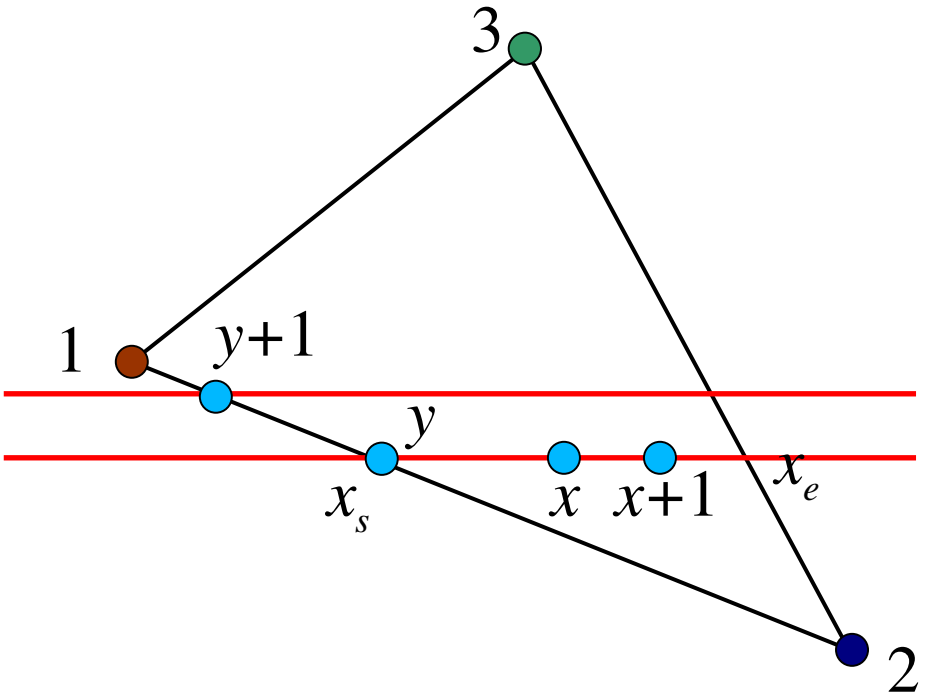
- Incremental calculation:

$$I_y = \frac{I_1 - I_2}{y_1 - y_2}$$

$$I_x = \frac{I_{x_e} - I_{x_s}}{x_e - x_s}$$

$$I_{y+1} = I_y + I_y$$

$$I_{x+1} = I_x + I_x$$

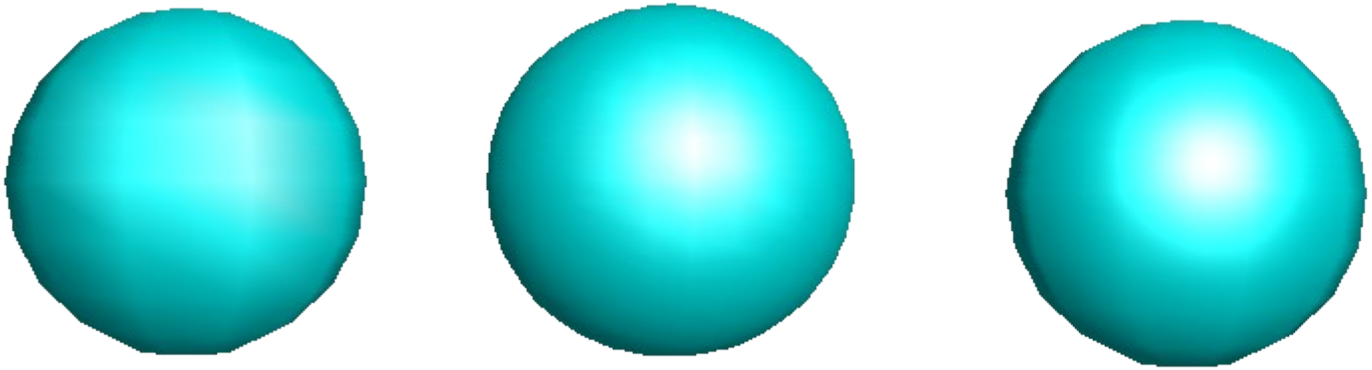


Gouraud Shading

- Fast and acceptable quality
- Problem:
  - Unnatural highlight shapes
  - Intensity streaks (mach bands)

Gouraud versus Phong Shading

- Intensity interpolation with lower number of polygons.



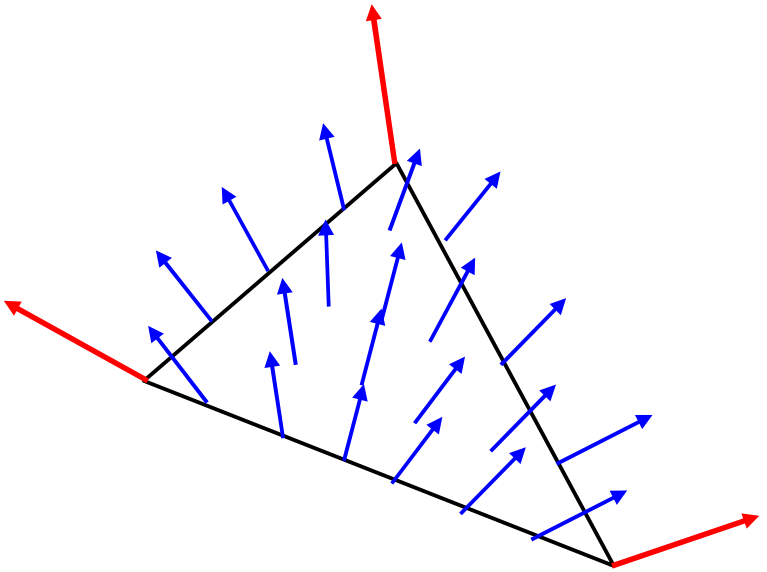
Gouraud
10x10
polygons

Gouraud
30x30
polygons

Phong
10x10
polygons

Normal Interpolation Rendering

- Phong rendering or shading.
- Idea similar to Gouraud but surface normals are interpolated for each pixel.
  - Calculate unit normals at vertices from adjacent surface normals
  - For each pixel of the polygon:
    - Interpolate the normal from the vertex normals
    - Calculate the intensity by applying an illumination model.
- Better results than Gouraud, but much expensive.



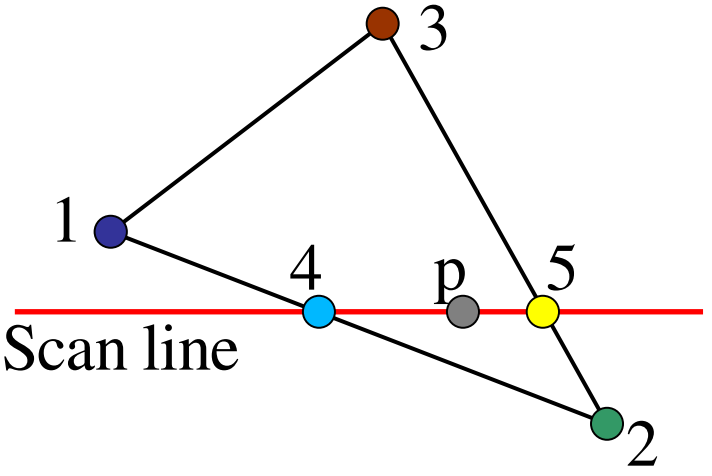
Phong surface rendering

- Normal vector interpolation with scanlines

$$N_4 = \frac{y_4 - y_2}{y_1 - y_2} N_1 + \frac{y_1 - y_4}{y_1 - y_2} N_2$$

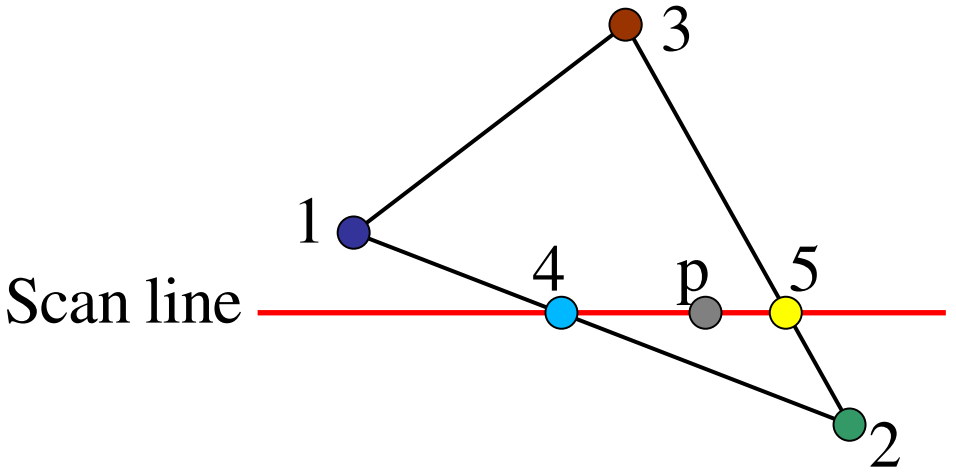
$$N_P = \frac{x_5 - x_P}{x_5 - x_4} N_4 + \frac{x_P - x_4}{x_5 - x_4} N_5$$

$$I_P = K_a I_a + \left(K_d \left(N_P \cdot L \right) + K_s \left(R_P \cdot V \right)^{n_s} \right) I_L$$



- Expensive since it requires calculation of intensity for each pixel.
- Accelarations based on Taylor expansion exists but still very expensive compared to Gouraud.

Example



$P_1=(1,2),\ P_2=(5,0),\ P_3=(3,4),\ P_4=(3,1),\ P_5=(4,1),\ P=(3.5,1)$

$N_1=(0,1,0),\ N_2=(0,0,1),\ N_3=(0,0.7,0.7)$

$V=(0,0,1),\quad L=(0,0.7,0.7)$

$K_a=0.1,\ I_a=0.5,\ K_d=0.5,\ K_s=0.2,\ n_s=1,\ I_L=0.6$

Find : I_P with Phong shading

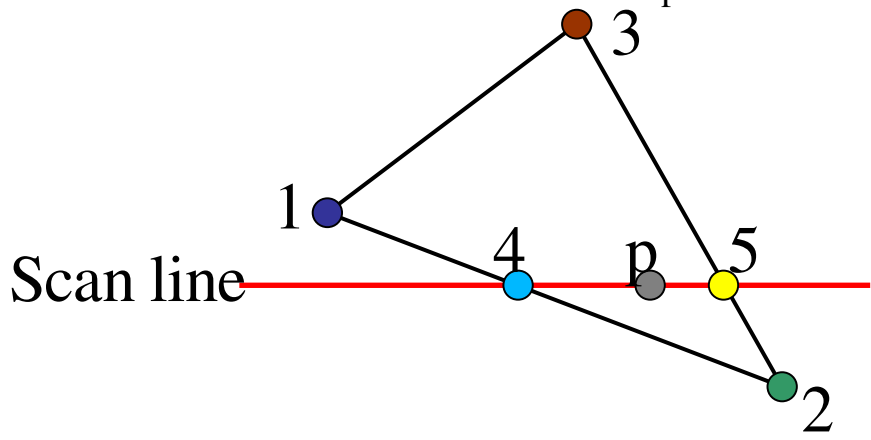
Example

First Step: Find the normal vectors at points 4, 5, and p: $\mathbf{N}_4$, $\mathbf{N}_5$, and $\mathbf{N}_p$

$$N_4=\frac{(y_4-y_2)}{(y_1-y_2)}N_1+\frac{(y_1-y_4)}{(y_1-y_2)}N_2$$
$$=0.5(0,1,0)+0.5(0,0,1)=(0,0.5,0.5),$$
unit vector $N_4=(0,0.7,0.7)$

$$N_5=\frac{(y_5-y_2)}{(y_3-y_2)}N_3+\frac{(y_3-y_5)}{(y_3-y_2)}N_2$$
$$=0.25(0,0.7,0.7)+0.75(0,0,1)=(0,0.18,0.93),$$
unit vector $N_5=(0,0.19,0.98)$

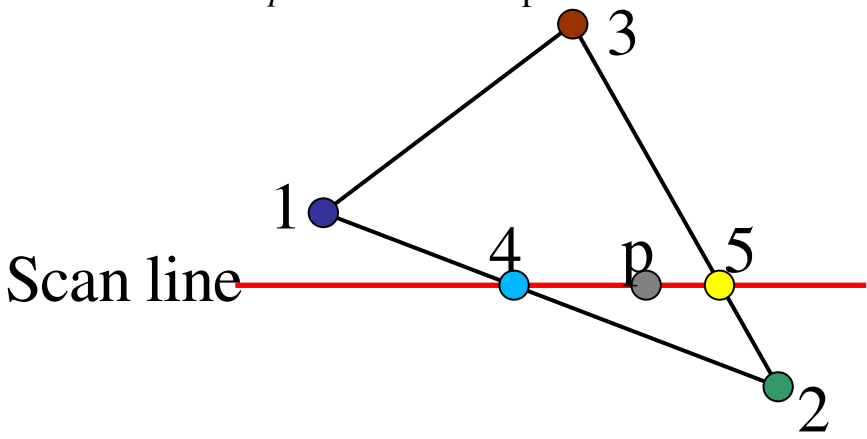
$$N_P=\frac{(x_5-x_P)}{(x_5-x_4)}N_4+\frac{(x_P-x_4)}{(x_5-x_4)}N_5$$
$$=0.5(0,0.7,0.7)+0.5(0,0.19,0.98)$$
$$=(0,0.45,0.84),$$
unit vector $N_P=(0,0.47,0.88)$



$P_1=(1,2)$, $P_2=(5,0)$, $P_3=(3,4)$,
$P_4=(3,1)$, $P_5=(4,1)$, $P=(3.5,1)$
$N_1=(0,1,0)$, $N_2=(0,0,1)$,
$N_3=(0,0.7,0.7)$
$V=(0,0,1)$, $L=(0,0.7,0.7)$
$K_a=0.1$, $I_a=0.5$, $K_d=0.5$,
$K_s=0.2$, $n_s=1$, $I_L=0.6$

Example

Second step: Find the illumination amount at point **p**, I_p ,using $\mathbf{N}_p$



$$I=K_a I_a+K_d\left(N\cdot L\right)I_L+K_s\left(V\cdot R\right)^{n_s} I_L,$$

$$R=2\left(N\cdot L\right)N-L$$

$$R_p=2\left(0.945\right)N_p-L$$

$$=\left(0,0.89,1.66\right)-\left(0,0.7,0.7\right)$$

$$=\left(0,0.19,0.96\right)$$

$$I_p=0.1\cdot 0.5+0.5\left(0.945\right)0.6+0.2\left(0.96\right)0.6$$

$$=0.45$$

$$N_p=\left(0,0.47,0.88\right)$$

$$V=\left(0,0,1\right),\qquad L=\left(0,0.7,0.7\right)$$

$$K_a=0.1,\quad I_a=0.5,\quad K_d=0.5,$$

$$K_s=0.2,\quad n_s=1,\quad I_L=0.6$$

We found I_p as 0.33 with Gouraud shading previously

OpenGL Illumination and Surface-Rendering Functions

OpenGL Lighting Functions

- OpenGL provides functions for
  - setting up point light sources and spotlights,
  - selecting surface reflection coefficients
  - choosing values for several parameters in the basic illumination model
- OpenGL supports constant intensity rendering (flat shading) and Gouraud surface rendering (smooth shading)
  - OpenGL does not support Phong shading.

Light Sources: GL_MAX_LIGHTS

- Up to eight light sources can be included in a standard OpenGL implementation
  - Some implementations may allow more than 8
  - This maximum number is a limitation on the number of light sources by which a single object is illuminated. When light attenuation and direction are accounted for, you may find that any given piece of geometry in your scene is only illuminated by a small handful of lights
  - If you have multiple light emitting objects, and your goal is to view the emission (not the effect of these lights on other objects) they can be modeled by texture mapping.

Light Sources

- Properties of a light source such as position, type, color, attenuation, and spotlight effects can be specified with the following OpenGL function:

  glLight* (lightName, lightProperty, propertyValue);

- Each light source is referenced by its identifier which is one of GL_LIGHT0, GL_LIGHT1, ..., GL_LIGHT7
- A light source is enabled with:

  glEnable (lightName);

Lighting

- Lighting is enabled globally in an OpenGL program with the following command:

```
glEnable (GL_LIGHTING);
```

Light Properties

glLight* (lightName, lightProperty, propertyValue);

lightProperty can be one of:

- GL_POSITION
- GL_AMBIENT
- GL_DIFFUSE
- GL_SPECULAR
- GL_CONSTANT_ATTENUATION
- GL_LINEAR_ATTENUATION
- GL_QUADRATIC_ATTENUATION
- GL_SPOT_DIRECTION
- GL_SPOT_CUTOFF
- GL_SPOT_EXPONENT

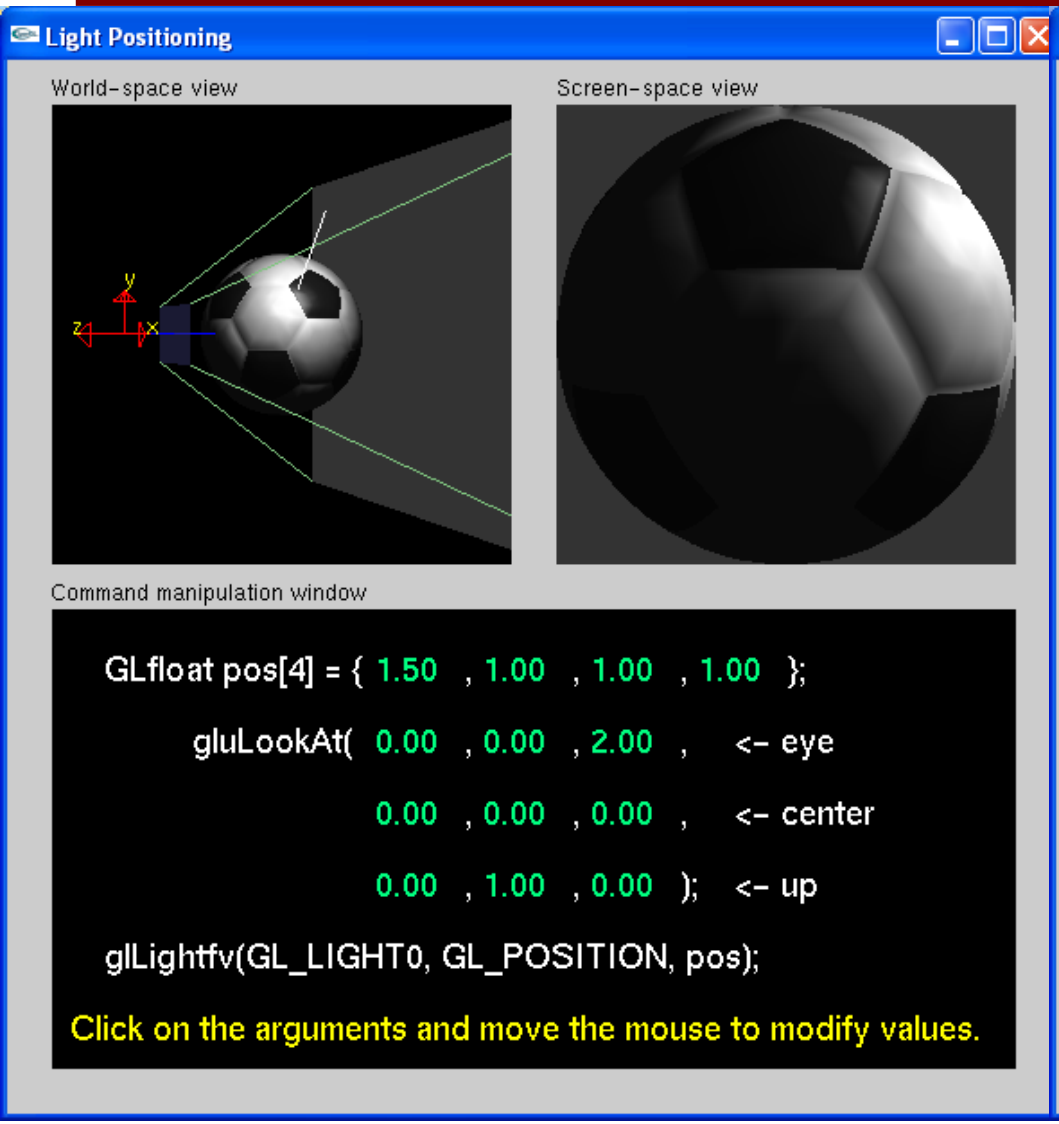
Light-Source Position and Type

- GL_POSITION property constant is used to specify both the position and type of a light source
- Two types of lights
  - Local light sources that are near the objects
  - Light sources at infinity
- **Remark**: The type of a light source is independent of the 3d position we assign to a light source. It is determined by a fourth parameter value.

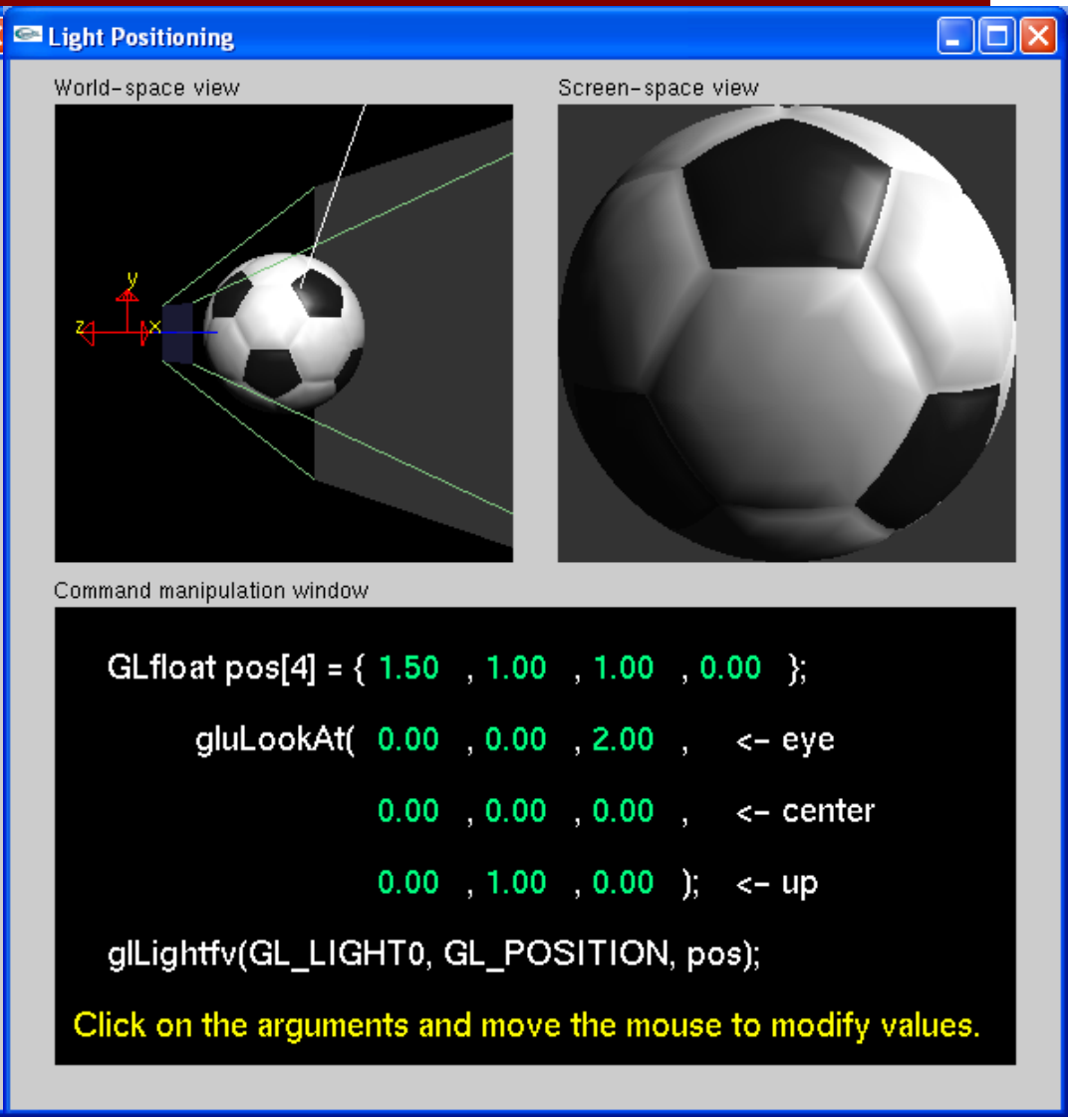
Light Sources: Local vs. Infinite

- For a local light source, the emitted light radiates in all directions and the position of the light is used in the lighting calculations
  - I.e., the light direction changes for every object
  - In OpenGL, it is referred to as “Positional Light”
- For a light source at infinity, the emitted light radiates in one direction only and this direction is applied to all objects in the scene
  - I.e., the light direction is constant for every object
  - The direction is given from the light source position to the coordinate origin
  - In OpenGL, it is referred to as “Directional Light”

Example



Positional light



Directional light

Applying transformations

- A light source is like an object in the scene and included in the scene description
- Geometric transformation matrices and viewing transformation matrices are applied to light positions as applied to other objects
  - If you want to fix your light position relative to the objects in your scene, you need to set the light position after you specify geometric and viewing transformations
  - If we want the light move as the view point moves, set its position before gluLookAt(...)

Light Source Colors

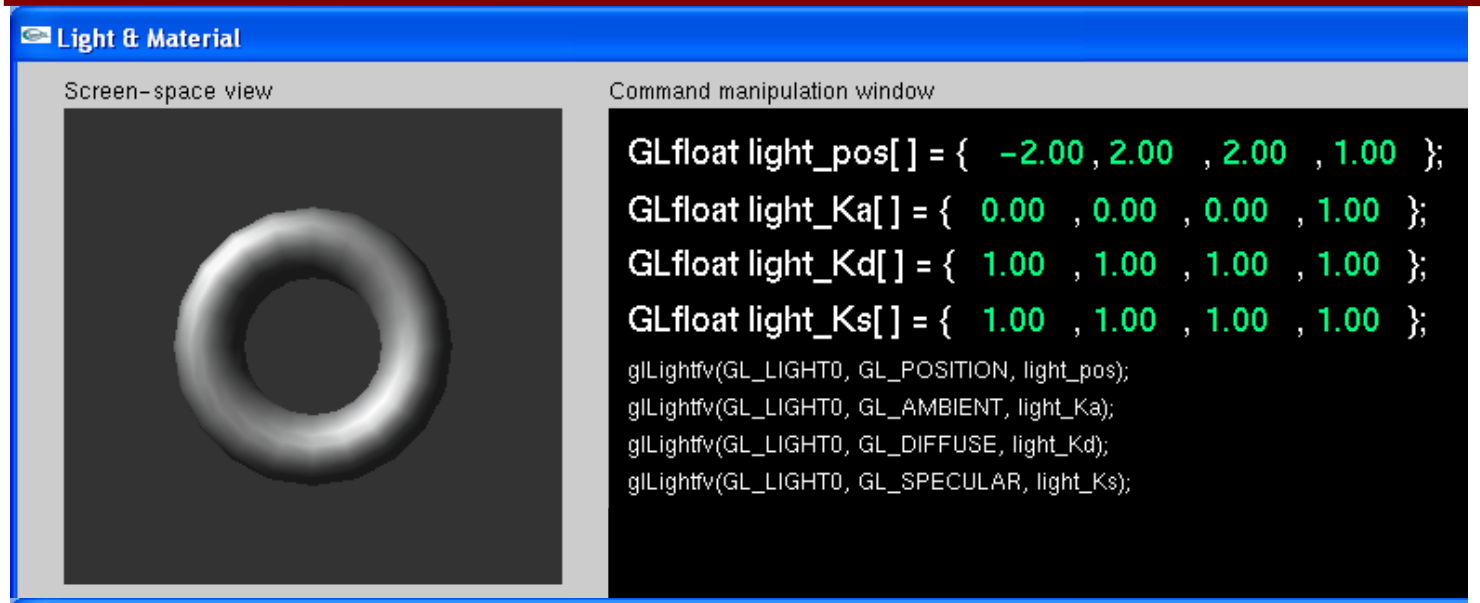
- You can specify three different color properties to a light source
  - This is unrealistically flexible compared to real-world lights which have one color
- We can specify the ambient, diffuse, and specular components of a light source with different colors.

Example

```
GLfloat blackColor [] = {0.0, 0.0, 0.0, 1.0};
GLfloat whiteColor [] = {1.0, 1.0, 1.0, 1.0};


glLightfv (GL_LIGHT3, GL_AMBIENT, blackColor);
glLightfv (GL_LIGHT3, GL_DIFFUSE, whiteColor);
glLightfv (GL_LIGHT3, GL_SPECULAR, whiteColor);
```

Example



Black ambient
White diffuse
White specular

Black ambient
Red diffuse
Blue specular

Radial-Intensity Attenuation

- In other words: distance base decrease in the lighting amount
- OpenGL provides functions to set the constants in the following equation

$$f_{atten}(d_l) = \frac{1}{a_0 + a_1 d_l + a_2 d_l^2}$$

```
glLightf (GL_LIGHT3, GL_CONSTANT_ATTENUATION, 1.5);
glLightf (GL_LIGHT3, GL_LINEAR_ATTENUATION, 0.75);
glLightf (GL_LIGHT3, GL_QUADRATIC_ATTENUATION, 0.4);
```

By default a_0=1.0, a_1=0.0, a_2=0.0, which means no attenuation.

Directional Local Light Sources

- a.k.a Spotlights
- In order to limit the light to a cone-shaped region of space
- The direction of the spotlight, the angular extent of the spotlight and the angular attenuation factor can be specified

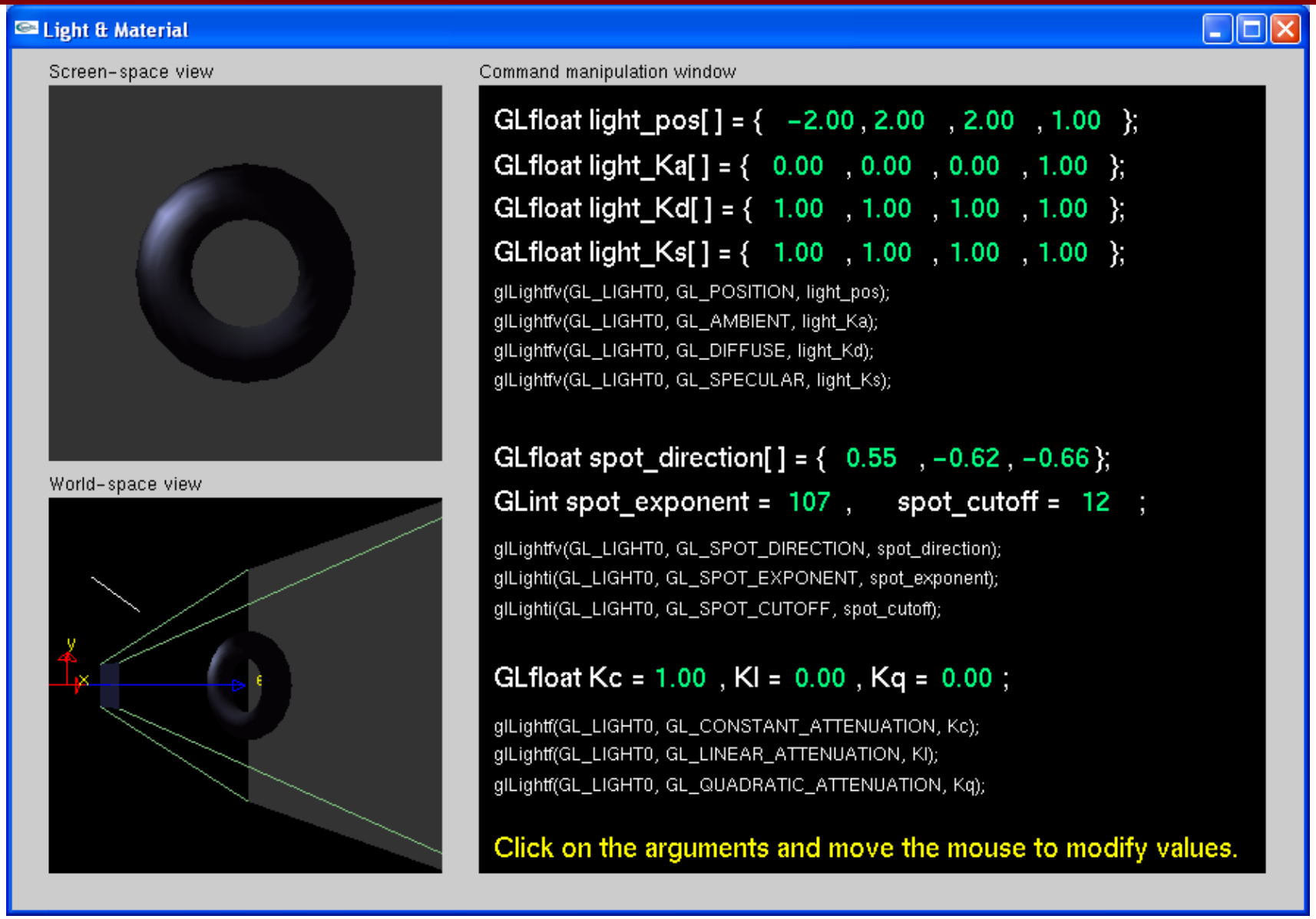
```
GLfloat dirVector [] = {1.0, 0.0, 0.0};
glLightf (GL_LIGHT3, GL_SPOT_DIRECTION, dirVector);
glLightf (GL_LIGHT3, GL_SPOT_CUTOFF, 30.0);
glLightf (GL_LIGHT3, GL_SPOT_EXPONENT, 2.5);
```

V_{light} →

θ_l →

→ $\cos^{a_l}\alpha = (V_{\text{obj}} \cdot V_{\text{light}})^{a_l}$

Example


Light & Material
Screen-space view
World-space view
Command manipulation window
GLfloat light_pos[ ] = { -2.00 , 2.00 , 2.00 , 1.00 };
GLfloat light_Ka[ ] = { 0.00 , 0.00 , 0.00 , 1.00 };
GLfloat light_Kd[ ] = { 1.00 , 1.00 , 1.00 , 1.00 };
GLfloat light_Ks[ ] = { 1.00 , 1.00 , 1.00 , 1.00 };
glLightfv(GL_LIGHT0, GL_POSITION, light_pos);
glLightfv(GL_LIGHT0, GL_AMBIENT, light_Ka);
glLightfv(GL_LIGHT0, GL_DIFFUSE, light_Kd);
glLightfv(GL_LIGHT0, GL_SPECULAR, light_Ks);
GLfloat spot_direction[ ] = { 0.55 , -0.62 , -0.66 };
GLint spot_exponent = 107 , spot_cutoff = 12 ;
glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, spot_direction);
glLighti(GL_LIGHT0, GL_SPOT_EXPONENT, spot_exponent);
glLighti(GL_LIGHT0, GL_SPOT_CUTOFF, spot_cutoff);
GLfloat Kc = 1.00 , Kl = 0.00 , Kq = 0.00 ;
glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, Kc);
glLightf(GL_LIGHT0, GL_LINEAR_ATTENUATION, Kl);
glLightf(GL_LIGHT0, GL_QUADRATIC_ATTENUATION, Kq);
Click on the arguments and move the mouse to modify values.