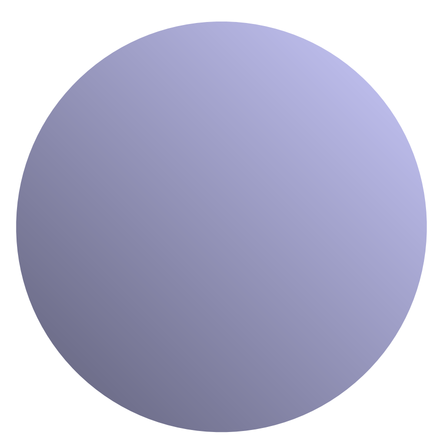
Lighting and Shading

Slides: Tamar Shinar, Victor Zordon

Why we need shading

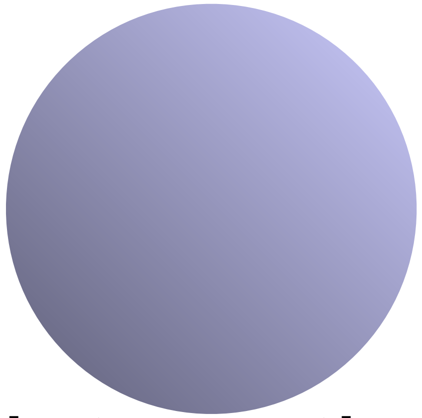
- Suppose we build a model of a sphere using many polygons and color each the same color. We get something like

- But we want



Shading

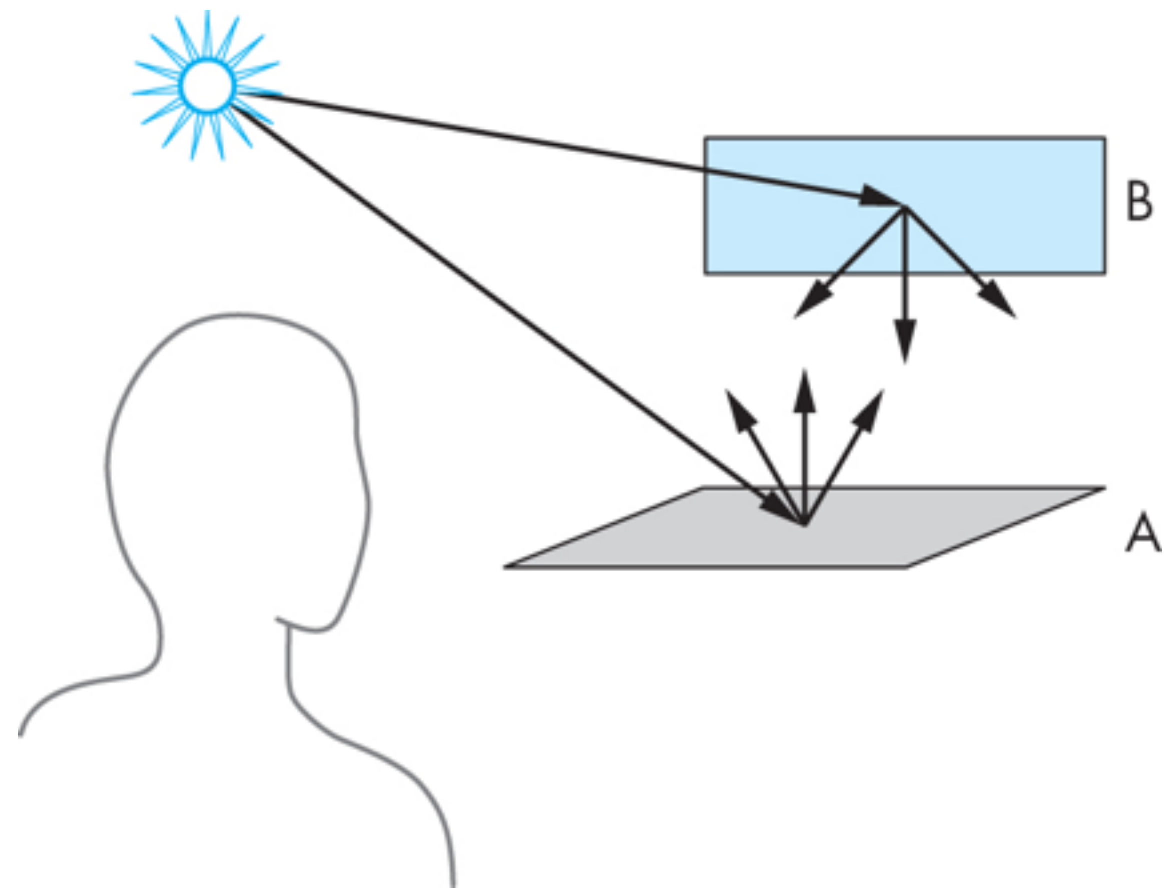
- Why does the image of a real sphere look like



- 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 (normal)

General rendering

- The most general approach is based on physics - using principles such as conservation of energy
- a surface either **emits** light (e.g., light bulb) or **reflects** light from other illumination sources, or both
- light interaction with materials is **recursive**
- the **rendering equation** is an integral equation describing the limit of this recursive process

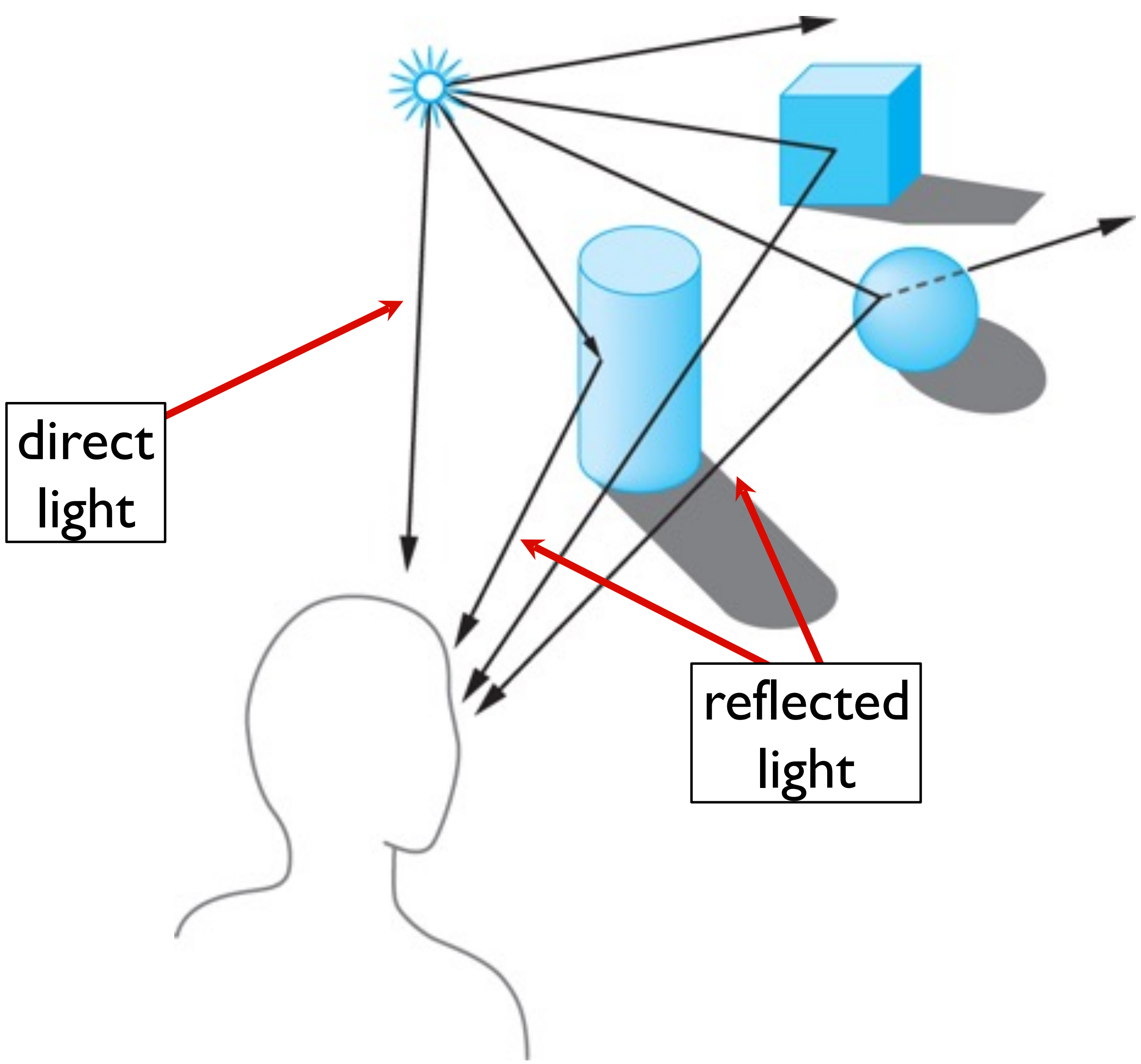


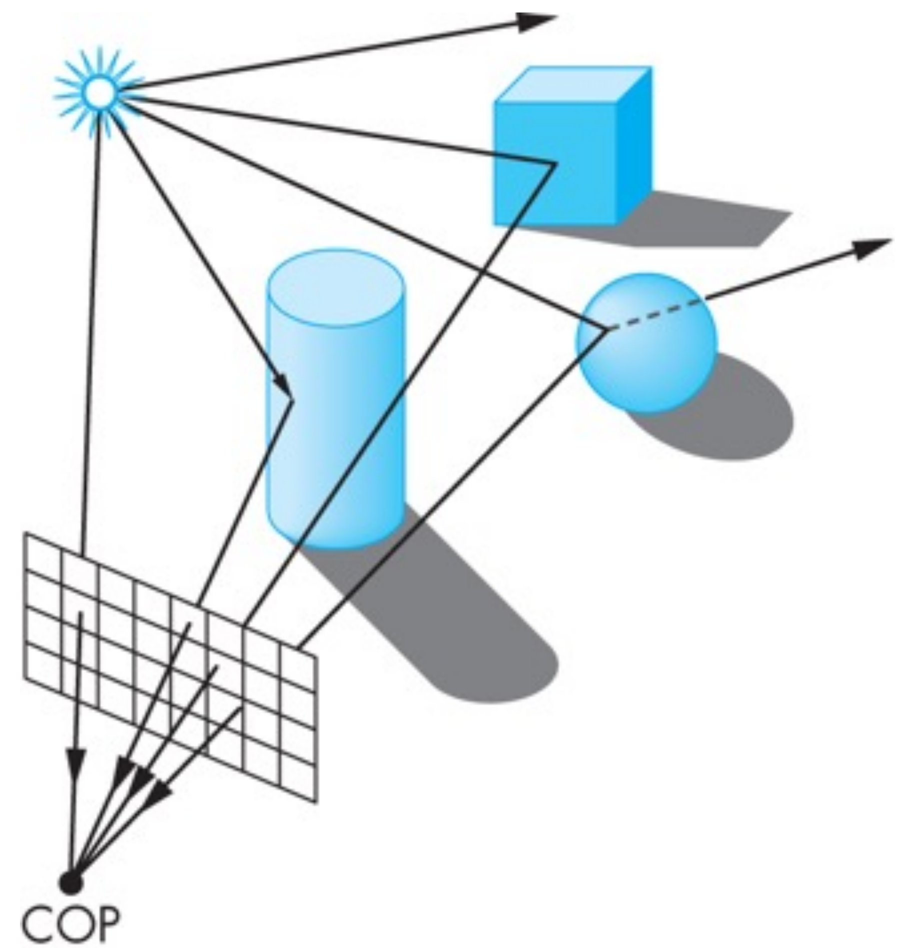
[Angel and Shreiner]

Fast local shading models

- the rendering equation can’t be solved analytically
- numerical methods aren’t fast enough for real-time
- for our fast graphics rendering pipeline, we’ll use a **local** model where shade at a point is independent of other surfaces
- use **Phong reflection model**
  - shading based on local light-material interactions

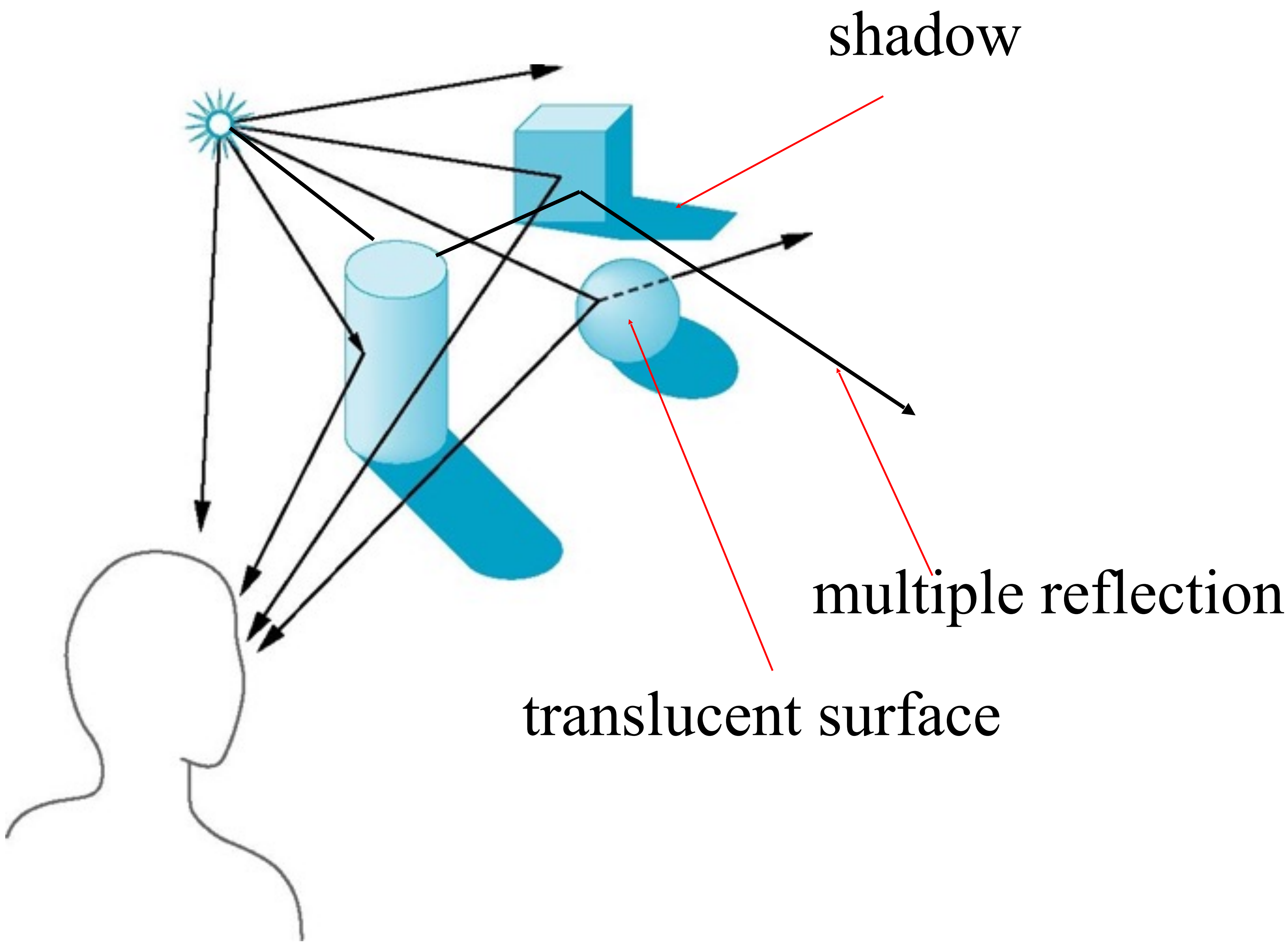
Local shading model





[Angel and Shreiner]

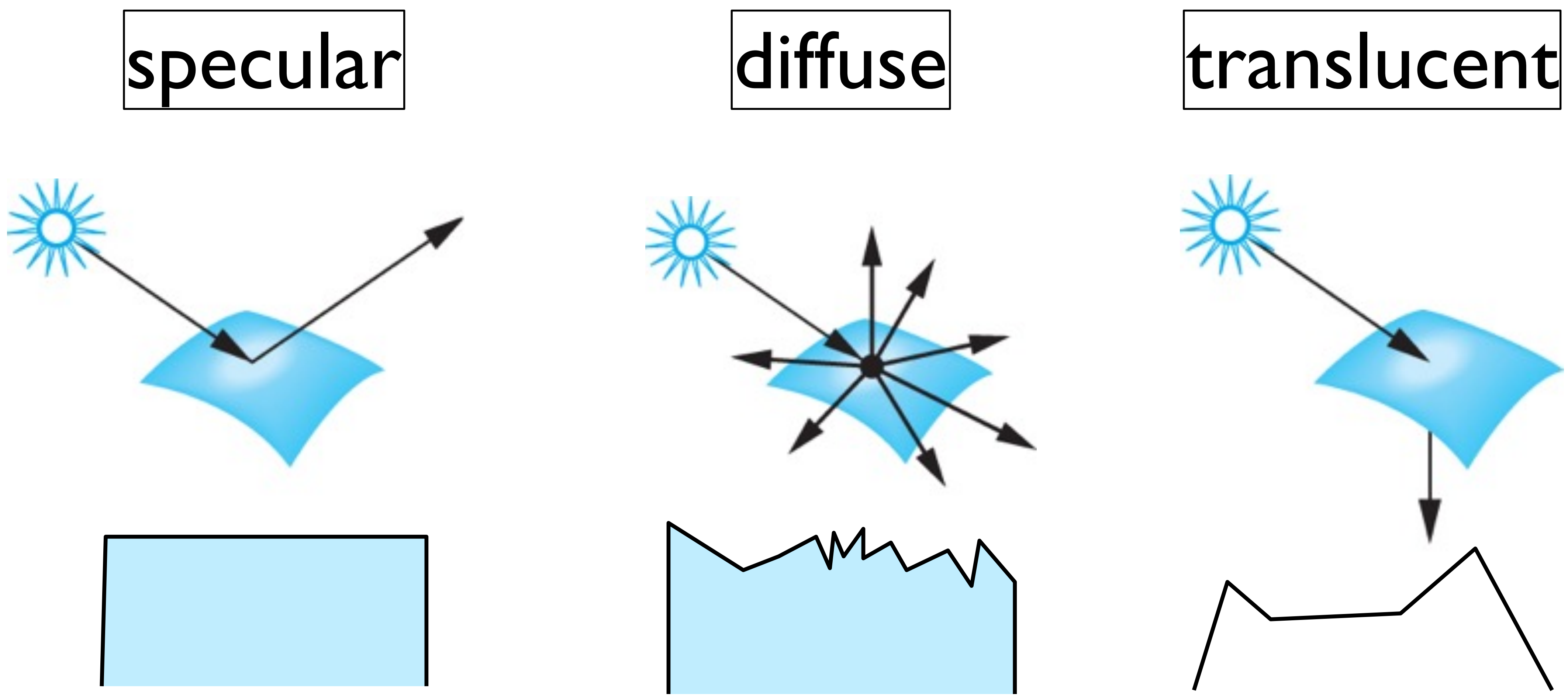
Global Effects



[Angel and Shreiner]

Light-material interactions

at a surface, light is absorbed, reflected, or transmitted

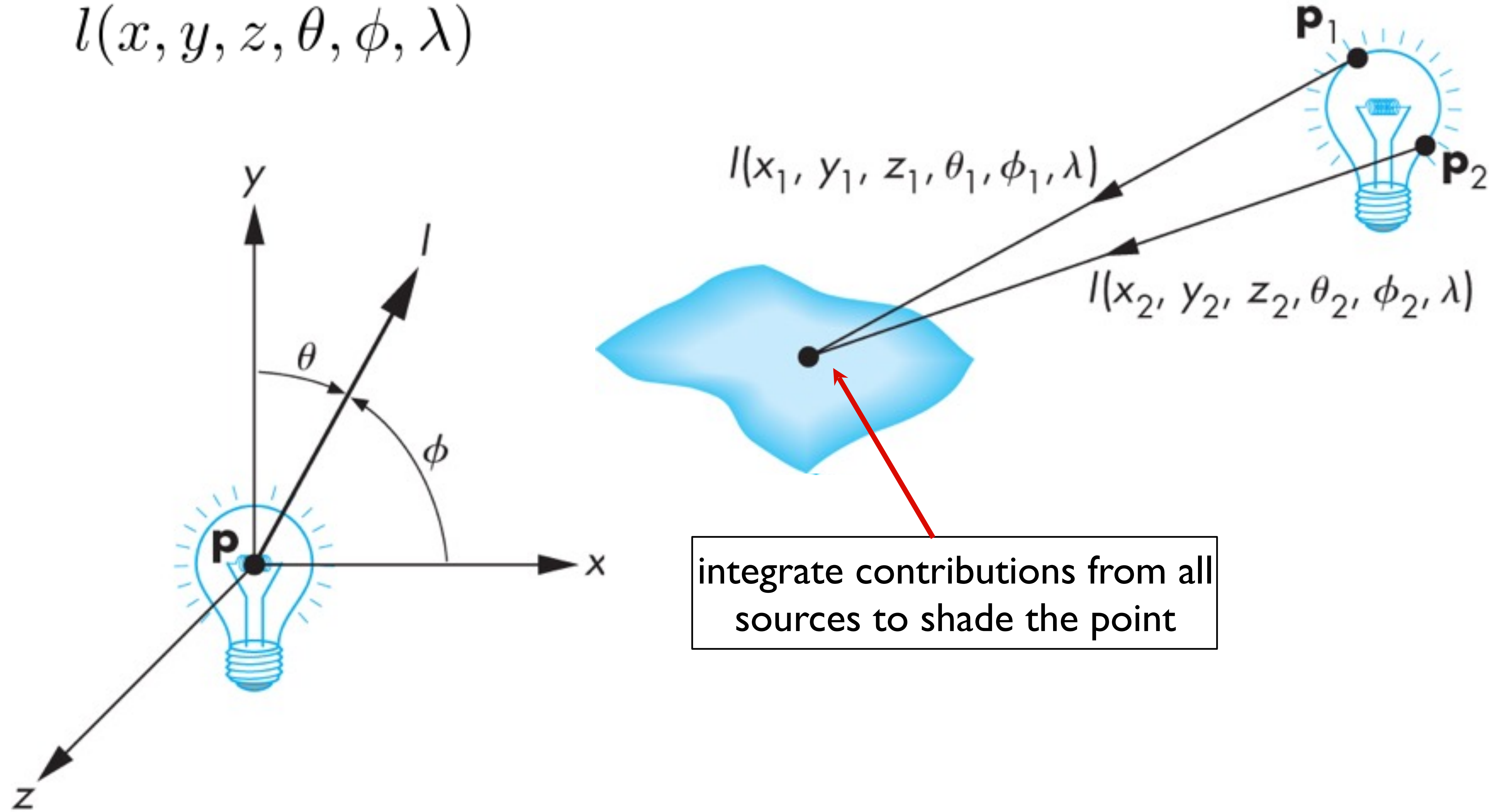


[Angel and Shreiner]

General light source

Illumination function:

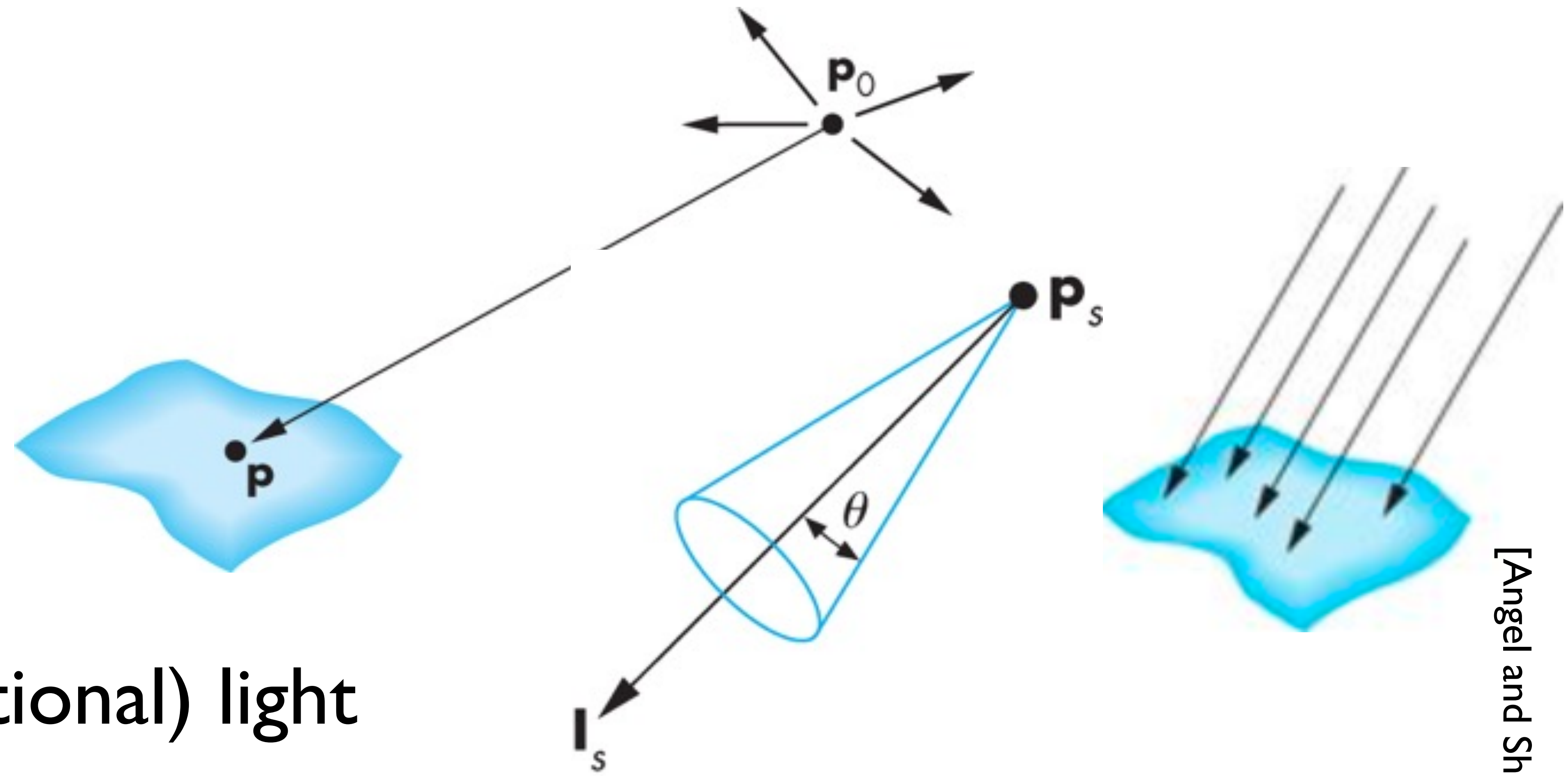
$$l(x, y, z, \theta, \phi, \lambda)$$



[Angel and Shreiner]

Idealized light sources

- Ambient light
- Point light
- Spotlight
- distant (directional) light



[Angel and Shreiner]

luminance: $\mathbf{L} = \begin{bmatrix} L_r \\ L_g \\ L_b \end{bmatrix}$

Ambient light source

- achieve a uniform light level
- no black shadows
- ambient light intensity at each point in the scene

$$\mathbf{L}_a = \begin{bmatrix} L_{ar} \\ L_{ag} \\ L_{ab} \end{bmatrix}$$

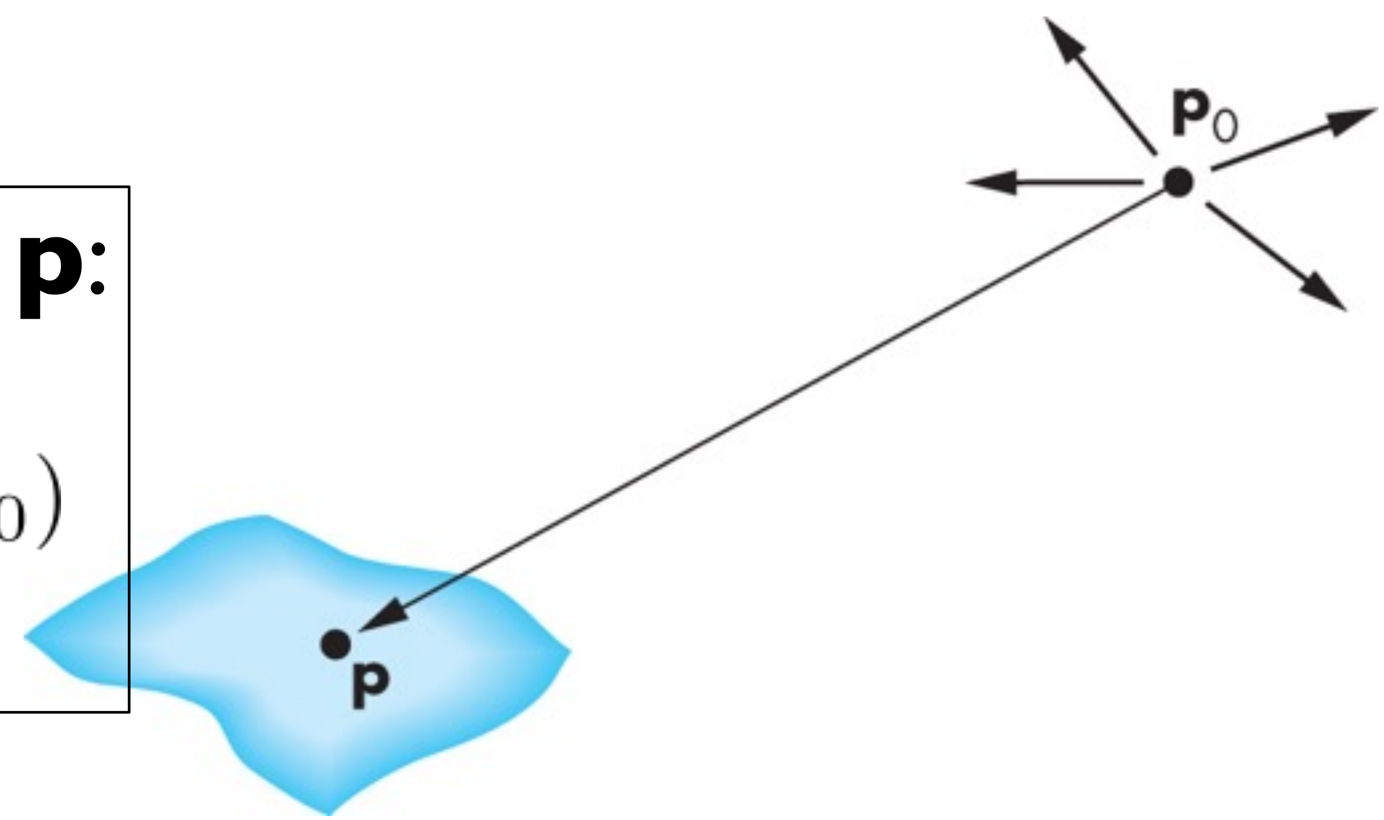
$$L_a$$

Point light source

$$\mathbf{L}(\mathbf{p}_0) = \begin{bmatrix} L_r(\mathbf{p}_0) \\ L_g(\mathbf{p}_0) \\ L_b(\mathbf{p}_0) \end{bmatrix} \qquad L(\mathbf{p}_0)$$

illumination intensity at **p**:

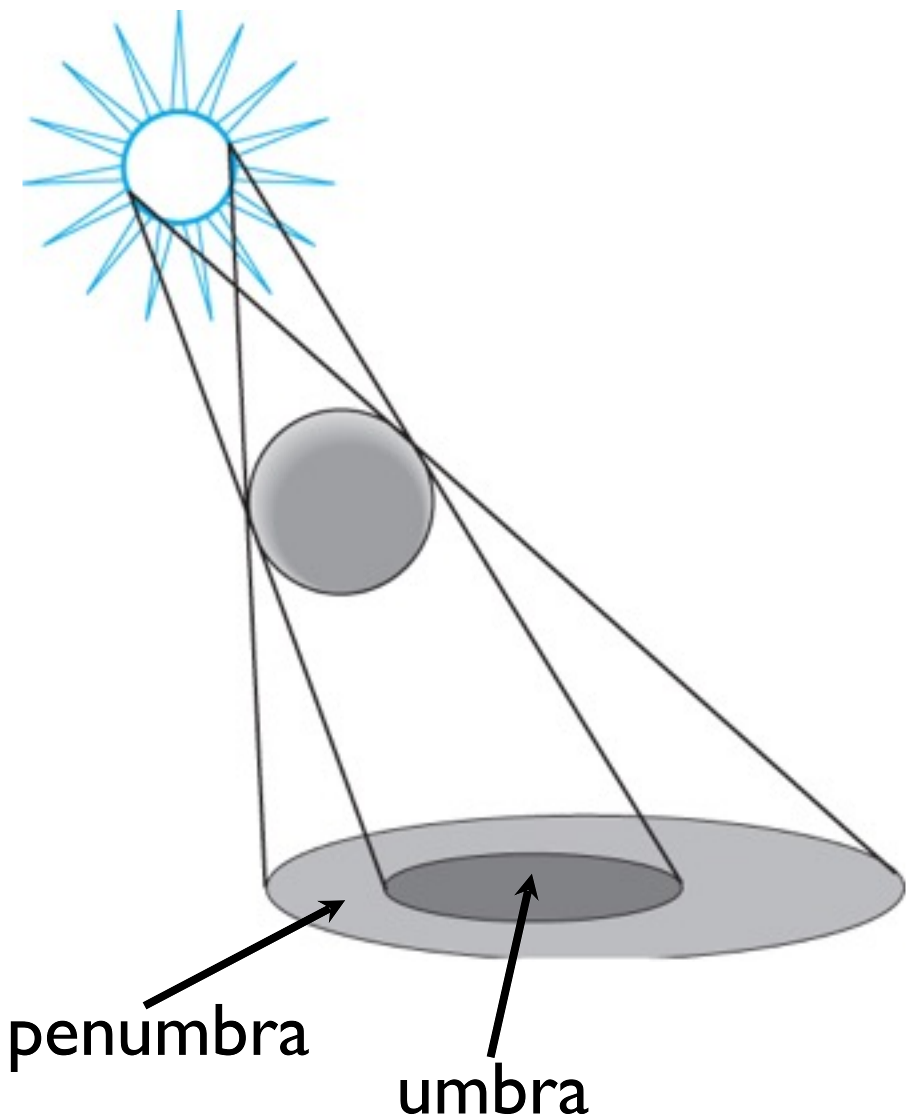
$$l(\mathbf{p}, \mathbf{p}_0) = \frac{1}{|\mathbf{p} - \mathbf{p}_0|^2} \mathbf{L}(\mathbf{p}_0)$$



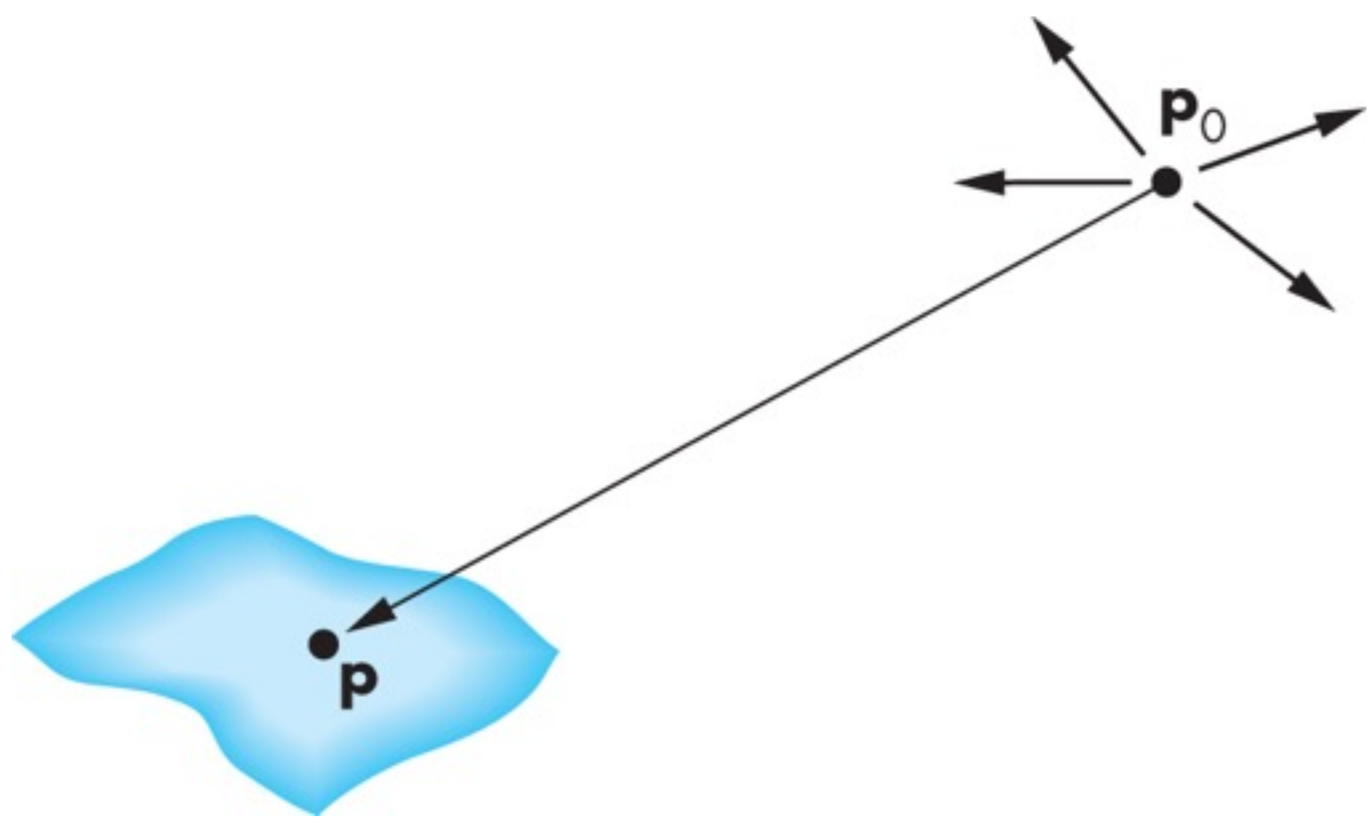
[Angel and Shreiner]

Point light source

Most real-world scenes have large light sources



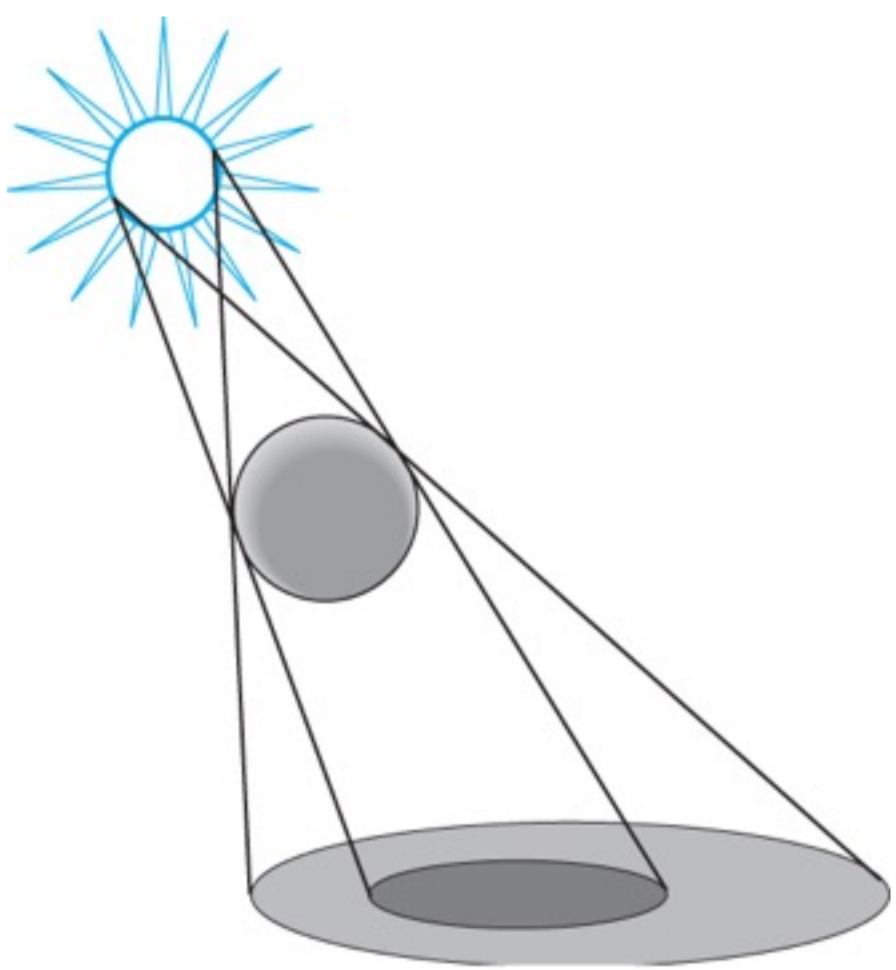
Point light sources alone aren't too realistic
- **add ambient light to mitigate high contrast**



[Angel and Shreiner]

Point light source

Most real-world scenes have large light sources



Point light sources alone aren't too realistic
- drop off intensity more slowly

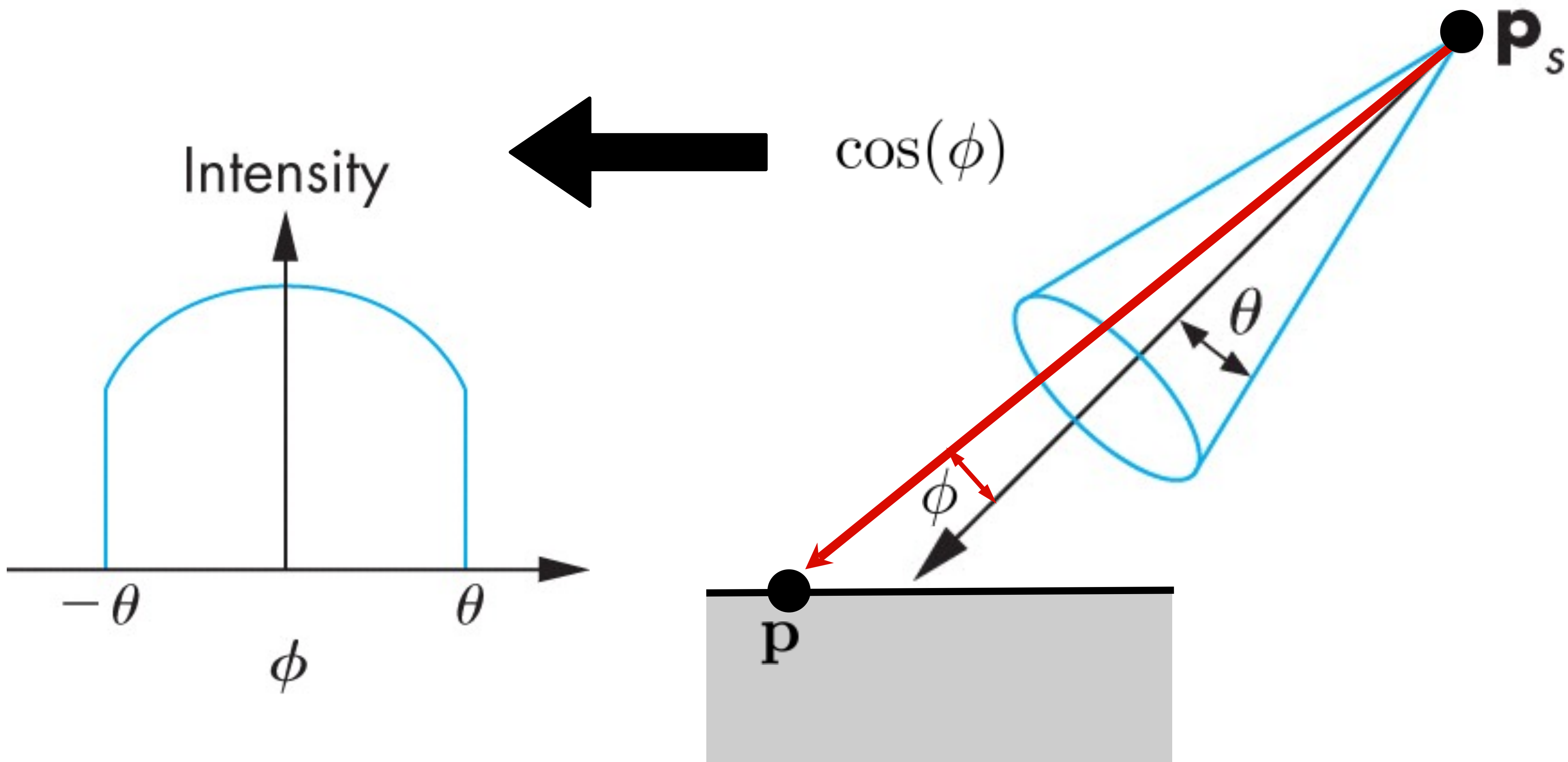
$$l(\mathbf{p}, \mathbf{p}_0) = \frac{1}{d^2}\mathbf{L}(\mathbf{p}_0)$$

$$\Downarrow$$

$$l(\mathbf{p}, \mathbf{p}_0) = \frac{1}{a + bd + cd^2}\mathbf{L}(\mathbf{p}_0)$$

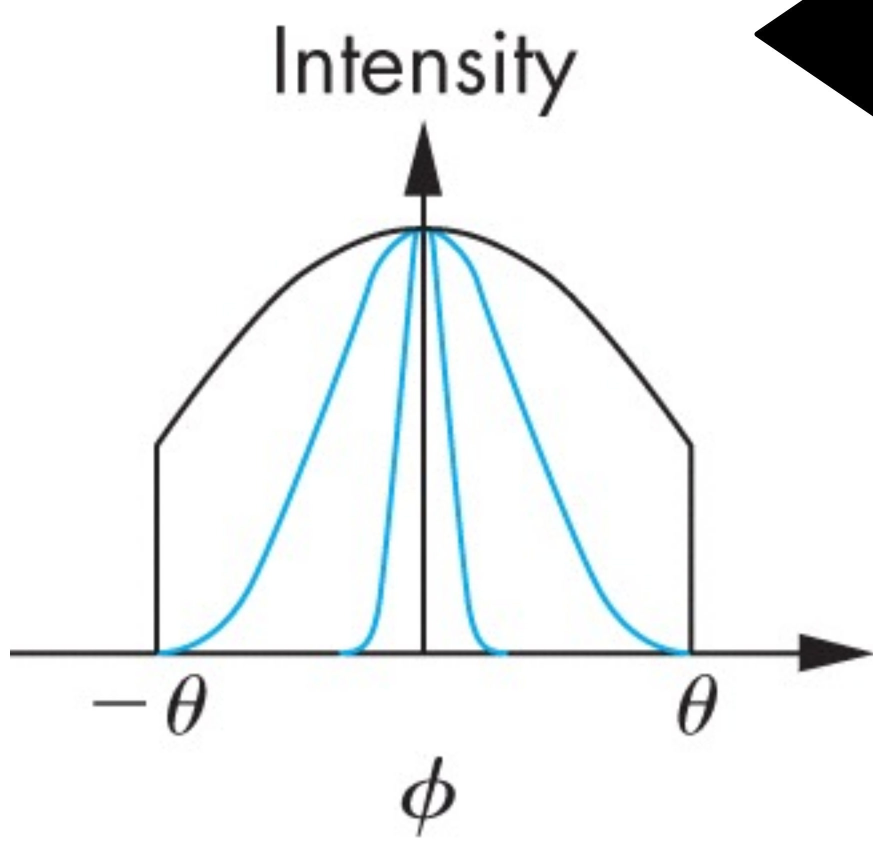
[Angel and Shreiner]

Spotlights

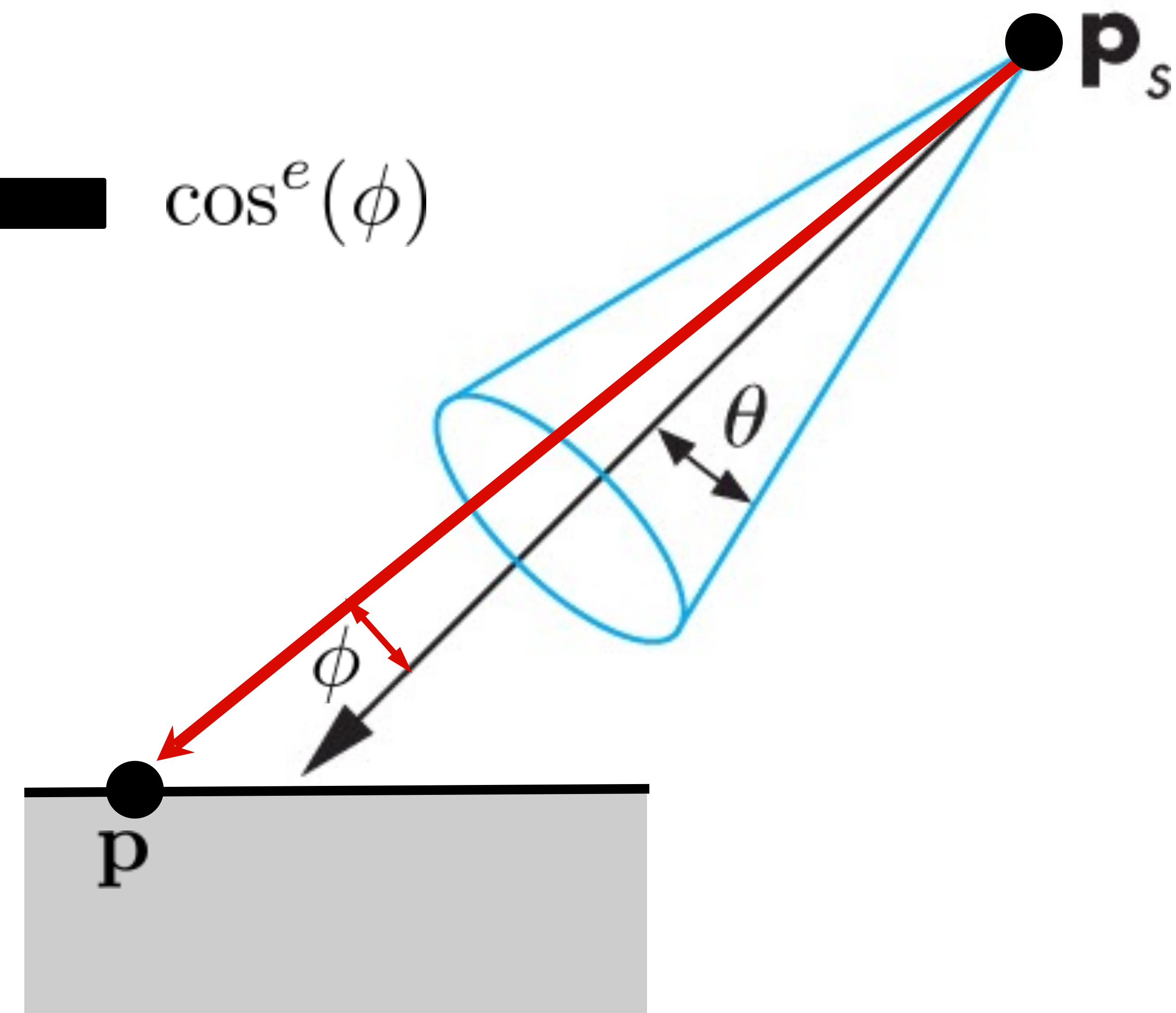


[Angel and Shreiner]

Spotlights



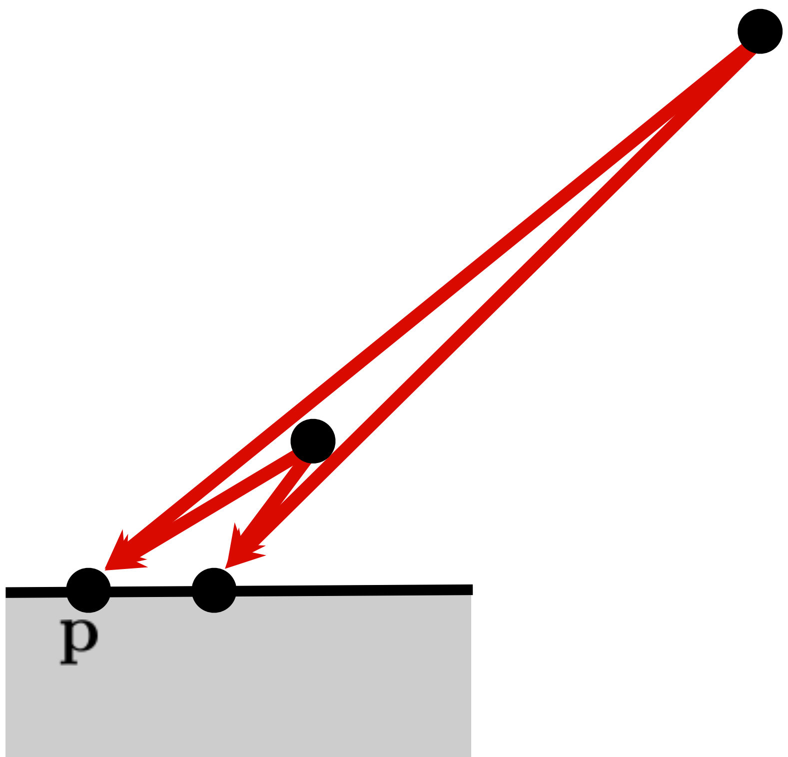
$\cos^e(\phi)$



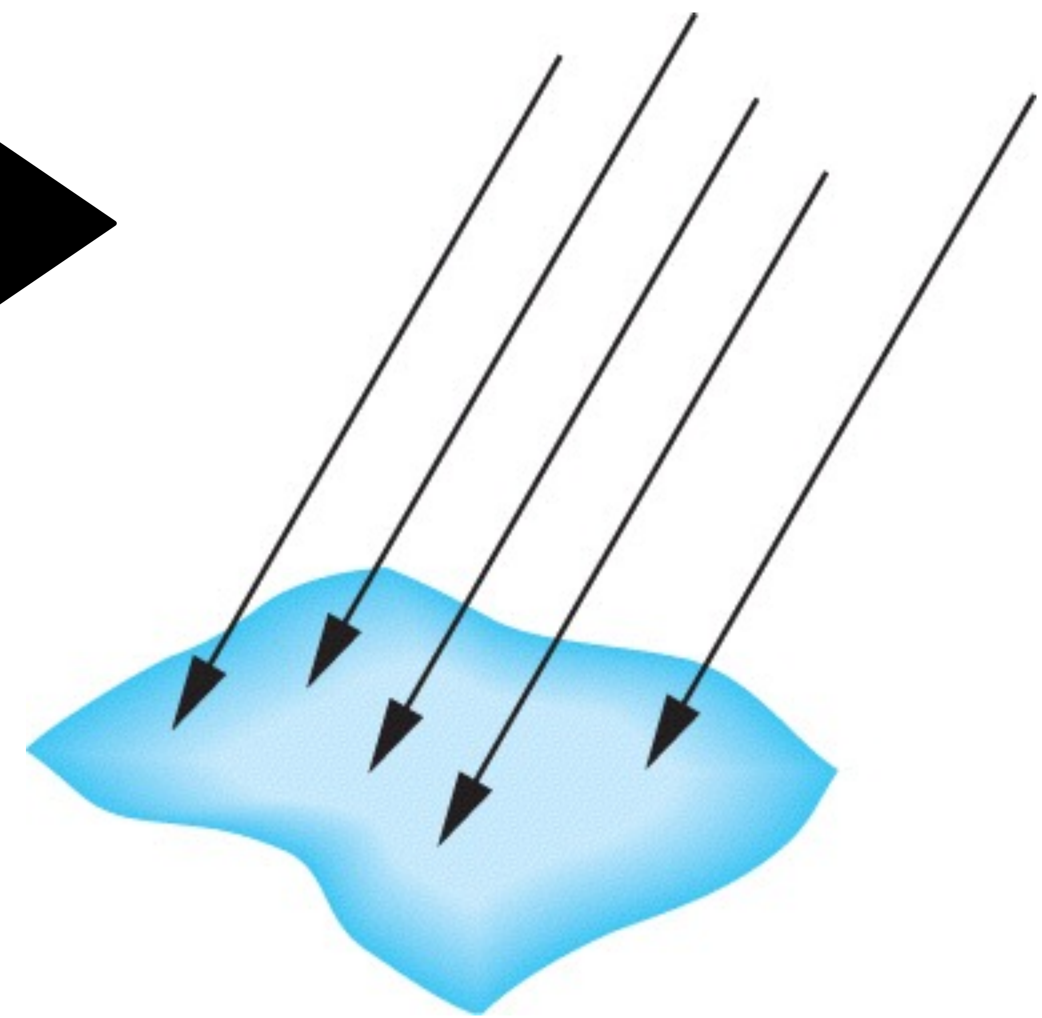
$\cos^e(\phi)\ l(\mathbf{p}, \mathbf{p}_s)$

[Angel and Shreiner]

Distant light source

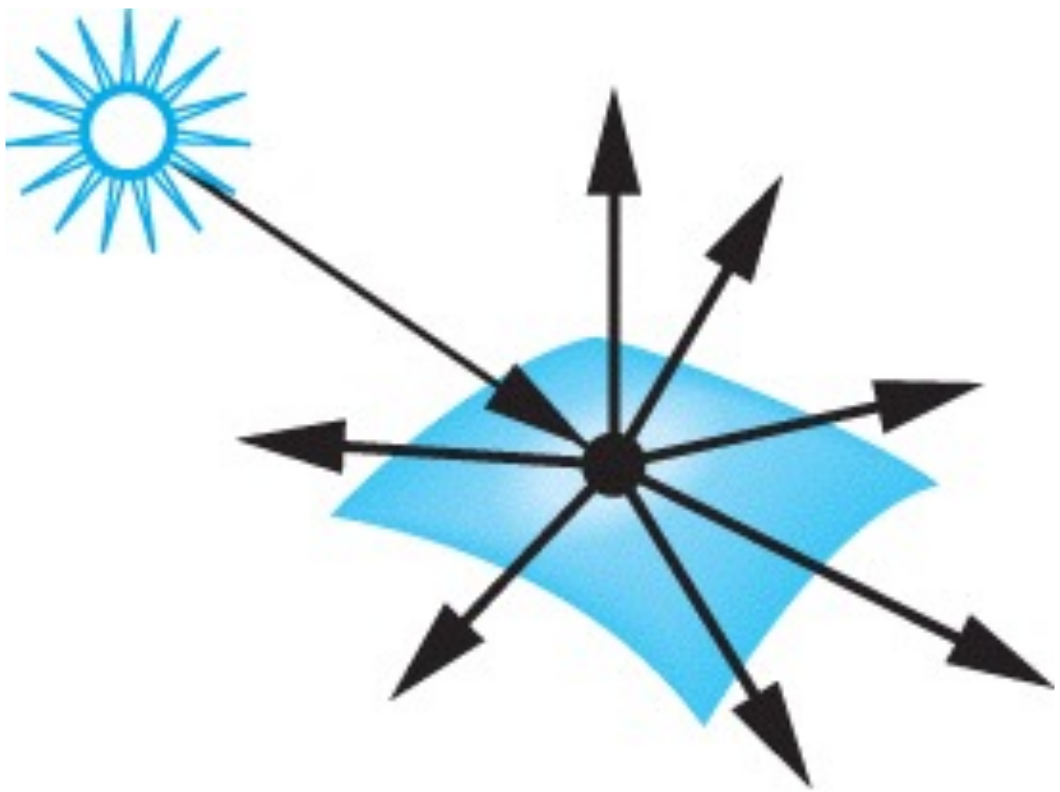


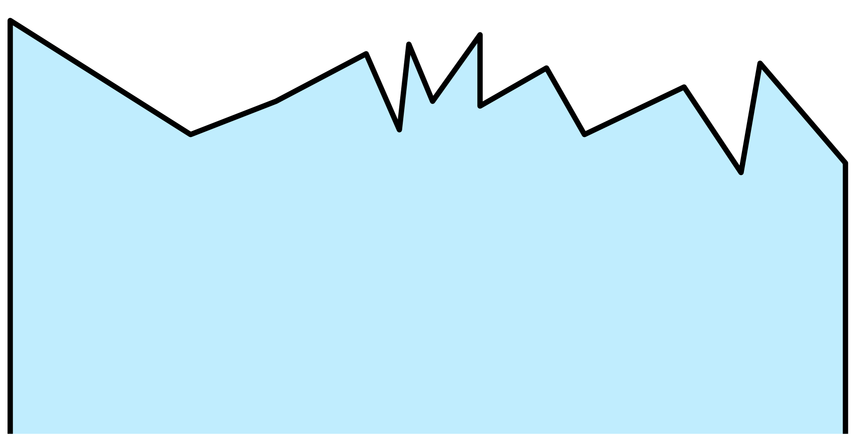
characterized by direction



[Angel and Shreiner]

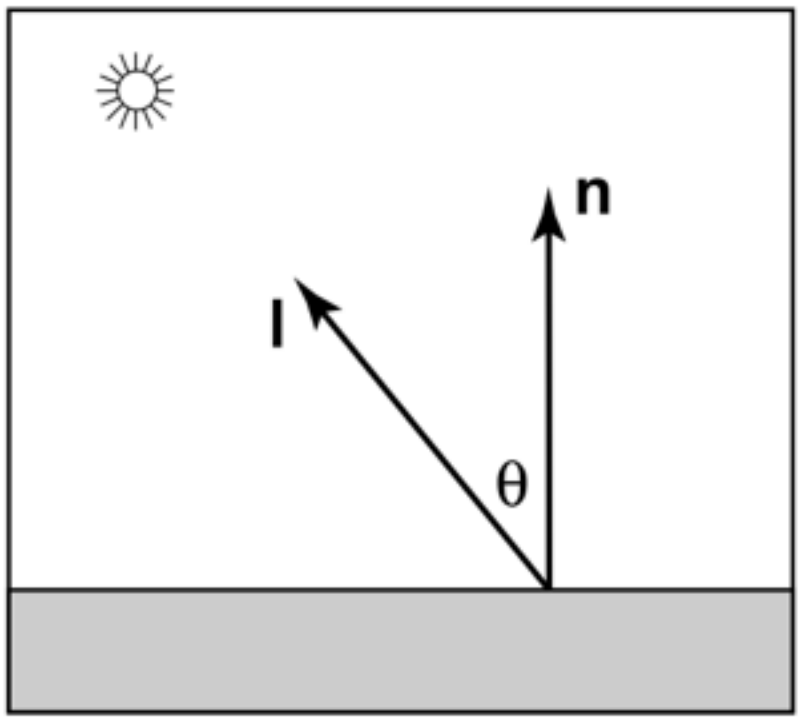
Lambertian Reflection Model

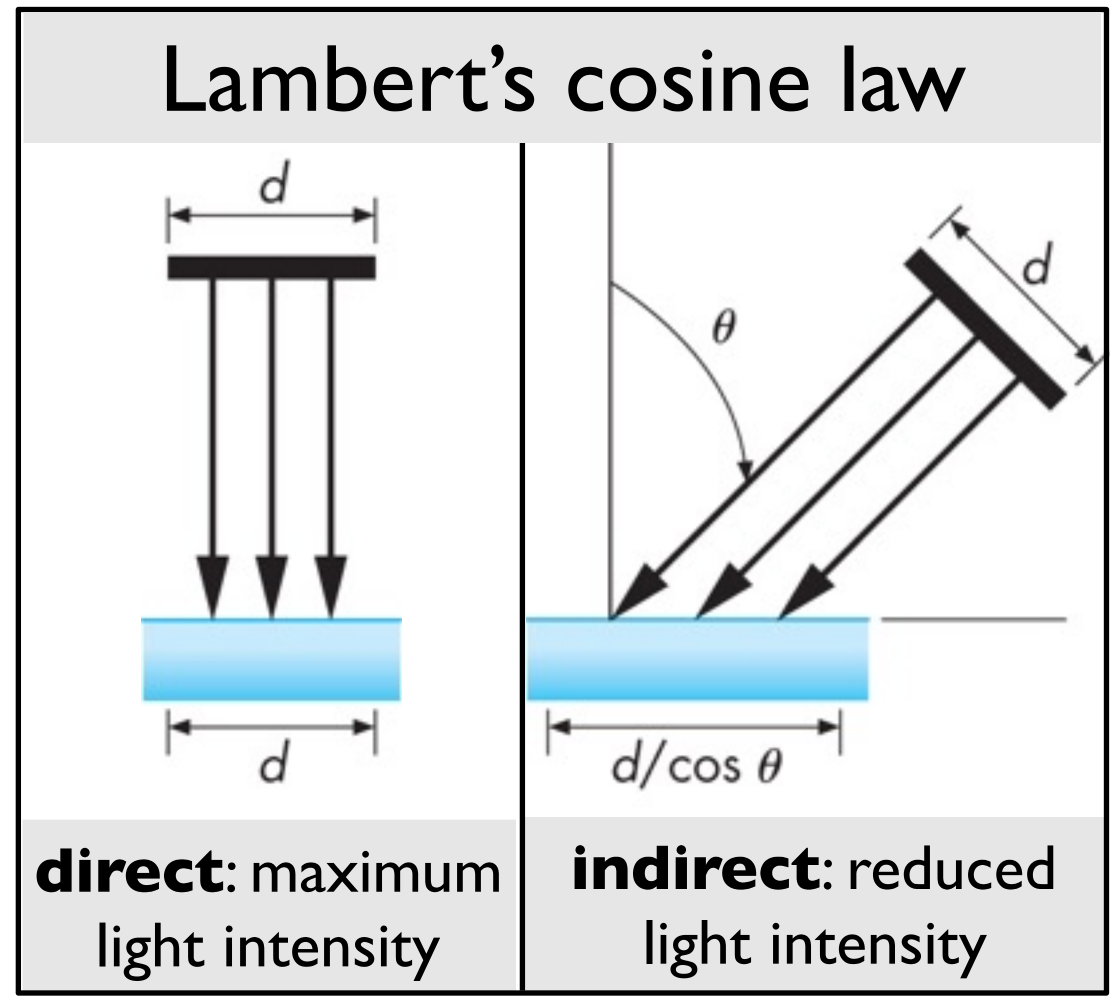




http://www.andreafisherpottery.com/cgi-bin/artistlnk.cgi?Jody_Folwell

Lambertian Reflection Model

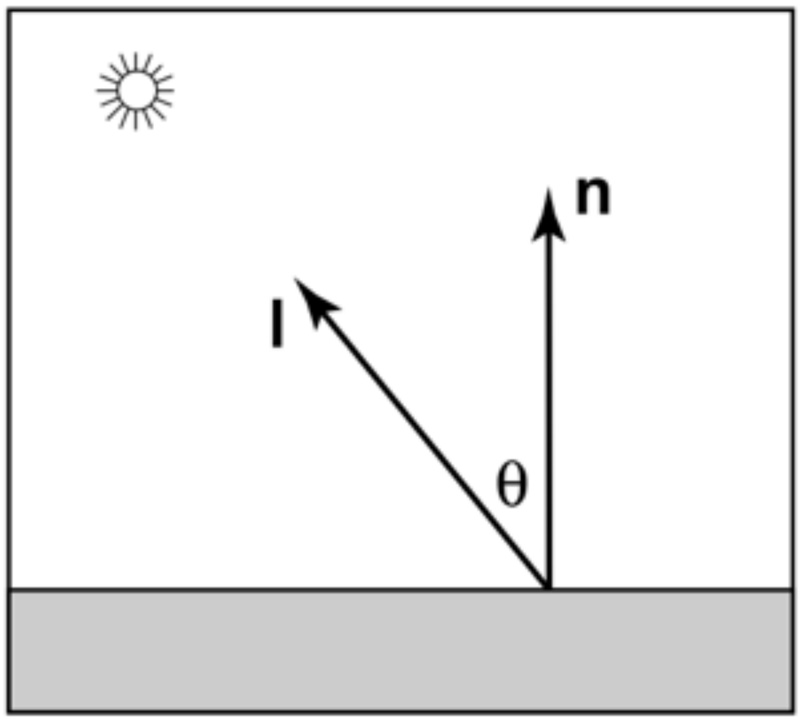




$$I \propto \cos\theta$$

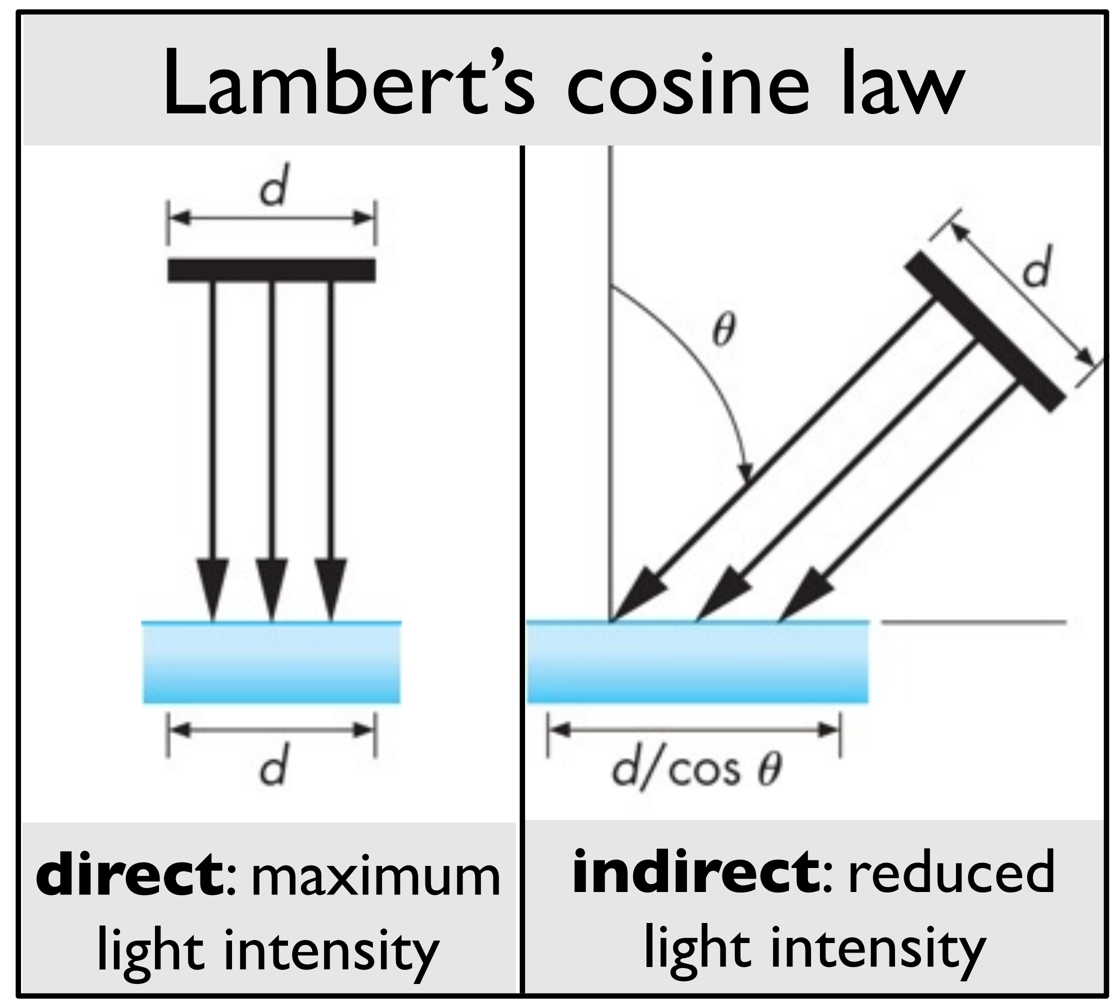
color intensity

Lambertian Reflection Model

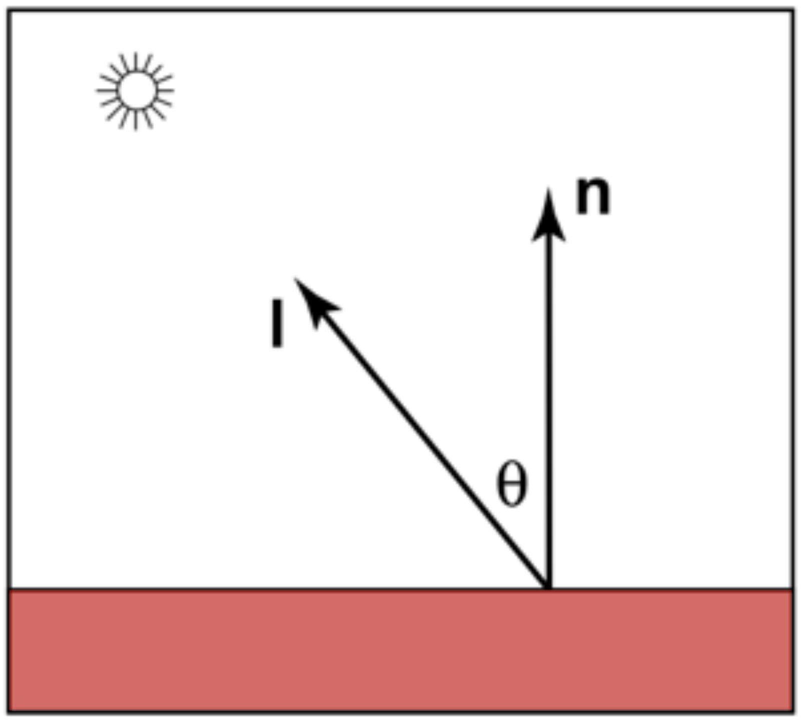


$$I \propto \mathbf{n} \cdot \mathbf{l}$$

color intensity



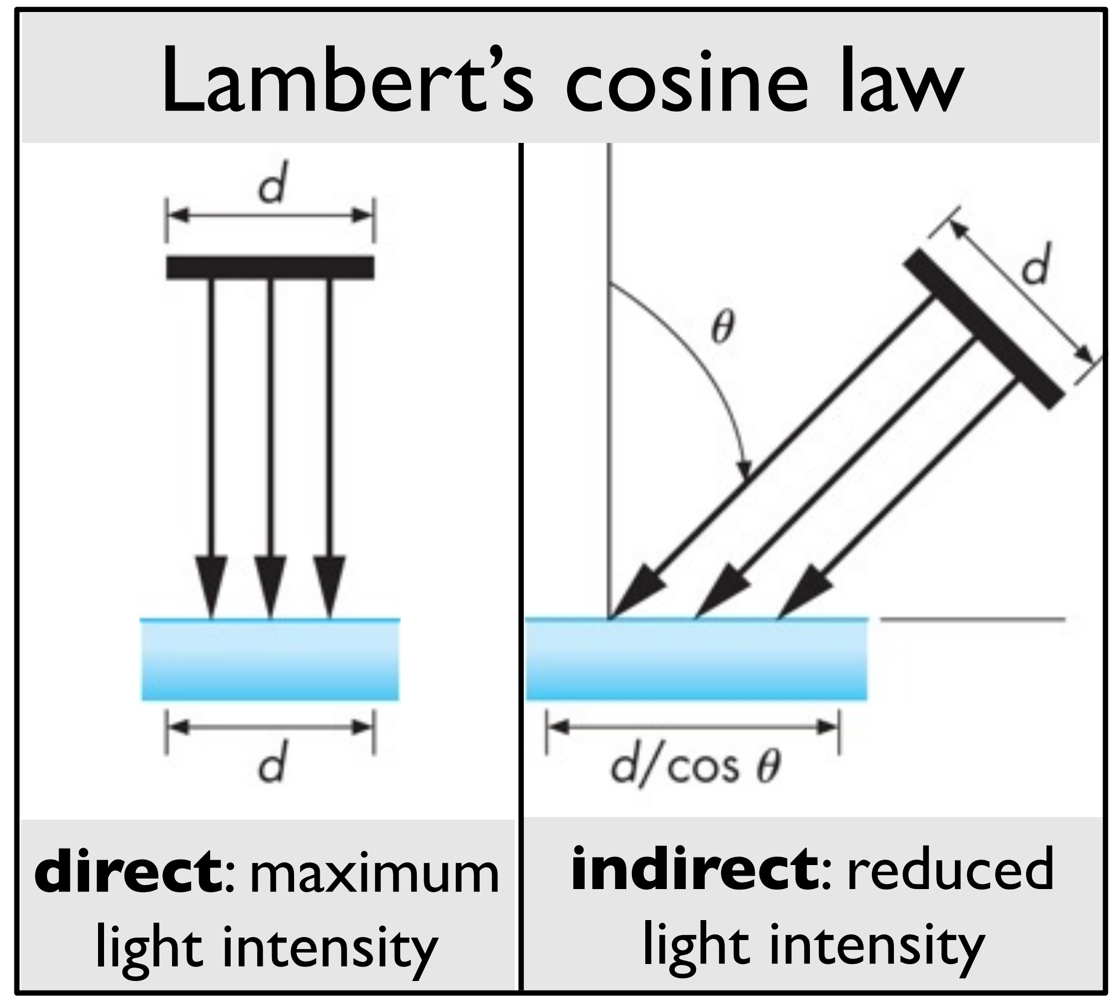
Lambertian Reflection Model



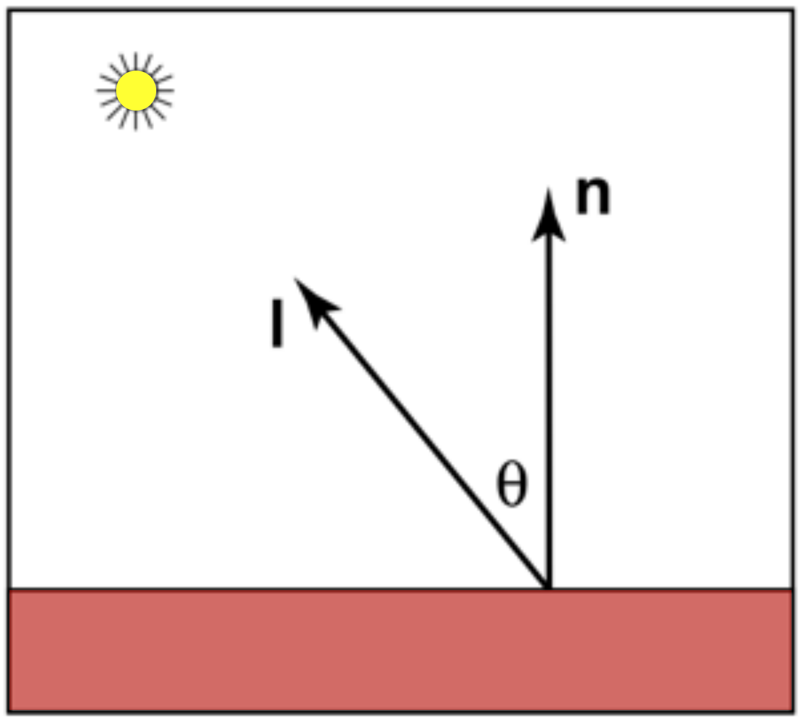
$$I \propto R\mathbf{n} \cdot \mathbf{l}$$

color intensity

reflectance



Lambertian Reflection Model

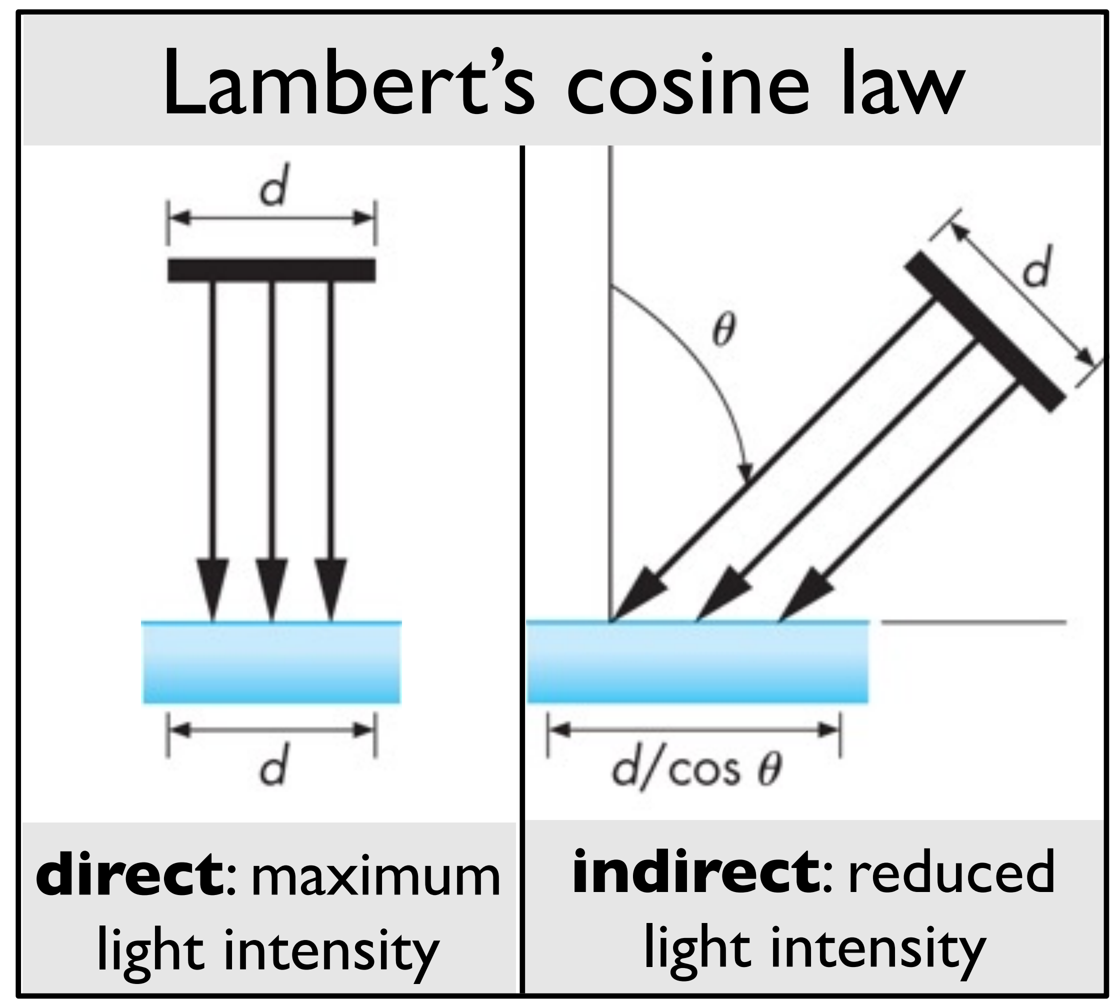


illumination

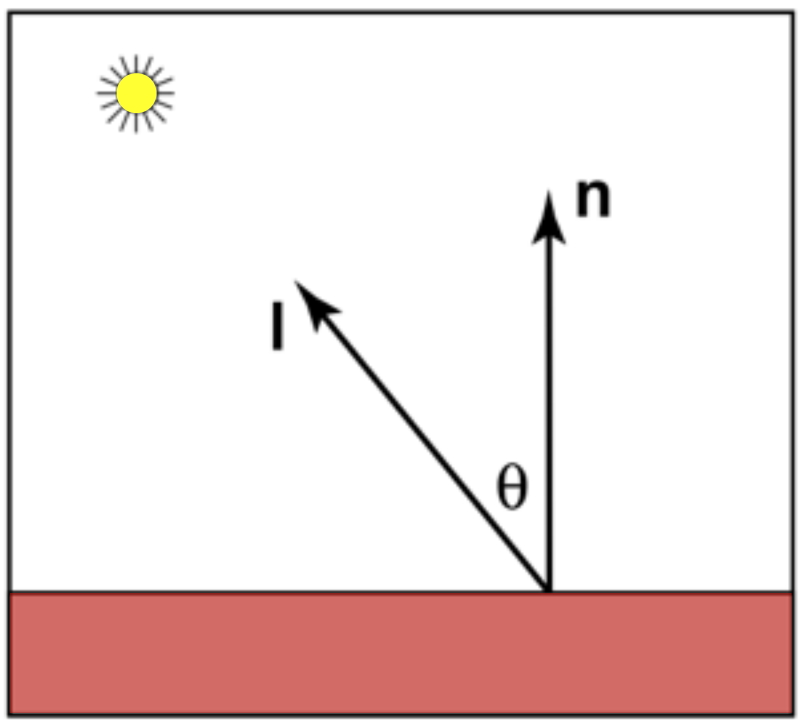
$$I = LR\mathbf{n} \cdot \mathbf{l}$$

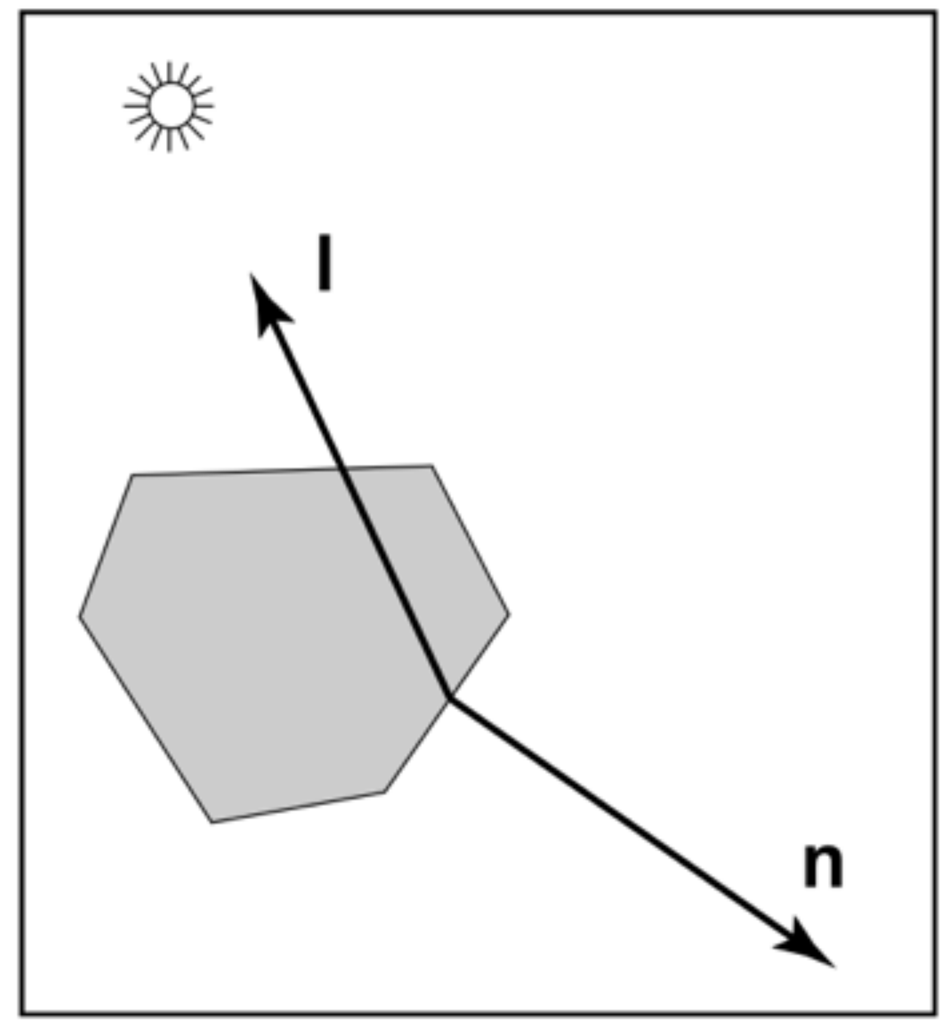
color intensity

reflectance



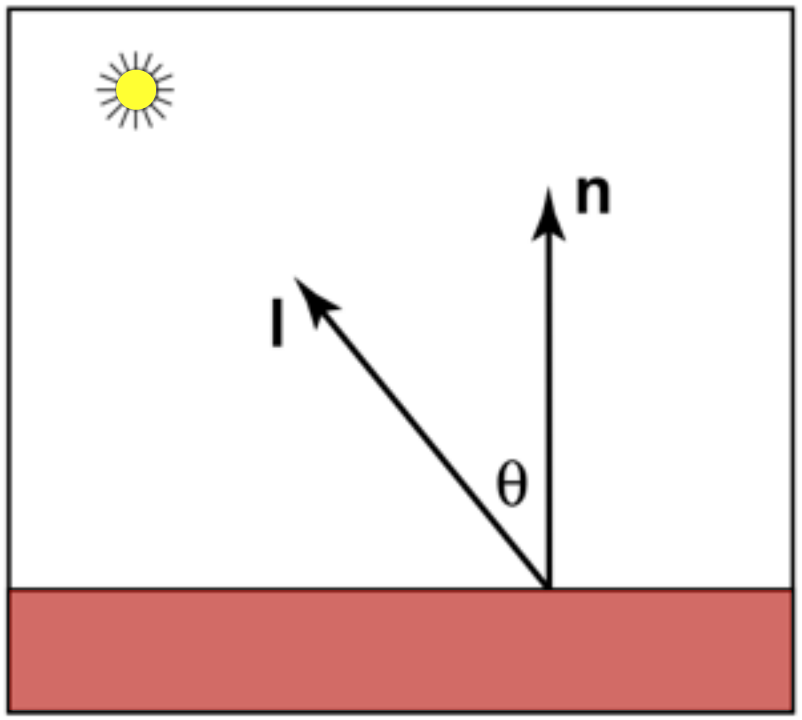
Lambertian Reflection Model



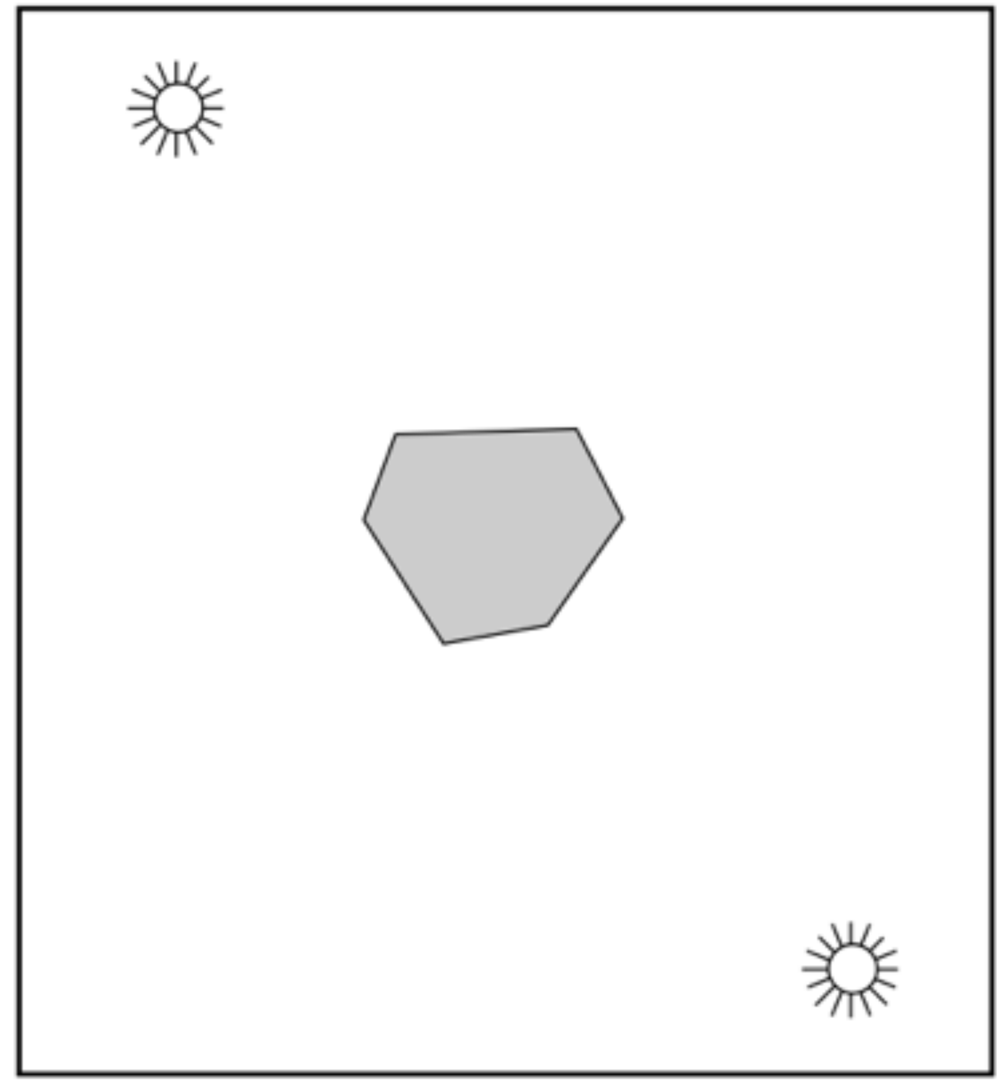


$$I = LR\max(0, \mathbf{n} \cdot \mathbf{l})$$

Lambertian Reflection Model



$$I = LR|\mathbf{n} \cdot \mathbf{l}|$$

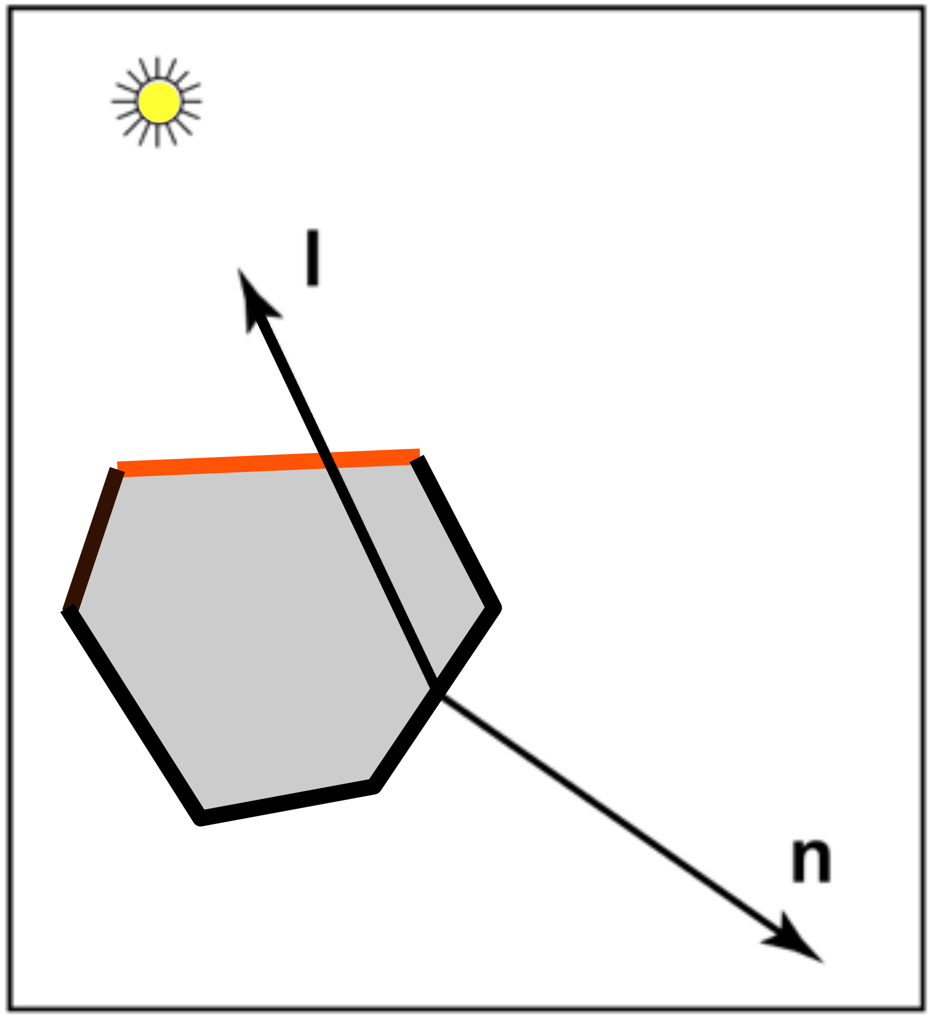


two-sided lighting

Ambient Reflection

$$I = LR \max(0, \mathbf{n} \cdot \mathbf{l})$$

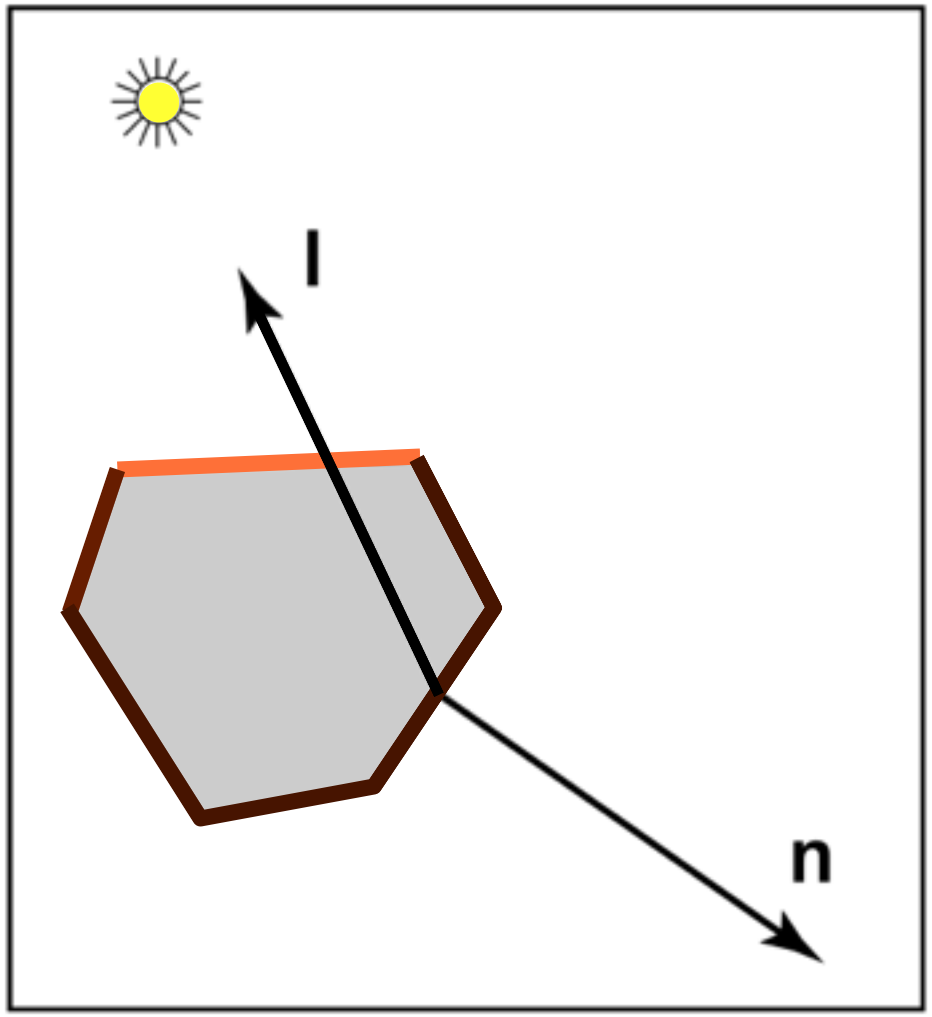
Surfaces facing away from the light will be totally **black**



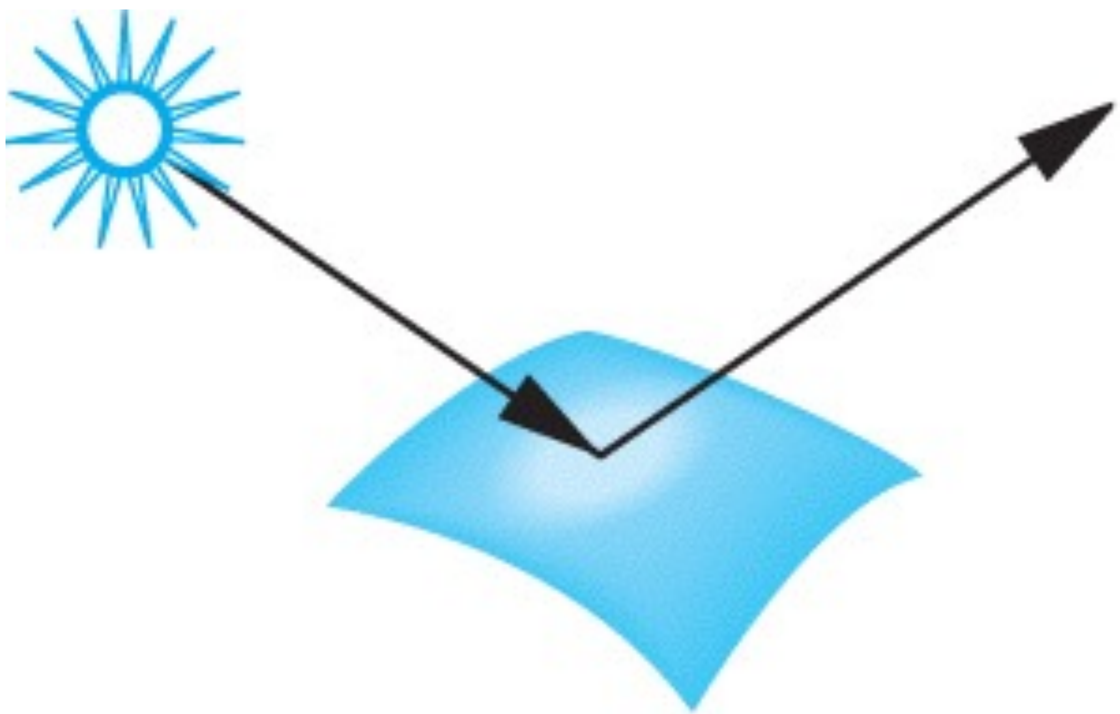
Ambient Reflection

$$I = L_a R_a + L_d R_d \max(0, \mathbf{n} \cdot \mathbf{l})$$

All surfaces get same amount of ambient light

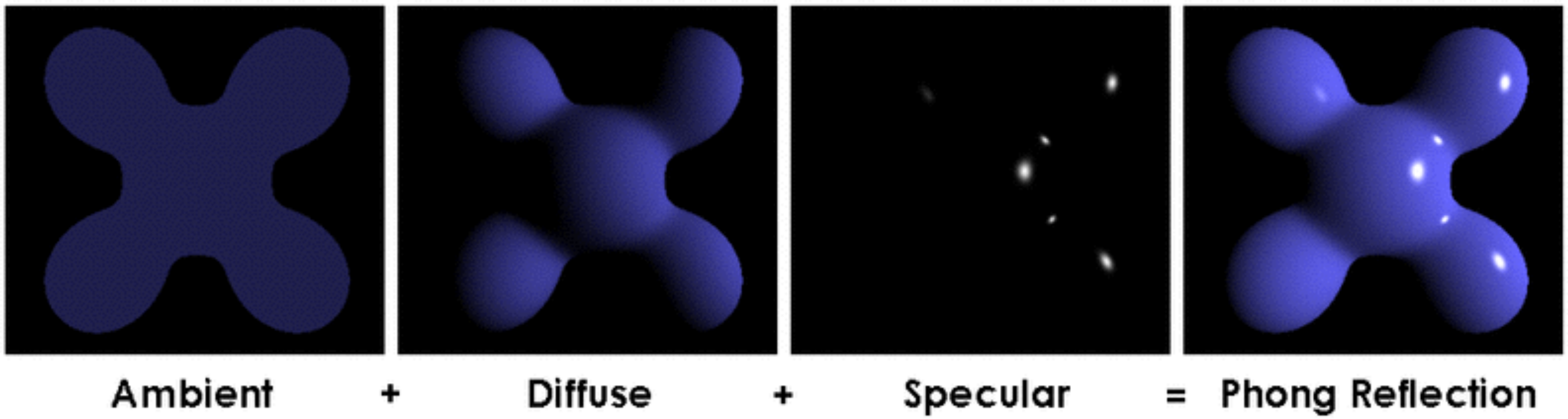


Phong Reflection Model



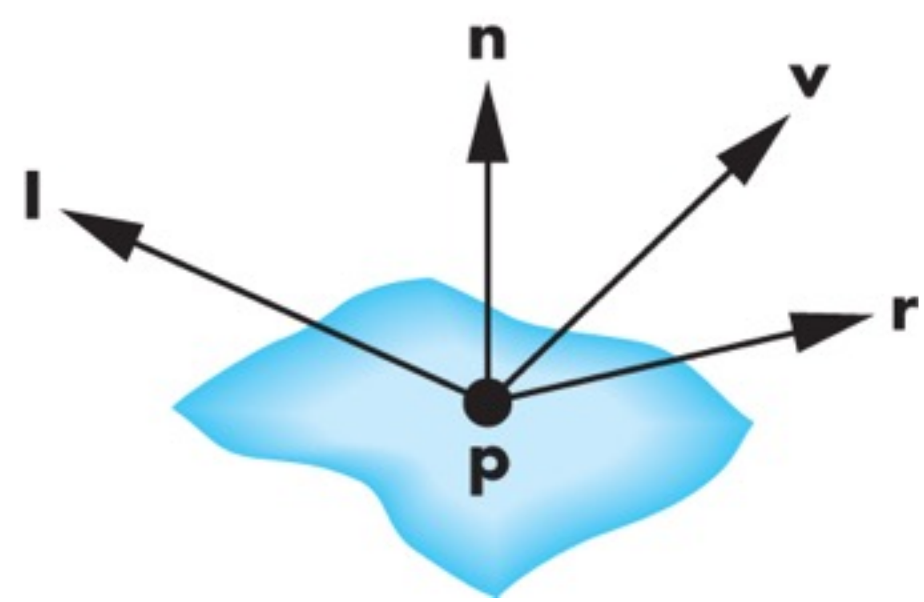
http://www.andreafisherpottery.com/cgi-bin/artistlnk.cgi?Jody_Folwell

Phong Reflection Model

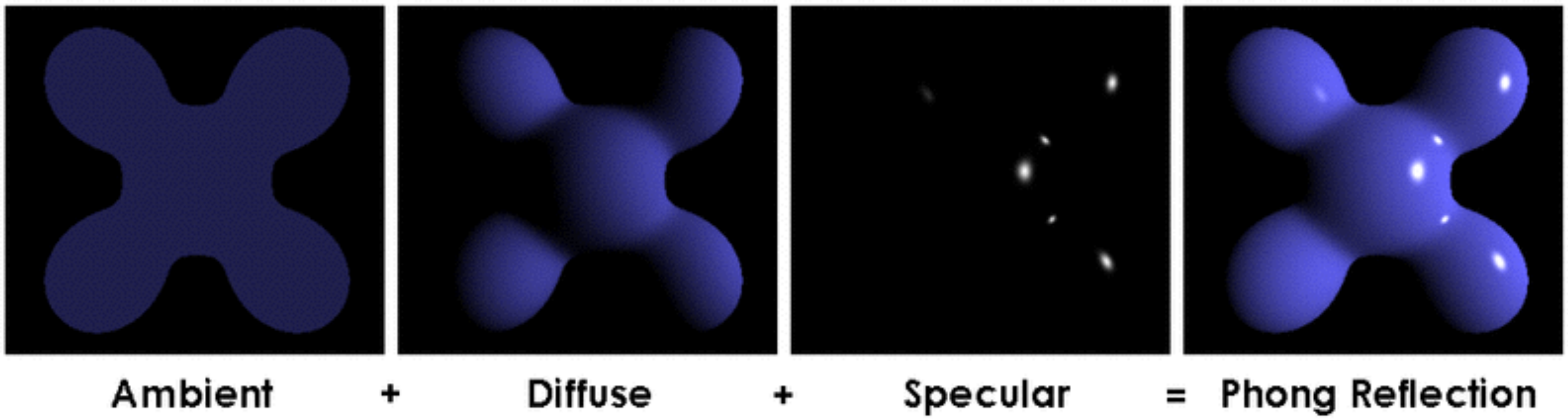


[Brad Smith, Wikimedia Commons]

- efficient, reasonably realistic
- 3 components
- 4 vectors



Phong Reflection Model



[Brad Smith, Wikimedia Commons]

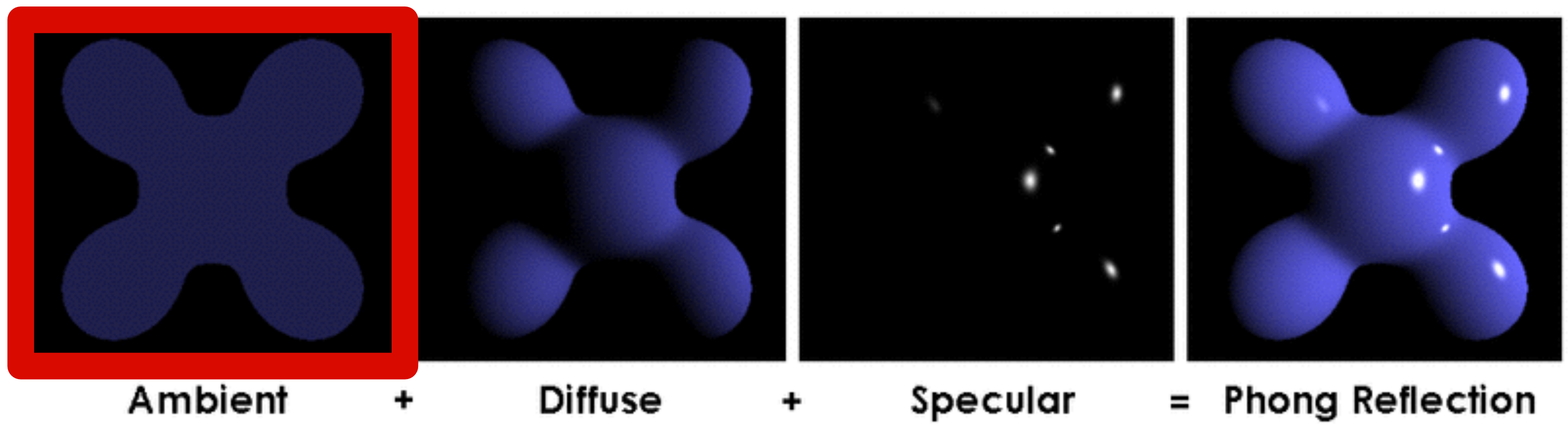
$$
\begin{aligned}
I &= I_a + I_d + I_s \\
&= R_a L_a + R_d L_d \max(0, \mathbf{l} \cdot \mathbf{n}) + R_s L_s \max(0, \cos\phi)^\alpha
\end{aligned}
$$

color intensity

reflectance

illumination

Ambient reflection

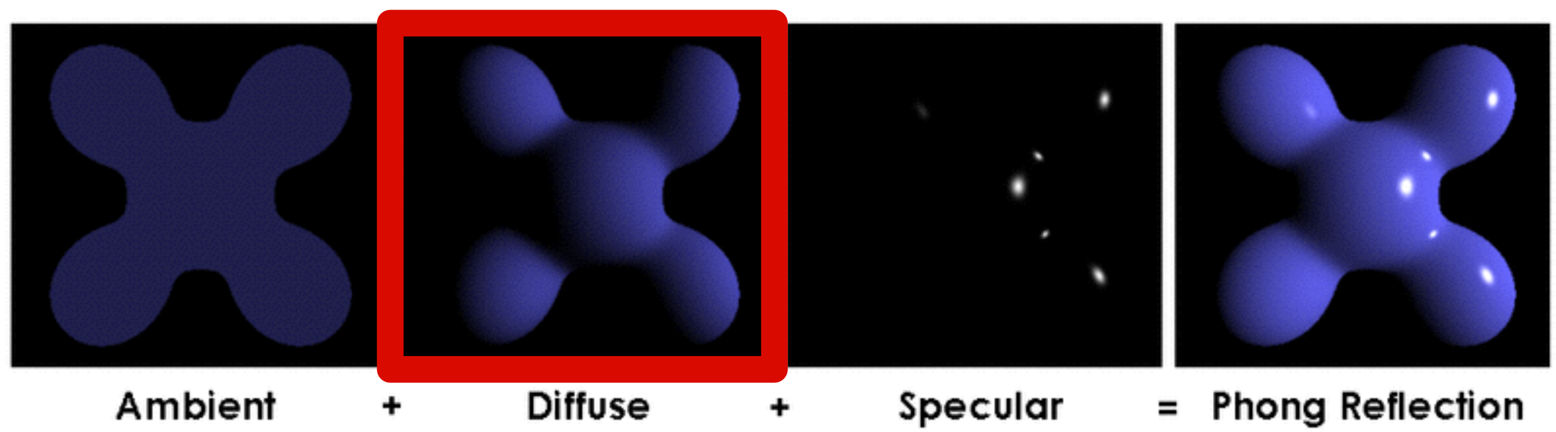


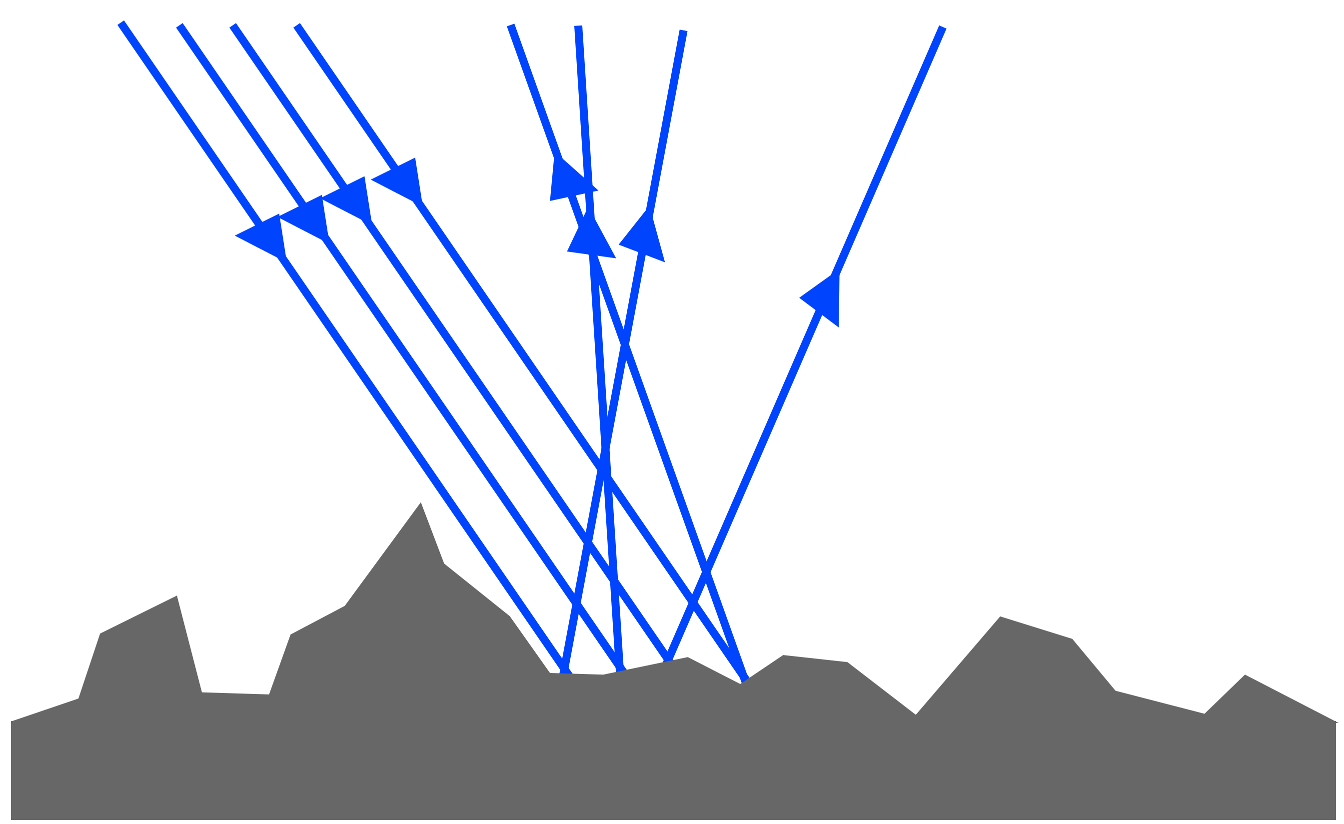
different ambient coefficients for different colors

$$I_a = R_a L_a, \qquad 0 \le R_a \le 1$$

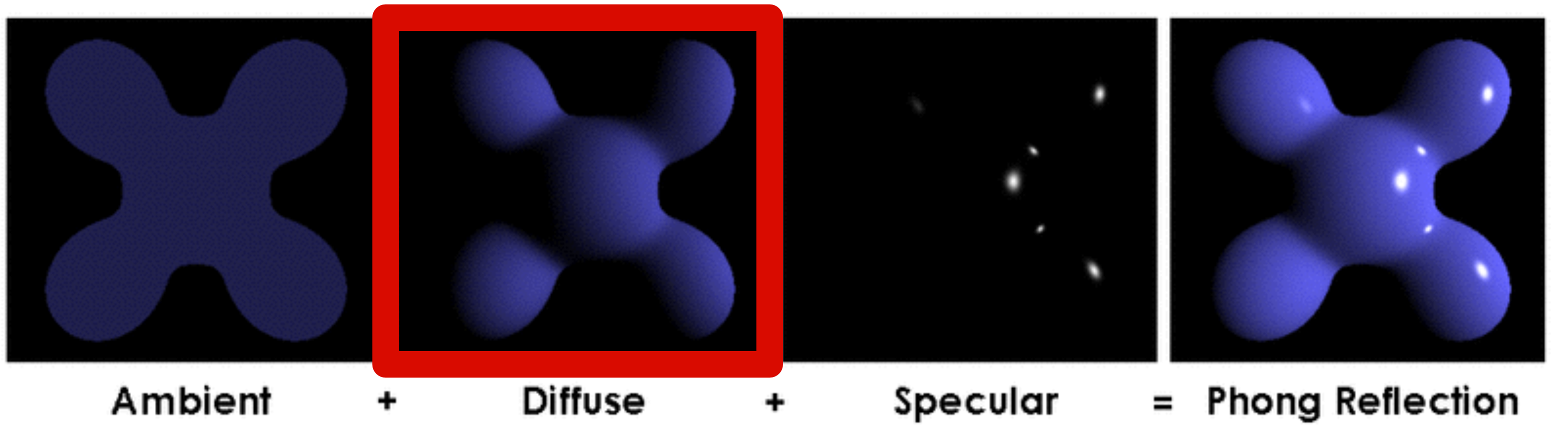
ambient reflection coefficient

Diffuse reflection



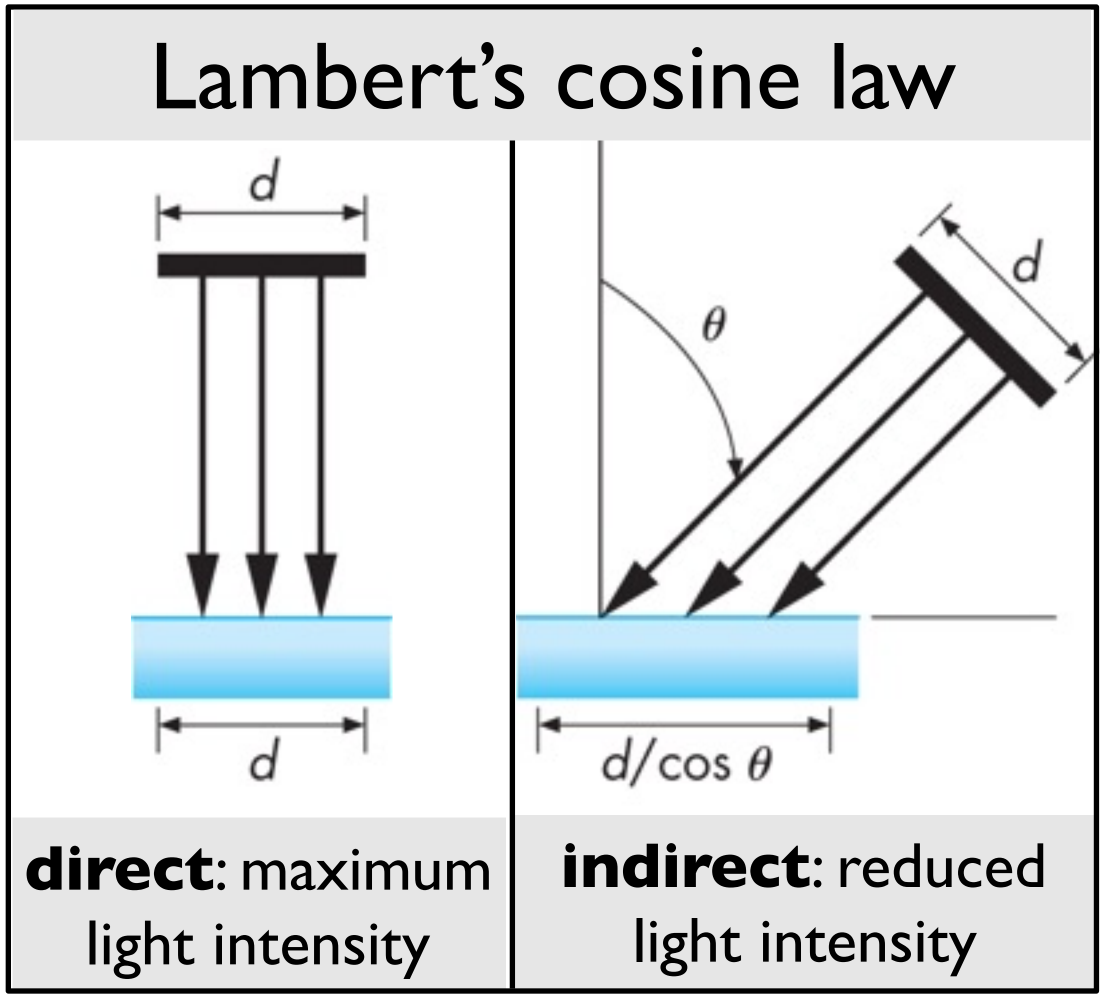


Diffuse reflection

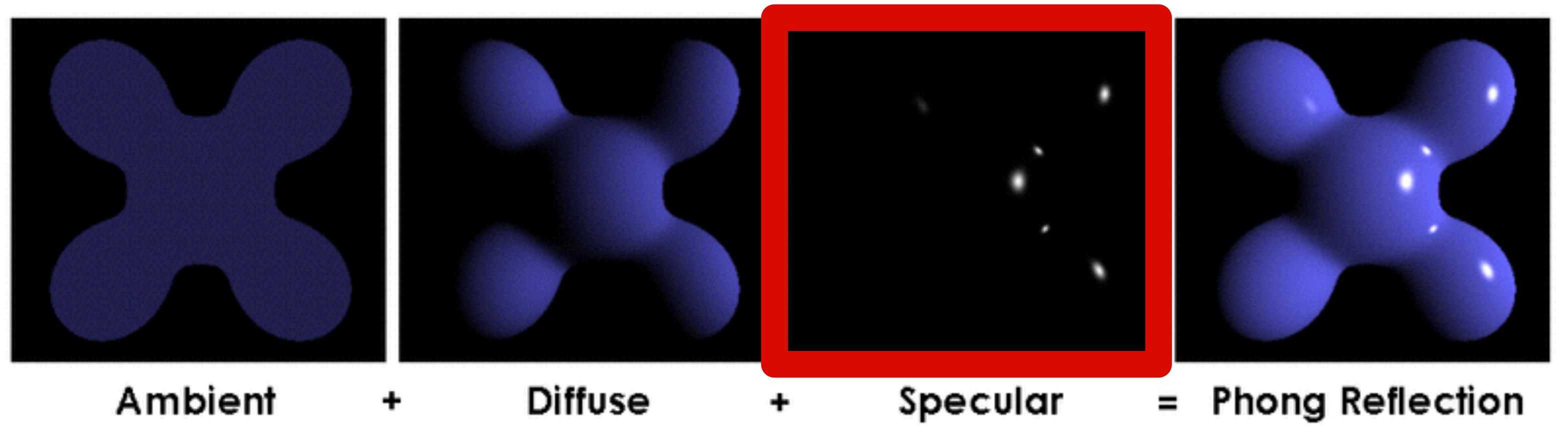


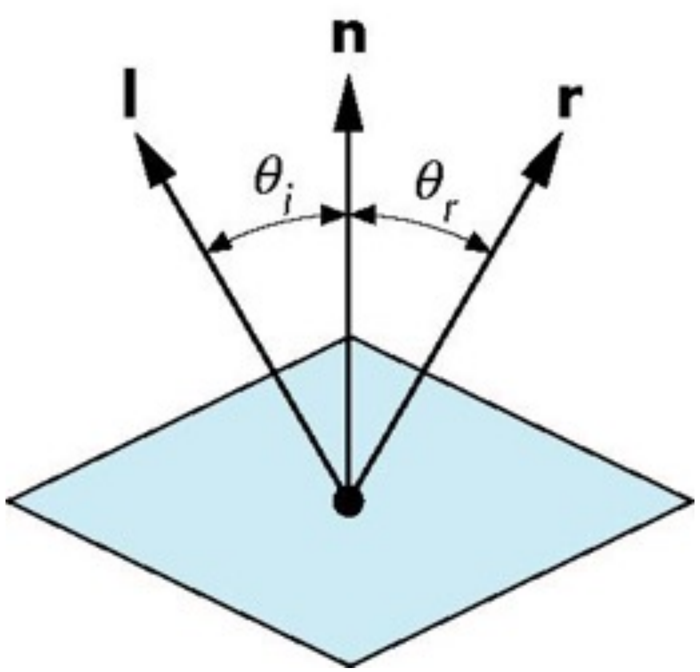
$$I_d = R_d L_d \max(0, \mathbf{l} \cdot \mathbf{n})$$

diffuse reflection coefficient



Specular reflection





Ideal reflector

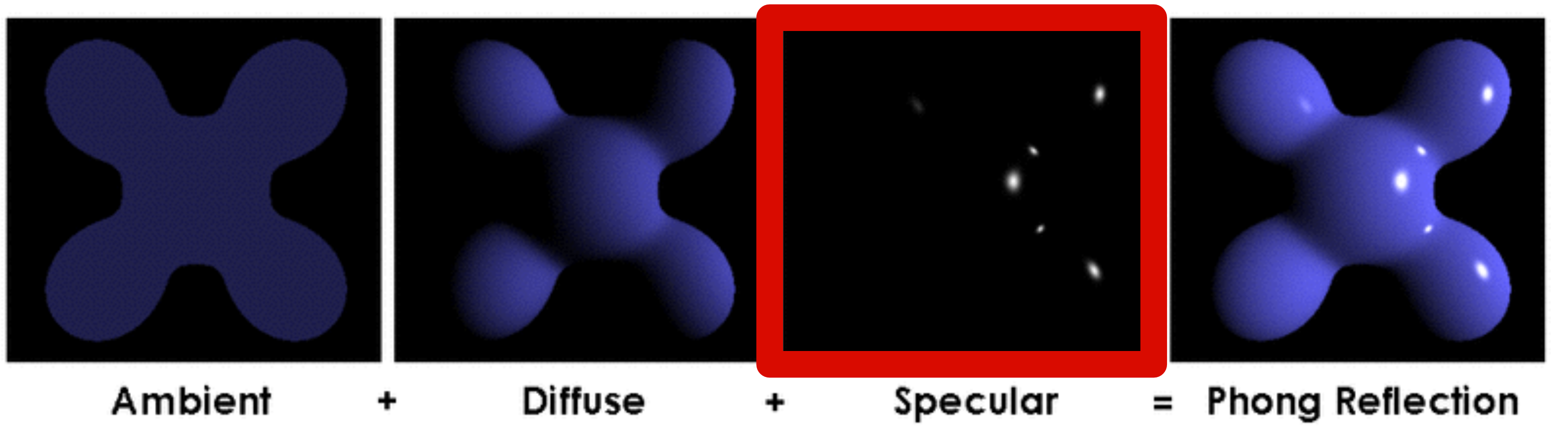
$$\theta_i = \theta_r$$

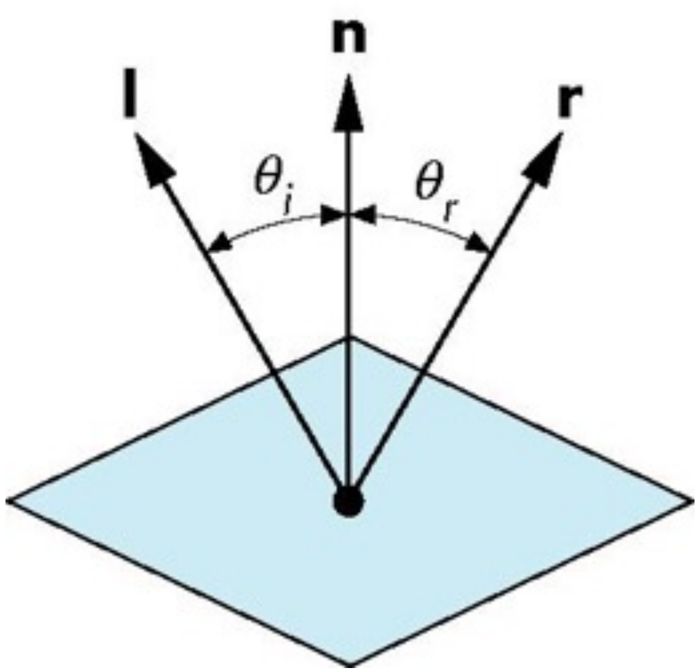
angle of incidence

angle of reflection

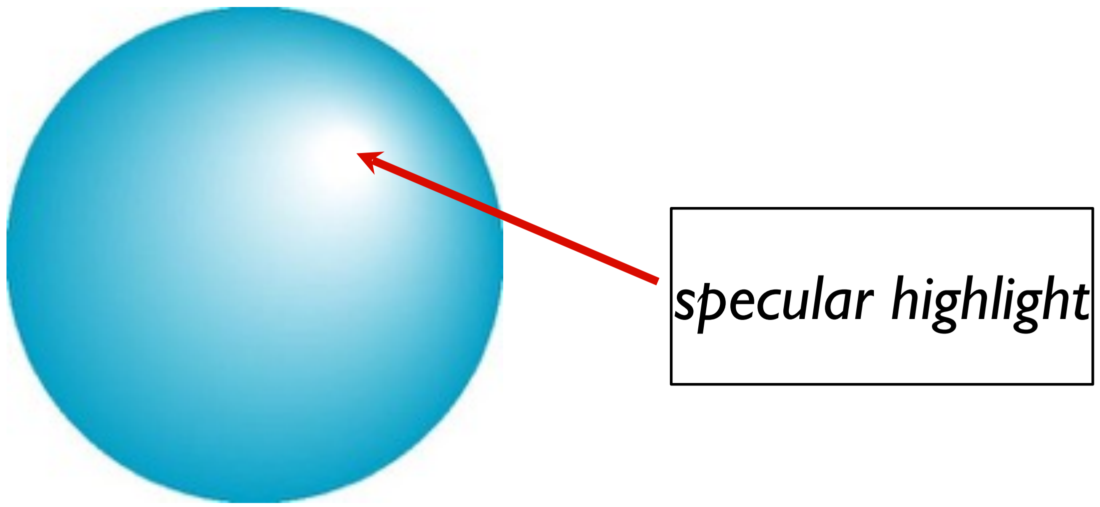
r is the mirror reflection direction

Specular reflection



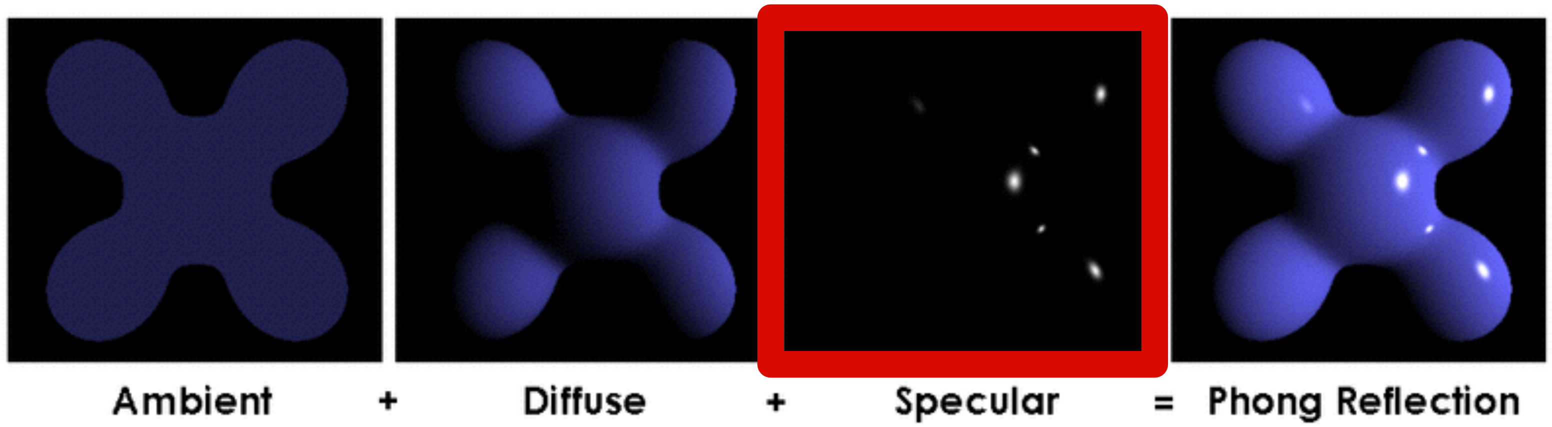


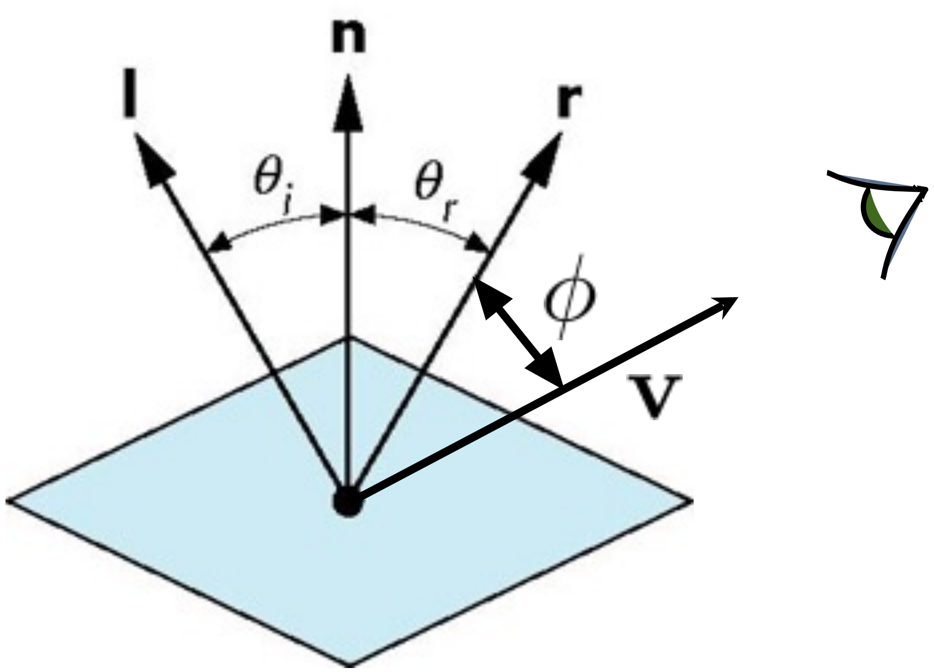
Specular surface



specular reflection is strongest in mirror reflection direction

Specular reflection





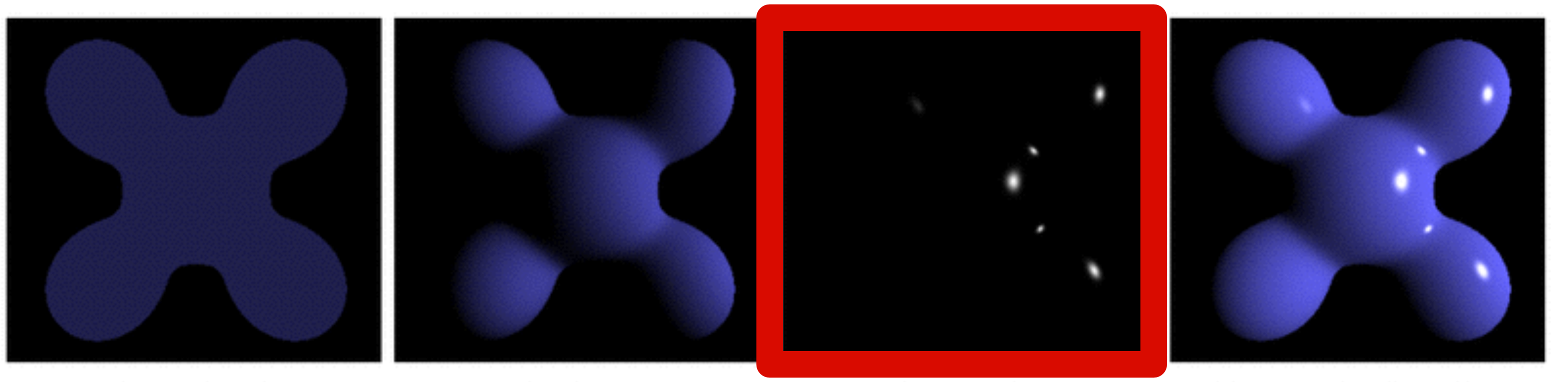
$$I_s = R_s L_s \cos^{\alpha} \phi$$

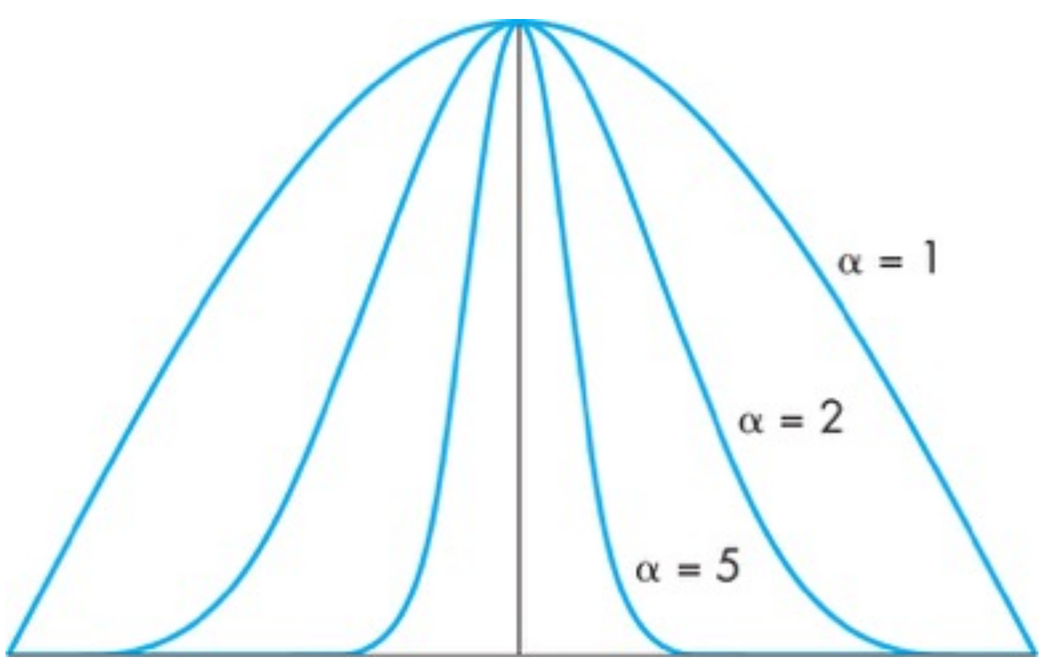
specular reflection coefficient

Phong exponent

specular reflection drops off with increasing angle ϕ

Specular reflection





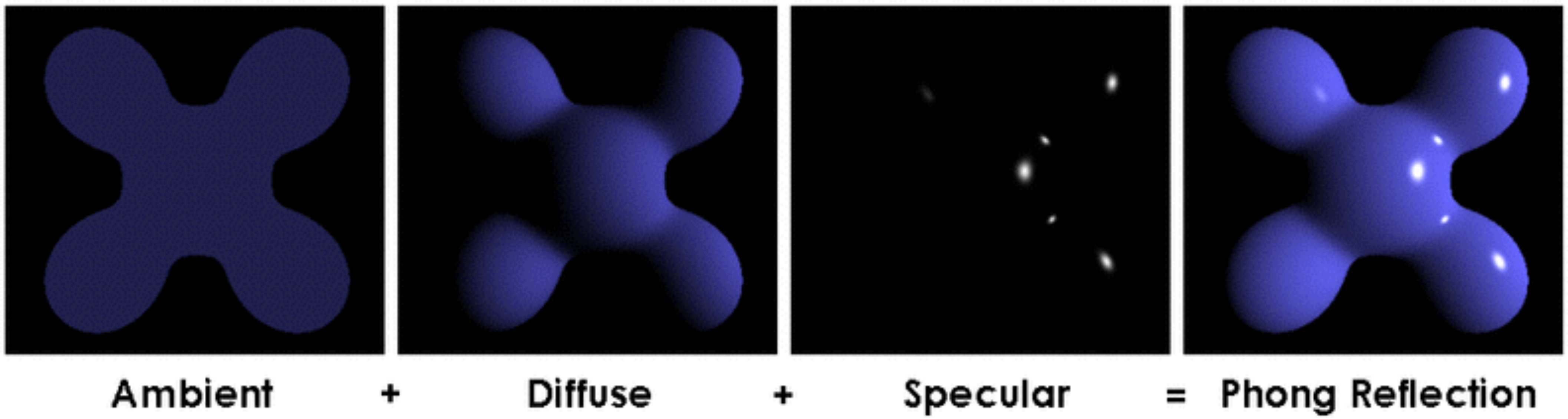
$$I_s = R_s L_s \max(0, \cos\phi)^{\alpha}$$

Phong exponent

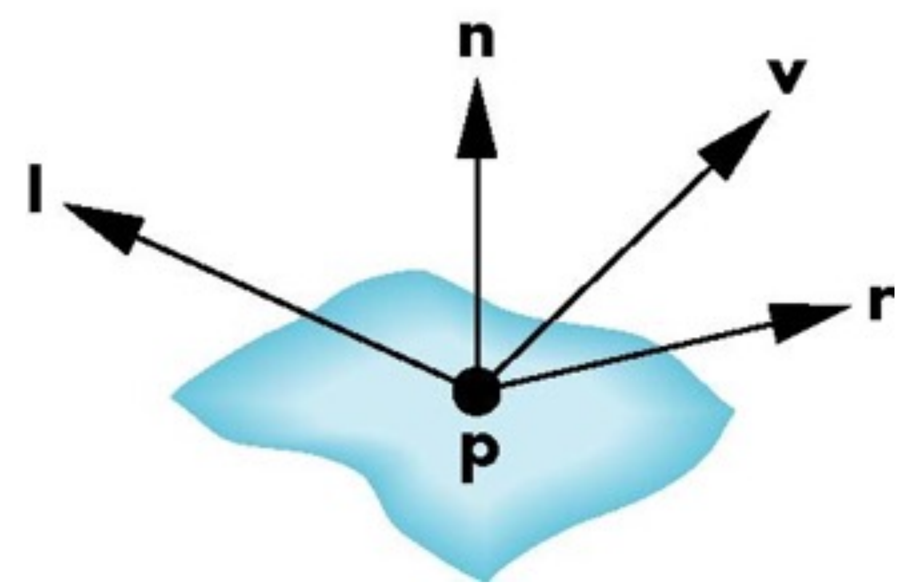
$\alpha = 5..10$ plastic

$\alpha = 100..200$ metal

Phong Reflection Model



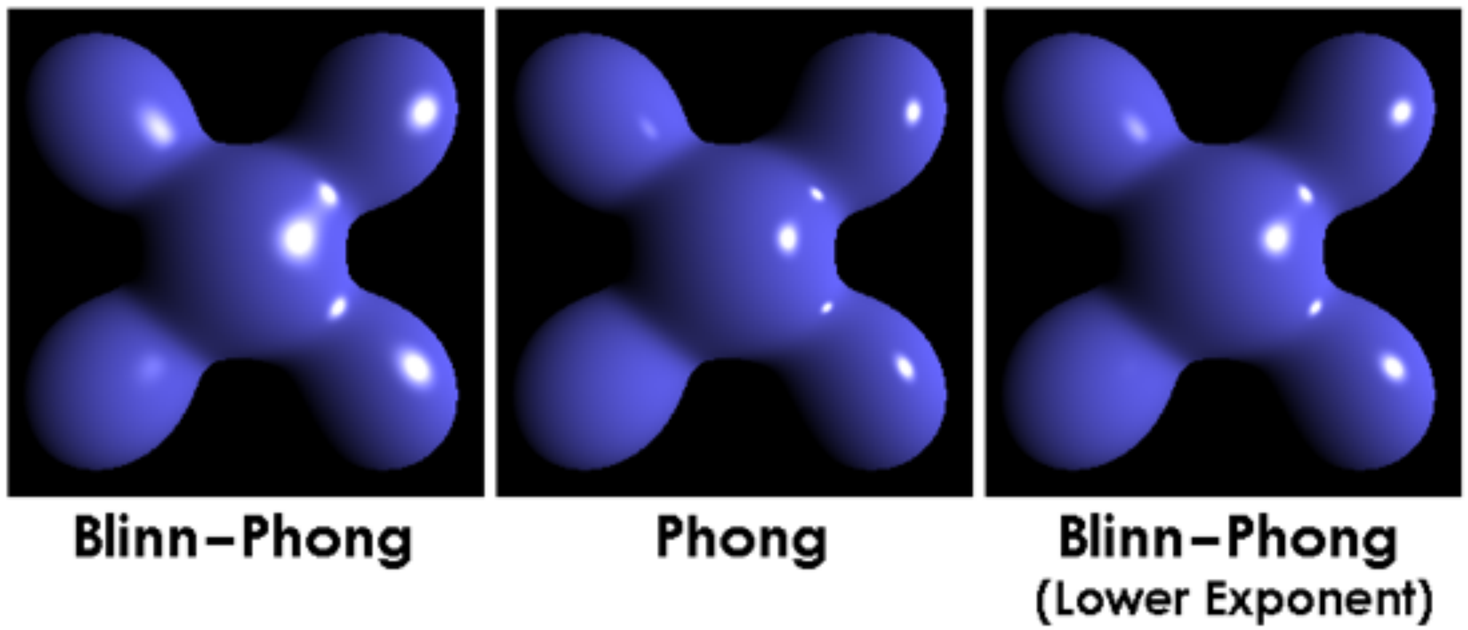
[Brad Smith, Wikimedia Commons]



$$
\begin{aligned}
I &= I_a + I_d + I_s \\
&= R_a L_a + R_d L_d \max(0, \mathbf{l} \cdot \mathbf{n}) + R_s L_s \max(0, \mathbf{v} \cdot \mathbf{r})^{\alpha}
\end{aligned}
$$

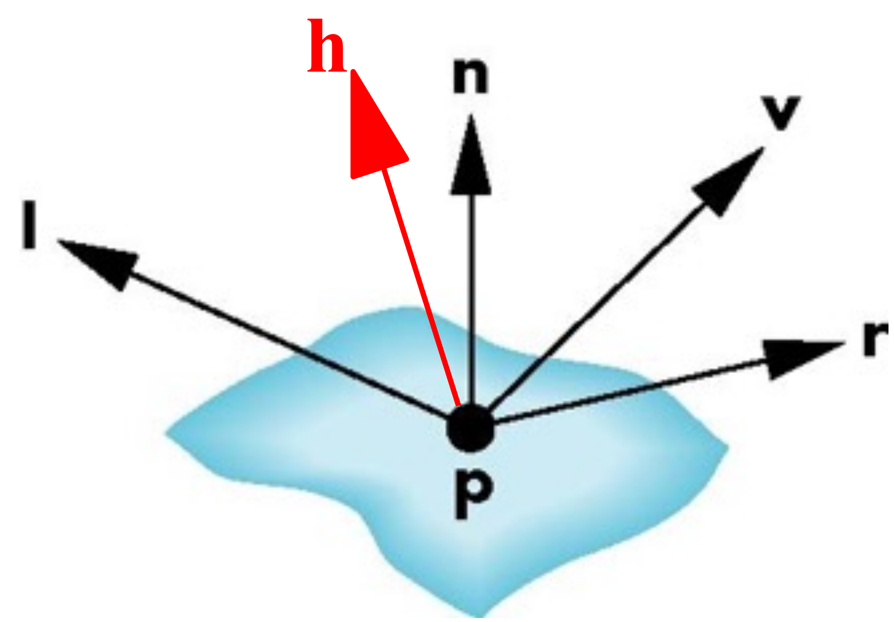
Ambient Diffuse Specular

Alternative: Blinn-Phong Model



[Brad Smith, Wikimedia Commons]

halfway vector $\boxed{\mathbf{h} = \dfrac{\mathbf{l} + \mathbf{v}}{|\mathbf{l} + \mathbf{v}|}}$



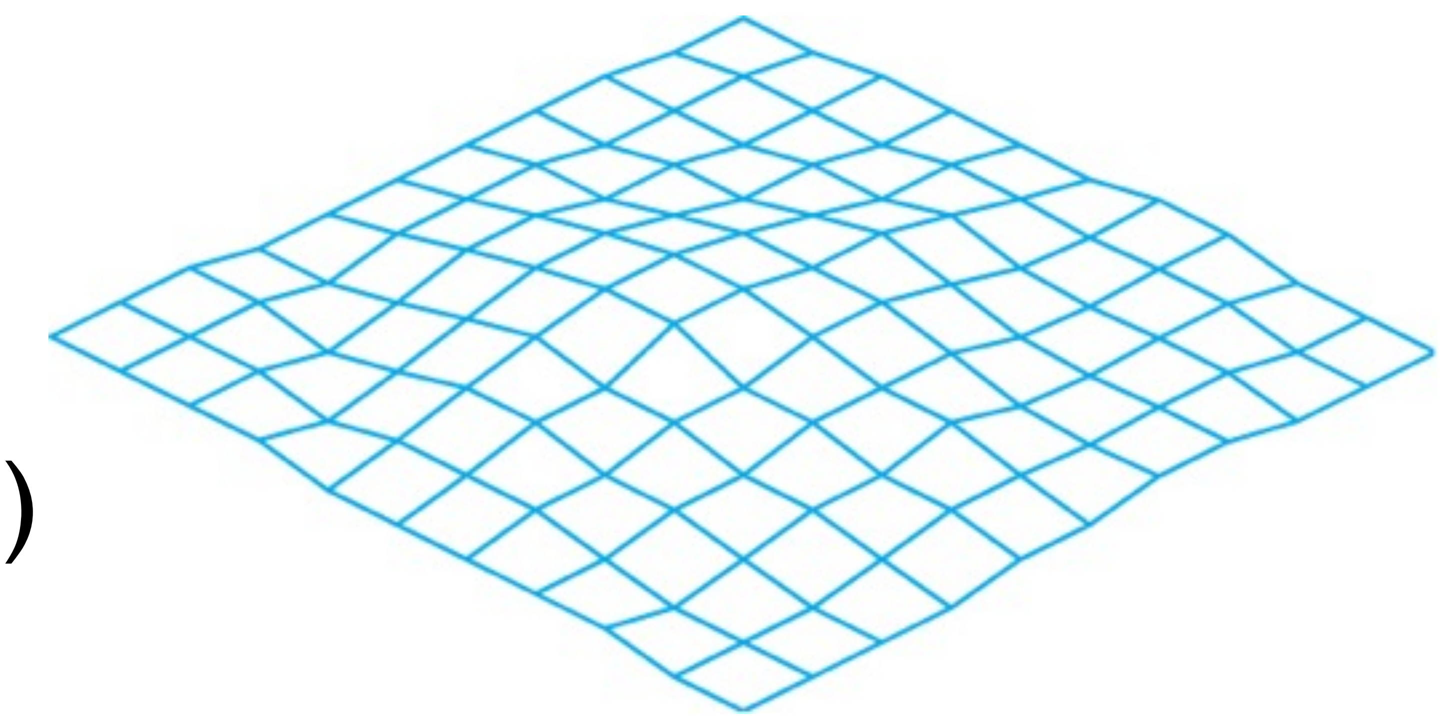
$$\begin{aligned} I &= I_a + I_d + I_s \\ &= \underbrace{R_a L_a}_{\text{Ambient}} + \underbrace{R_d L_d \max(0, \mathbf{l} \cdot \mathbf{n})}_{\text{Diffuse}} + \underbrace{R_s L_s \max(0, \mathbf{h} \cdot \mathbf{n})^{\alpha}}_{\text{Specular}} \end{aligned}$$

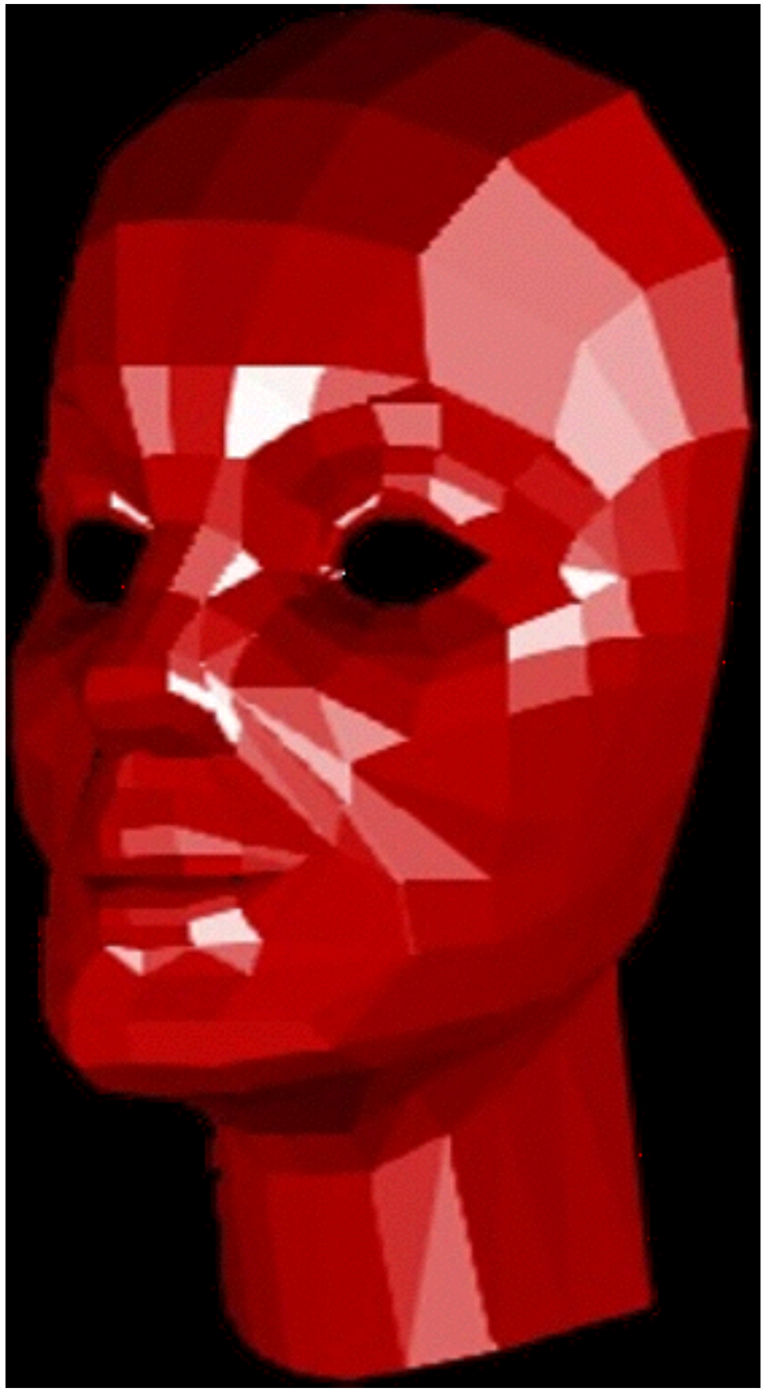
Shading Polygonal Geometry

Smooth surfaces are often approximated by polygons

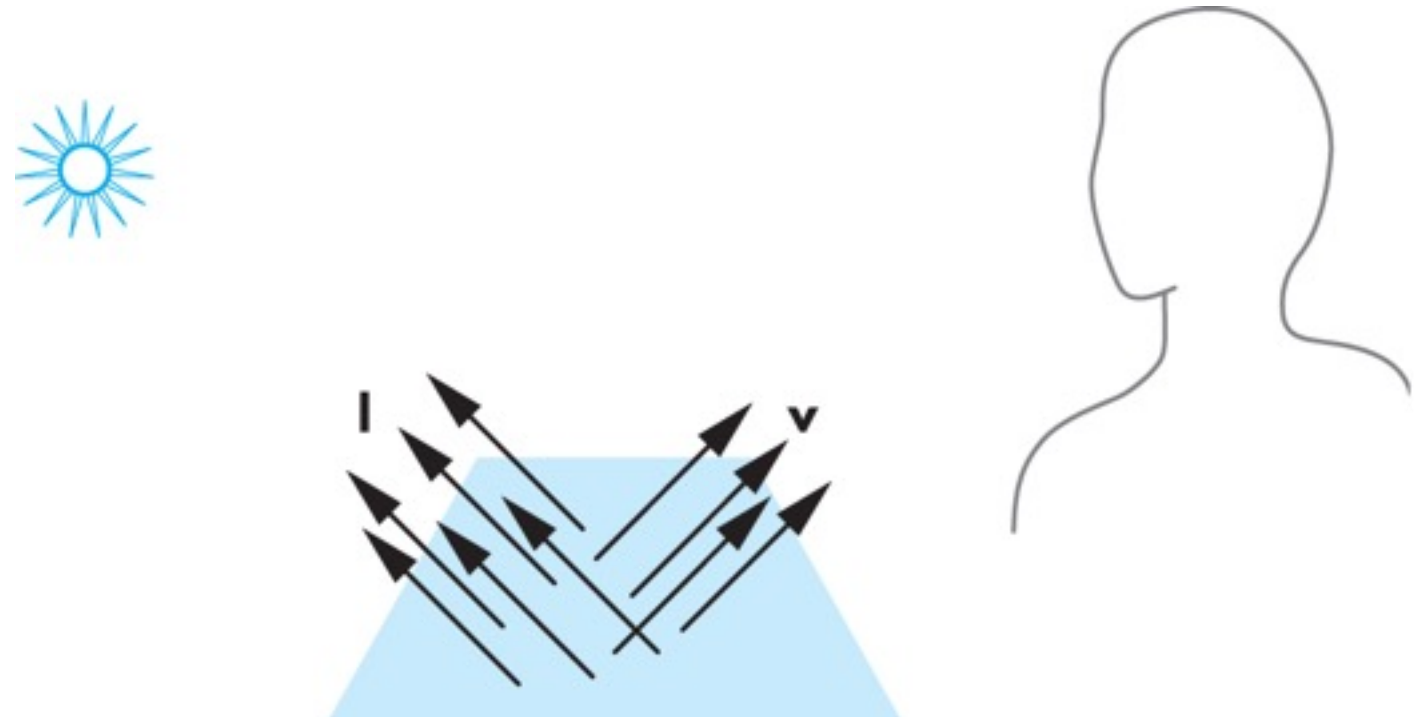
Shading approaches:

1. Flat
2. Smooth (Gouraud)
3. Phong





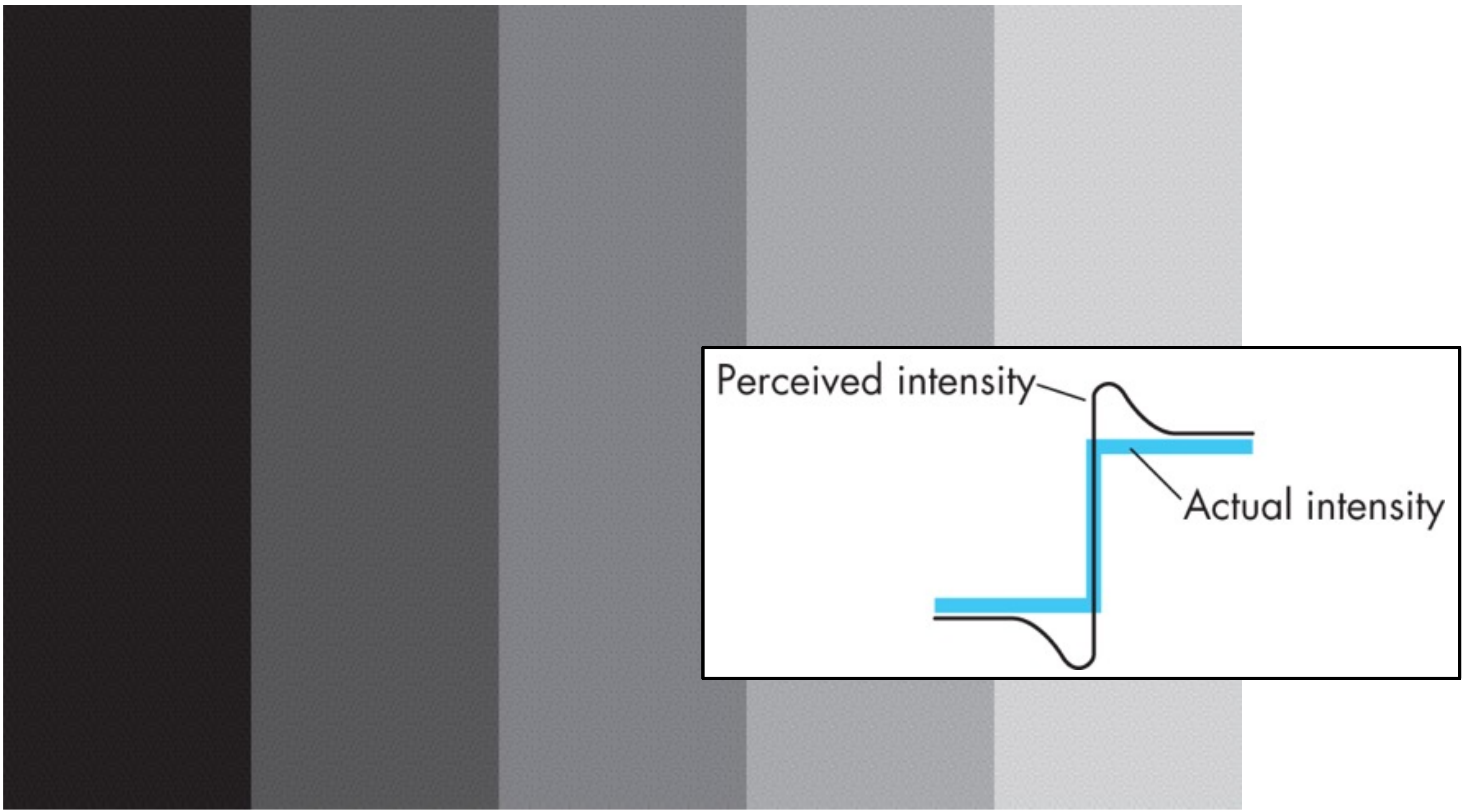
Flat Shading



do the shading calculation once per **polygon**

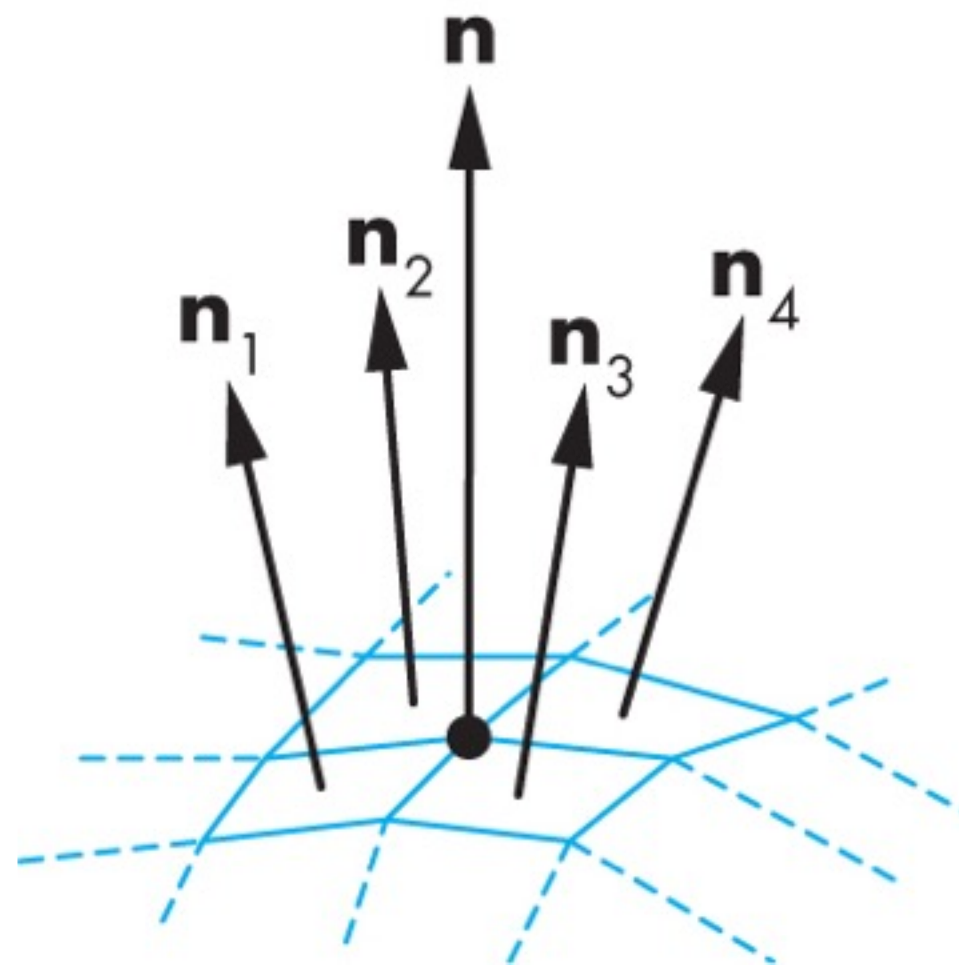
valid for light at ∞ and viewer at ∞ and faceted surfaces

Mach Band Effect



Smooth Shading

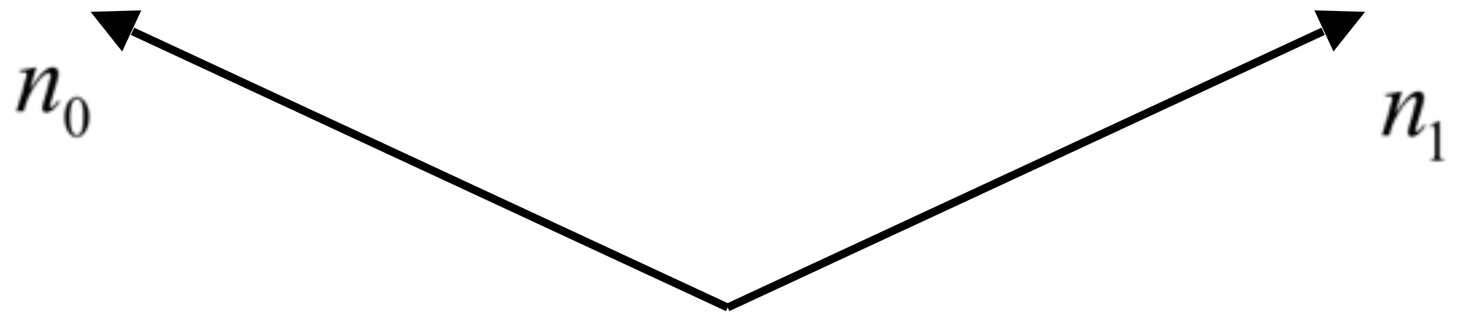
$$\mathbf{n} = \frac{\mathbf{n}_1 + \mathbf{n}_2 + \mathbf{n}_3 + \mathbf{n}_4}{||\mathbf{n}_1 + \mathbf{n}_2 + \mathbf{n}_3 + \mathbf{n}_4||}$$



do the shading calculation once per **vertex**

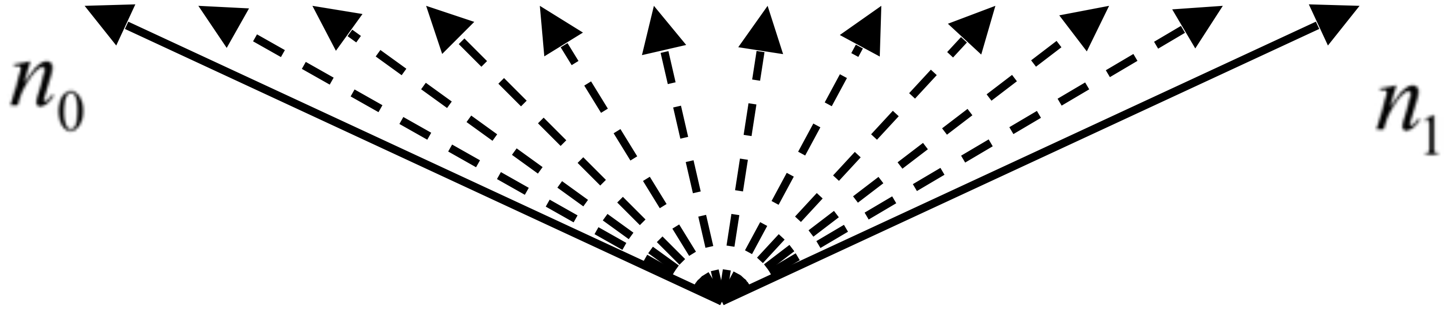
Interpolating Normals

- Must renormalize



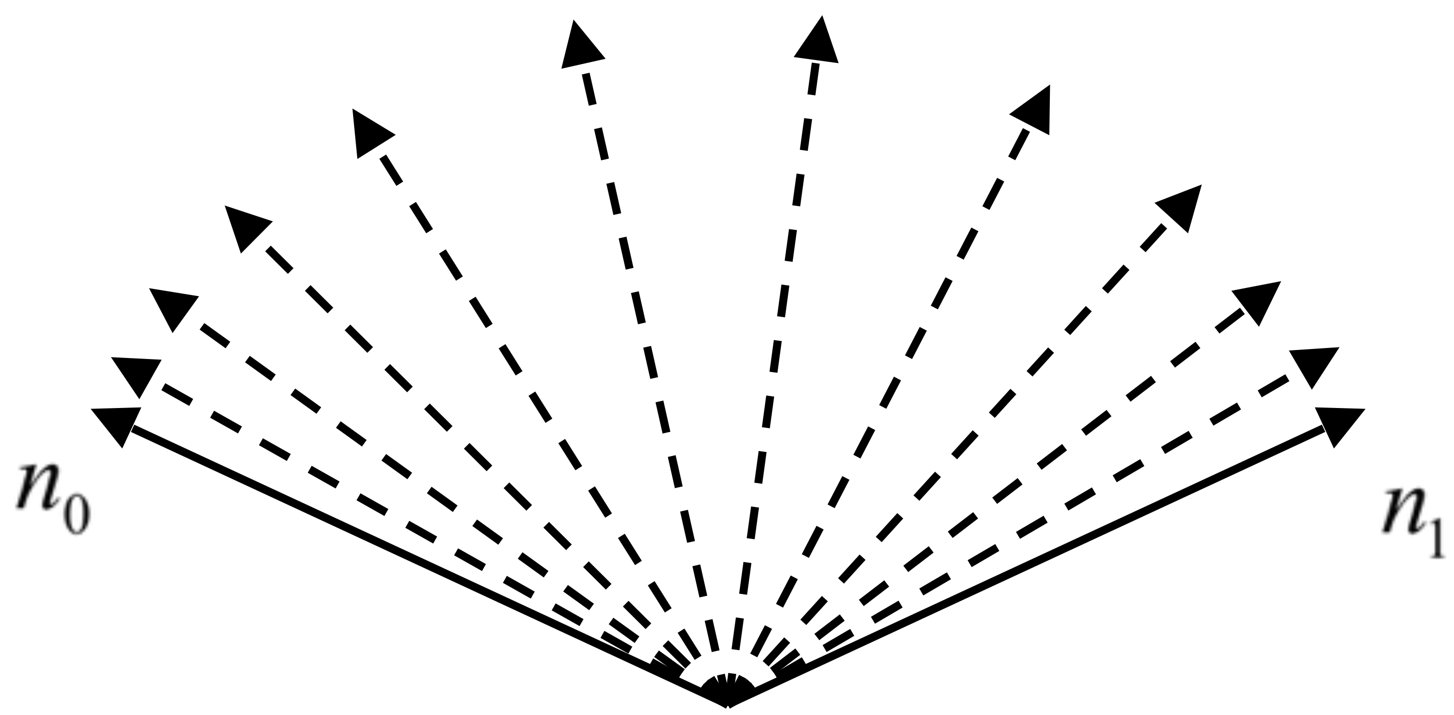
Interpolating Normals

- Must renormalize

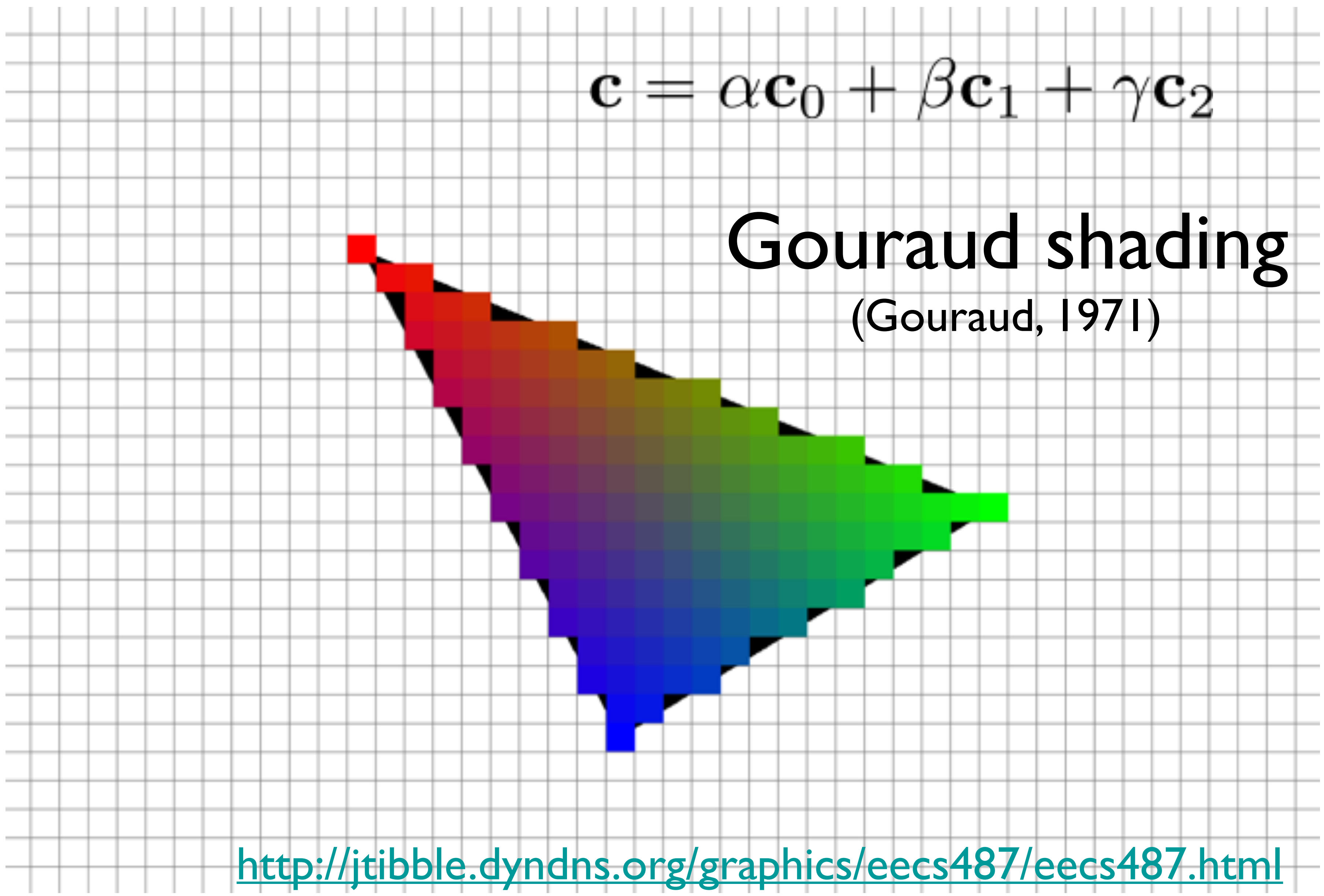


Interpolating Normals

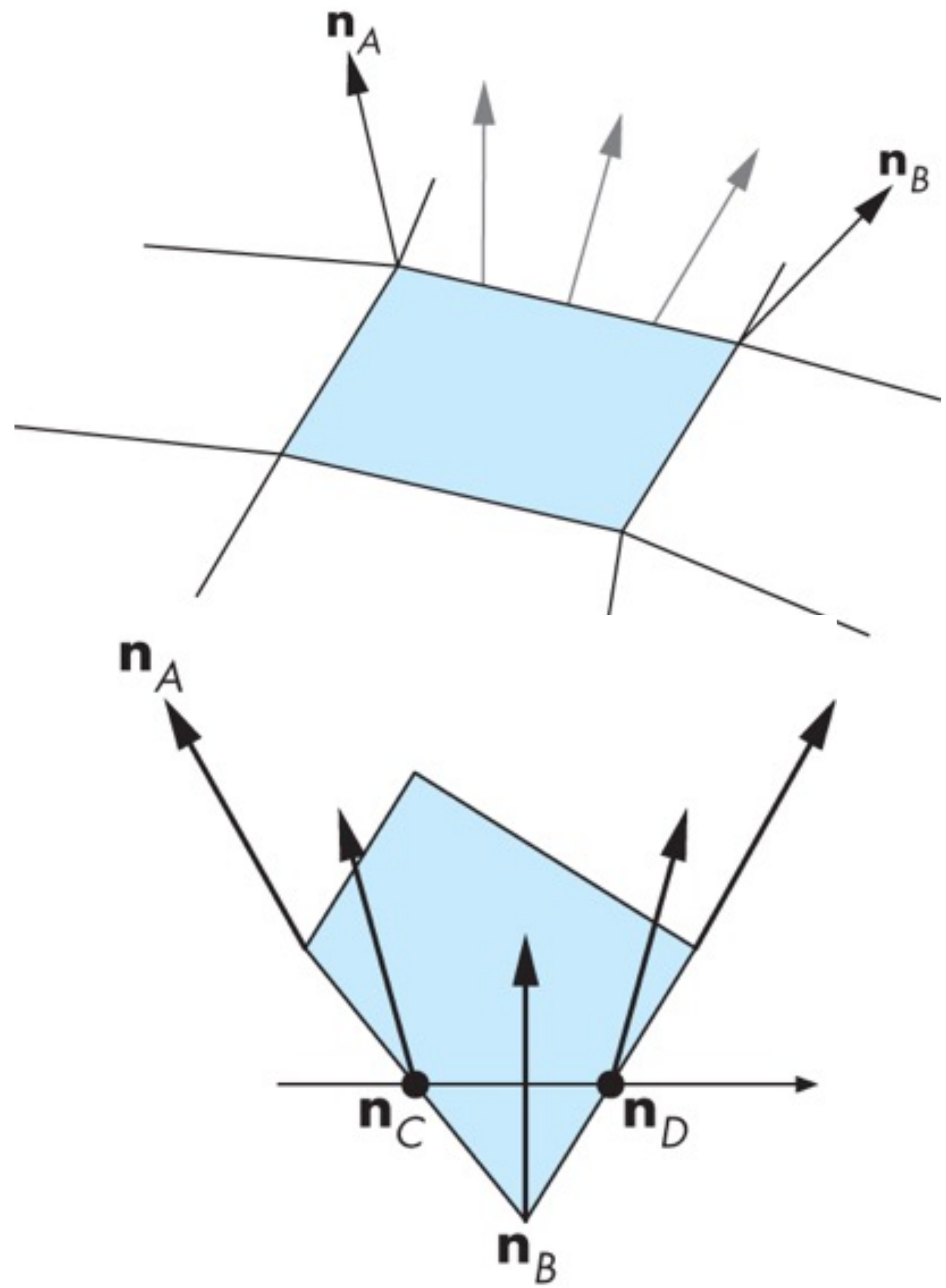
- Must renormalize



We can interpolate attributes using barycentric coordinates

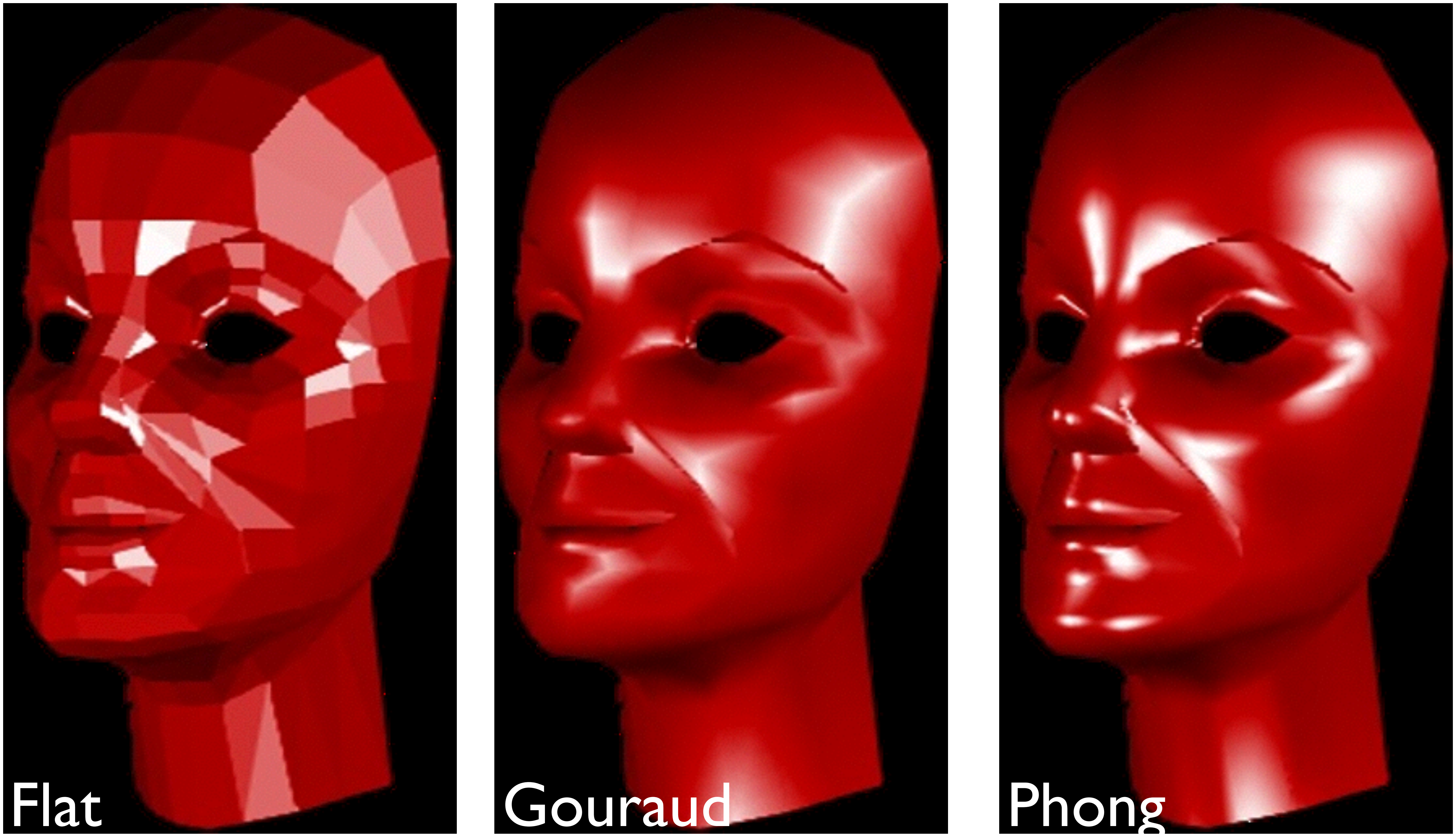


Phong Shading

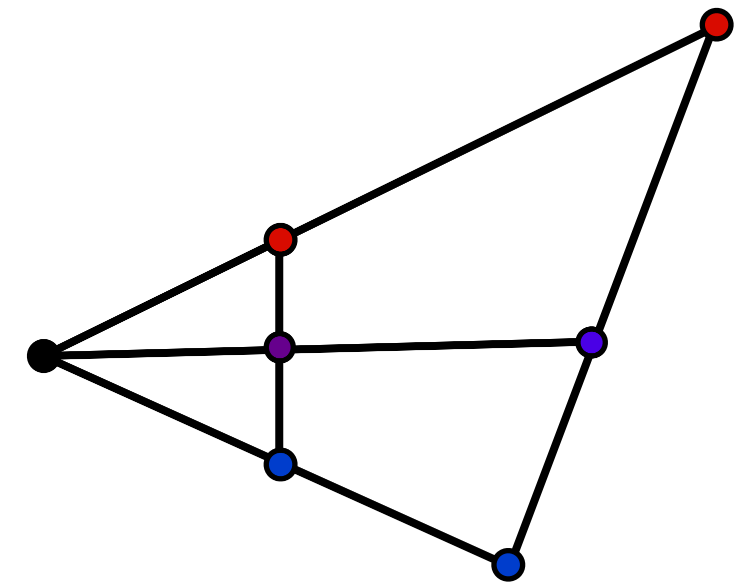


do the shading calculation once per **fragment**

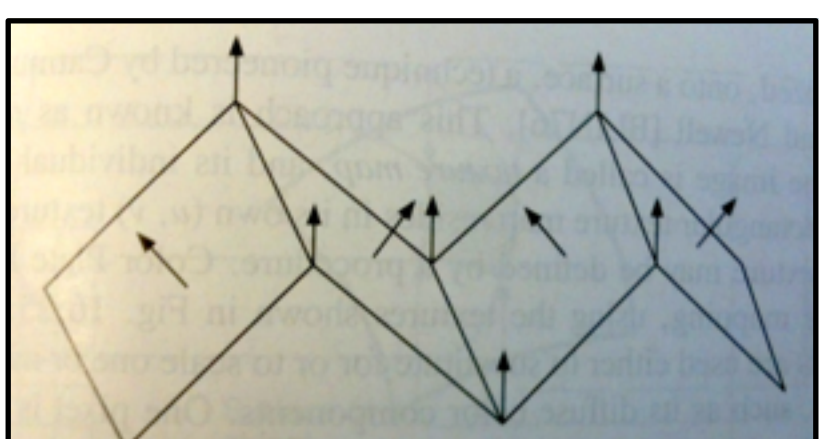
Comparison

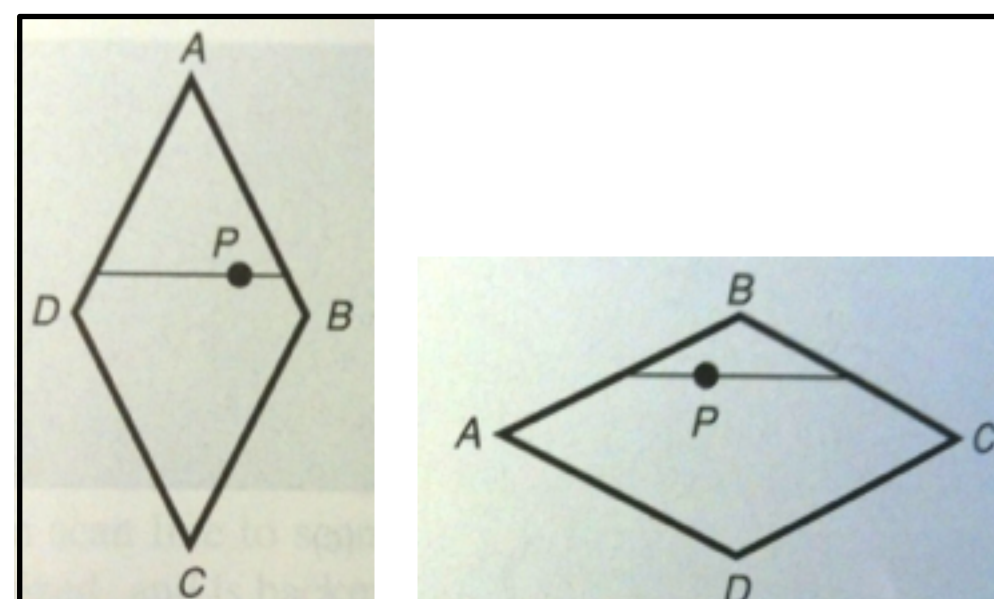


Problems with Interpolated Shading



- Polygonal silhouette
- Perspective distortion
- Orientation dependence
- Unrepresentative surface normals

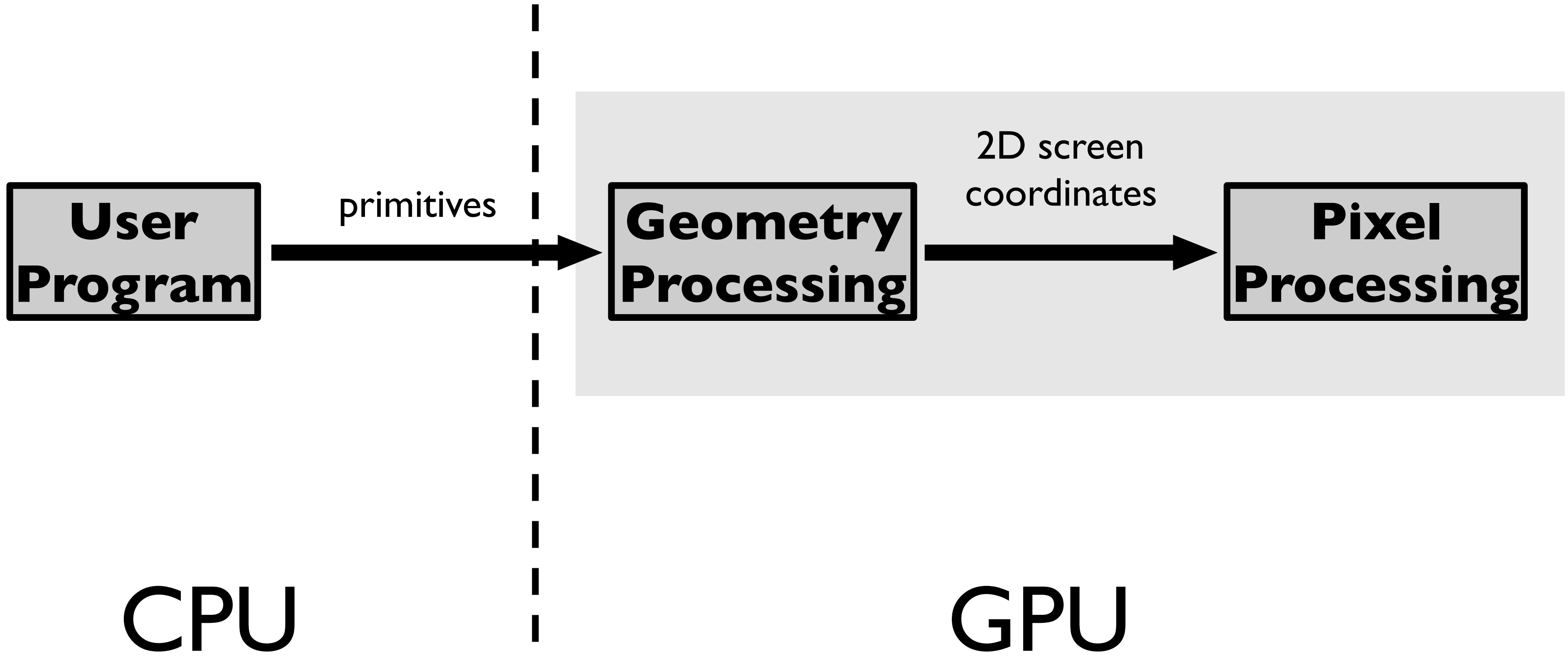




[Foley, van Dam, Feiner, Hughes]

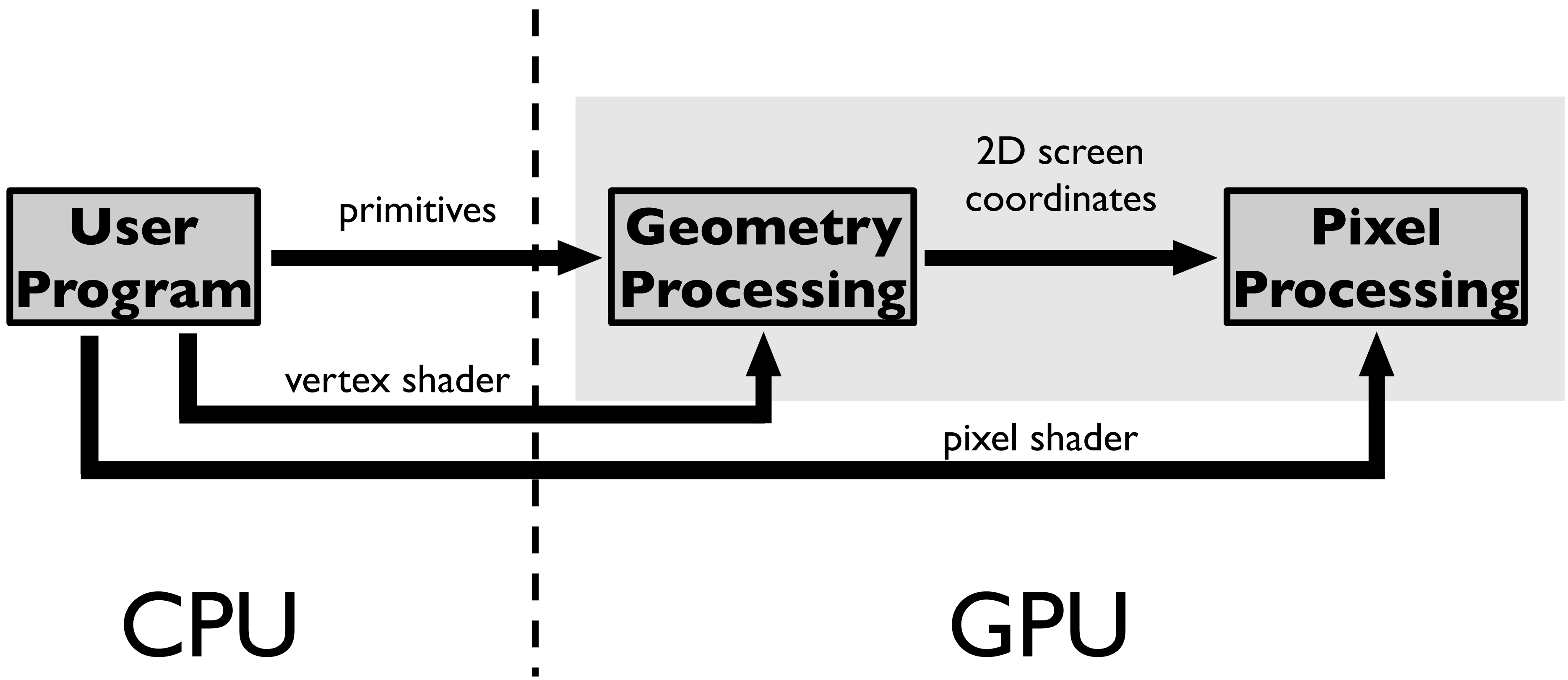
Programmable Shading

Fixed-Function Pipeline



Control pipeline through GL state variables

Programmable Pipeline



Supply shader programs to be executed on GPU as part of pipeline

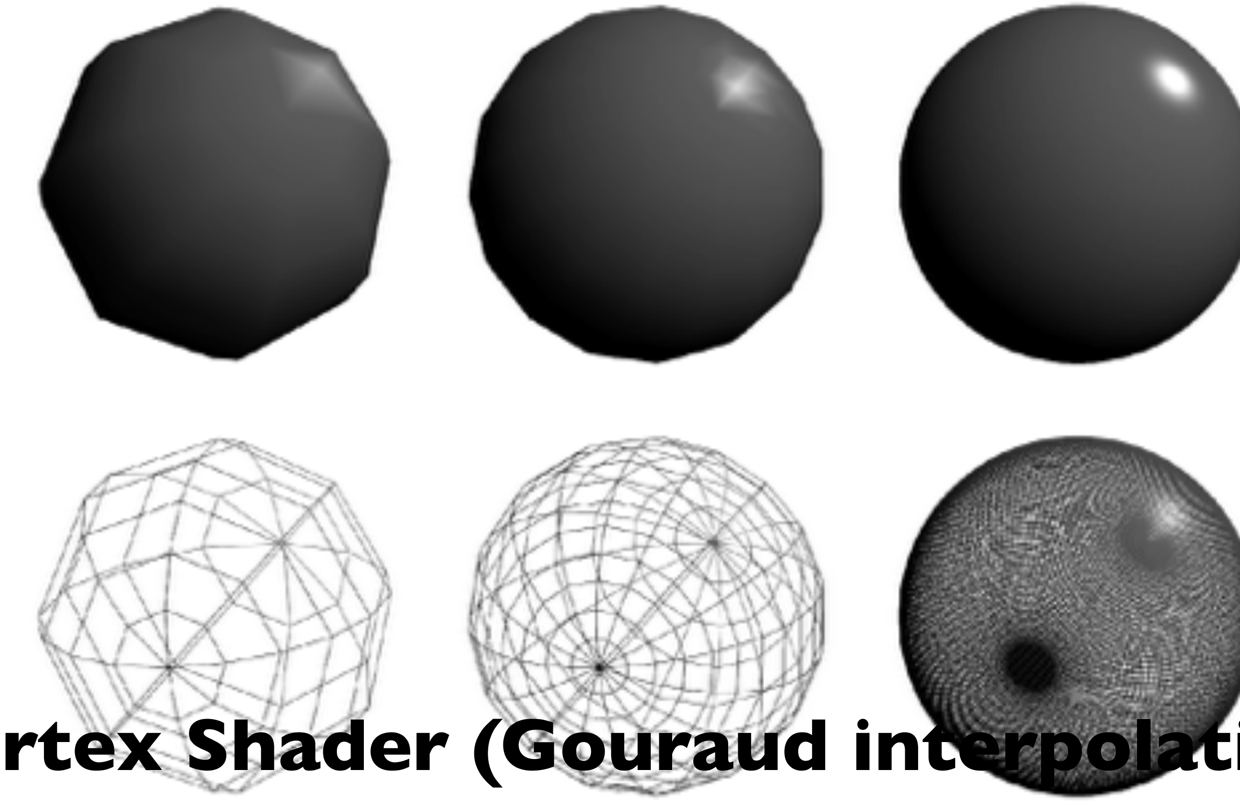
Phong reflectance in vertex and pixel shaders using GLSL

```
void main(void)
{
    vec4 v = gl_modelView_Matrix * gl_Vertex;
    vec3 n = normalize(gl_NormalMatrix * gl_Normal);
    vec3 l = normalize(gl_lightSource[0].position - v);
    vec3 h = normalize(l - normalize(v));

    float p = 16;
    vec4 cr = gl_FrontMaterial.diffuse;
    vec4 cl = fl_LightSource[0].diffuse;
    vec4 ca - vec4(0.2, 0.2, 0.2, 1.0);

    vec4 color;
    if (dot(h,n) > 0)
        color = cr * (ca + cl * max(0,dot(,n,l)))
            + cl* pow(dot(h,n), p);
    else
        color = cr * (ca + cl * max(0,dot(,n,l)));

    gl_FrontColor = color;
    gl_Position = ftransform();
}
```



Vertex Shader (Gouraud interpolation)

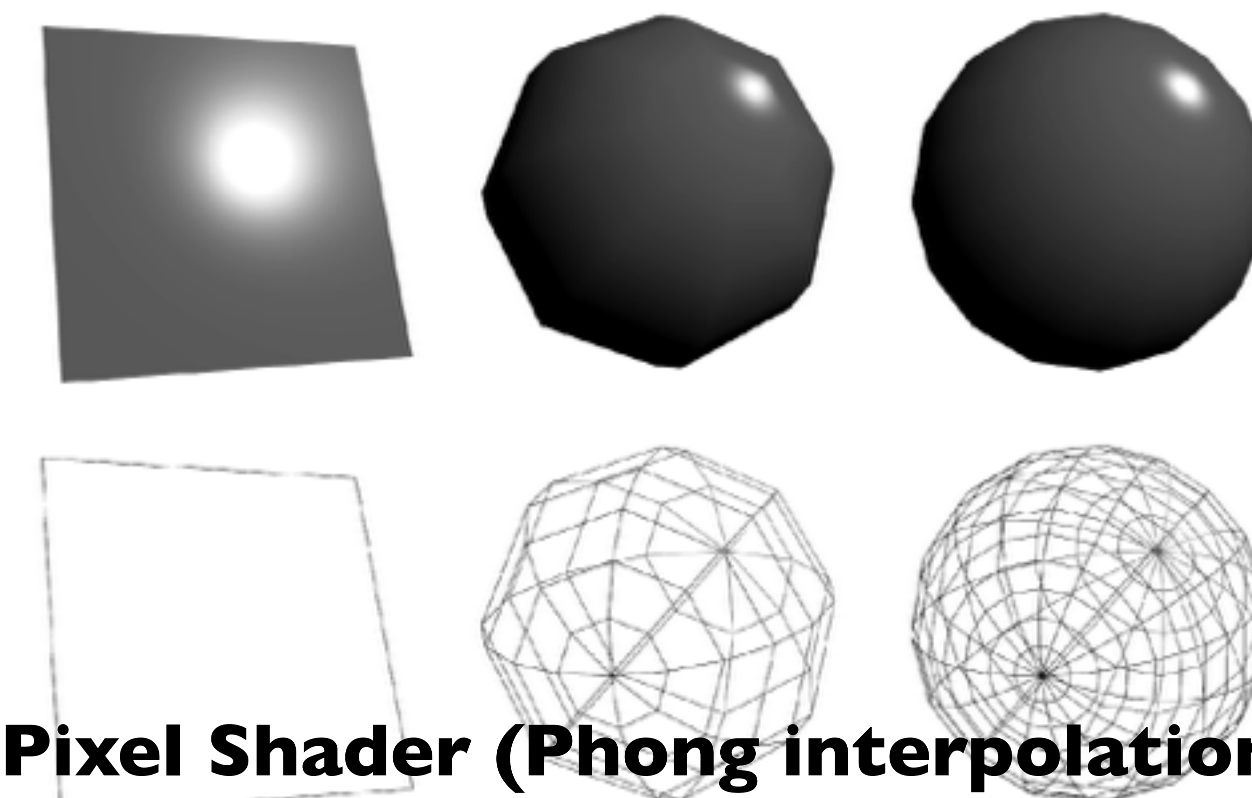
```
varying vec4 v;
varying vec3 n;

void main(void)
{
    vec3 l = normalize(gl_lightSource[0].position - v);
    vec3 h = normalize(l - normalize(v));

    float p = 16;
    vec4 cr = gl_FrontMaterial.diffuse;
    vec4 cl = fl_LightSource[0].diffuse;
    vec4 ca - vec4(0.2, 0.2, 0.2, 1.0);

    vec4 color;
    if (dot(h,n) > 0)
        color = cr * (ca + cl * max(0,dot(,n,l)))
            + cl* pow(dot(h,n), p);
    else
        color = cr * (ca + cl * max(0,dot(,n,l)));

    gl_FragColor = color;
}
```

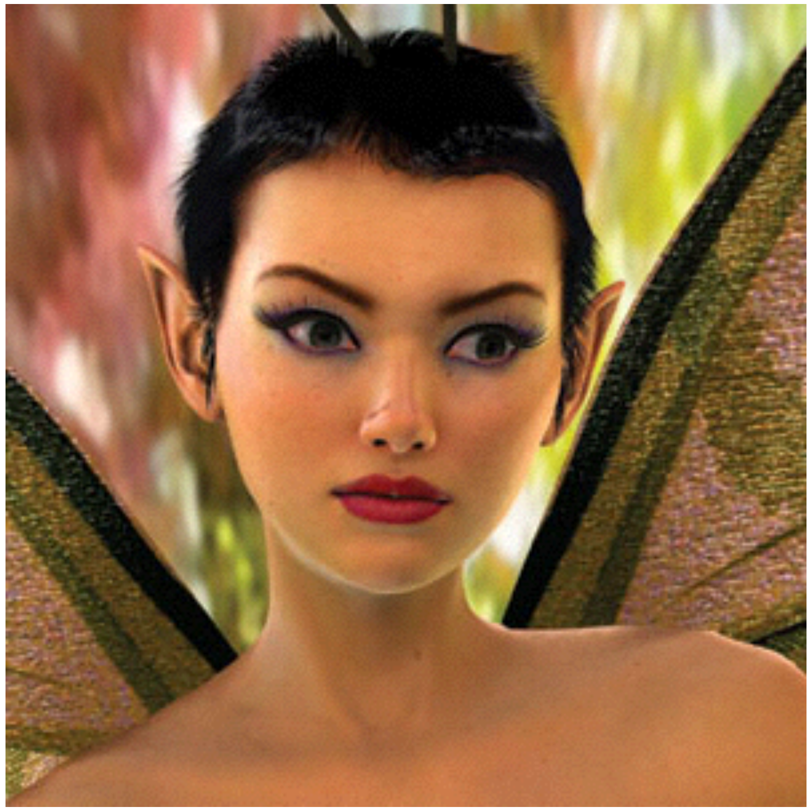


Pixel Shader (Phong interpolation)

[Shirley and Marschner]

Rusty car shader, NVIDIA

Call of Juarez DX10 Benchmark, ATI



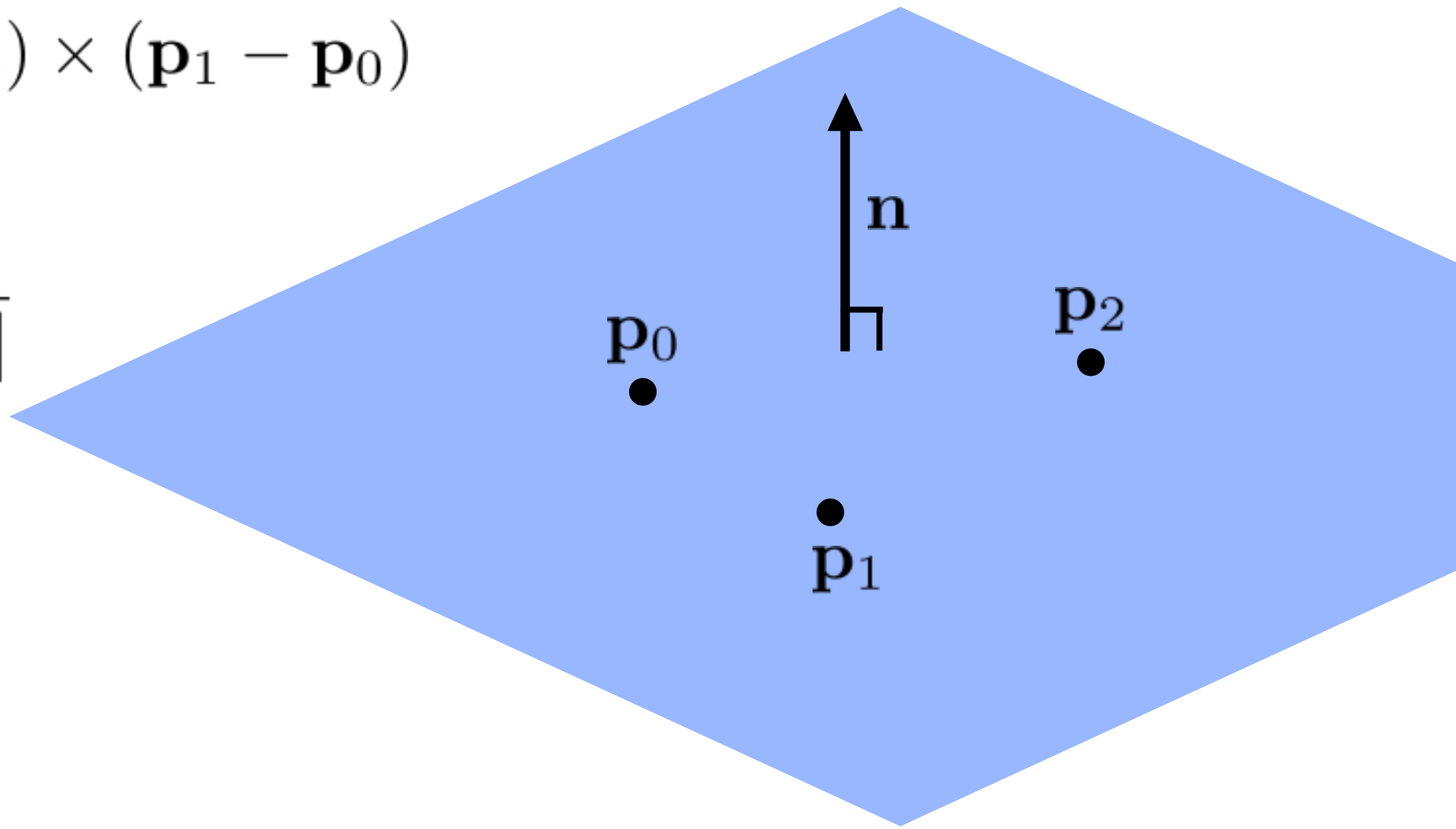
Dawn, NVIDIA

Computing Normal Vectors

Plane Normals

$$\mathbf{v} = (\mathbf{p}_2 - \mathbf{p}_0) \times (\mathbf{p}_1 - \mathbf{p}_0)$$

$$\mathbf{n} = \frac{\mathbf{v}}{||\mathbf{v}||}$$



Implicit function normals

$$f(\mathbf{p}) = 0$$

$$\nabla f(\mathbf{p})$$

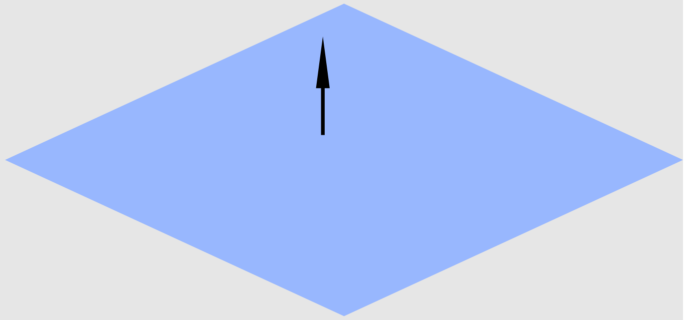
$$\nabla f = \begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \\ \frac{\partial f}{\partial z} \end{pmatrix}$$

sphere

$$\mathbf{p} \cdot \mathbf{p} - r^2 = 0$$

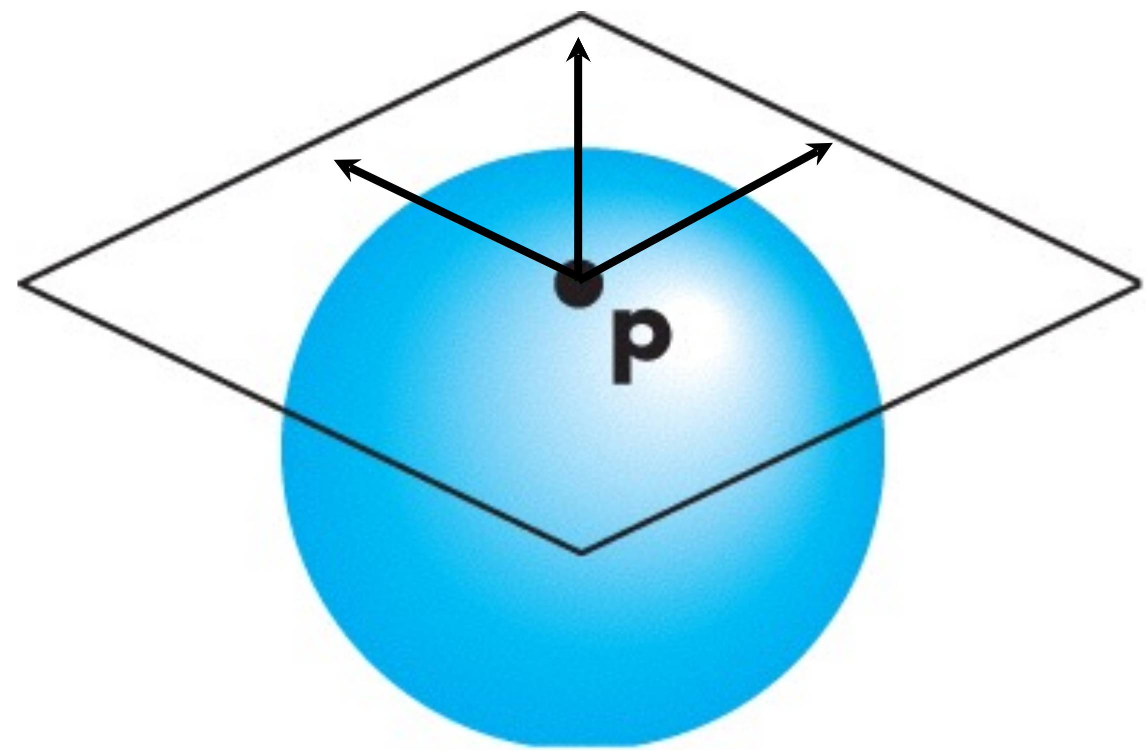
plane

$$\mathbf{n} \cdot (\mathbf{p} - \mathbf{p}_0) = 0$$



Parametric form

$$\mathbf{p}(u,v) = \begin{pmatrix} x(u,v) \\ y(u,v) \\ z(u,v) \end{pmatrix}$$



tangent vectors $\quad \dfrac{\partial \mathbf{p}}{\partial u} \quad \dfrac{\partial \mathbf{p}}{\partial v}$

normal $\quad \dfrac{\frac{\partial \mathbf{p}}{\partial u} \times \frac{\partial \mathbf{p}}{\partial v}}{\left|\left|\frac{\partial \mathbf{p}}{\partial u} \times \frac{\partial \mathbf{p}}{\partial v}\right|\right|}$