Computer Graphics

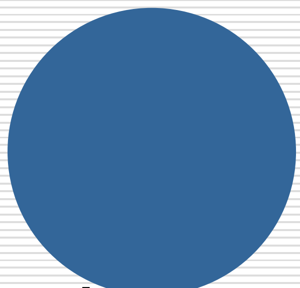
Bing-Yu Chen
National Taiwan University

Illumination and Shading

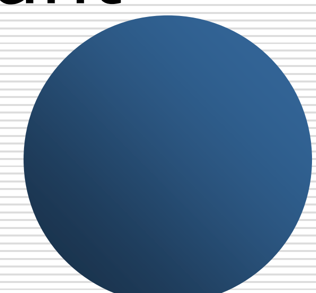
- Illumination Models
- Shading Models for Polygons
- Surface Detail
- Shadows
- Transparency
- Global Illumination
- Recursive Ray Tracing
- Radiosity
- The Rendering Pipeline

Why We Need Shading ?

- Suppose we build a model of a sphere using many polygons and color it with only one color. We get something like

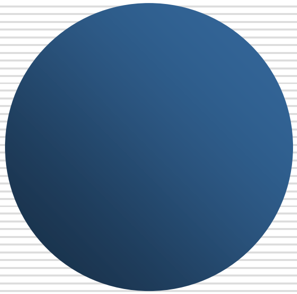


- But we want



Shading

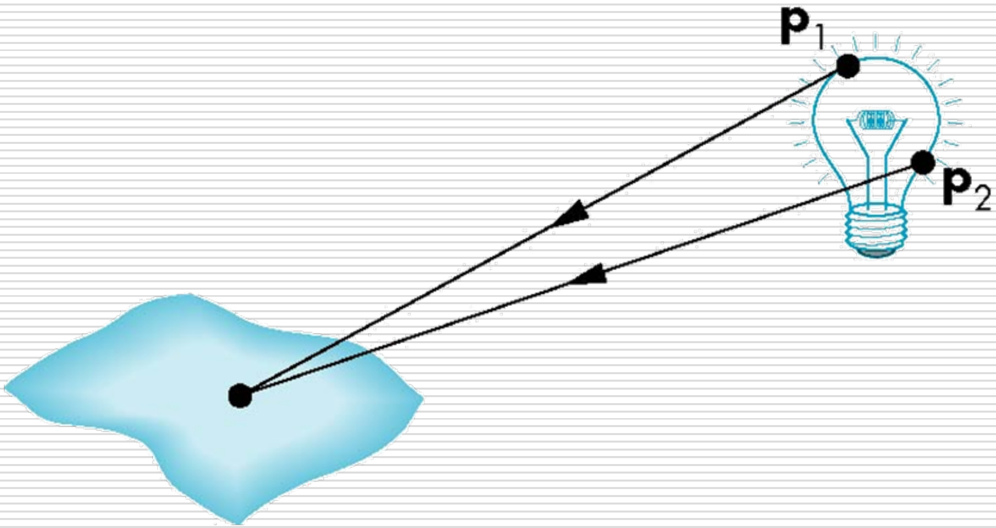
- Why does the image of a real sphere look like



- Light-material interactions cause each point to have a different color or shade
- Need to consider
  - Light sources
  - Material properties
  - Location of viewer
  - Surface orientation

Light Sources

- General light sources are difficult to work with because we must integrate light coming from all points on the source

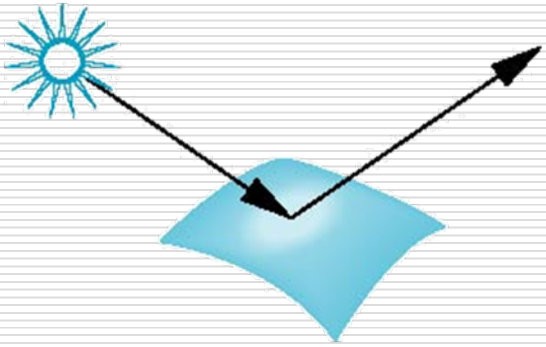


Simple Light Sources

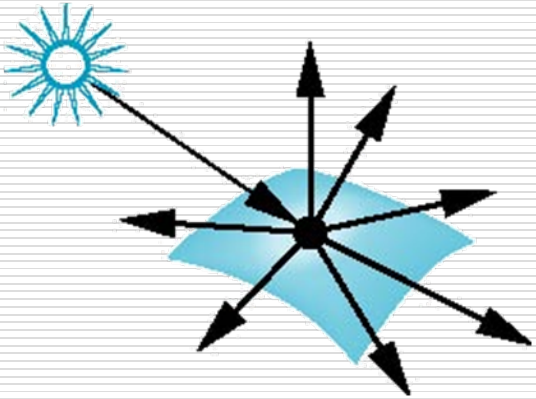
- Point source
  - Model with position and color
  - Distant source = infinite distance away (parallel)
- Spotlight
  - Restrict light from ideal point source
- Ambient light
  - Same amount of light everywhere in scene
  - Can model contribution of many sources and reflecting surfaces

Surface Types

- The smoother a surface, the more reflected light is concentrated in the direction a perfect mirror would reflected the light
- A very rough surface scatters light in all directions

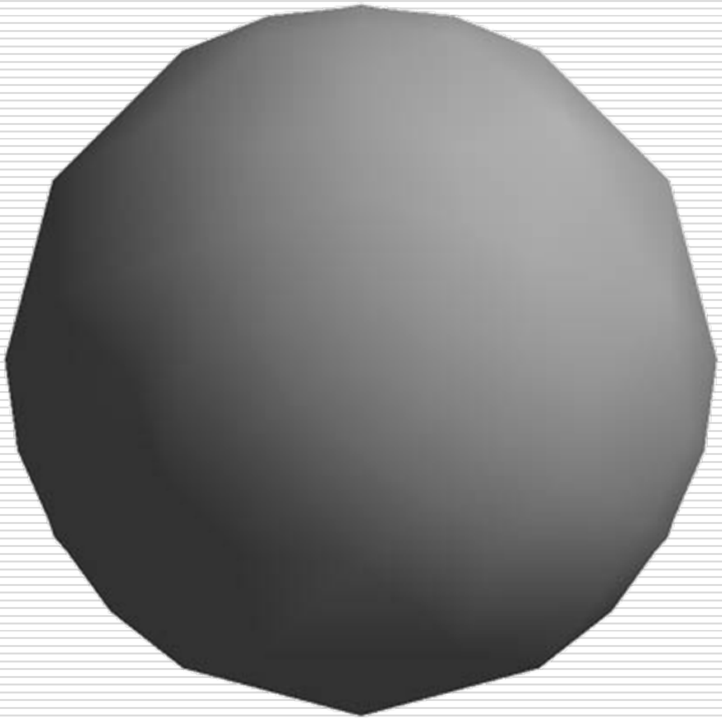


smooth surface



rough surface

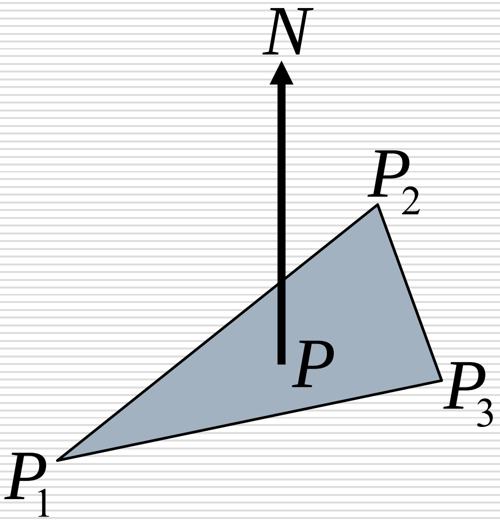
What is Normal?



Recall: Normal for Triangle

- Plane $N \bullet (P - P_1) = 0$

$$N = P_1P_2 \times P_1P_3$$
$$= (P_3 - P_1) \times (P_2 - P_1)$$

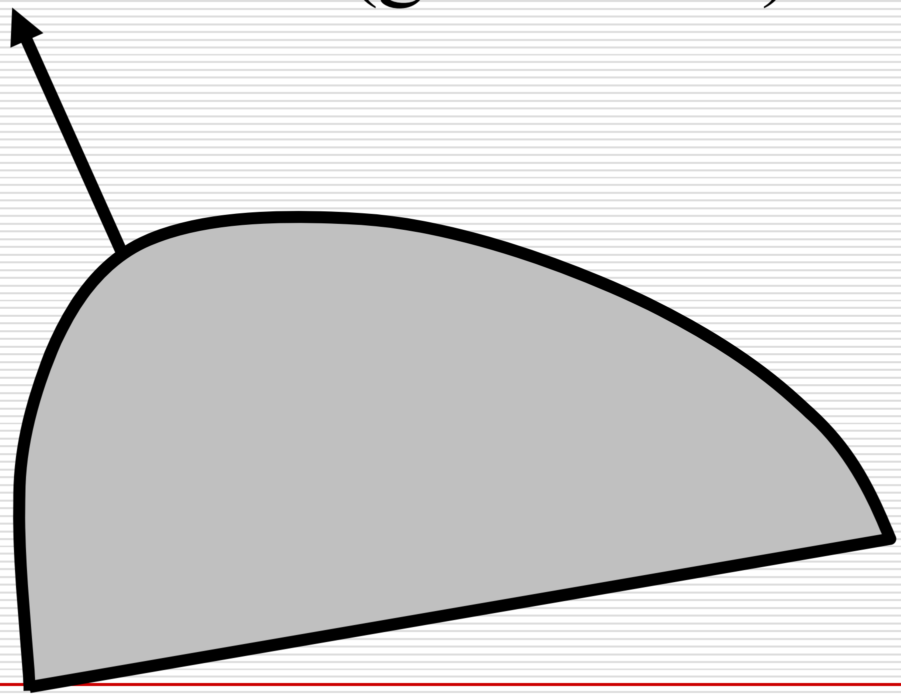


- Normalize $N \leftarrow N / |N|$

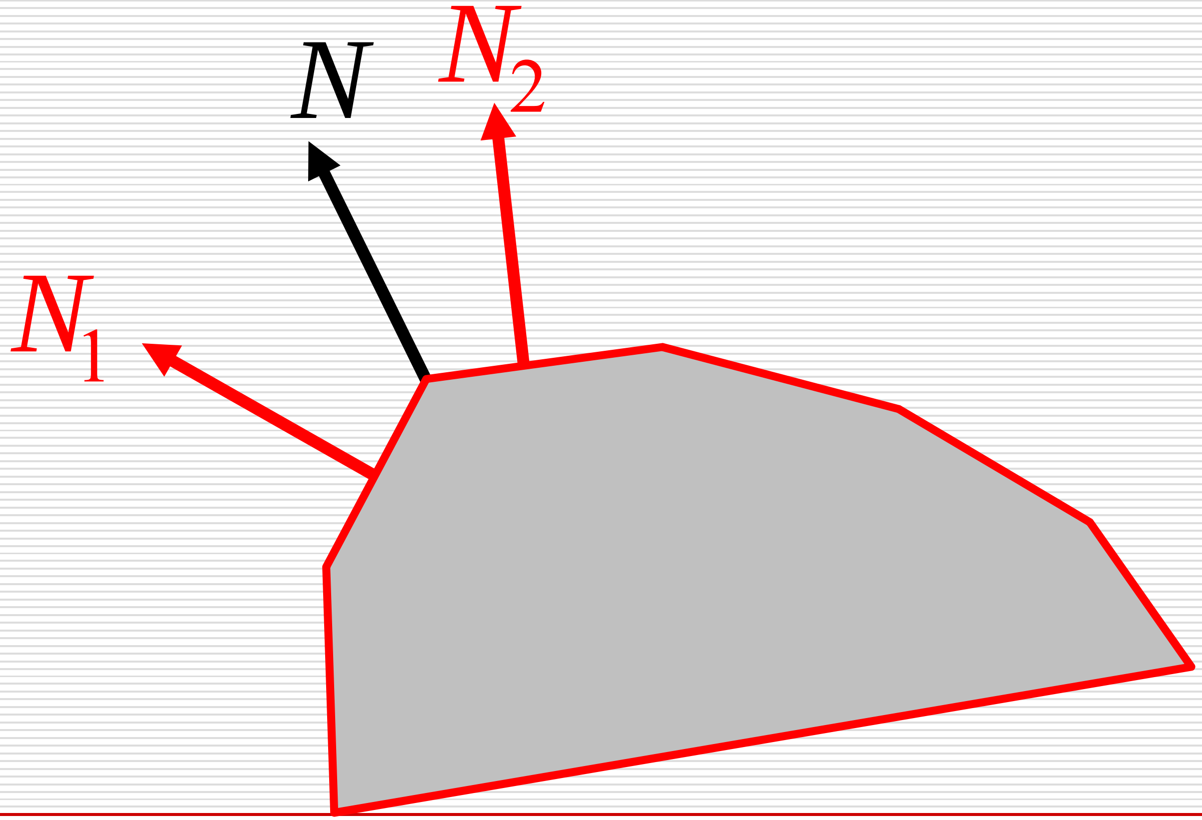
- Note that
  - right-hand rule determines outward face

Using Average Normals

N = true (geometric) normal

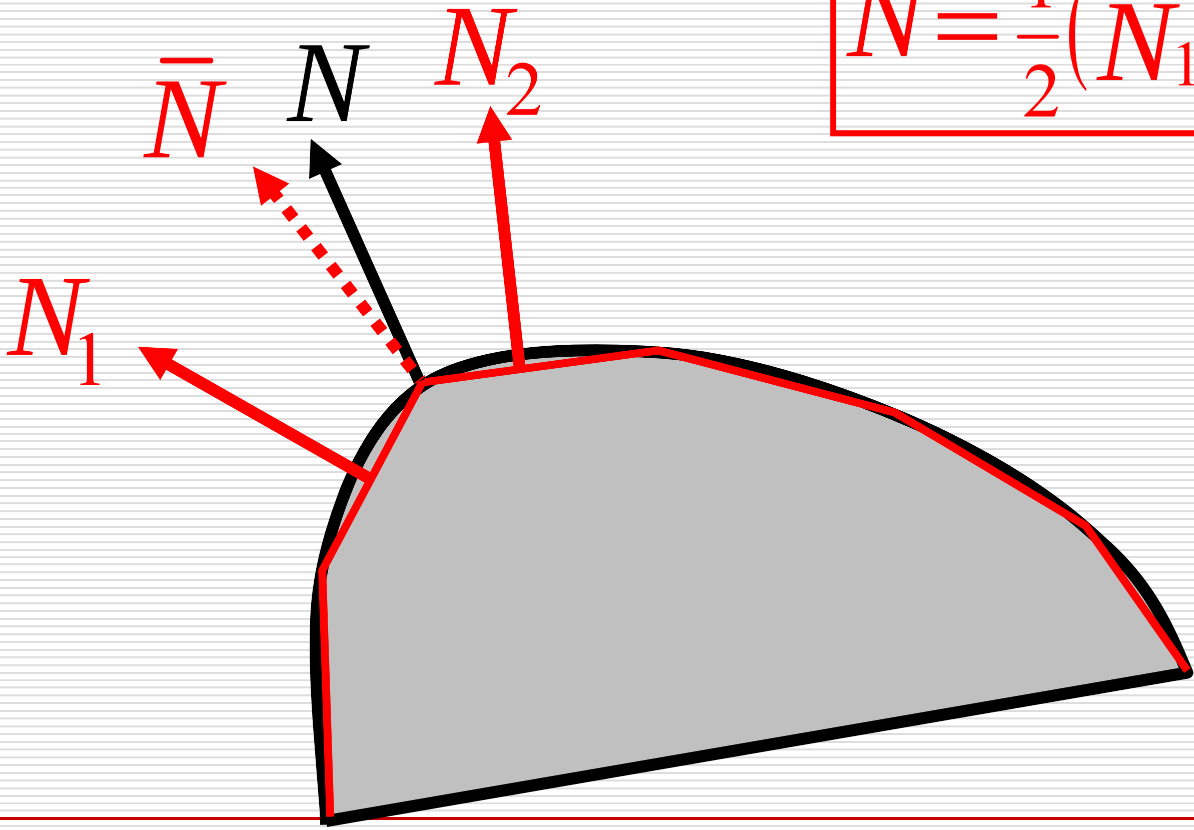


Using Average Normals



Using Average Normals

$$\bar{N} = \frac{1}{2}(N_1 + N_2)$$

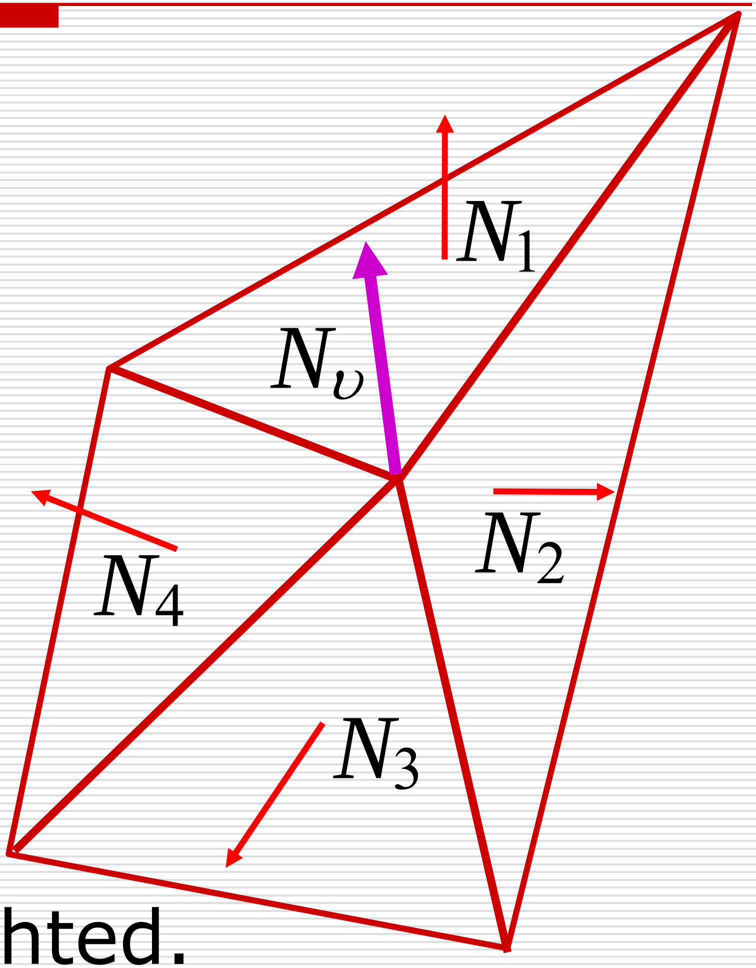


Using Average Normals

$$N_\upsilon = \frac{(N_1 + N_2 + N_3 + N_4)}{\|N_1 + N_2 + N_3 + N_4\|}$$

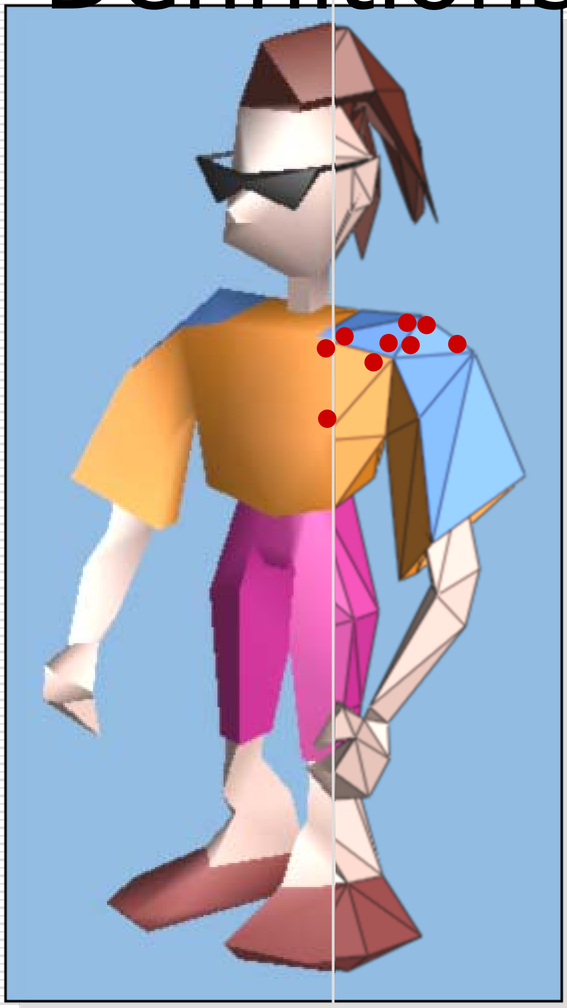
More generally,

$$N_\upsilon = \frac{\sum_{i=1}^{n} N_i}{\left| \sum_{i=1}^{n} N_i \right|}$$



It can also be area-weighted.

Definitions of Triangle Meshes



$\{f_1\} : \{ v_1 , v_2 , v_3 \}$
$\{f_2\} : \{ v_3 , v_2 , v_4 \}$
…

connectivity

$\{v_1\} : (x,y,z)$
$\{v_2\} : (x,y,z)$
…

geometry

$\{f_1\}$: *"skin material"*
$\{f_2\}$: *"brown hair"*
…

face attributes

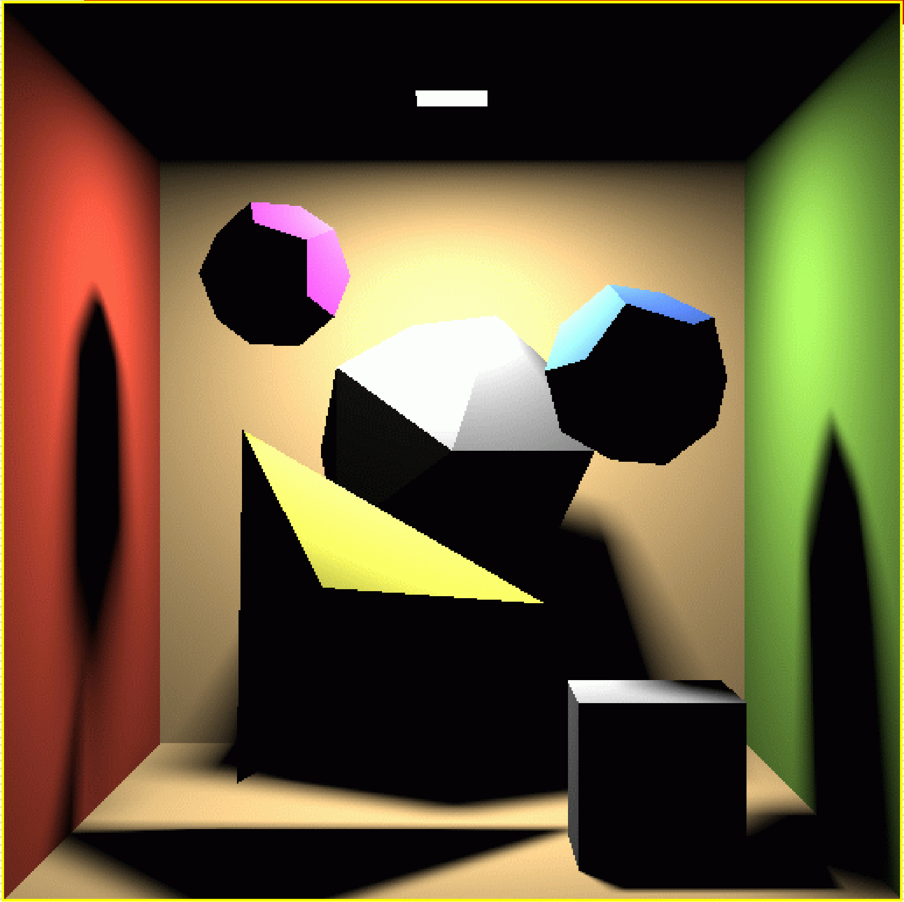
$\{v_2,f_1\} : (n_x,n_y,n_z)\ (u,v)$
$\{v_2,f_2\} : (n_x,n_y,n_z)\ (u,v)$
…

corner attributes

Illumination (Shading) Models

- Interaction between light sources and objects in scene that results in perception of intensity and color at eye
- Local vs. global models
  - Local: perception of a particular primitive only depends on light sources **directly** affecting that one primitive
    - Geometry
    - Material properties
    - Shadows cast (global?)
  - Global: also take into account **indirect** effects on light of other objects in the scene
    - Light reflected/refracted
    - Indirect lighting

Local vs. Global Models

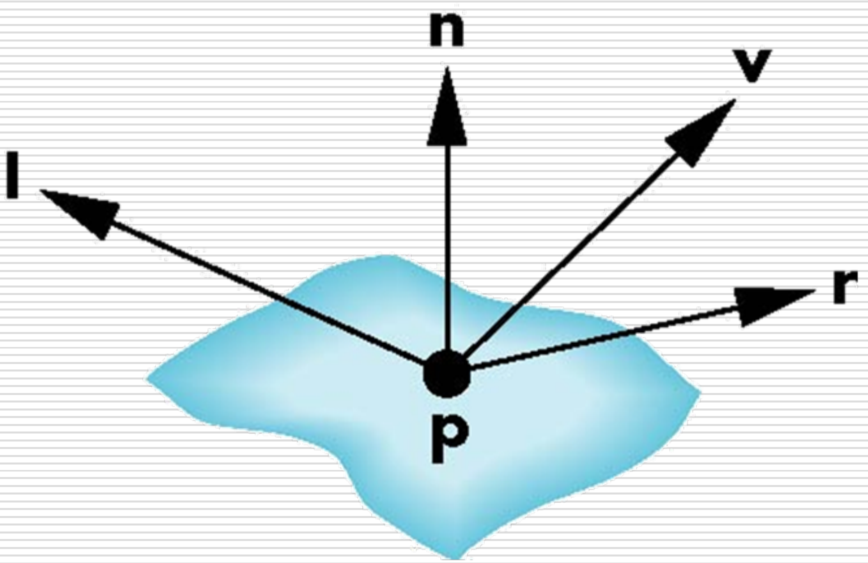


direct lighting

indirect lighting

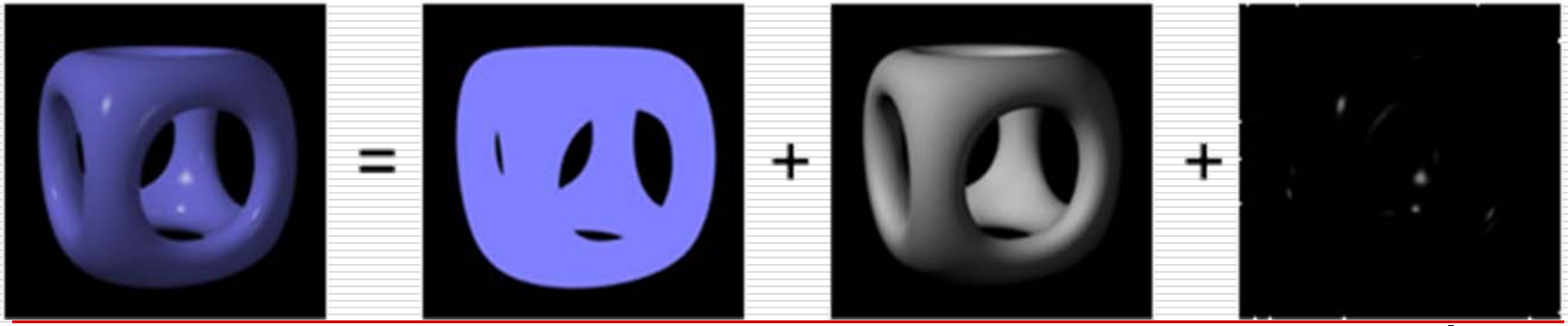
The Phong Illumination Model

- A simple model that can be computed rapidly
- Has three components
  - Diffuse
  - Specular
  - Ambient
- Uses four vectors
  - To source
  - To viewer
  - Normal
  - Perfect reflector



Basics of Local Shading

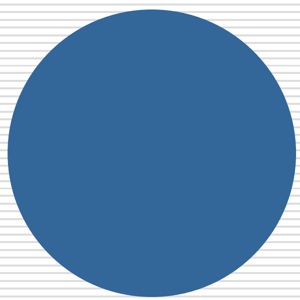
- Diffuse reflection
  - light goes everywhere; colored by object color
- Specular reflection
  - happens only near mirror configuration; usually white
- Ambient reflection
  - constant accounted for other source of illumination

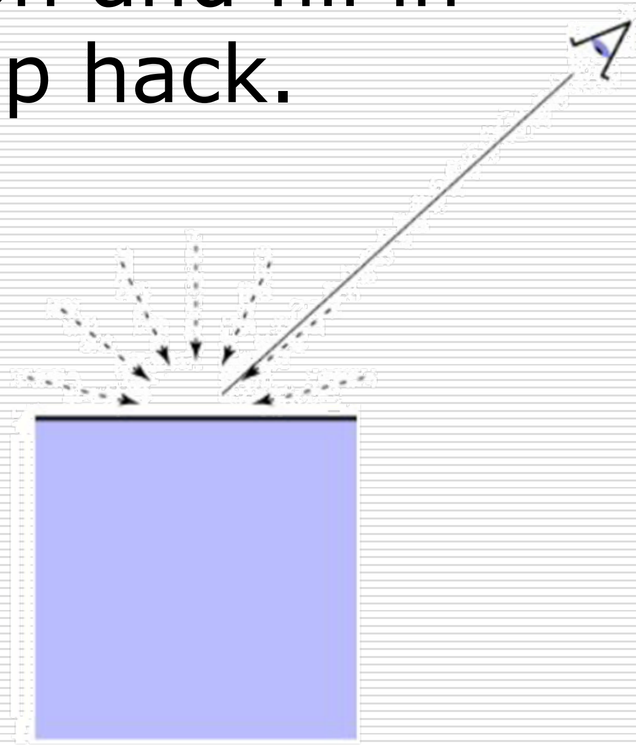


ambient diffuse specular

Ambient Shading

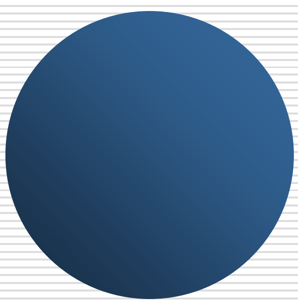
- add constant color to account for disregarded illumination and fill in black shadows; a cheap hack.

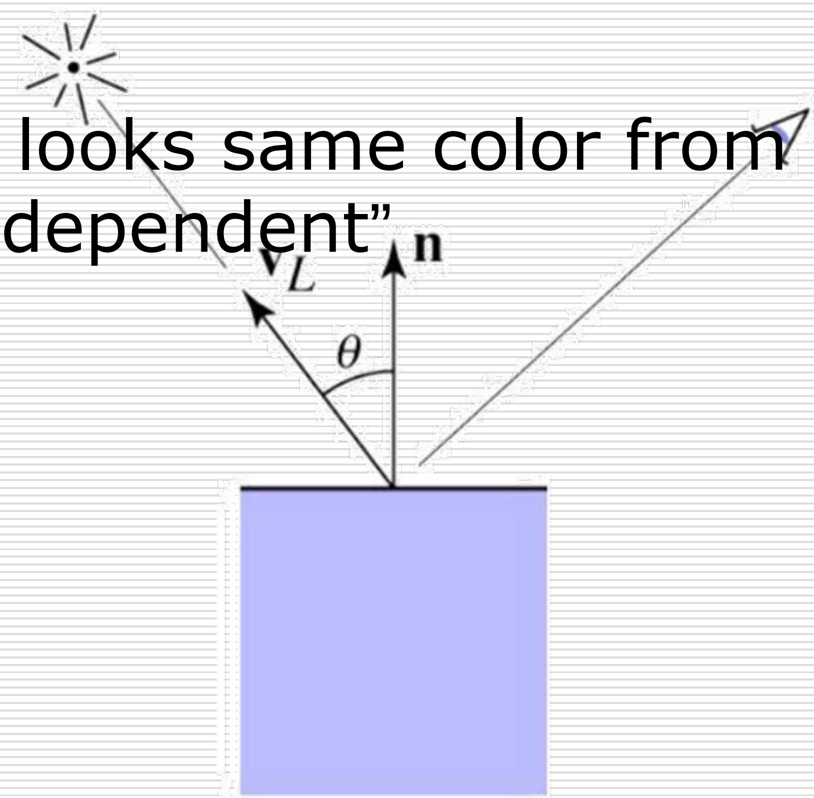




Diffuse Shading

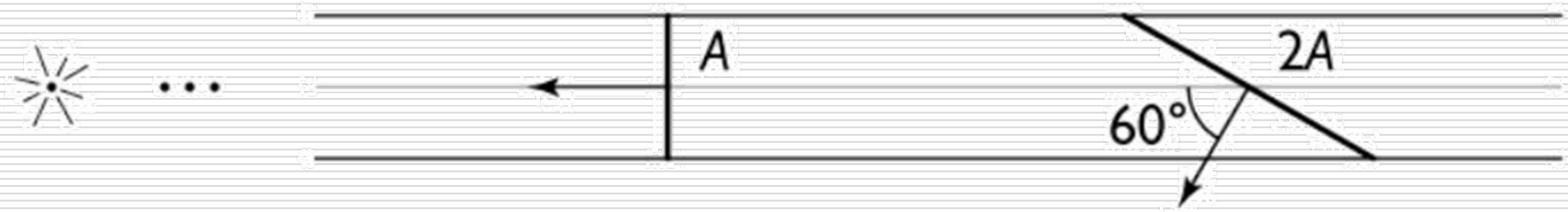
- Assume light reflects equally in all directions
  - Therefore surface looks same color from all views; “view independent”





Diffuse shading

- Illumination on an oblique surface is less than on a normal one (Lambertian cosine law)

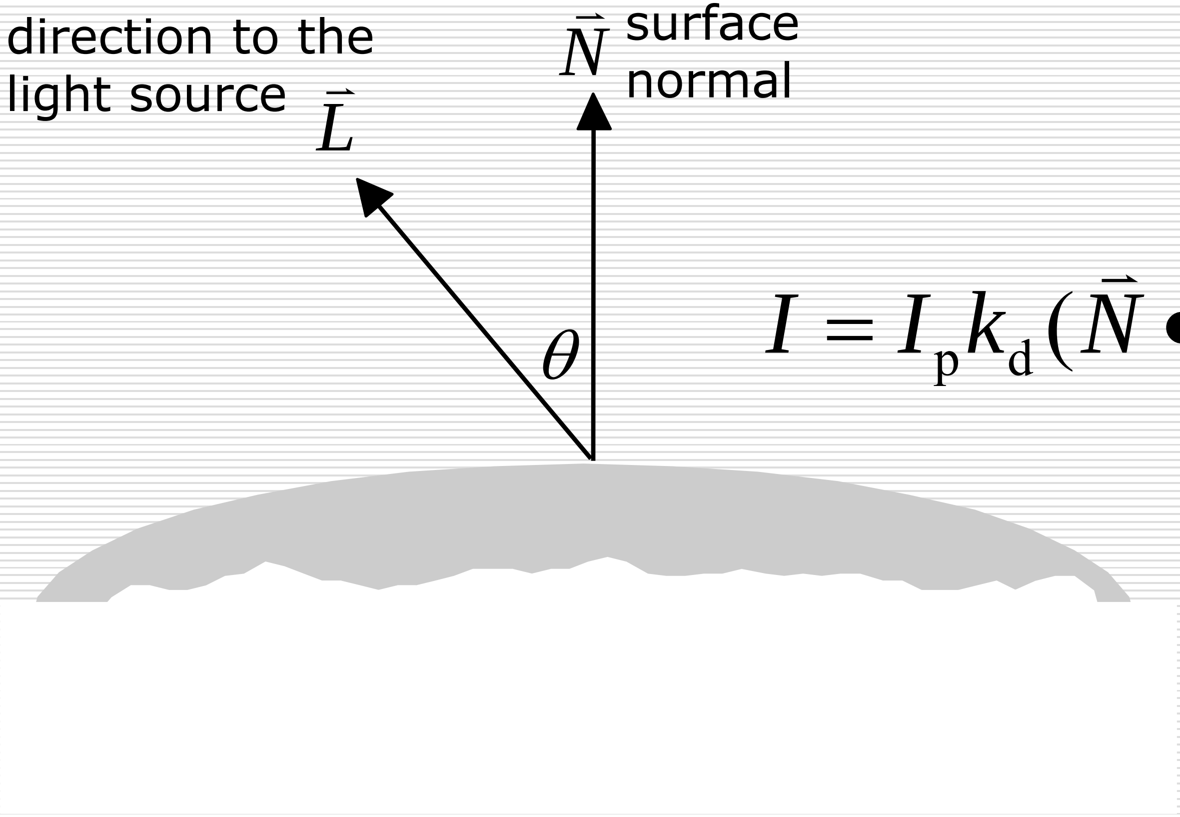


  - Generally, illumination falls off as cosθ

Illumination Models

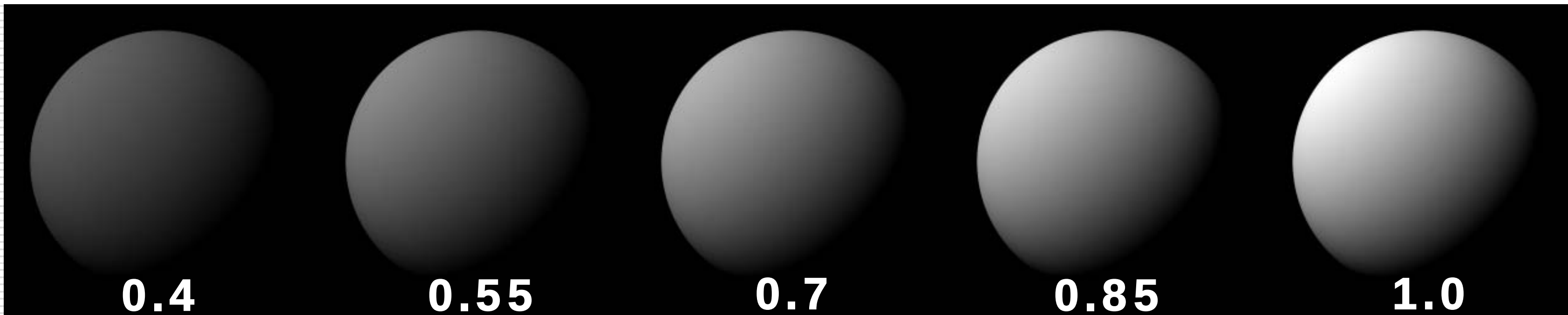
- Ambient Light: $I = I_a k_a$
  - I_a: intensity of the ambient light
  - k_a: ambient-reflection coefficient: 0 ~ 1
- Diffuse Reflection: $I = I_p k_d \cos\theta$
  - I_p: point light source's intensity
  - k_d: diffuse-reflection coefficient: 0 ~ 1
  - θ: angle: 0° ~ 90°

Diffuse Reflection

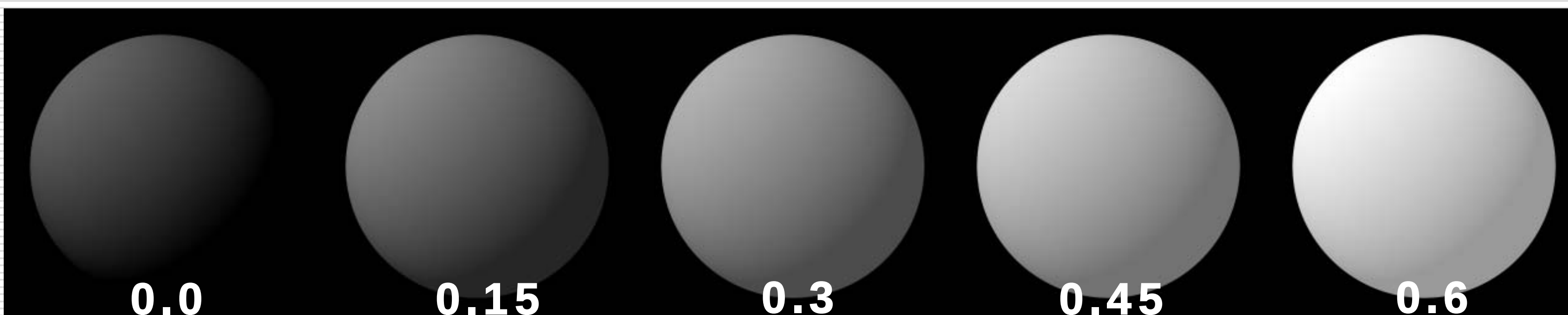


$$I = I_{\mathrm{p}} k_{\mathrm{d}} (\vec{N} \bullet \vec{L})$$

Examples



diffuse-reflection model with different k_{d}



ambient and diffuse-reflection model with different k_{a}
and $I_{\mathrm{a}} = I_{\mathrm{p}} = 1.0, k_{\mathrm{d}} = 0.4$

Light-Source Attenuation

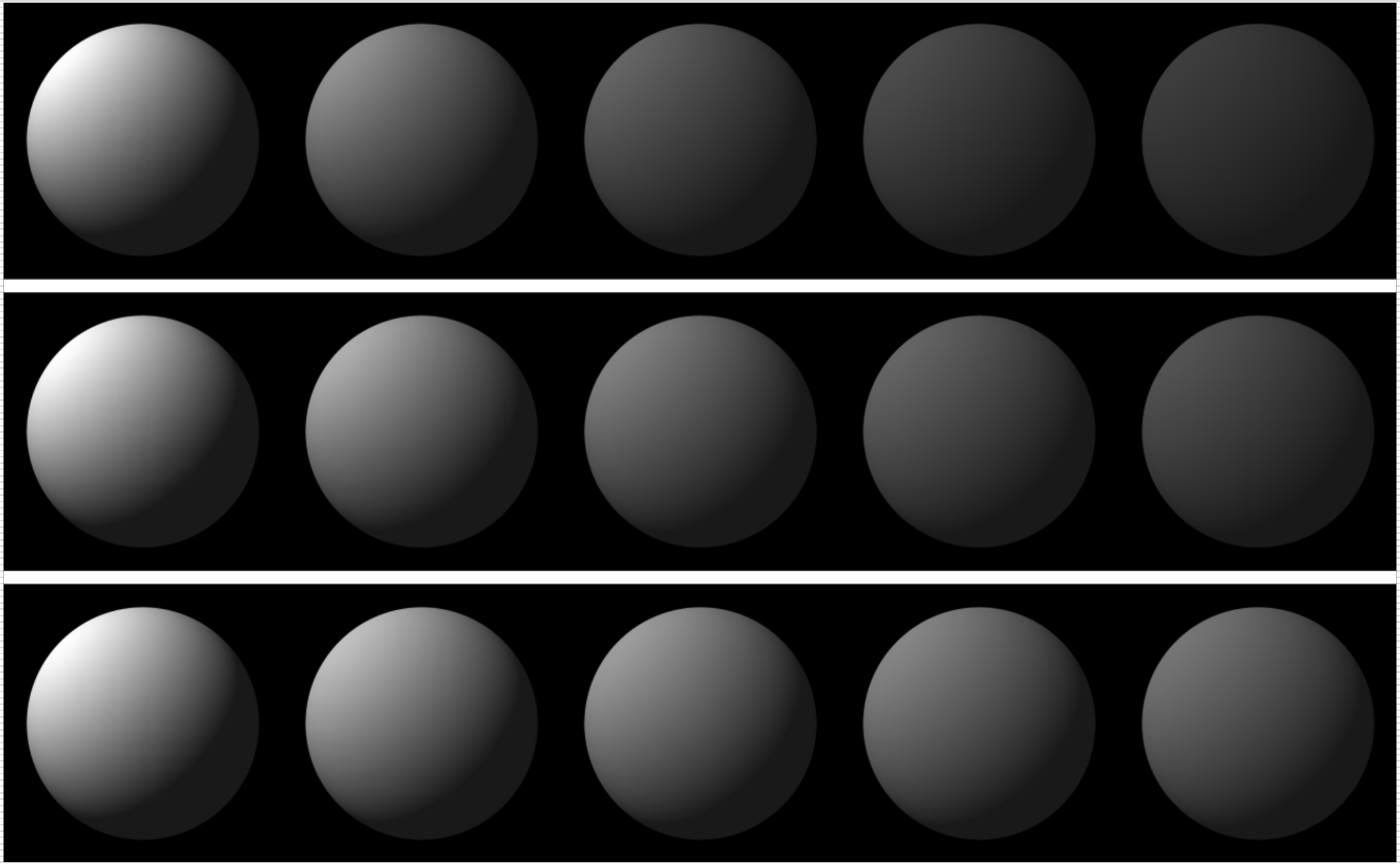
- $I = I_a k_a + f_{att} I_p k_d (\vec{N} \bullet \vec{L})$
  - f_{att} : light-source attenuation factor
  - if the light is a point source

$$f_{att} = \frac{1}{d_L^2}$$

  - where d_L is the distance the light travels from the point source to the surface

$$f_{att} = \min\left(\frac{1}{c_1 + c_2 d_L + c_3 d_L^2}, 1\right)$$

Examples



$$\frac{1}{d_L^2}$$

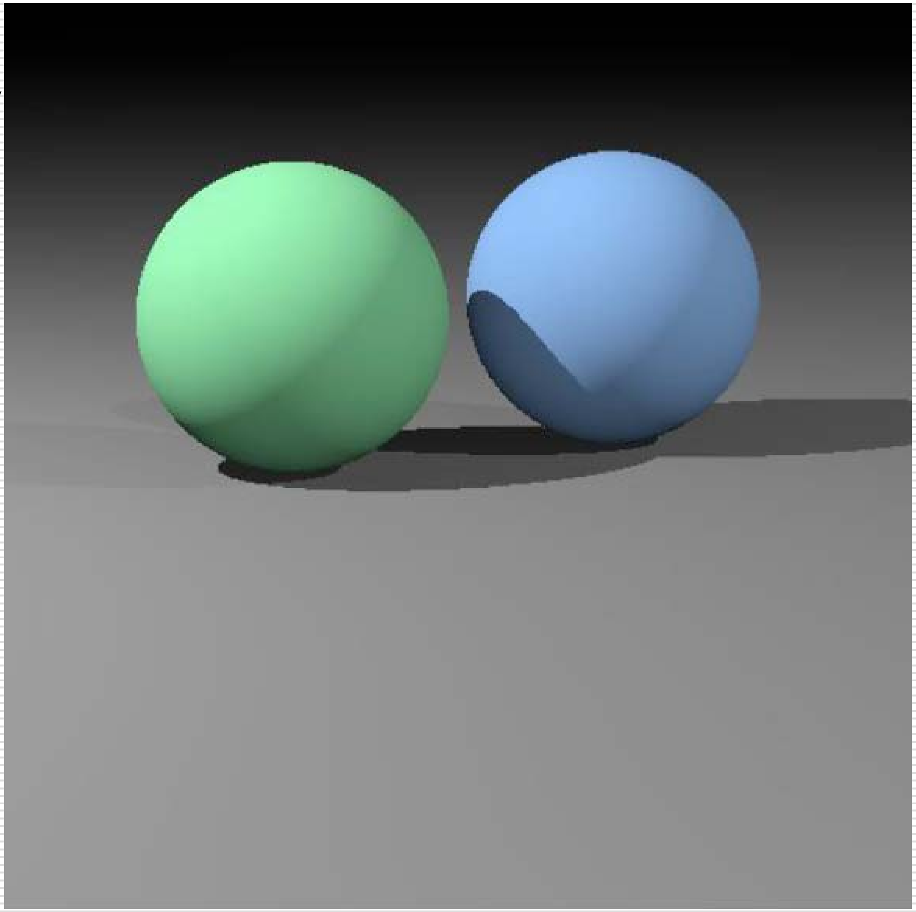
$$\frac{1}{d_L}$$

Colored Lights and Surfaces

- If an object's **diffuse color** is $O_{\mathrm{d}} = (O_{\mathrm{dR}}, O_{\mathrm{dG}}, O_{\mathrm{dB}})$ then $I = (I_{\mathrm{R}}, I_{\mathrm{G}}, I_{\mathrm{B}})$ where for the red component
$$I_{\mathrm{R}} = I_{\mathrm{aR}} k_{\mathrm{a}} O_{\mathrm{dR}} + f_{\mathrm{att}} I_{\mathrm{pR}} k_{\mathrm{d}} O_{\mathrm{dR}} (\vec{N} \bullet \vec{L})$$
however, it should be
$$I_{\lambda} = I_{\mathrm{a}\lambda} k_{\mathrm{a}} O_{\mathrm{d}\lambda} + f_{\mathrm{att}} I_{\mathrm{p}\lambda} k_{\mathrm{d}} O_{\mathrm{d}\lambda} (\vec{N} \bullet \vec{L})$$
where λ is the **wavelength**

Diffuse Shading

- For color objects, apply the formula for each color channel separately

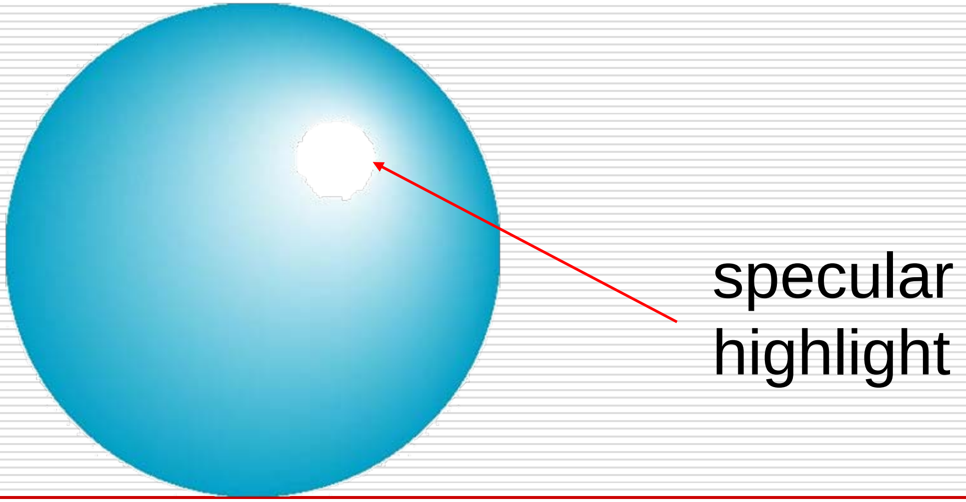


Specular Shading

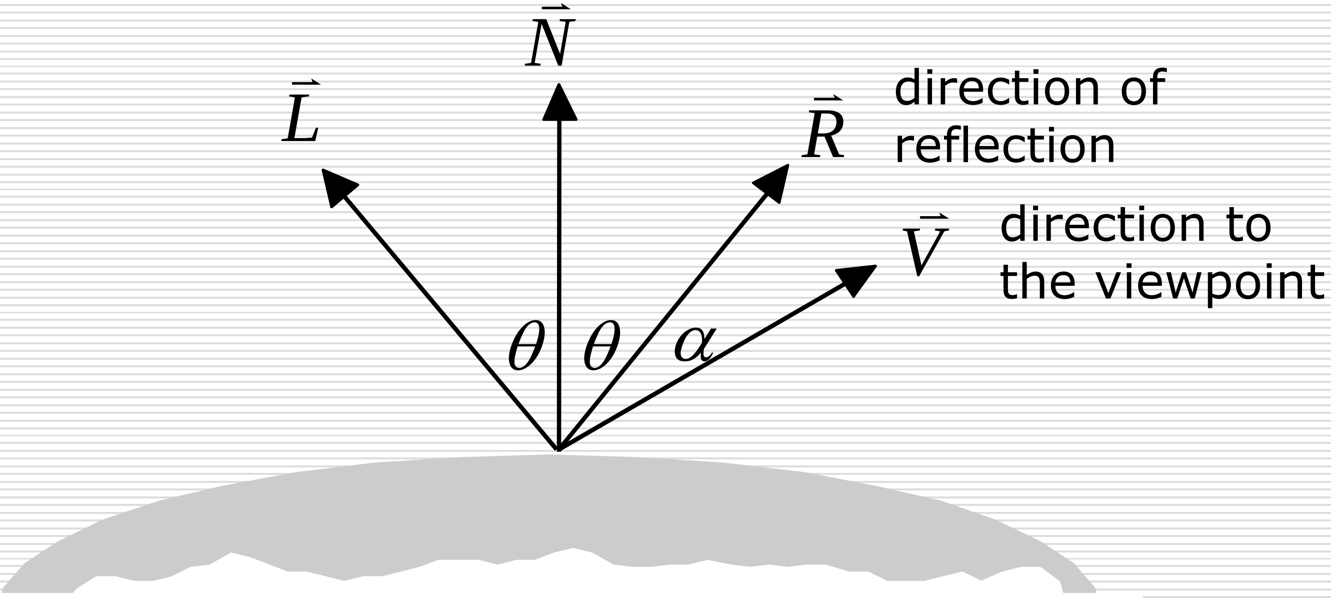
- Some surfaces have highlights, mirror like reflection; view direction dependent; especially for smooth shinny surfaces

Specular Surfaces

- Most surfaces are neither ideal diffusers nor perfectly specular (ideal refectors)
- Smooth surfaces show specular highlights due to incoming light being reflected in directions concentrated close to the direction of a perfect reflection



Specular Reflection



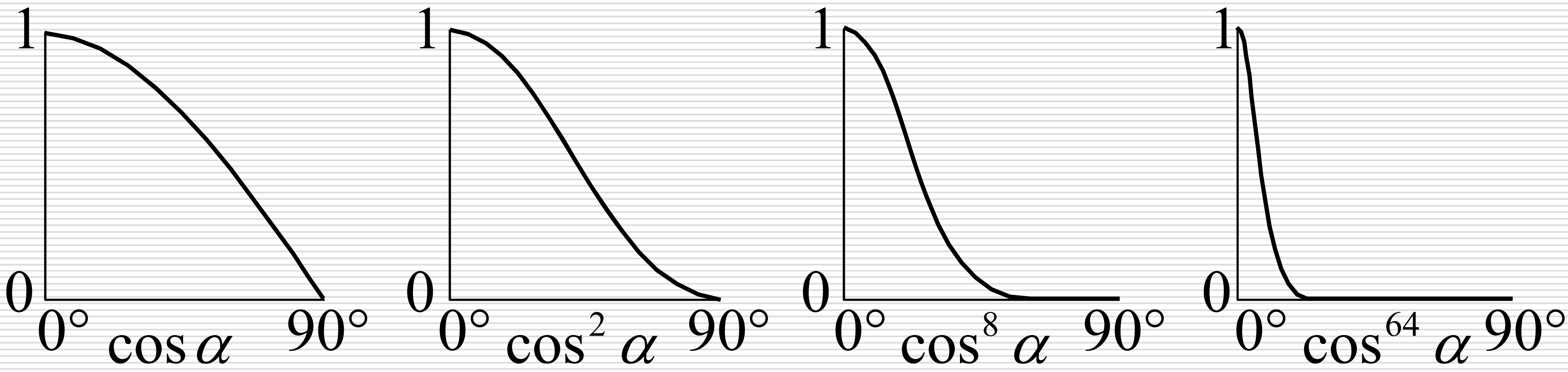
The Phong Illumination Model

- $I_\lambda = I_{a\lambda} k_a O_{d\lambda} + f_{att} I_{p\lambda} [k_d O_{d\lambda} \cos\theta + W(\theta) \cos^n \alpha]$
  - $W(\theta) = k_s$: specular-reflection coefficient:0~1
- so, the Eq. can be rewritten as
  $I_\lambda = I_{a\lambda} k_a O_{d\lambda} + f_{att} I_{p\lambda} [k_d O_{d\lambda} (\vec{N} \bullet \vec{L}) + k_s (\vec{R} \bullet \vec{V})^n]$
- consider the object's **specular color**

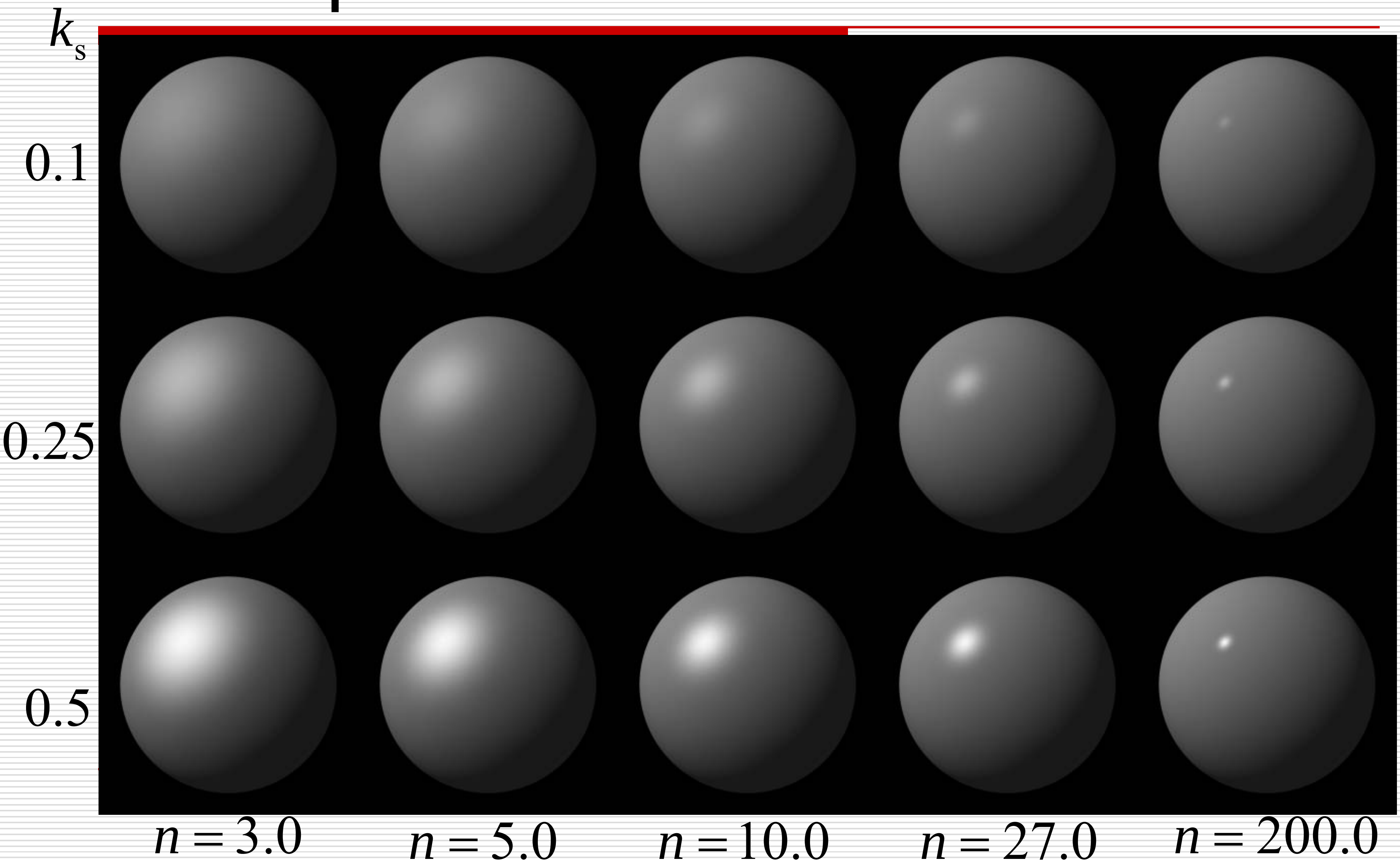
$$I_\lambda = I_{a\lambda} k_a O_{d\lambda} + f_{att} I_{p\lambda} [k_d O_{d\lambda} (\vec{N} \bullet \vec{L}) + k_s O_{s\lambda} (\vec{R} \bullet \vec{V})^n]$$

  - $O_{s\lambda}$: specular color

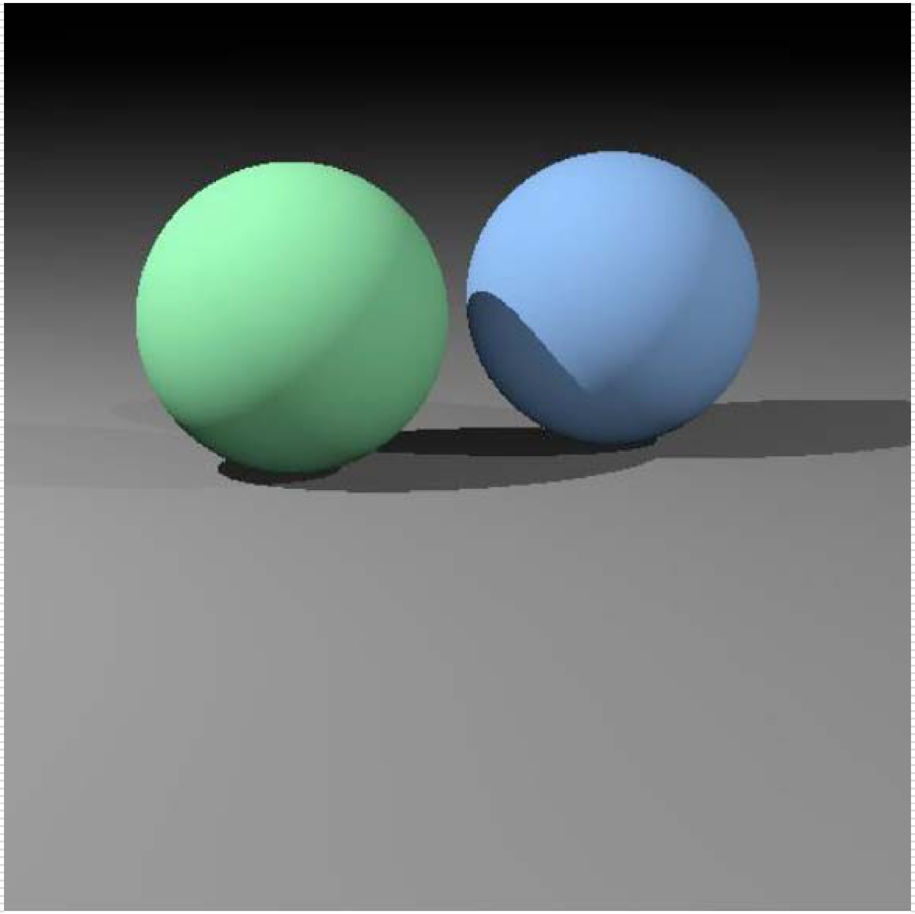
The Phong Illumination Model

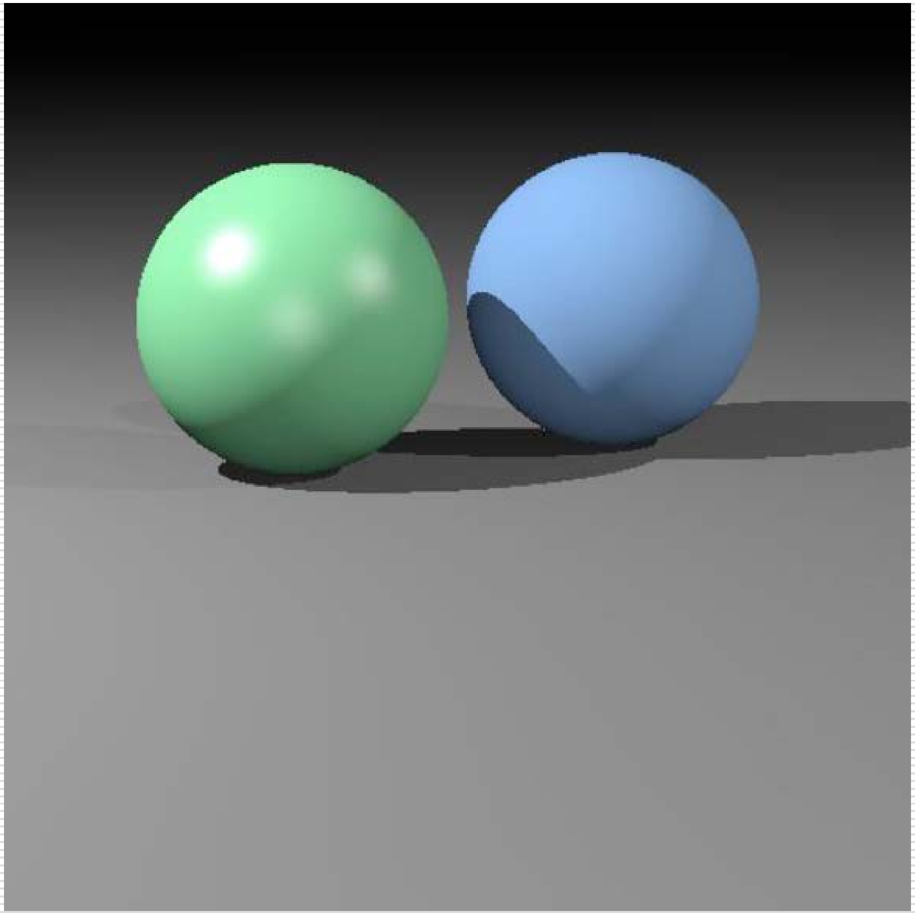


Examples



Specular Shading



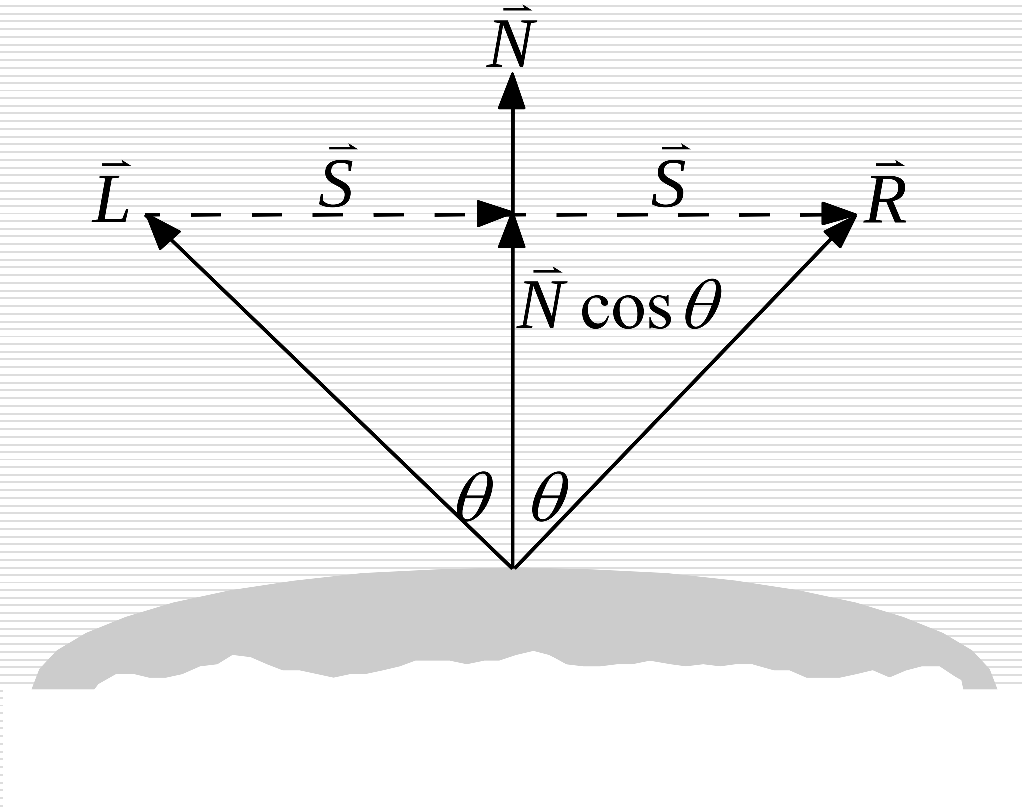


diffuse

diffuse + specular

Calculating the Reflection Vector

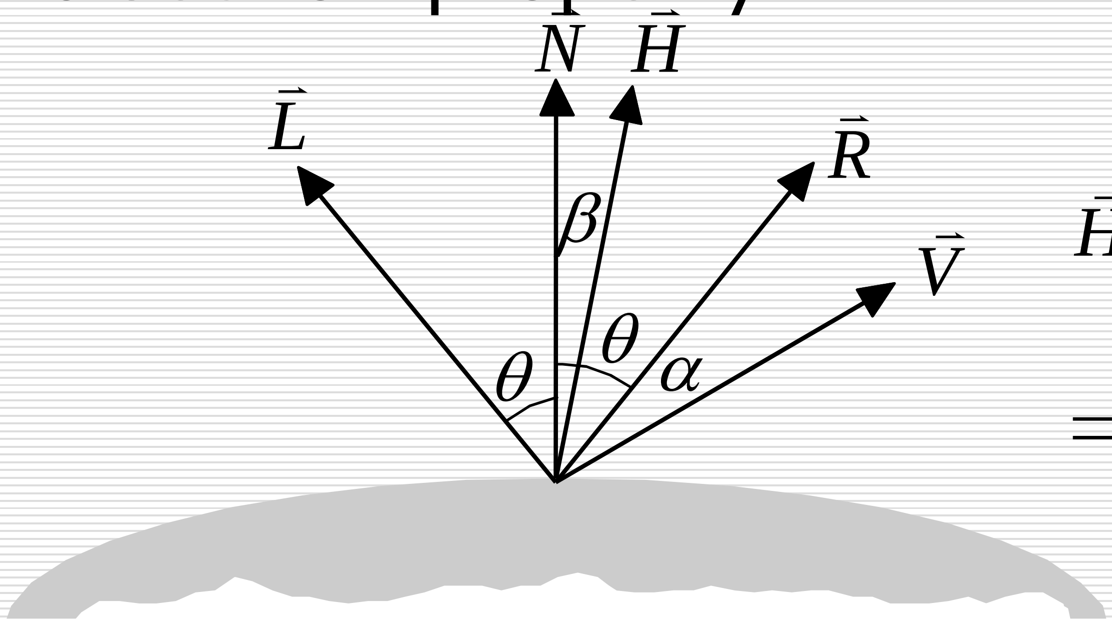
- Fall off gradually from the perfect reflection direction



$$
\begin{aligned}
\vec{R} &= \vec{N}\cos\theta + \vec{S} \\
&= \vec{N}\cos\theta + \vec{N}\cos\theta - \vec{L} \\
&= 2\vec{N}\cos\theta - \vec{L} \\
&= 2\vec{N}(\vec{N} \bullet \vec{L}) - \vec{L}
\end{aligned}
$$

The Halfway Vector (Blinn-Phong)

- Rather than computing reflection directly; just compare to normal bisection property.



$$\vec{H} = \frac{\vec{L} + \vec{V}}{\left| \vec{L} + \vec{V} \right|}$$

$$\Rightarrow \cos\alpha \approx \vec{N} \bullet \vec{H}$$

Multiple Light Sources

- If there are m light sources, then

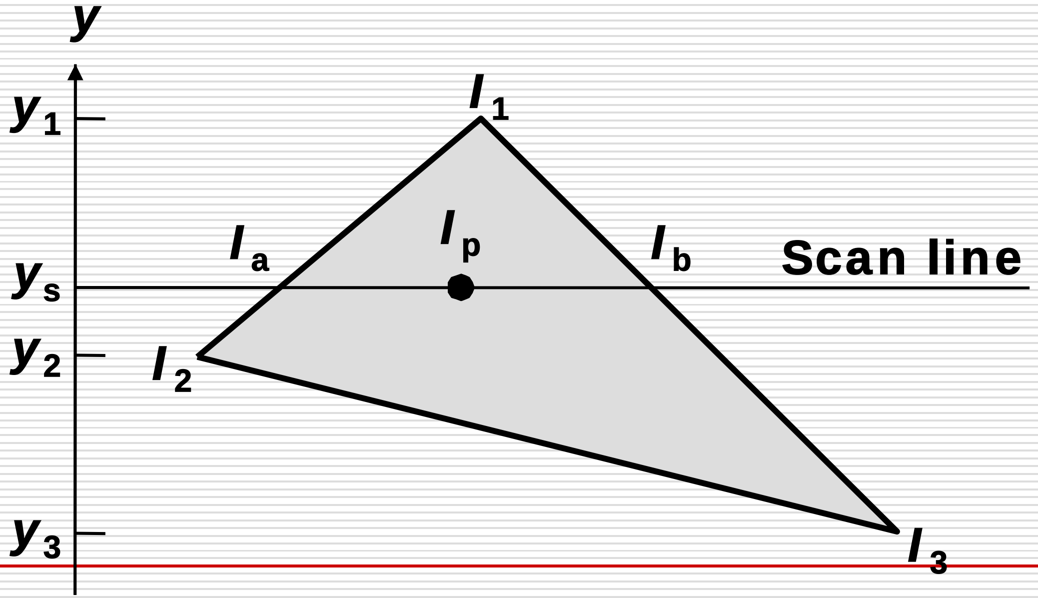
$$I_\lambda = I_{a\lambda} k_a O_{d\lambda} + \sum_{1 \le i \le m} f_{att_i} I_{p\lambda_i} [k_d O_{d\lambda} (\vec{N} \bullet \vec{L}_i) + k_s O_{s\lambda} \cos^n \alpha_i]$$

$$\approx I_{a\lambda} k_a O_{d\lambda} + \sum_{1 \le i \le m} f_{att_i} I_{p\lambda_i} [k_d O_{d\lambda} (\vec{N} \bullet \vec{L}_i) + k_s O_{s\lambda} (\vec{R}_i \bullet \vec{V})^n]$$

$$\approx I_{a\lambda} k_a O_{d\lambda} + \sum_{1 \le i \le m} f_{att_i} I_{p\lambda_i} [k_d O_{d\lambda} (\vec{N} \bullet \vec{L}_i) + k_s O_{s\lambda} (\vec{N} \bullet \vec{H}_i)^n]$$

Computing Lighting at Each Pixel

- Most accurate approach: Compute component illumination at each pixel with individual positions, light directions, and viewing directions
- But this could be expensive...



Shading Models for Polygons

- Flat Shading
  - Faceted Shading
  - Constant Shading
- Gouraud Shading
  - Intensity Interpolation Shading
  - Color Interpolation Shading
- Phong Shading
  - Normal-Vector Interpolation Shading

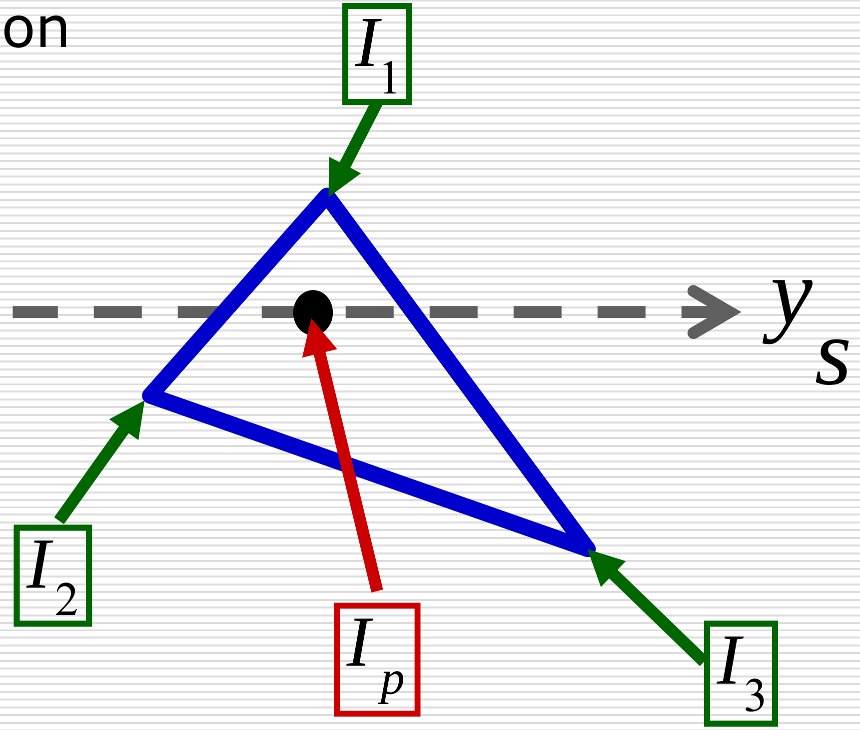
Flat Shading

- Assumptions
  - The light source is at infinity
  - The viewer is at infinity
  - The polygon represents the actual surface being modeled and is not an approximation to a curved surface

Flat Shading

- Compute constant shading function, over each polygon
- Same normal and light vector across whole polygon
- Constant shading for polygon

$$I_p = I$$

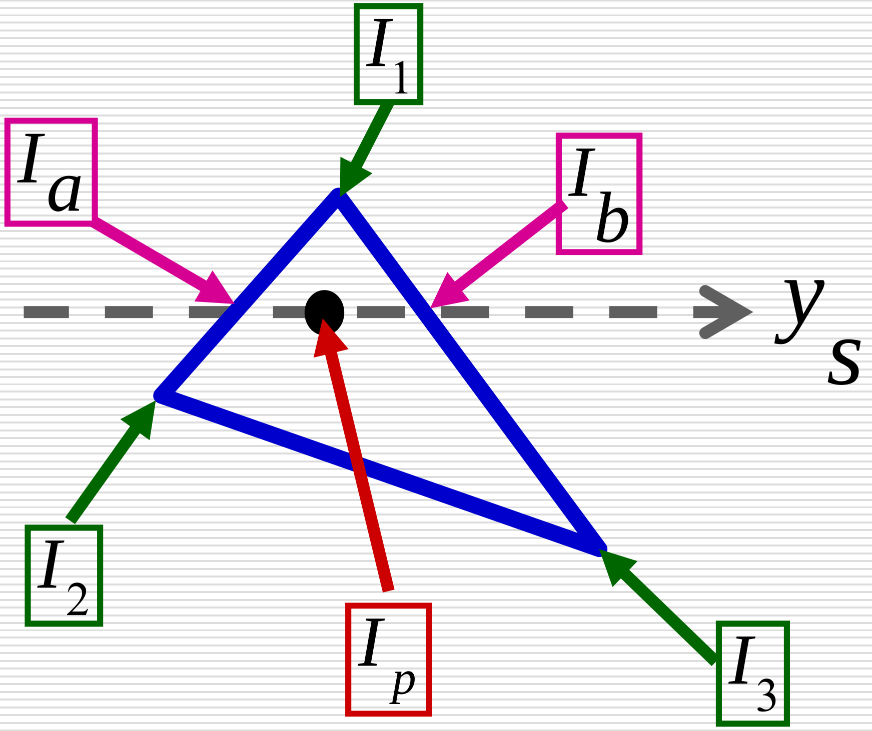


Intensity Interpolation (Gouraud)

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

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

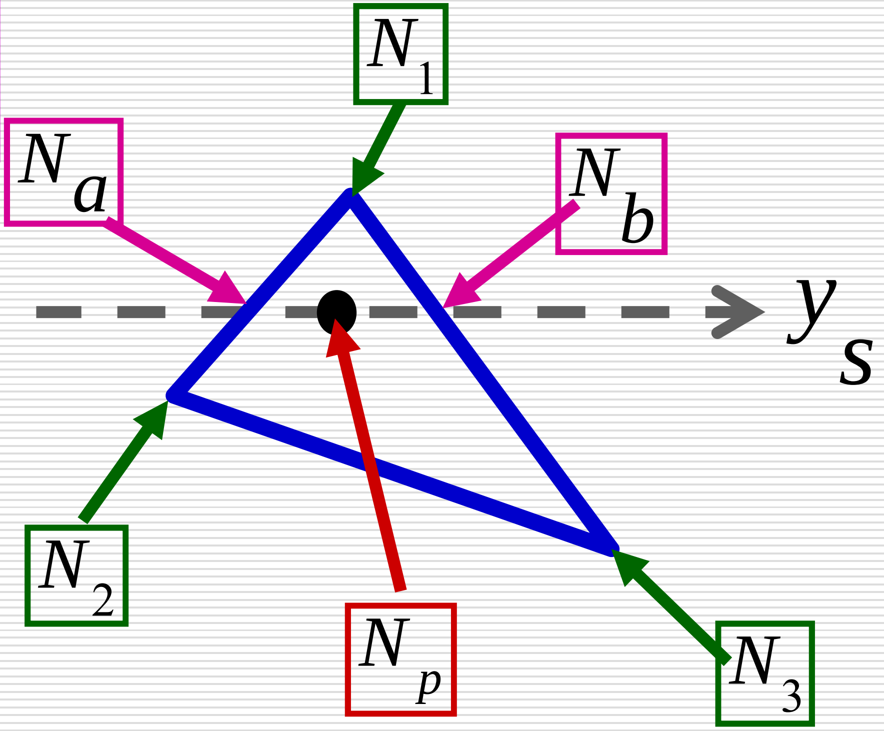
$$I_p = I_a \frac{x_b - x_p}{x_b - x_a} + I_b \frac{x_p - x_a}{x_b - x_a}$$



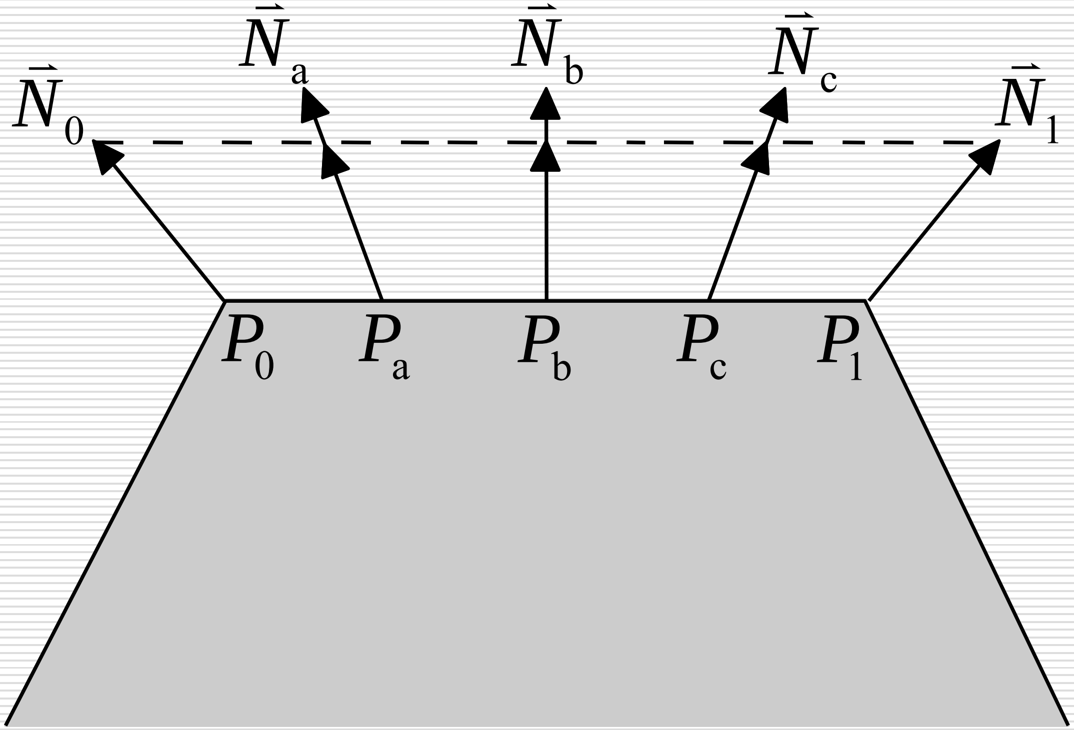
Normal Interpolation (Phong)

$$N_a = N_1 \frac{y_s - y_2}{y_1 - y_2} + N_2 \frac{y_1 - y_s}{y_1 - y_2}$$

$$N_b = N_1 \frac{y_s - y_3}{y_1 - y_3} + N_3 \frac{y_1 - y_s}{y_1 - y_3}$$



Normal Interpolation (Phong)



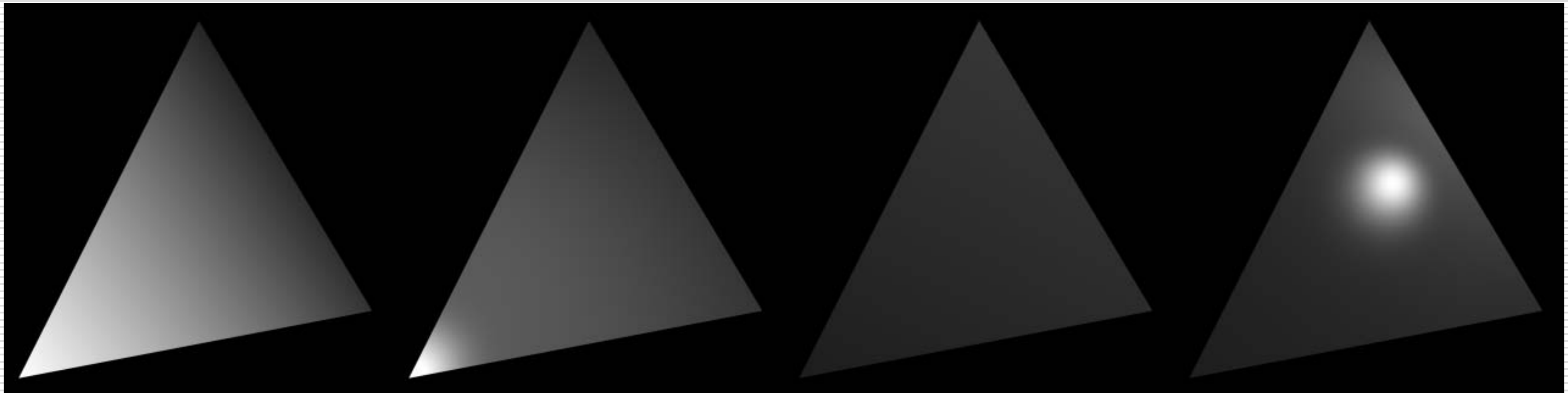
Normal Interpolation (Phong)

$$\tilde{N}_p = \frac{N_a}{\|N_a\|}\left[\frac{x_b - x_p}{x_b - x_a}\right] + \frac{N_b}{\|N_b\|}\left[\frac{x_p - x_a}{x_b - x_a}\right]$$

$$N_p = \frac{\tilde{N}_p}{\left\|\tilde{N}_p\right\|}$$

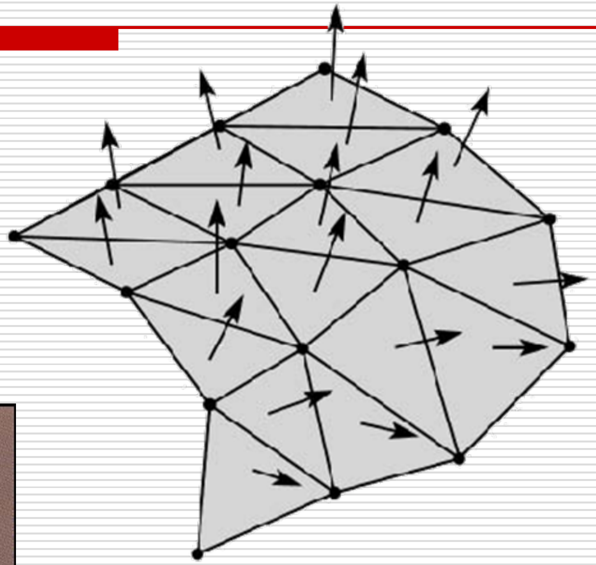
Normalizing makes this a unit vector

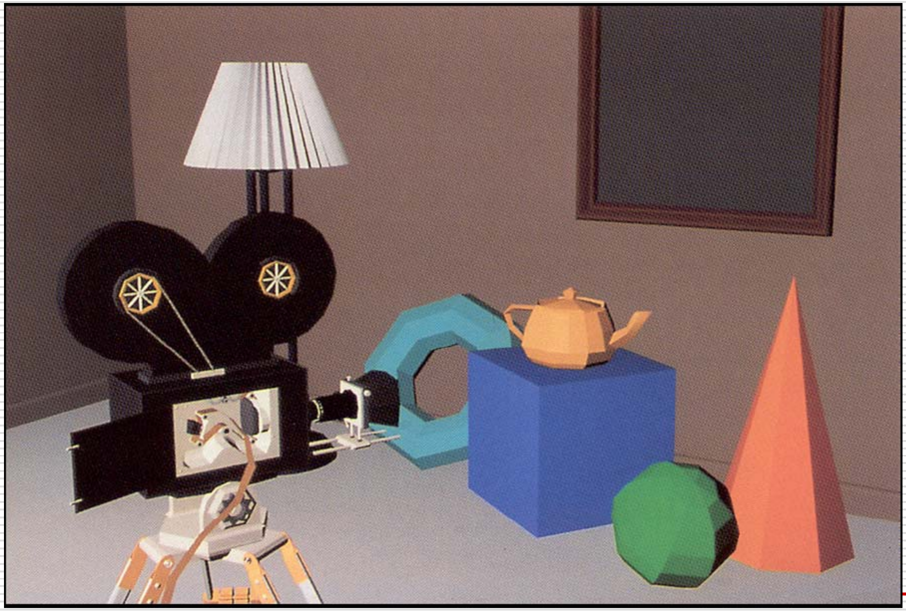
Gouraud v.s. Phong Shading



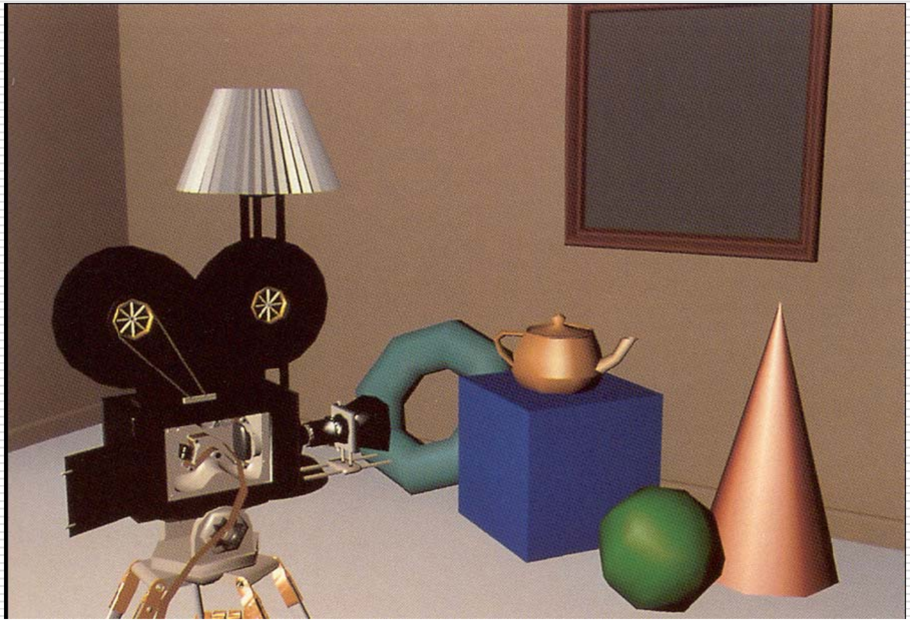
Gouraud Phong Gouraud Phong

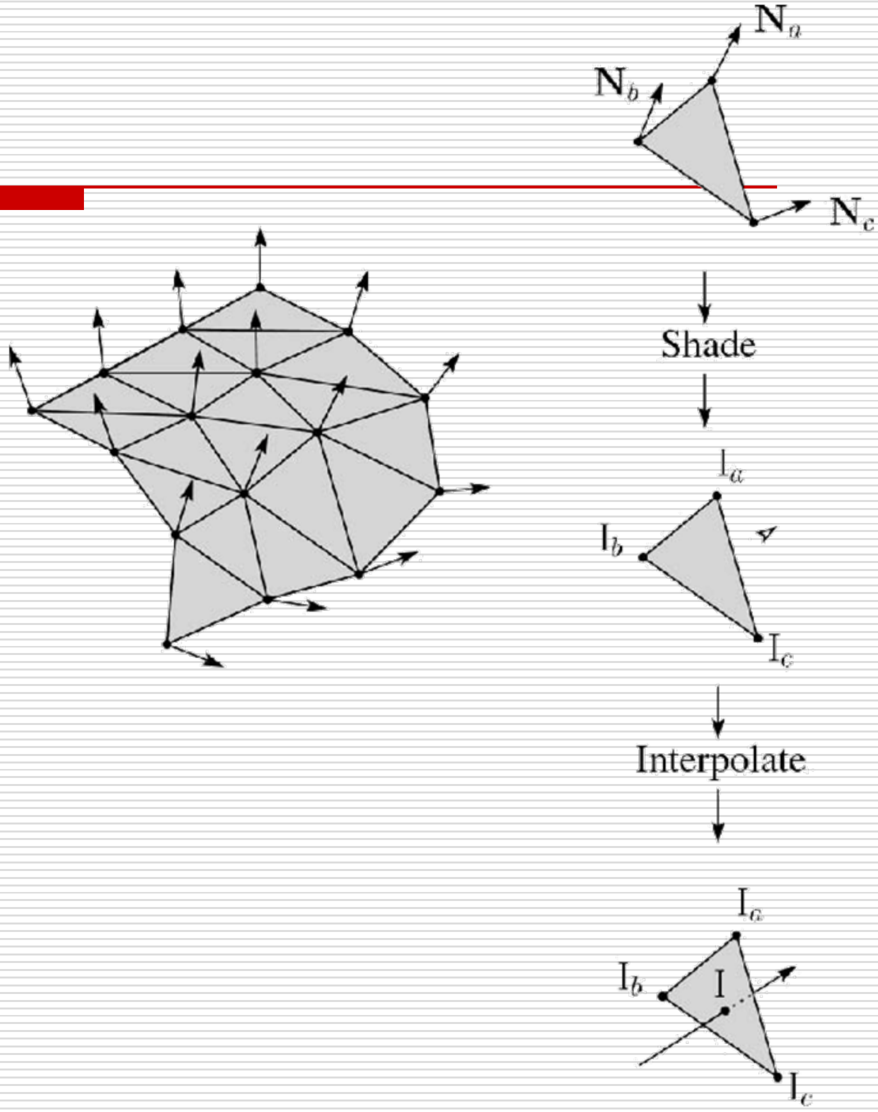
Flat Shading



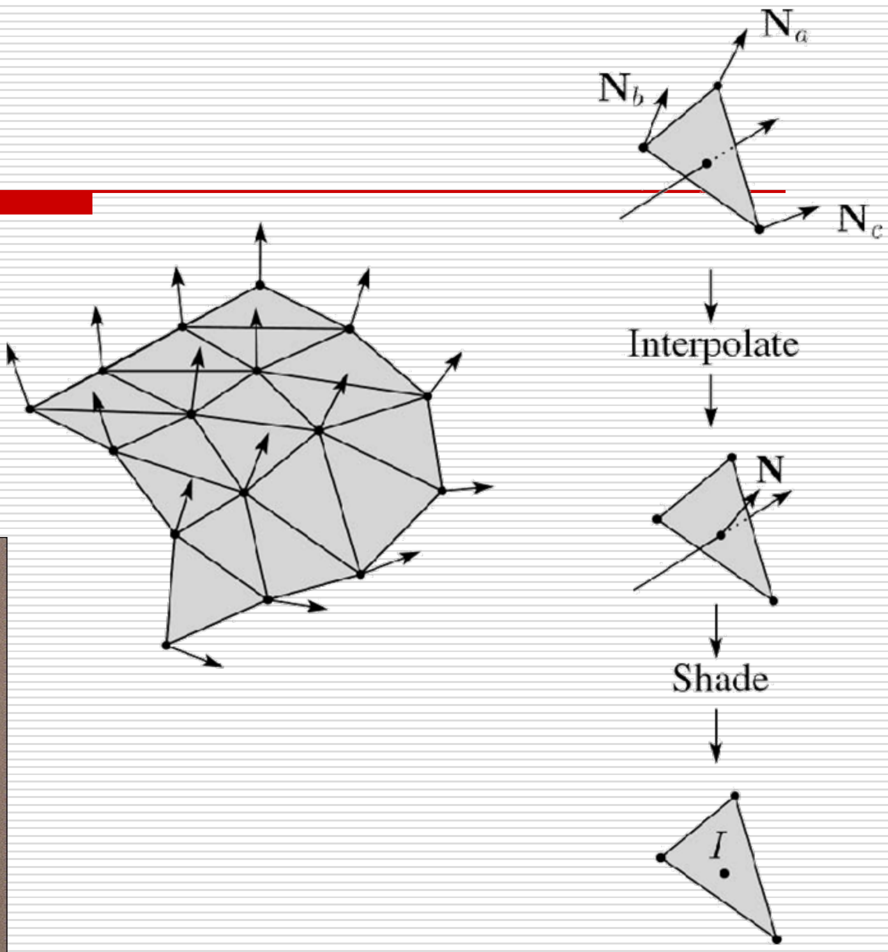


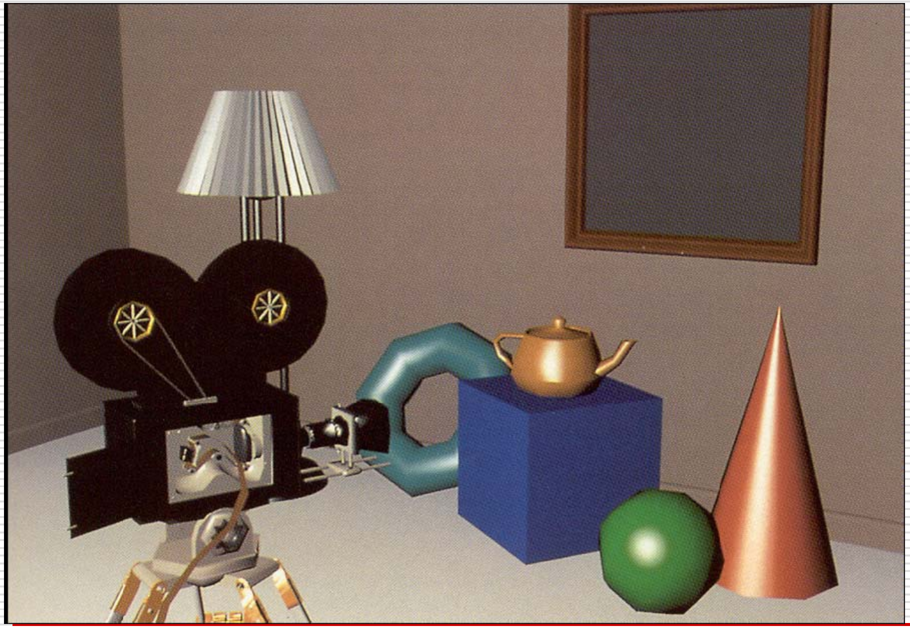
Gouraud Shading





Phong Shading



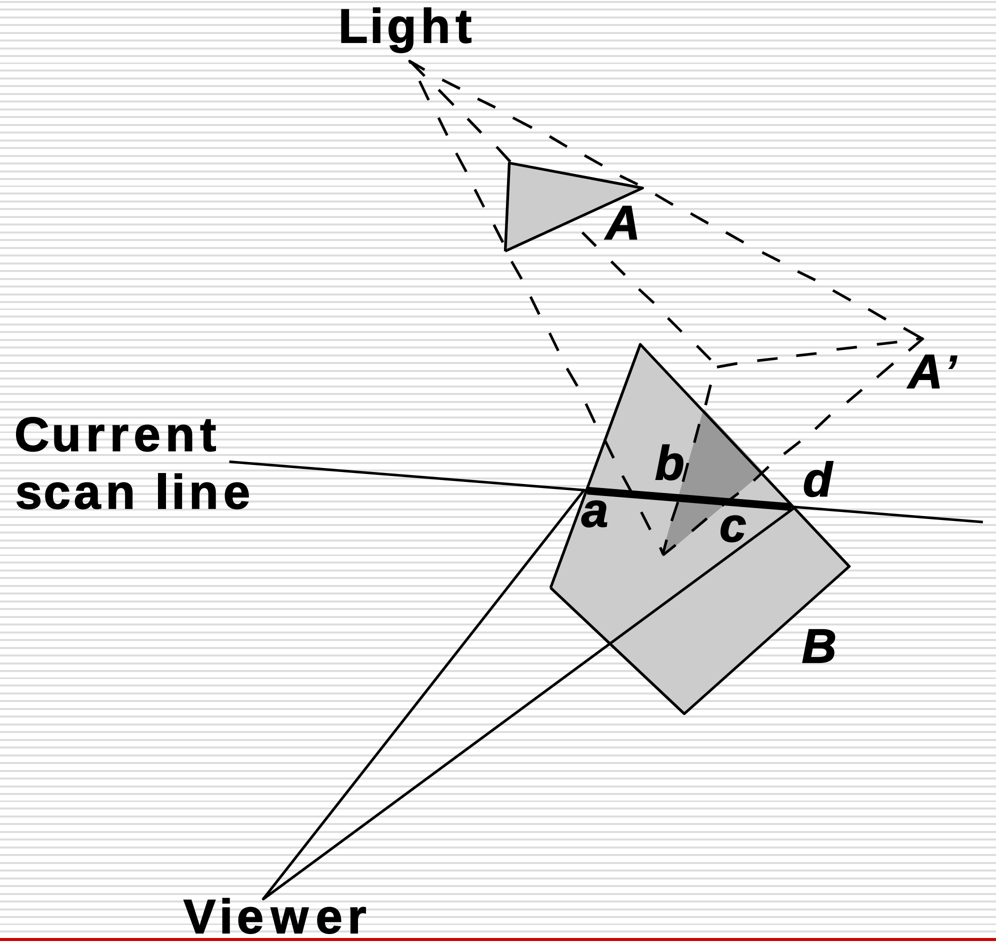


Shadows

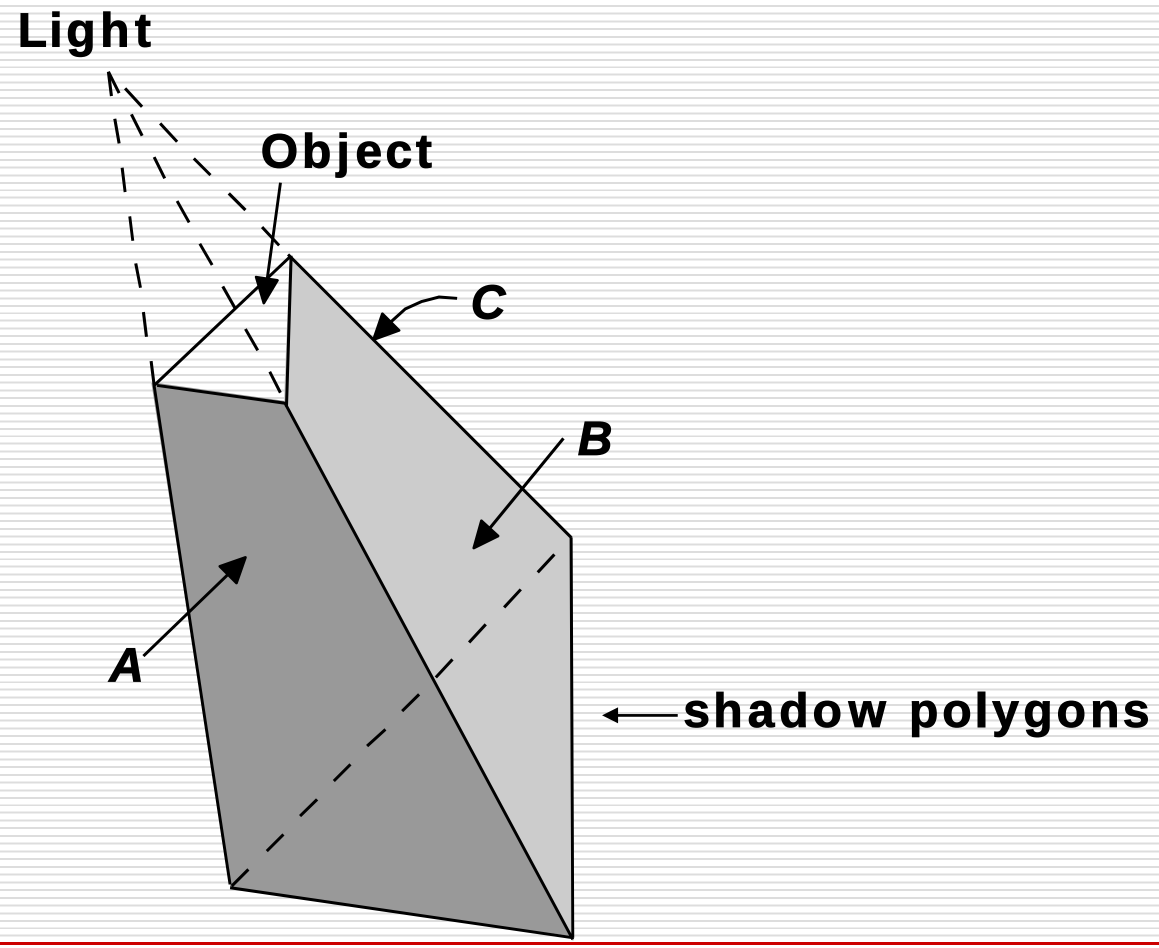
- $I_\lambda = I_{a\lambda} k_a O_{d\lambda} + \sum_{1 \le i \le m} S_i f_{att_i} I_{p\lambda_i} [k_d O_{d\lambda} (\vec{N} \bullet \vec{L}_i) + k_s O_{s\lambda} (\vec{R}_i \bullet \vec{V})^n]$

  - $S_i = \begin{cases} 0, \text{if light } i \text{ is blocked at this point} \\ 1, \text{if light } i \text{ is not blocked at this point} \end{cases}$

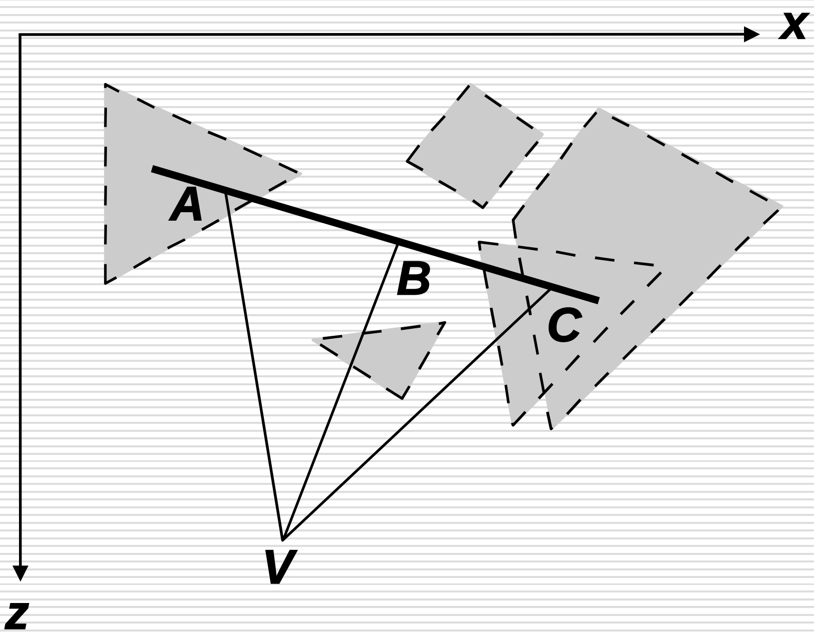
Scan-Line Generation of Shadows

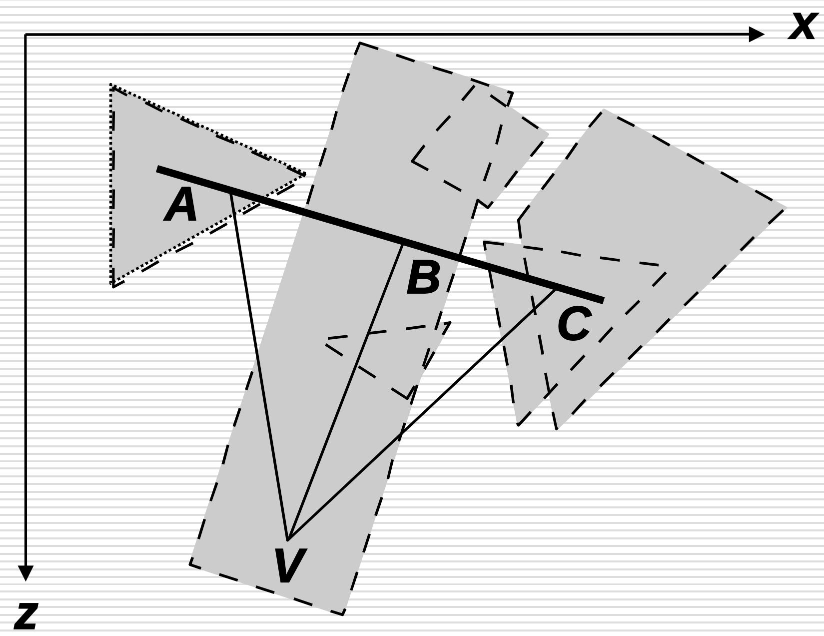


Shadow Volumes

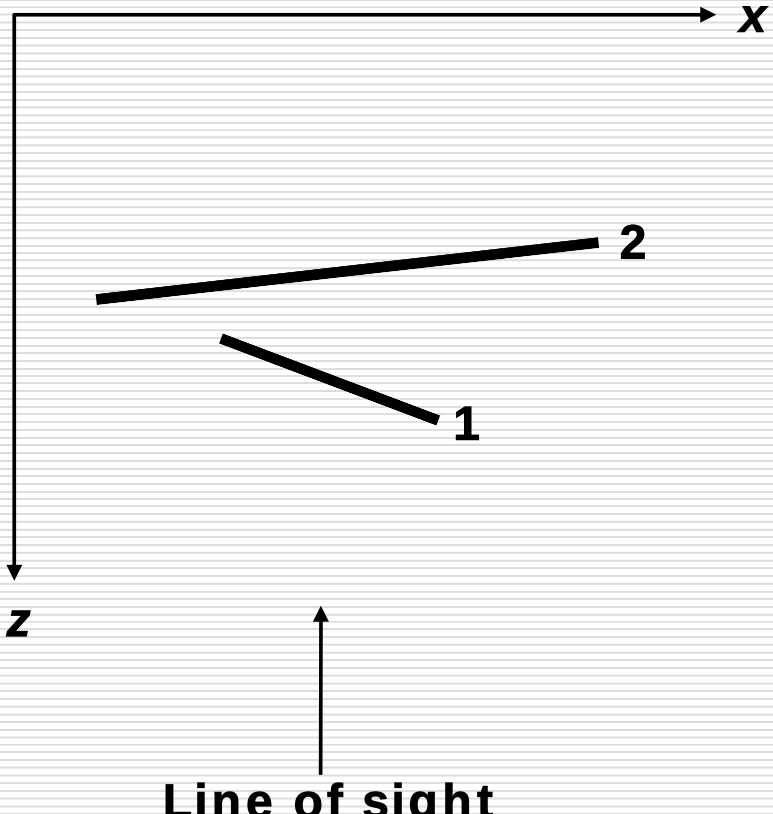


Shadow Volumes





Transparency

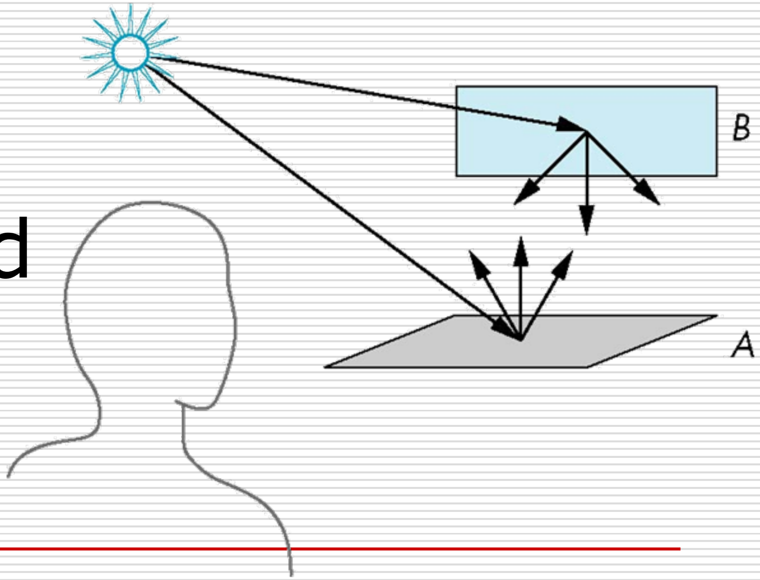


- interpolated transparency
- $I_\lambda = (1 - k_{t1}) I_{\lambda 1} + k_{t1} I_{\lambda 2}$
  - k_{t1}: transparency: 0 ~ 1
- filtered transparency
- $I_\lambda = I_{\lambda 1} + k_{t1} O_{t\lambda} I_{\lambda 2}$
  - $O_{t\lambda}$: transparency color

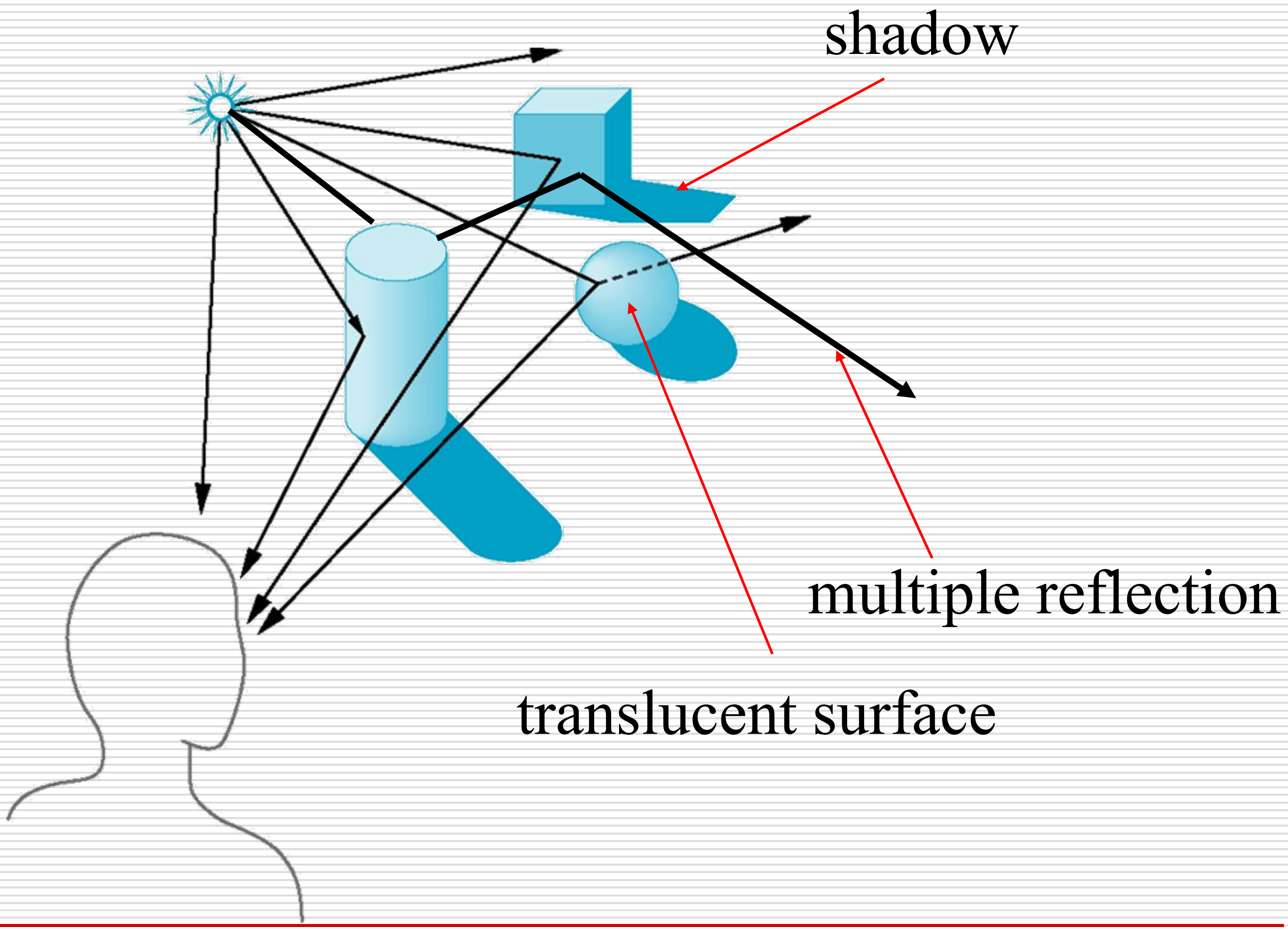
Scattering

- Light strikes A
  - Some scattered
  - Some absorbed
- Some of scattered light strikes B
  - Some scattered
  - Some absorbed
- Some of this scattered light strikes A

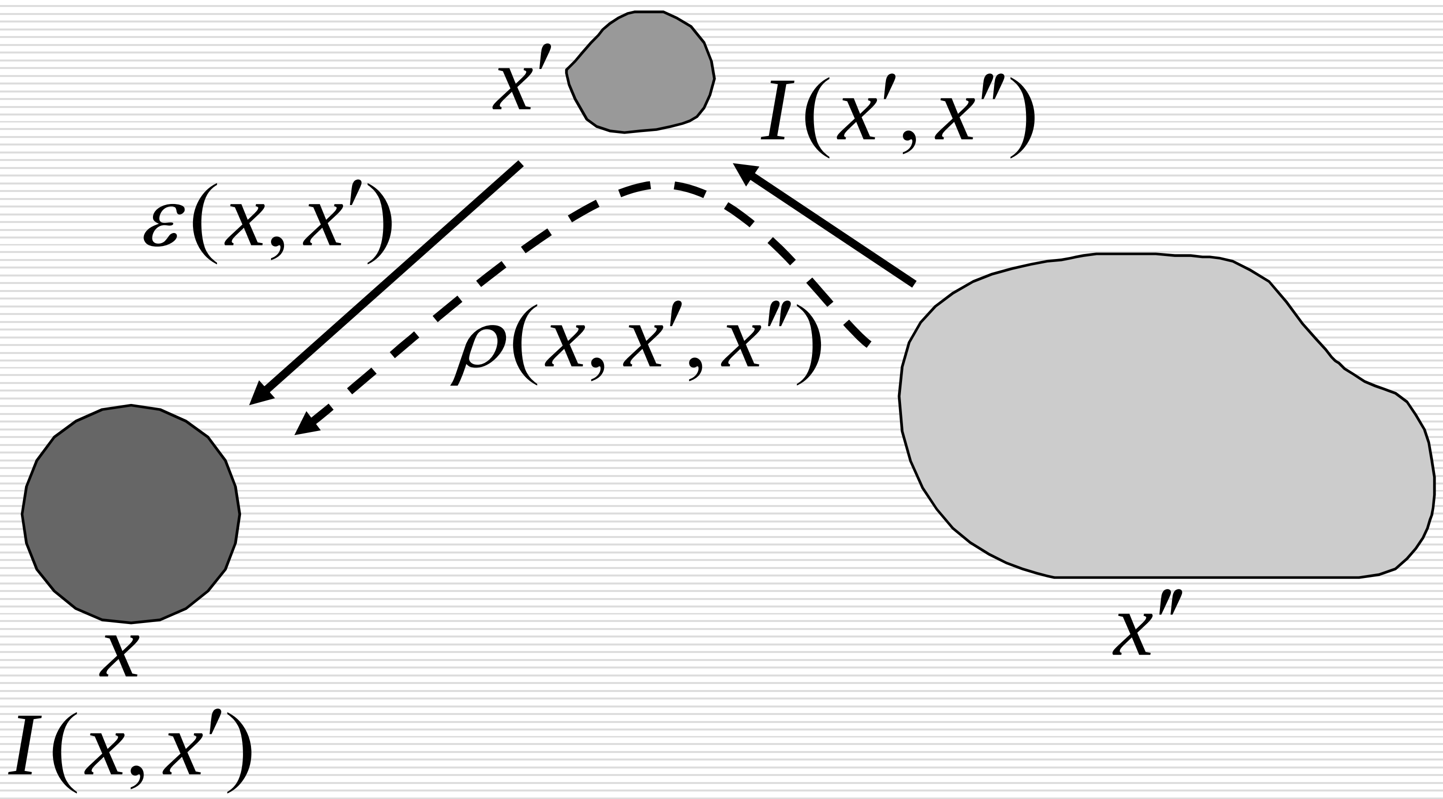
  and so on



Global Effects



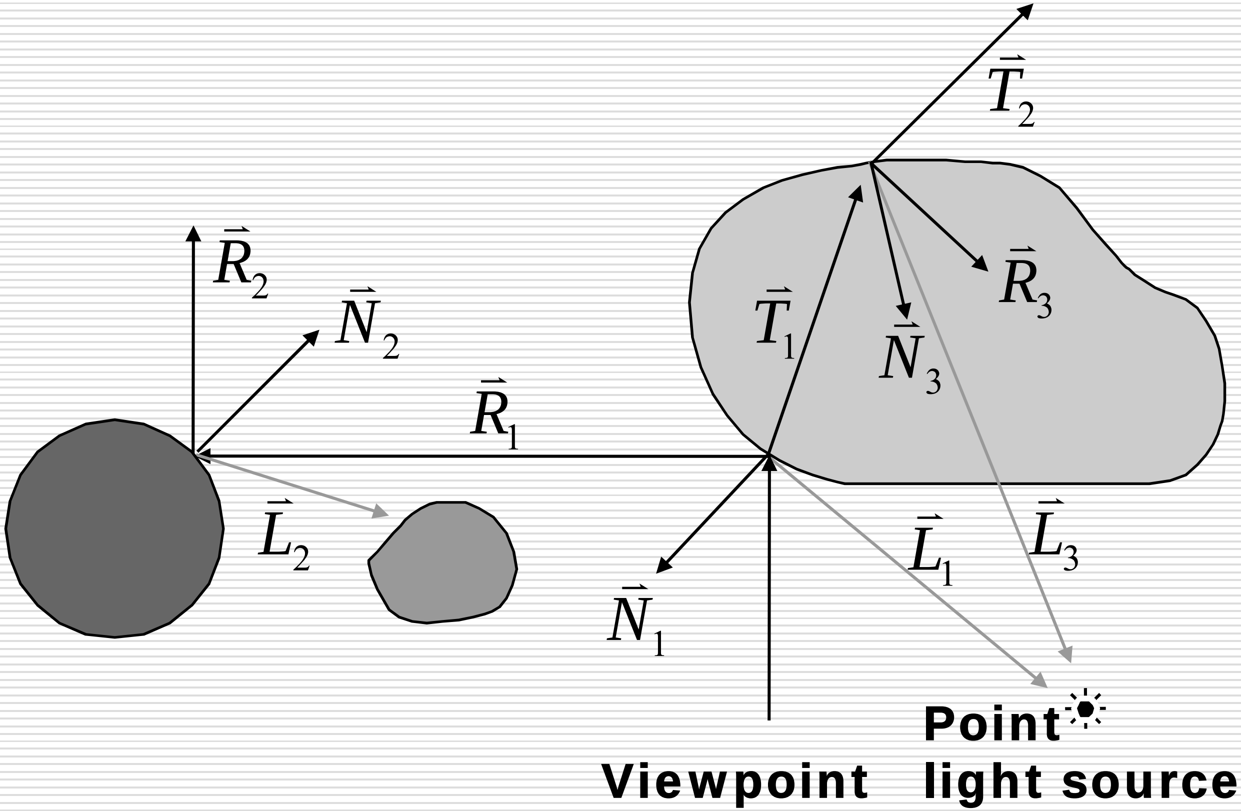
Global Illumination



The Rendering Equation

- $I(x,x') = g(x,x')\left[\varepsilon(x,x') + \int_S \rho(x,x',x'')I(x',x'')dx''\right]$
  - $I(x,x')$: intensity passing from x' to x
  - $\varepsilon(x,x')$: emitted light intensity from x' to x
  - $\rho(x,x',x'')$: intensity of light reflected from x'' to x from the surface at x'
  - $g(x,x') = \begin{cases} 0, & \text{if } x' \text{ is invisible from } x \\ 1/r^2, & \text{if } x' \text{ is visible from } x \end{cases}$
  - r : the distance between x' and x
  - S : all surfaces

Recursive Ray Tracing



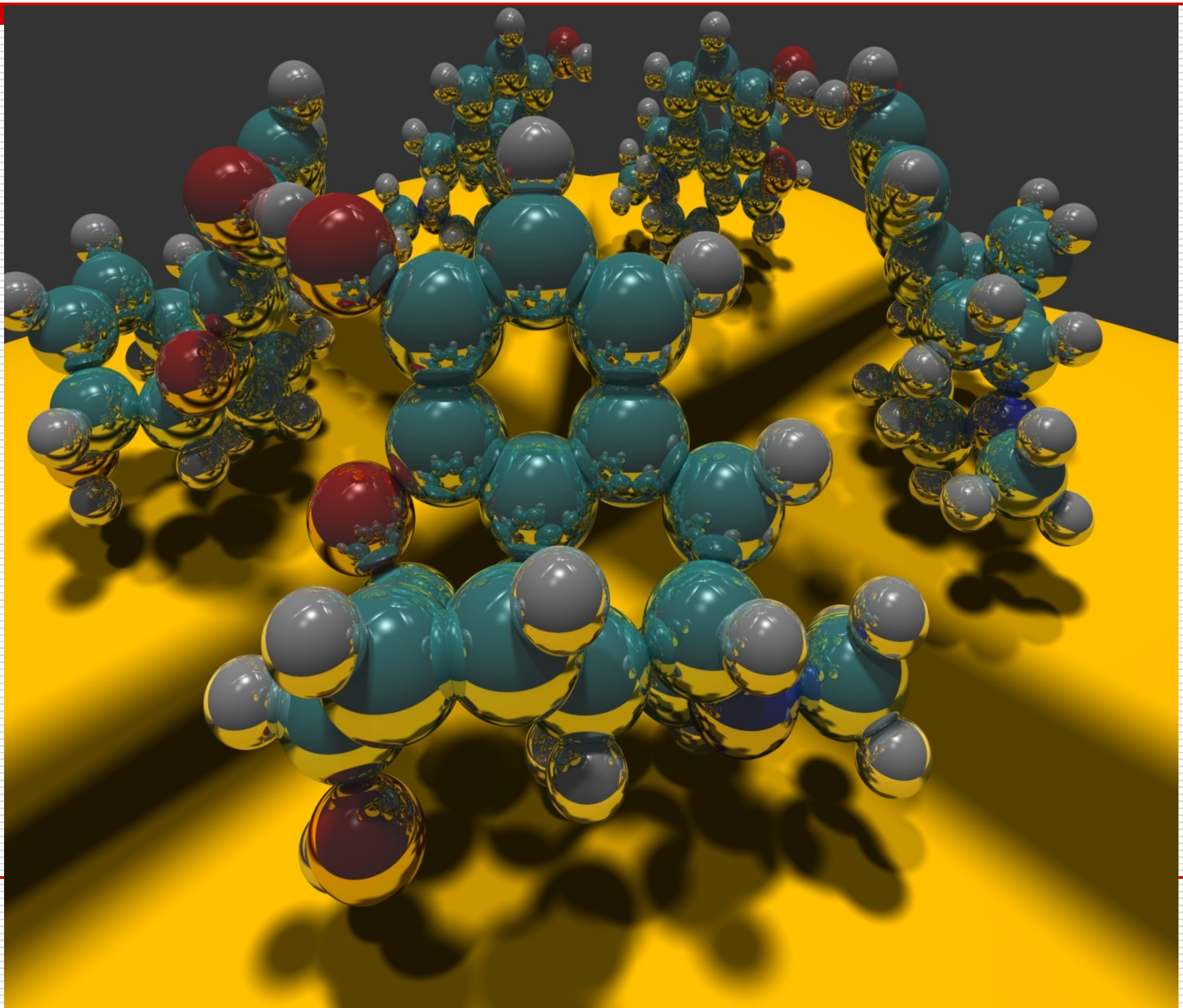
$\vec{N}_i$ **surface normal**

$\vec{R}_i$ **reflected ray**

$\vec{L}_i$ **shadow ray**

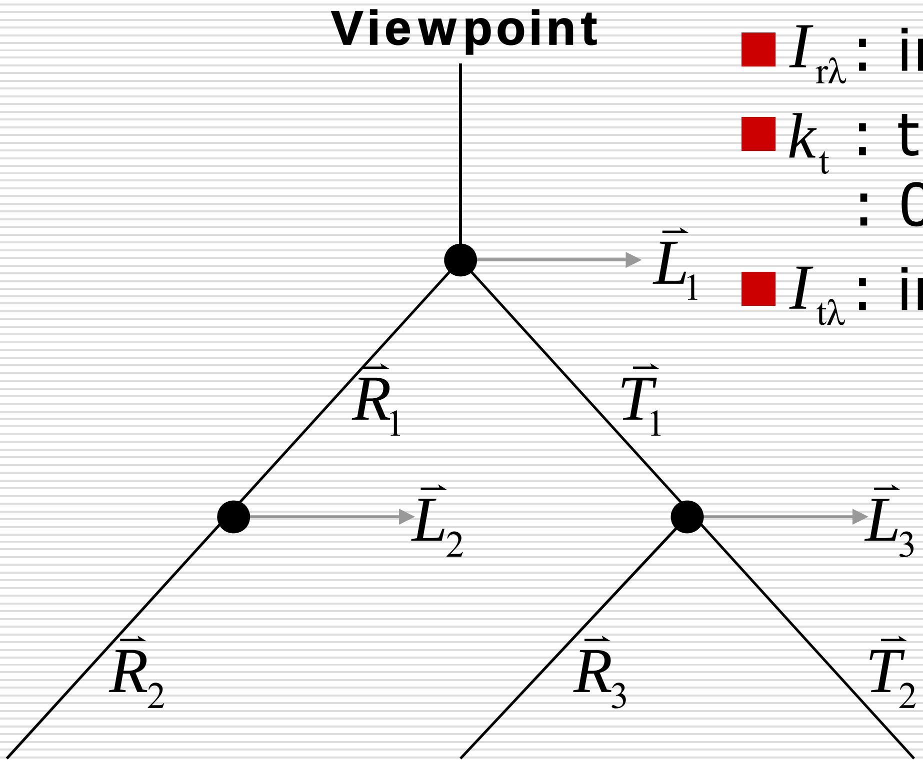
$\vec{T}_i$ **transmitted ray**

Reflection



Refraction

The Ray Tree



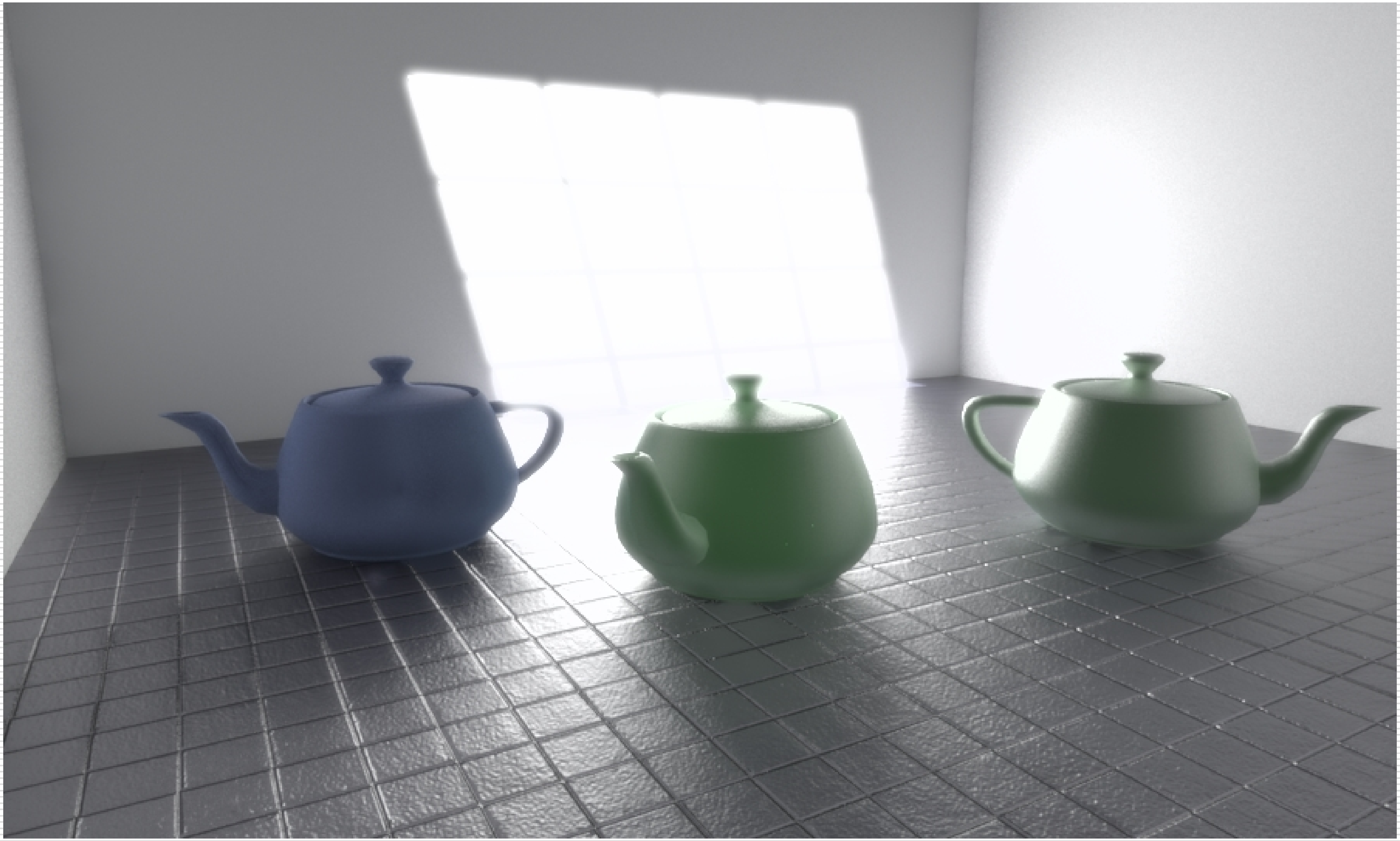
- $I_{r\lambda}$: intensity of reflected ray
- k_t : transmission coefficient
  : 0 ~ 1
- $I_{t\lambda}$: intensity of transmitted ray

$$I_\lambda = I_{a\lambda} k_a O_{d\lambda} + \sum_{1 \le i \le m} S_i f_{att_i} I_{p\lambda_i} [k_d O_{d\lambda} (\vec{N} \bullet \vec{L}_i) + k_s (\vec{N} \bullet \vec{H}_i)^n] + k_s I_{r\lambda} + k_t I_{t\lambda}$$

Ray Tracing Results

Ray Tracing Results

Ray Tracing Results



Ray Tracing Results

Ray Tracing Results

The Radiosity Equation

- $$B_i = E_i + \rho_i \sum_{1 \le j \le n} B_j F_{j-i} \frac{A_j}{A_i}$$
  - B_i: radiosity of patch i
  - E_i: rate at which light is emitted from patch i
  - ρ_i: reflectivity of patch i
  - F_{j-i} : form factor (configuration factor)
  - A_i: area of patch i
- since $A_i F_{i-j} = A_j F_{j-i}$
- thus $B_i = E_i + \rho_i \sum_{1 \le j \le n} B_j F_{i-j}$

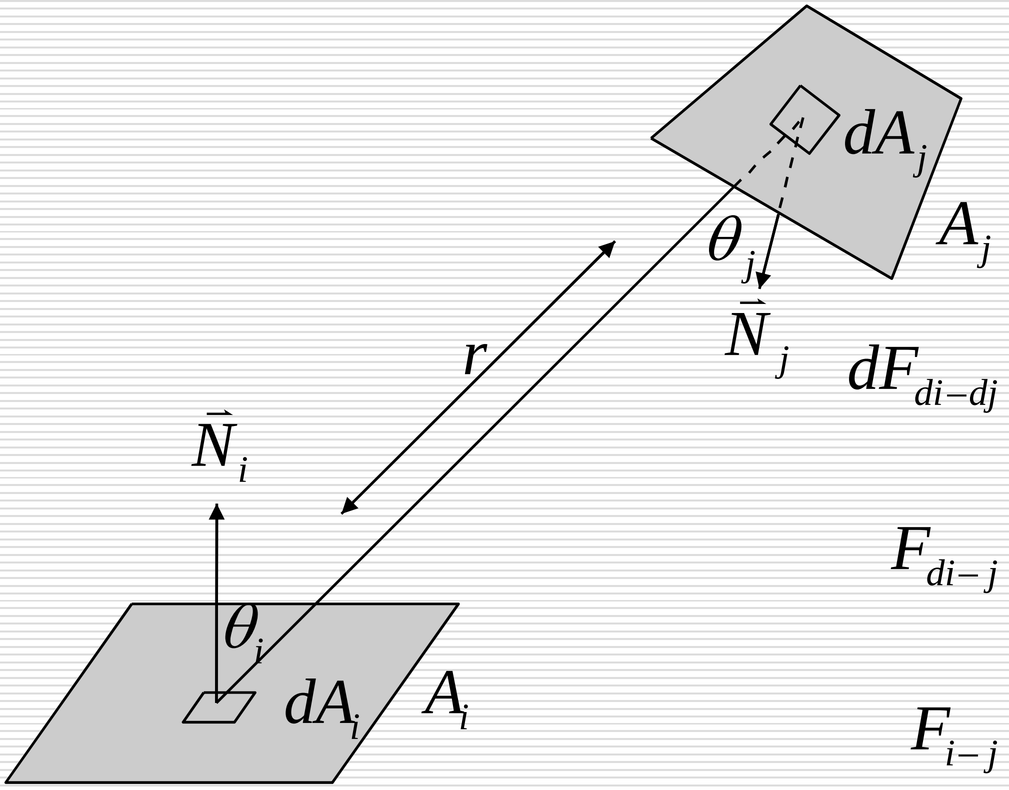
The Radiosity Equation

- rearranging terms $B_i - \rho_i \sum_{1 \le j \le n} B_j F_{i-j} = E_i$
- therefore

$$\begin{bmatrix} 1-\rho_1 F_{1-1} & -\rho_1 F_{1-2} & \cdots & -\rho_1 F_{1-n} \\ -\rho_2 F_{2-1} & 1-\rho_2 F_{2-2} & \cdots & -\rho_2 F_{2-n} \\ \vdots & \vdots & \cdots & \vdots \\ -\rho_n F_{n-1} & -\rho_n F_{n-2} & \cdots & 1-\rho_n F_{n-n} \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ E_n \end{bmatrix}$$

- progressive refinement

Computing Form Factors



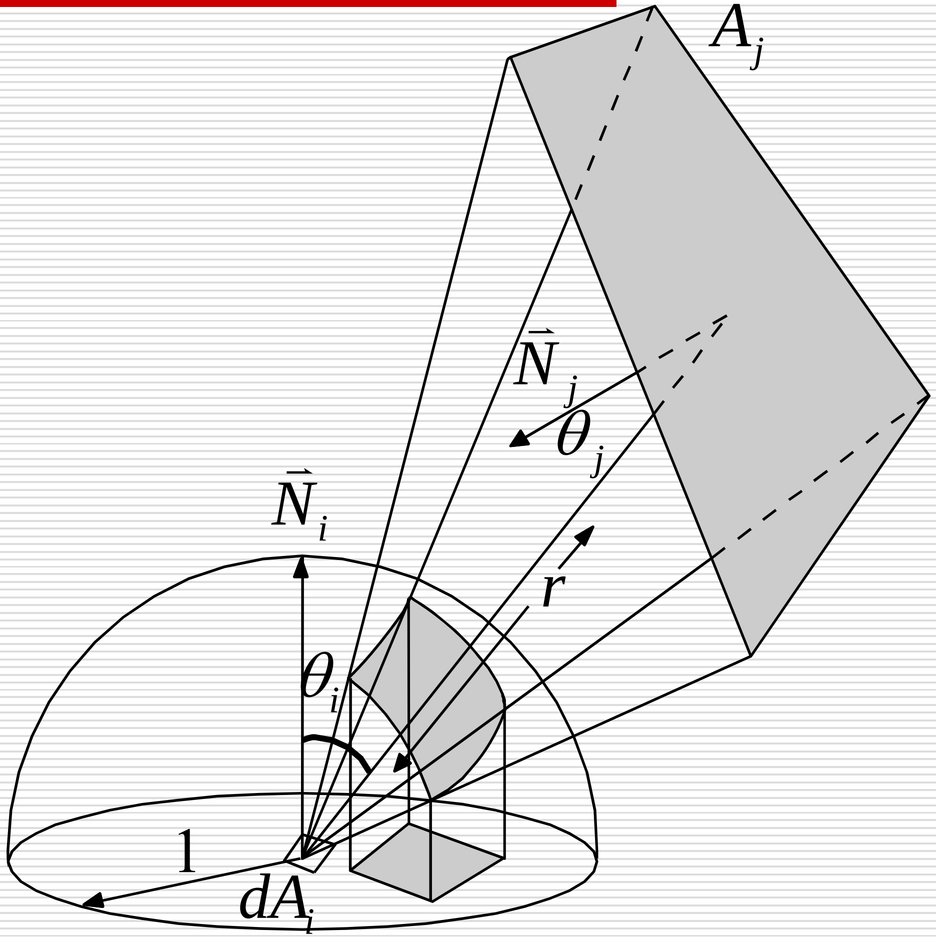
visible or invisible
1 **0**

$$dF_{di-dj} = \frac{\cos\theta_i \cos\theta_j}{\pi r^2} H_{ij} dA_j$$

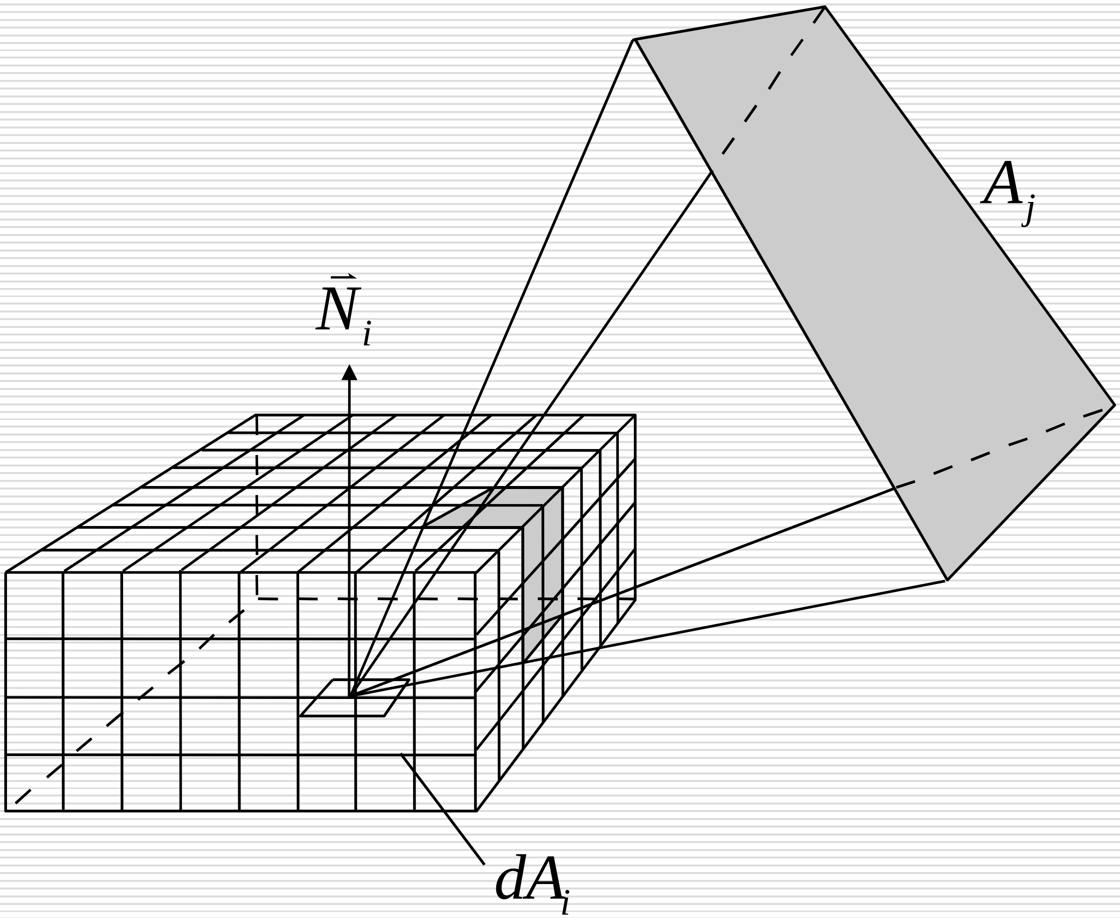
$$F_{di-j} = \int_{A_j} \frac{\cos\theta_i \cos\theta_j}{\pi r^2} H_{ij} dA_j$$

$$F_{i-j} = \frac{1}{A_i} \int_{A_i} \int_{A_j} \frac{\cos\theta_i \cos\theta_j}{\pi r^2} H_{ij} dA_j dA_i$$

Hemisphere



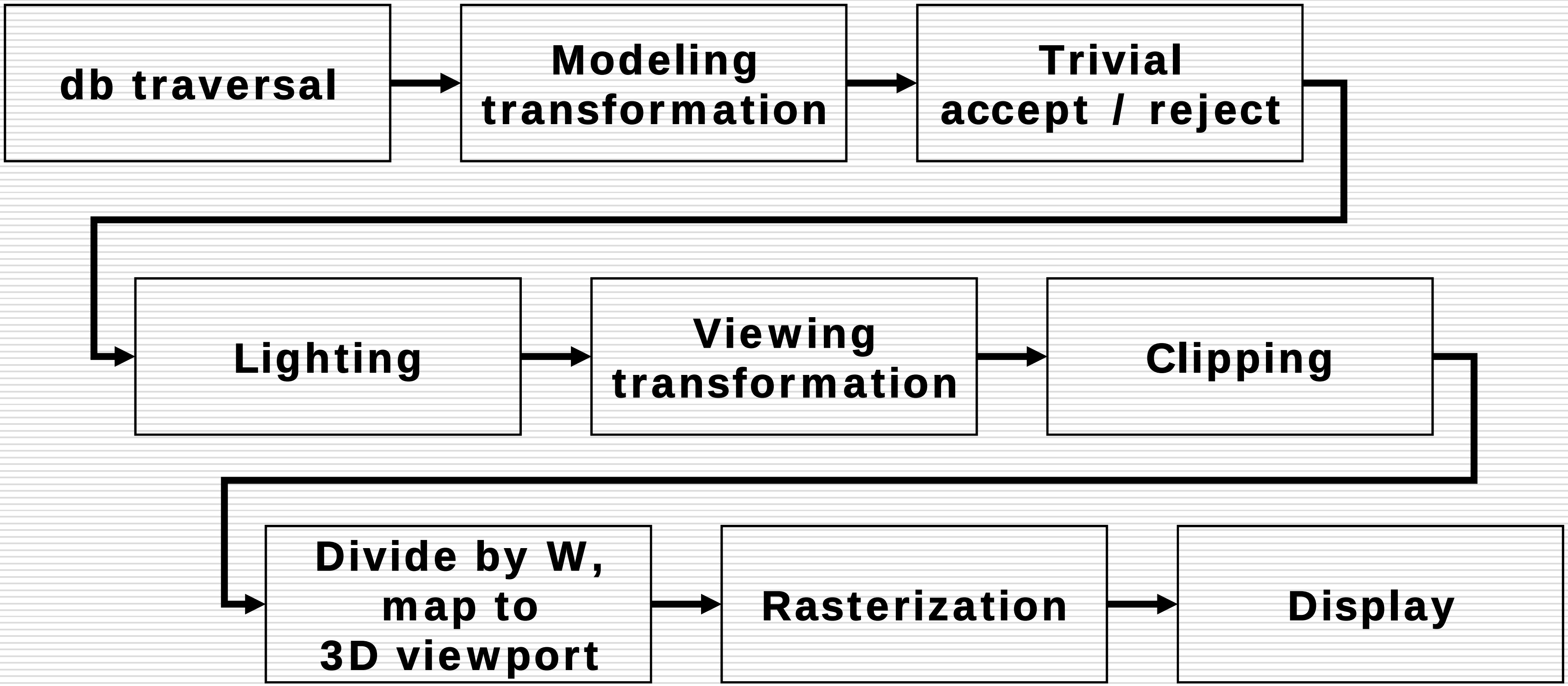
Hemicube



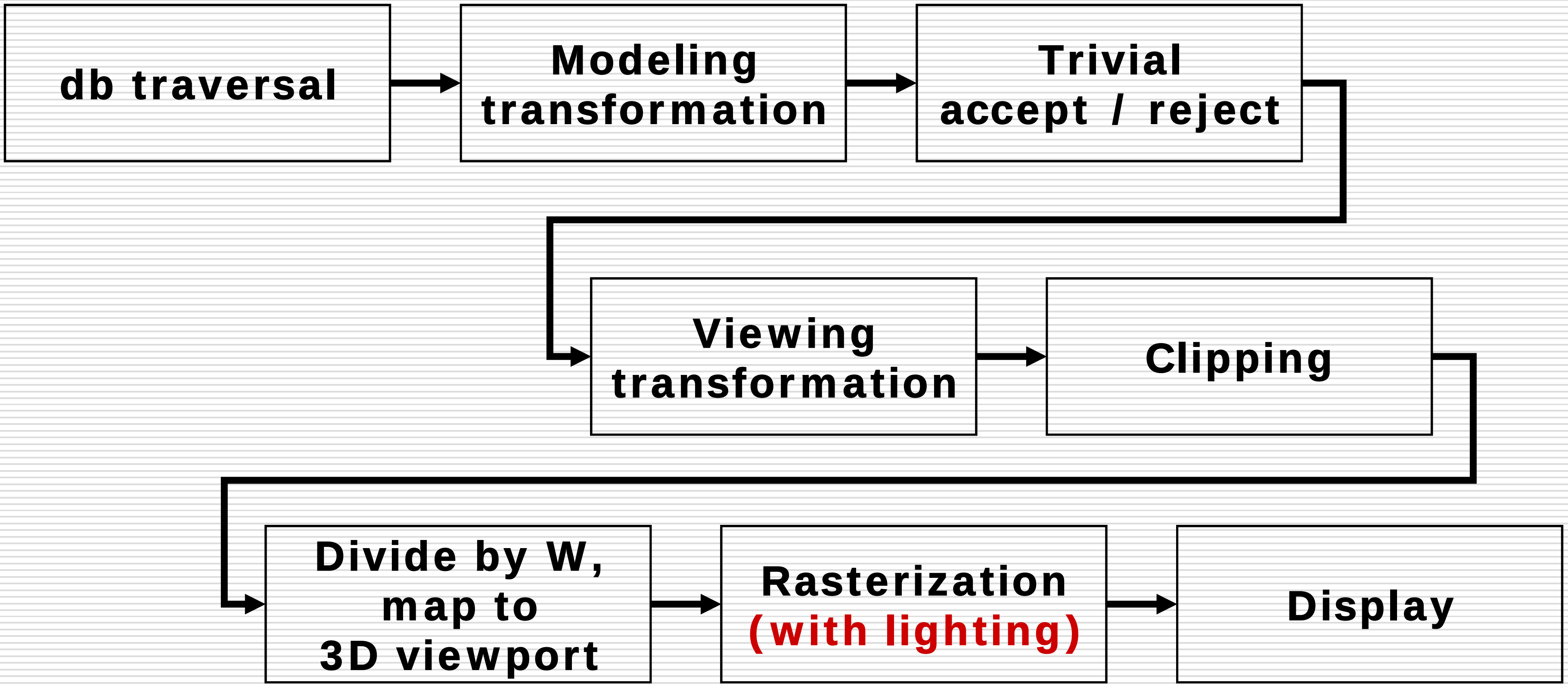
The Rendering Pipeline

- Local Illumination Pipelines
  - z-buffer and Gouraud shading
  - z-buffer and Phong shading
  - list-priority algorithm and Phong shading
- Global Illumination Pipelines
  - radiosity
  - ray tracing

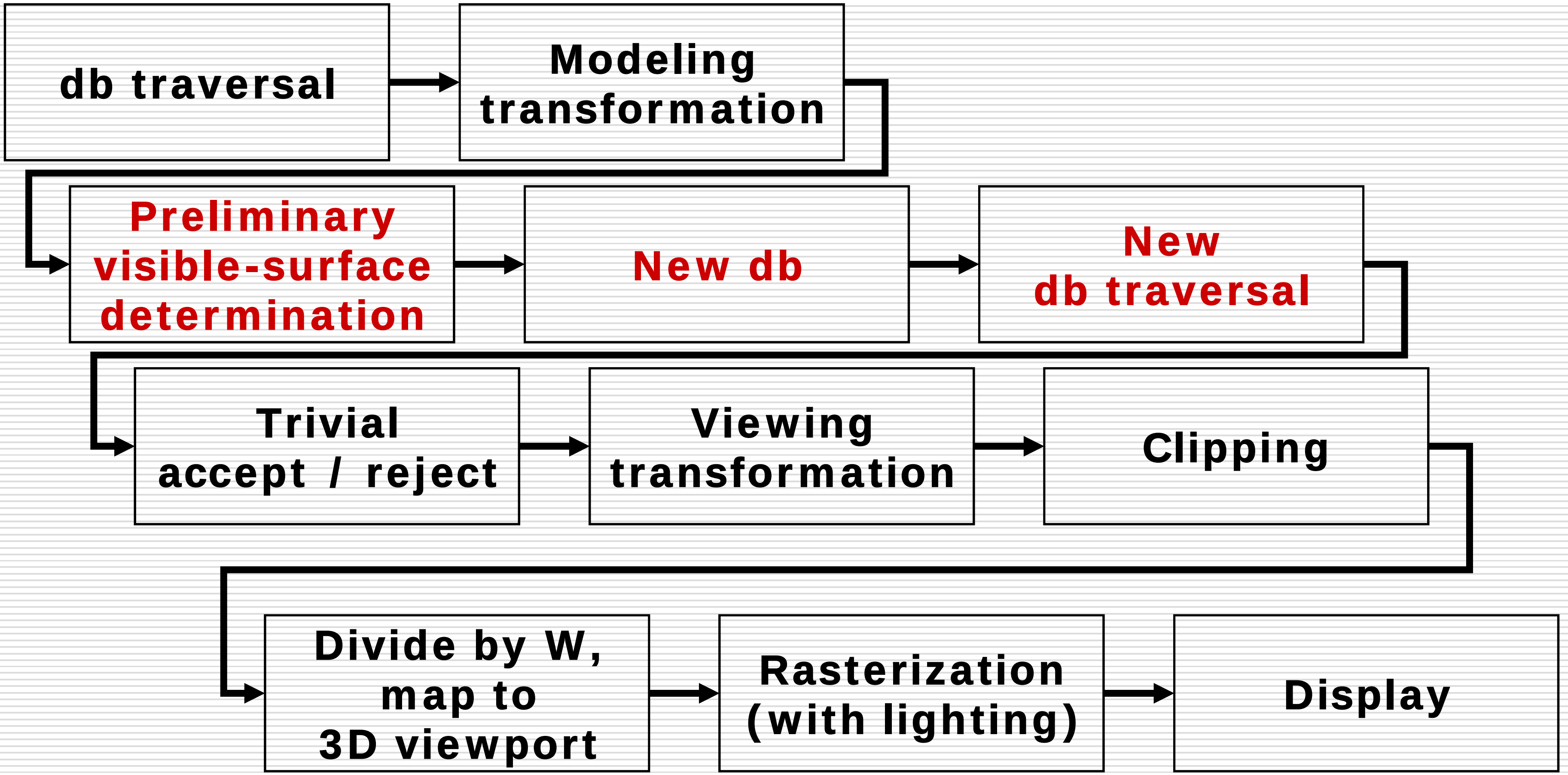
Rendering Pipeline for z-buffer & Gouraud shading



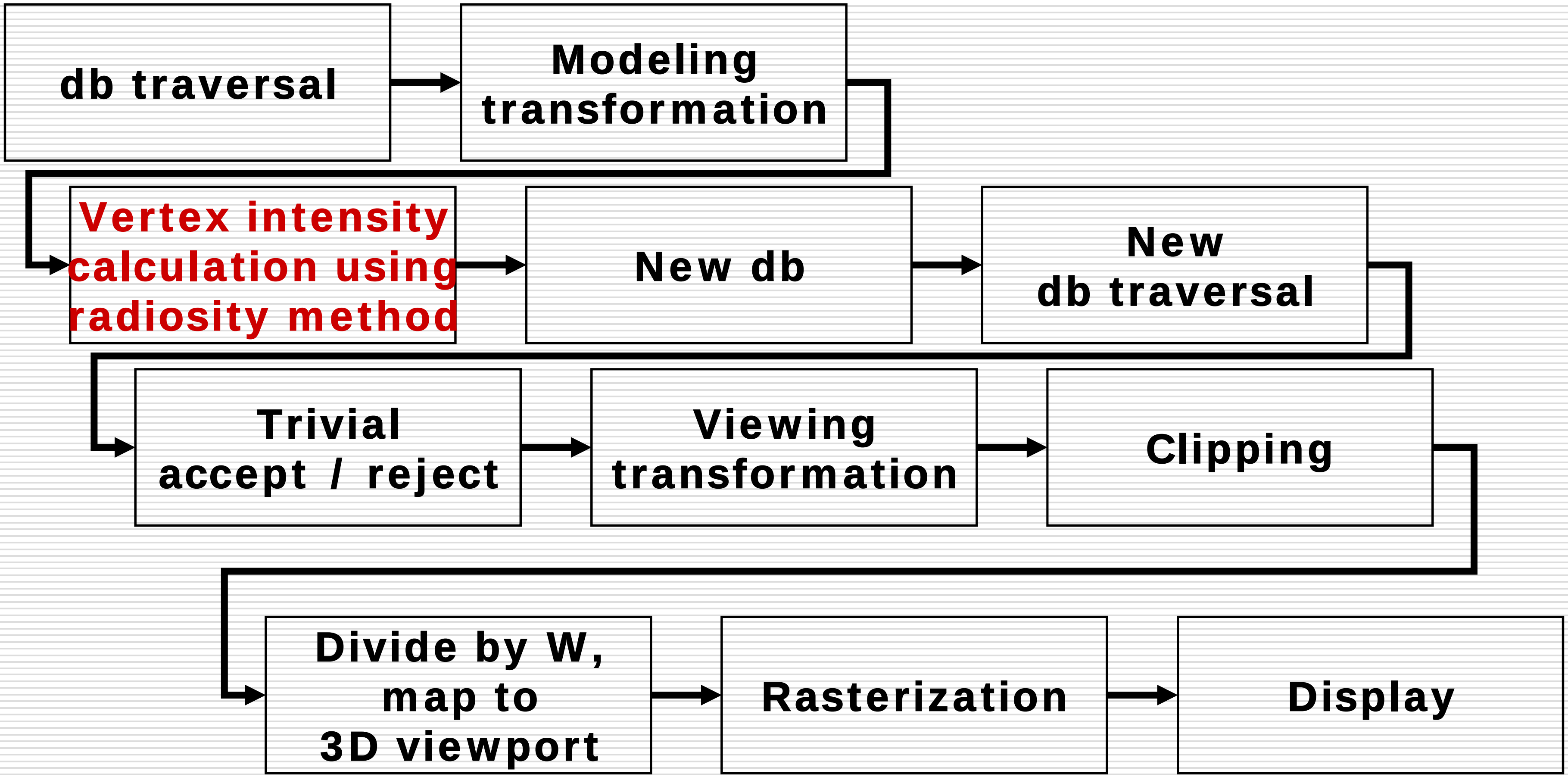
Rendering Pipeline for z-buffer & Phong shading



Rendering Pipeline for list-priority algorithm & Phong shading



Rendering Pipeline for radiosity & Gouraud shading



Rendering Pipeline for ray tracing