# Illumination II

*Local illumination model:* only consider the effect of light arriving directly from light sources, and use crude approximations for indirect lighting effects.

*Luminance function:*

$$I = (I_r, I_g, I_b)$$

*Components of OpenGL local illumination model:*

glLightfv(source,parameter,pointer-to-array)

glLightf(source,parameter,value)

- ambient emission

$$I_a = \begin{pmatrix} I_{ar} \\ I_{ag} \\ I_{ab} \end{pmatrix}$$

- point emission

$$I(p_0) = \begin{pmatrix} I_r(p_0) \\ I_g(p_0) \\ I_b(p_0) \end{pmatrix}$$

GLfloat ambient0[]=1.0,0.0,0.0,1.0

GLfloat diffuse0[]=1.0,0.0,0.0,1.0

GLfloat specular0[]=1.0,1.0,1.0,1.0

GLfloat light0-pos[]=1.0,2.0,1.0,1.0

glLightfv(GL-LIGHT0,GL-AMBIENT,ambient0)

glLightfv(GL-LIGHT0,GL-POSITION,light0-pos)

*Attenuation function:*

$$I(p, p_0) = \frac{1}{a + bd + cd^2} I(p_0)$$

where $d = |p - p_0|^2$.

glLightf(GL-LIGHT0,GL-CONSTANT-ATTENUATION,a)

# The Phong Illumination Model

Components of Phong illumination or reflection model:

- *Emission:* to model objects that glow
- *Ambient reflection:* A simple way to model indirent reflection. All surfaces in all positions and orientations are illuminated equally.
- *Diffuse reflection:* The shading produced by dull smooth objects.
- *Specular reflection:* The bright spots appearing on smooth shiny (e.g., metallic or polished) surfaces.

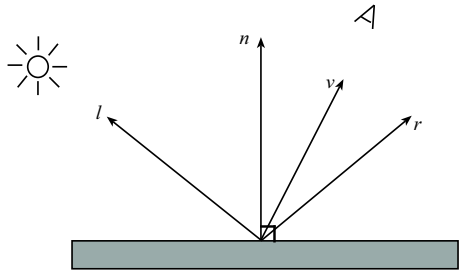
*Question:* What is the amount of light that is transmitted (either by emission or reflection) from each point in the direction of the viewer.

*Question:* This is achieved by first associating reflectivity or material properties to all the modelled objects in the scene, and then applying a Phong reflection calculation to determine the transmitted light intensity.

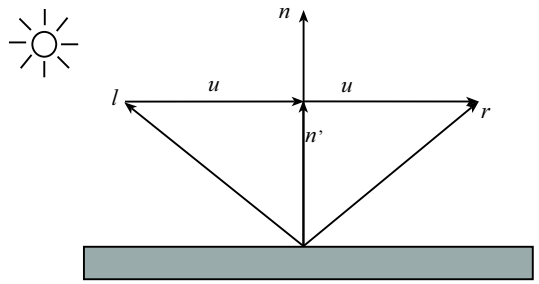
# Relevant Vectors

The shading of a point on a surface is a function of the relatonship between the viewer, light sources, and surface. The following vectors are relevant to direct illumination. All vectors are assumed to be normalized to unit length.

- *Normal vector:* A vector $\vec{n}$ that is perpendicular to the surface and directed outwards from the surface.
- *View vector:* A vector $\vec{v}$ that points in the direction of the viewer.
- *Light vector:* A vector $\vec{l}$ that points towards the light source.
- *Reflection vector:* A vector $\vec{r}$ that indicates the direction of pure reflection of the light vector.



# The Reflection Vector




Reflection vector

$$\vec{n}' = (\vec{n} \cdot \vec{l})\vec{n}$$

$$\vec{u} = \vec{n}' - \vec{l}$$

$$\vec{r} = \vec{l} + 2\vec{u} = \vec{l} + 2(\vec{n}' - \vec{l}) = 2(\vec{n} \cdot \vec{l})\vec{n} - \vec{l}$$

# Specular Reflection

Most objects are not perfect Lambertian reflector. One of the most common deviation is for smooth metallic or highly polished objects. They tend to have *specular highlights* (or "shiny spots").

There are common models of specular reflection: Phong model and *halfway vector model* (OpenGL).

Define $\vec{h}$ = Normalize $(\vec{l} + \vec{v})$.

The parameters of surface that control specular reflection are $k_s$, the surface's *coefficient of specular reflection*, and $\alpha$, *shininess*.

The formula for the specular component is

$$I_s = k_s(\vec{n} \cdot \vec{h})^{\alpha} L_s C_s$$

# Phong Illumination Equation

The total illumination of a point in OpenGL is computed for the supported Light sources and is calculated

$$I = I_e + I_a + \frac{1}{a + bd + cd^2}(I_d + I_s)$$
$$= I_e + k_a L_a C_a + \frac{1}{a + bd + cd^2}(k_d \max(0, \vec{n} \cdot \vec{l}) L_d C_d + k_s (\vec{n} \cdot \vec{h})^{\alpha} L_s C_s),$$

where $d$ is the distance from the object to the light source. The reflection material properties for front/back of each surface is specified by OpenGL using functions

glMaterialfv(face,type,pointer-to-array)

glMaterialf(face,value)

GLfloat ambient[]=0.1,0.25,0.0,1.0

GLfloat diffuse[]=0.1,0.25,0.0,1.0

GLfloat specular[]=1.0,0.0,1.0,1.0

GLfloat emission[]=0.0,0.8,0.0,1.0

glMaterialfv(GL-front-and-back,GL-specular,specular)

glMaterialf(GL-front-and-back,GL-shininess, 100.0)

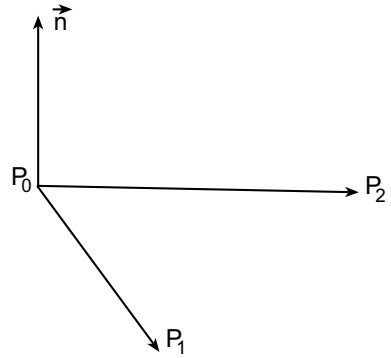
For multiple light sources, we add up the ambient, diffuse, and specular components for each light source.

# Different Normals

## Normals by Cross Products

Given any three non-collinear points, $P_0$, $P_1$, $P_2$, on a polygon, a normal of the polygon is given through a cross product

$$\vec{n} = (P_1 - P_0) \times (P_2 - P_0).$$

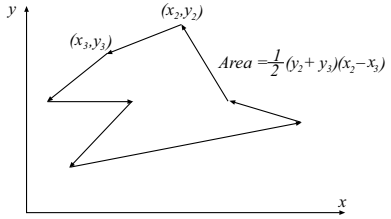


# Normals by Area

A polygon is given by $n$ points $P_0$, $P_1$, . . . , $P_{n-1}$. If we can determine a plane equation

$$ax + by + cz + d = 0$$

from these points, then $(a, b, c)$ is the normal vector of the polygon.



$$a = \frac{1}{2}\sum_{i=1}^{n}(z_i + z_{i+1})(y_i - y_{i+1})$$

$$b = \frac{1}{2}\sum_{i=1}^{n}(x_i + x_{i+1})(z_i - z_{i+1})$$

$$c = \frac{1}{2}\sum_{i=1}^{n}(y_i + y_{i+1})(x_i - x_{i+1})$$

Then normalizing $(a, b, c)$ is $\vec{n}$.

# Normals for Implicitly Defined Surfaces

Given a surface defined by an *implicit representation*, i.e., defined by some equation

$$f(x, y, z) = 0$$

then the normal at some point is given by *gradient vector*

$$\vec{n} = \begin{pmatrix} \partial f/\partial x \\ \partial f/\partial y \\ \partial f/\partial z \end{pmatrix}$$

# Normals for Parametric Surfaces

Surfaces in computer graphics are most often represented parametrically. The *parametric representation* of a surface is defined by three functions of 2 variables or *parameters*:

$$x = \phi_x(u, v),$$
$$y = \phi_y(u, v),$$
$$z = \phi_z(u, v).$$

Then the normal of the surface at a point is defined as

$$\vec{n} = \frac{\partial \phi}{\partial u} \times \frac{\partial \phi}{\partial v}$$

where

$$\frac{\partial \phi}{\partial u} = \begin{pmatrix} \partial \phi_x / \partial u \\ \partial \phi_y / \partial u \\ \partial \phi_z / \partial u \end{pmatrix} \qquad \frac{\partial \phi}{\partial v} = \begin{pmatrix} \partial \phi_x / \partial v \\ \partial \phi_y / \partial v \\ \partial \phi_z / \partial v \end{pmatrix}$$

# Shading

Shading algorithms apply lighting models to polygons, through interpolation from the vertices.

OPENGL:

glShadeModel(GL-FLAT)

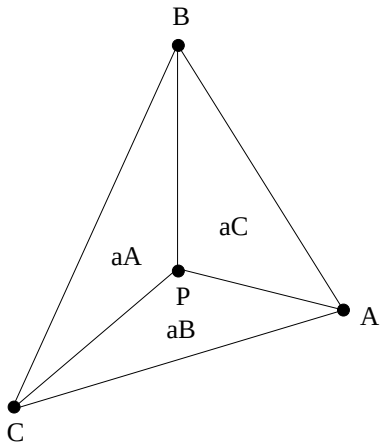
*Gouraud Shading:* Lighting in only computed at the vertices, and the colors are interpolated across the (convex) polygon

*Phong Shading:* A normal is specified at each vertex, and this normal is interpolated across the polygon. At each pixel, a lighting model is calculated.

# Gouraud Shading

- *Gouraud shading* interpolates colors across a polygon from the vertices
- Lighting calculations are only performed at the vertices
- Highlights can be missed or blurred
- Common in hardware renderers; model that OpenGL supports
- Gouraud shading is well-defined only for triangles...
  Equivalent to a *barycentric combination*
- Barycentric combinations are also *affine combinations*...
  Triangular Gouraud shading is *invariant* under affine transformations

# Triangle Gouraud Shading




2

$$\alpha_A = D(P, B, C)/D(A, B, C)$$

$$\alpha_B = D(A, P, C)/D(A, B, C)$$

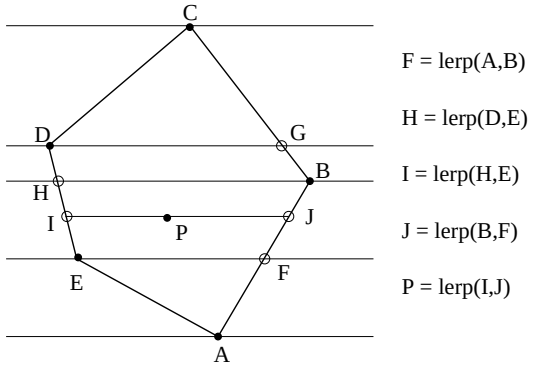
$$\alpha_C = D(A, B, P)/D(A, B, C)$$

$$\alpha_A + \alpha_B + \alpha_C = 1$$

$$P = \alpha_A\, A + \alpha_B\, B + \alpha_C\, C$$

$$D(A,B,C) = \begin{vmatrix} 1 & 1 & 1 & 1 \\ 1 & x_A & y_A & z_A \\ 1 & x_B & y_B & z_B \\ 1 & x_C & y_C & z_C \end{vmatrix}$$

- Gouraud shading for polygons with more than three vertices:
  - Sort the vertices by $y$ coordinate
  - Slice the polygon into trapezoids with parallel top and bottom
  - Interpolate colors along each edge of the trapezoid...
  - Interpolate colors along each scanline



- Gouraud shading gives *bilinear* interpolation within each trapezoid
- Since rotating the polygon can result in a different trapezoidal decomposition, $n$-sided Gouraud interpolation is *not affine invariant*

glShadeModel(GL-SMOOTH)

# Phong Shading

- *Phong Shading* interpolates lighting model parameters, *not* colors
- Much better rendition of highlights
- A *normal* is specified at each vertex of a polygon
- Vertex normals are independent of the polygon normal
- Vertex normals should relate to the surface being approximated by the polygon
- The normal is interpolated across the polygon (using Gouraud techniques).
- At each pixel,
  - Interpolate the normal...
  - Interpolate other shading parameters...
  - Compute the view and light vectors...
  - Evaluate the lighting model
- The lighting model does not have to be the Phong lighting model...
- Normal interpolation is nominally done by vector addition and renormalization
- Several "fast" approximations are possible
- The view and light vectors may also be interpolated or approximated

# Reading Assignment and News

Chapter 6 pages 275 - 304, of Recommended Text.

On Wednesday October 22 (tomorrow) Peter Djeu shall conduct a recitation class from 2:30pm - 4:00pm in ACES 2.402, about subdivision surfaces. Please also track the News section of the Course Web Pages for the most recent Announcements related to this course.

(http://www.cs.utexas.edu/users/bajaj/graphics23/cs354/)