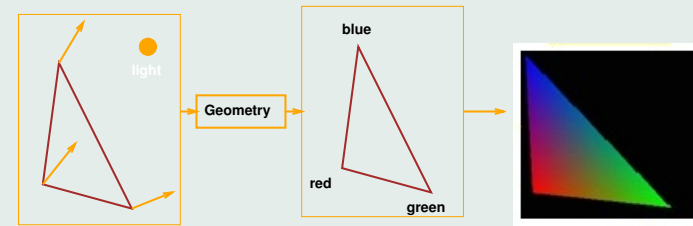


## Visual Appearance

- Simple Lighting Models
- Fog Models
- Gamma Correction
- Transparency and Alpha Blending

## How to Compute Lighting?



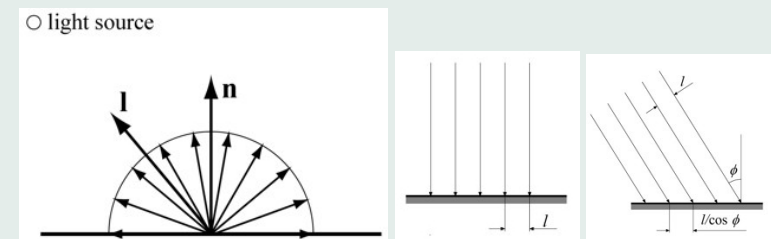
- Compute lighting at vertices, then interpolate over triangle
- Can set colors per vertex manually, but need more realism. Use
  - Light sources
  - Material properties
  - Geometrical relationships

## Local Lighting Model

$$i = i_{amb} + i_{diff} + i_{spec}$$

- Linear combination of ambient, diffuse and specular lighting models
- Not a physically realistic model (except diffuse)
- **Local Model**, only illumination from designated light sources considered

## Diffuse Reflection

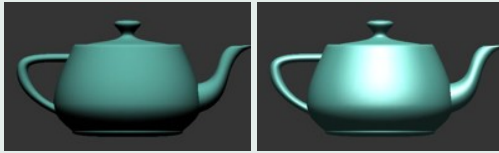


- Diffuse reflection is computed using **Lambert's law**:

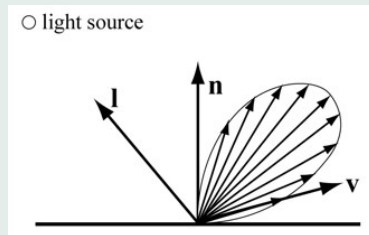
$$i_{diff} = (\vec{\mathbf{n}} \cdot \vec{\mathbf{l}}) m_{diff} \otimes s_{diff}$$

- Photons are scattered equally in all directions (viewer independent)
- Models **dull, matte** surfaces.

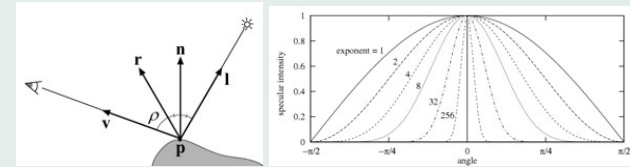
## Specular Reflection



- Diffuse is dull, specular model models shiny surfaces, with highlights
- Viewer dependent



## Specular Reflection:Phong's Model



- Use reflected ray  $\vec{r}$ , then dot prod. with view vector  $\vec{v}$

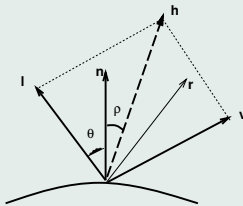
$$\vec{r} = -\vec{I} + 2((\vec{n} \cdot \vec{I})\vec{n})$$

$$i_{spec} = (\vec{r} \cdot \vec{v})^{m_{shi}} = \cos^2 \rho^{m_{shi}}$$

- In practice,

$$i_{spec} = \max(0, (\vec{r} \cdot \vec{v})^{m_{shi}}) m_{spec} \otimes S_{spec}$$

## Specular Reflection:Blinn's Model



- Instead of computing reflected ray  $\vec{r}$ , compute  $\vec{h}$ , the halfway vector between  $\vec{v}$  and  $\vec{I}$ , and compute its angle with  $\vec{n}$

$$\vec{h} = \frac{\vec{v} + \vec{I}}{\|\vec{v} + \vec{I}\|}$$

- Thus, we get

$$i_{spec} = \max(0, (\vec{h} \cdot \vec{n})^{m_{shi}}) m_{spec} \otimes S_{spec}$$

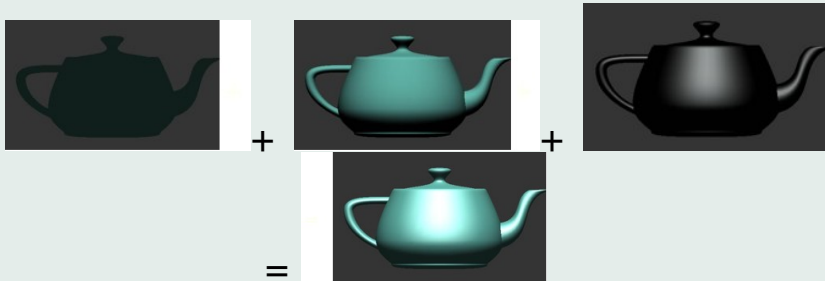
## Ambient Reflection

- No physical basis.
- Attempts to model interactions of light between objects with a **constant intensity, non-directional light source**.
- Used in conjunction with other models.

$$i_{amb} = m_{amb} \otimes S_{amb}$$

## Combined (Local) Model

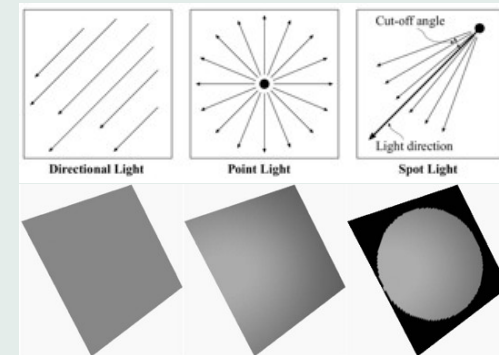
$$i = i_{amb} + i_{diff} + i_{spec}$$



⇒ This is a **hack**, little to do with how reality works!

## Light Source Types and Attenuation

- Lighting as a function of **distance**,  $1/(a + bt + ct^2)$
- **Multiple Lights**: sum their contributions
- Light sources can be **Directional**, **Point** or **Spot** lights



## Lighting vs. Shading

- Lighting: interaction between light and matter
- Shading: Evaluate lighting at object points and determine pixel's colors.
- Polygon Shading Algorithms:
  - Constant (Flat, Faceted)
  - Gouraud : Interpolate vertex **colors** (per vertex shading)
  - Phong: Interpolate vertex **normals** (per pixel shading)

## Polygon Rendering Algorithms

- Constant (Flat) Shading
- Interpolated Shading (Gouraud, Phong)

## Constant (Flat, Faceted) Shading

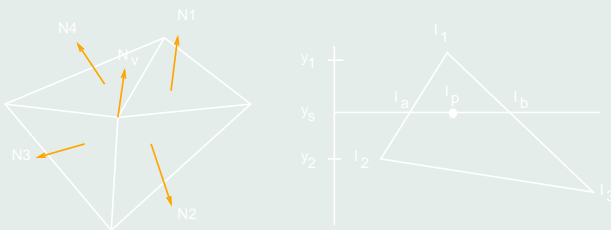
- A **constant** intensity for entire polygon.
- **Assumptions:**
  - ⇒ Light source is at infinity,  $\vec{N} \bullet \vec{L}$  is constant.
  - ⇒ Viewer is at infinity,  $\vec{N} \bullet \vec{V}$  is constant.
  - ⇒ Polygon represents the exact surface being modeled.

Extremely fast to compute, illumination equation is evaluated once per polygon, but produces unrealistic images in most cases.

## Interpolated Shading: (Wylie, Romney, Evans, Erdahl, Gouraud, Phong)

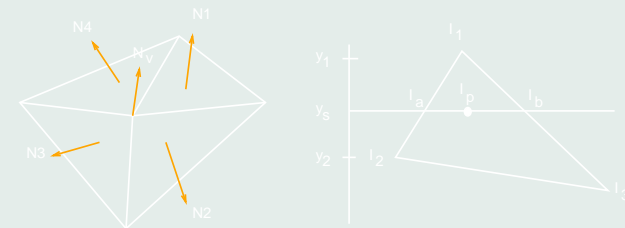
- Interpolates shading intensities at each point on the polygon, using the intensities computed at the **vertices** of the polygon.
- Developed for triangular facets originally and generalize to any polygon by Gouraud, and called **Gouraud shading**
- Can be easily integrated with a scan-line algorithm.
- **Interpolation** replaces the more expensive application of the illumination model at each interior point of the polygon.
- Assumes polygon represents the exact surface being modeled.

## Gouraud Shading



- Face Normals are averaged to obtain vertex normals.
- Vertex Intensities ( $I_1, I_2, I_3$ ) are obtained by application of a lighting model.
- Intensities at interior points are obtained as follows:

## Gouraud Shading

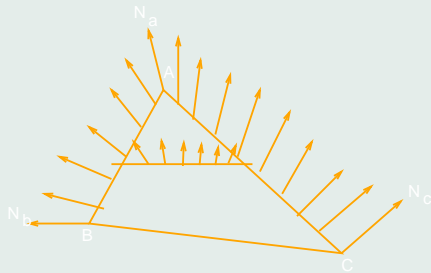


$$I_a = I_2 + (I_1 - I_2) \frac{y_s - y_2}{y_1 - y_2}$$

$$I_b = I_3 + (I_1 - I_3) \frac{y_s - y_3}{y_1 - y_3}$$

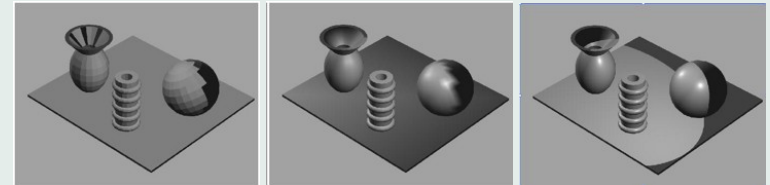
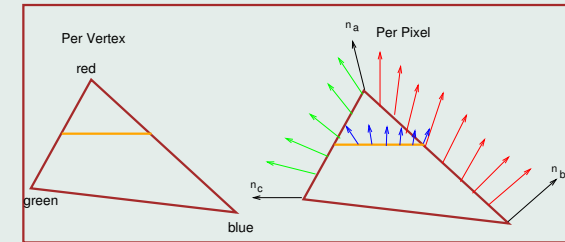
$$I_p = I_a + (I_b - I_a) \frac{x_p - x_a}{x_b - x_a}$$

## Phong Shading



- Compute Normals at vertices of polygon.
- Interpolate each component of normal at any interior point of polygon
- Apply illumination model.
- Can easily be incorporated into a scanline algorithm.

## Gouraud vs. Phong Shading



## Interpolated Shading: Problems

- **Polygonal Silhouette:** Interpolated shading only smooths out discontinuities across adjacent faces and has no effect on the polygonal geometry at the silhouette.
- **Perspective Distortion:** Vertex Intensities are computed in Object Coordinates, whereas shading is performed in Screen Coordinates, after Perspective transformation.
- T Vertex problem.

## Fog Models

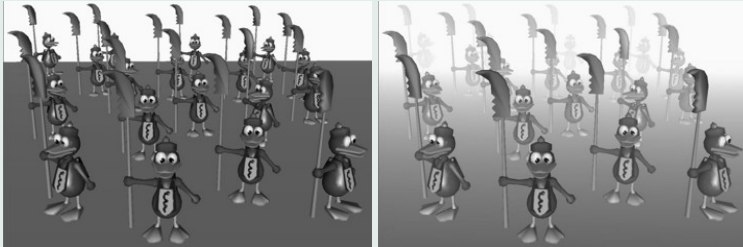
- Simple atmospheric effect
- A little better realism
- Help in determining distances
- **Fog Color:** color of surface:

$$c_p = fc_s + (1 - f)c_f, f \in [0, 1]$$

⇒ As  $f$  increases, effect of fog decreases

## Computing Fog

- **Linear:**  $f = \frac{z_{end} - z_p}{z_{end} - z_{start}}$
- **Exponential:**  $f = e^{-d_f z_p}$
- **Exponential Squared:**  $f = e^{(-d_f z_p)^2}$
- OpenGL implements all three types of fog.



## Transparency and Alpha Compositing

- Relatively simplistic, in real-time rendering systems
- **Alpha blending** is employed (mixing two colors)
- Alpha ( $\alpha$ ) is another component in the frame buffer, or on triangle.
- Represents the opacity (1.0 is totally opaque, 0.0 is totally transparent)

## The “Over” Operator

$$c_o = \alpha c_s + (1 - \alpha) c_d$$

- $c_o$  is the transparent (source) object color,  $c_d$  the pixel (destination) color before blending,  $\alpha$ , the opacity
- Usually requires **sorting**, render **back to front**
- $(R, G, B, \alpha)$  in textures as well
- Usually premultiplied alphas used ( $c_o = c'_s + (1 - \alpha_s) c_d$ ).
- Methods to compute transparency: A-Buffer (coverage masks), multiple passes to sort transparent objects (**depth peeling**)