CS230 : Computer Graphics

Lecture 6: Lighting and Shading

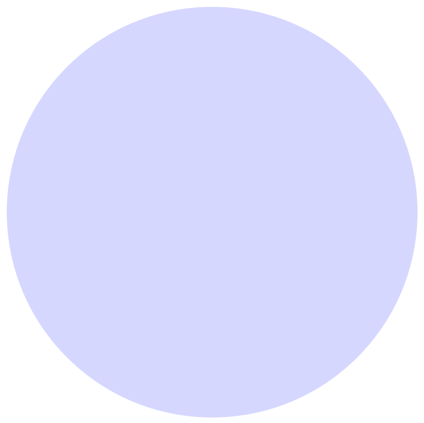
Tamar Shinar
Computer Science & Engineering
UC Riverside

Notes

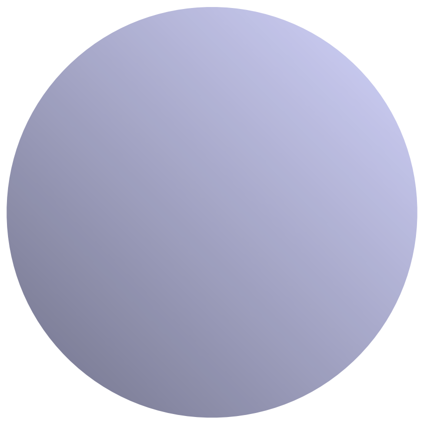
- HBC, Chapter 17
- Angel, Chapter 5
- some slides courtesy of V. Zordan

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



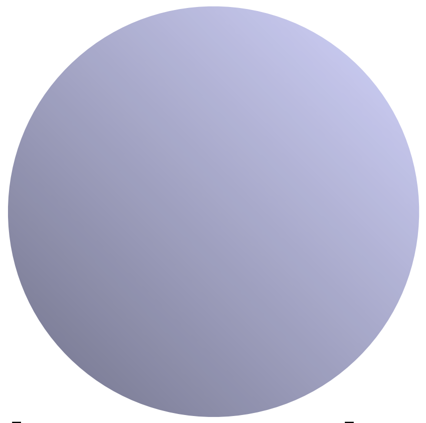
- But we want



The more realistically lit sphere has gradations in its color that give us a sense of its three-dimensionality

Shading

- Why does the image of a real sphere look like

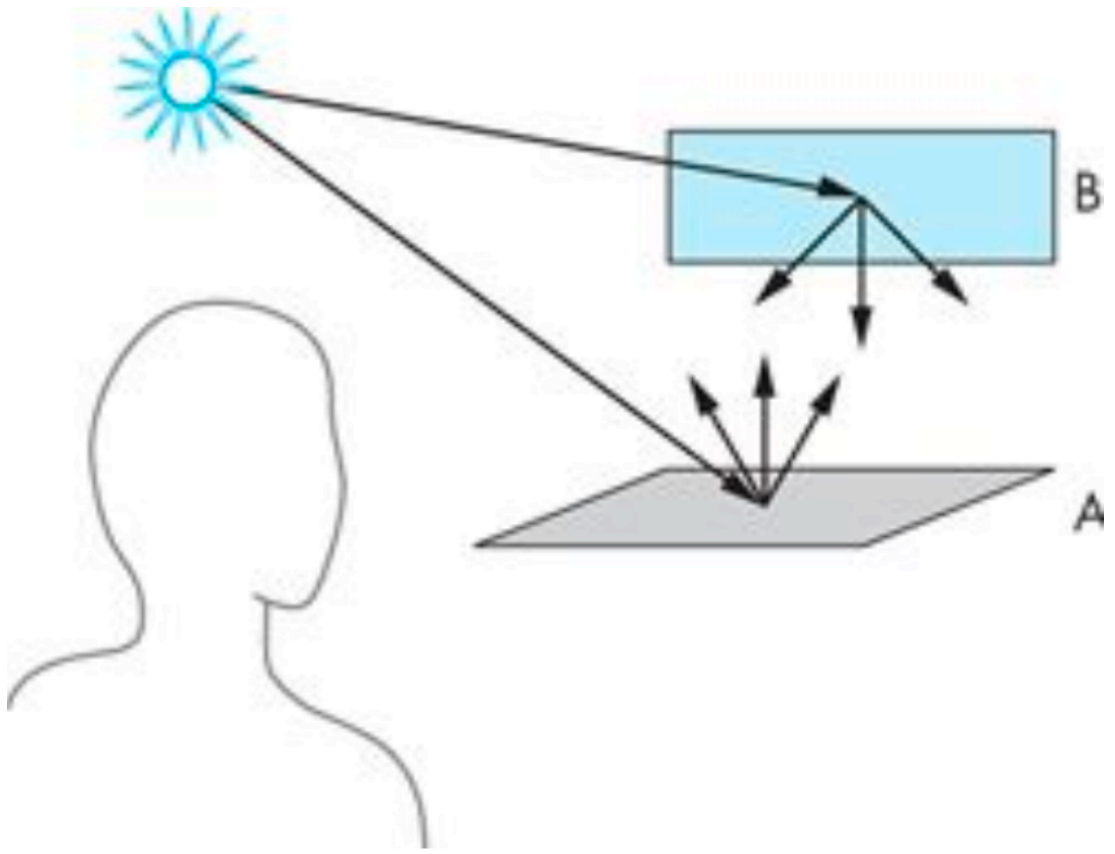


- Light-material interactions cause each point to have a different color or shade
- Need to consider
  - Light sources
  - Material properties
  - Location of viewer
  - Surface orientation (normal)

We are going to develop a **local** lighting model by which we can shade a point independently of the other surfaces in the scene
our **goal** is to add this to a fast graphics pipeline architecture

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 for 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

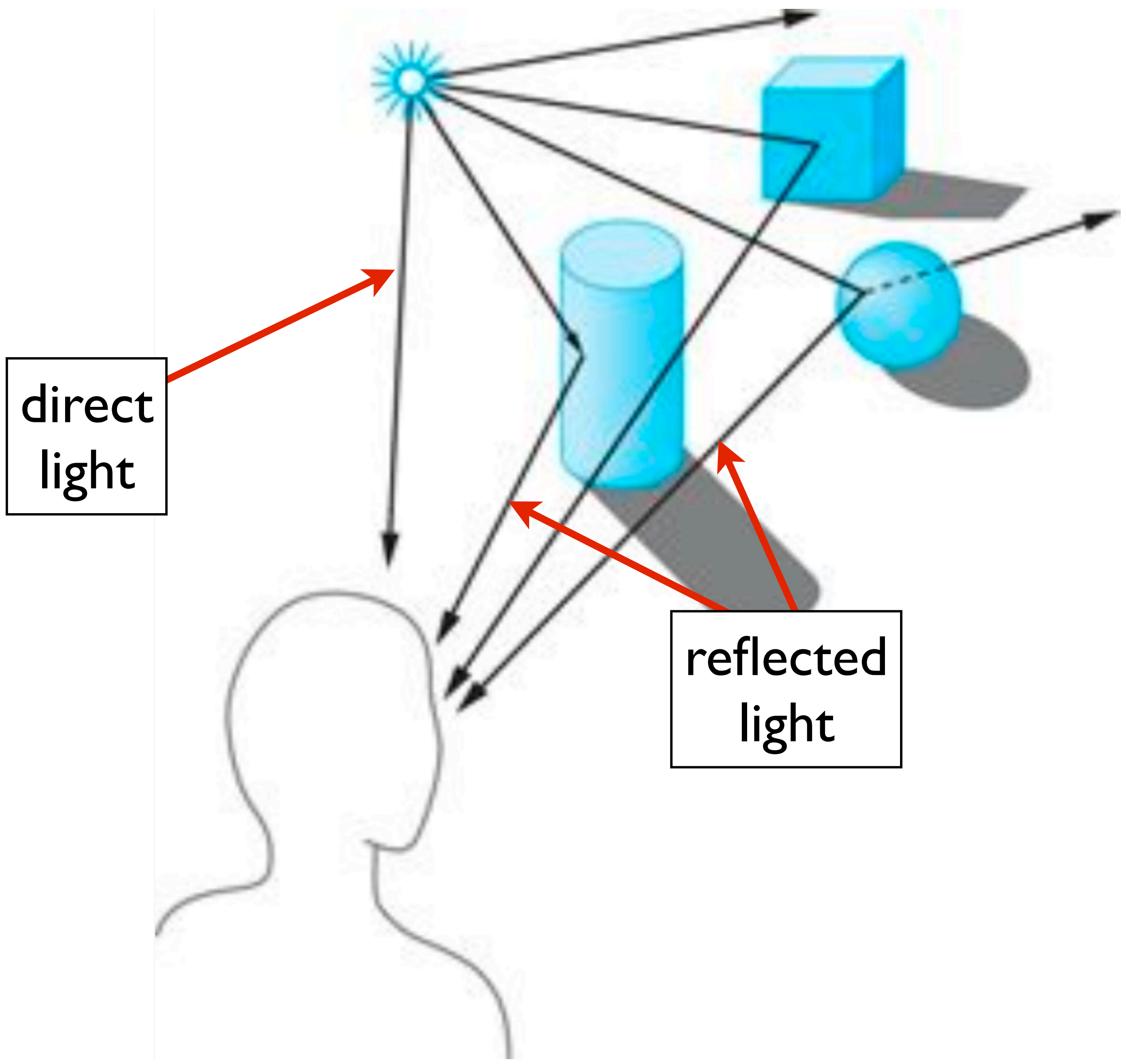
http://en.wikipedia.org/wiki/Rendering_equation

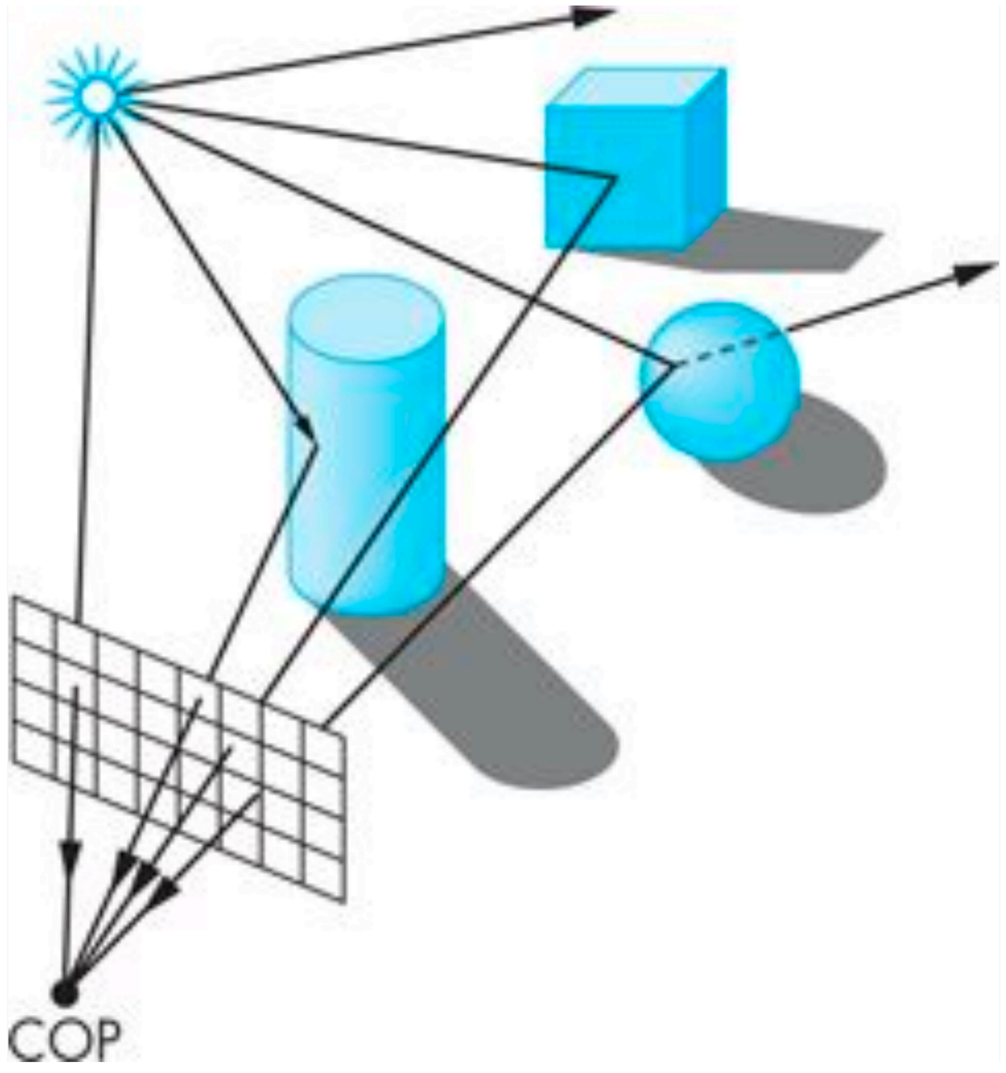
Fast local shading models

- 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

some approximations to the rendering equation include **radiosity** and **ray tracing**, but they are still not as fast as the local model in the pipeline architecture

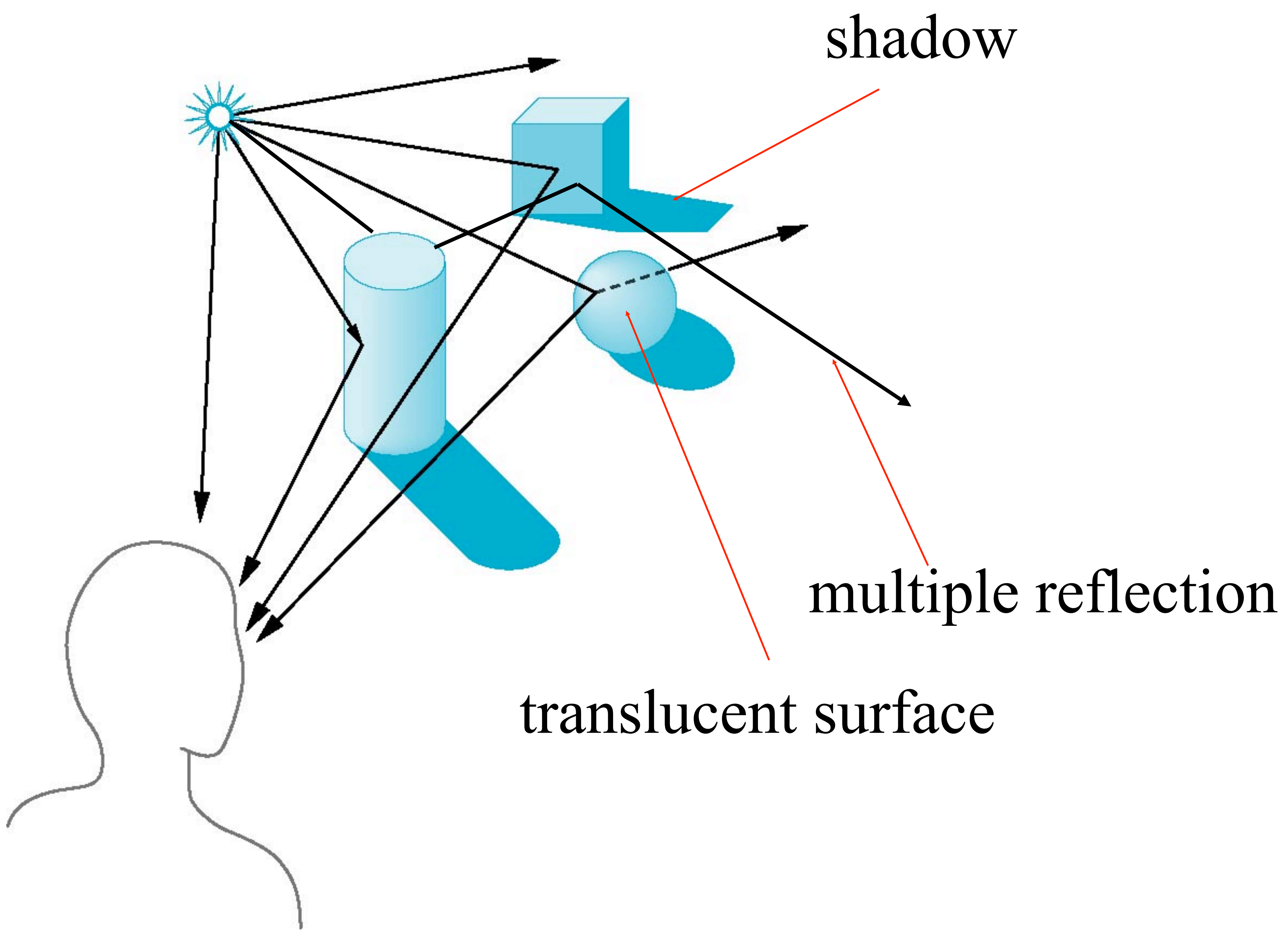
Local shading model





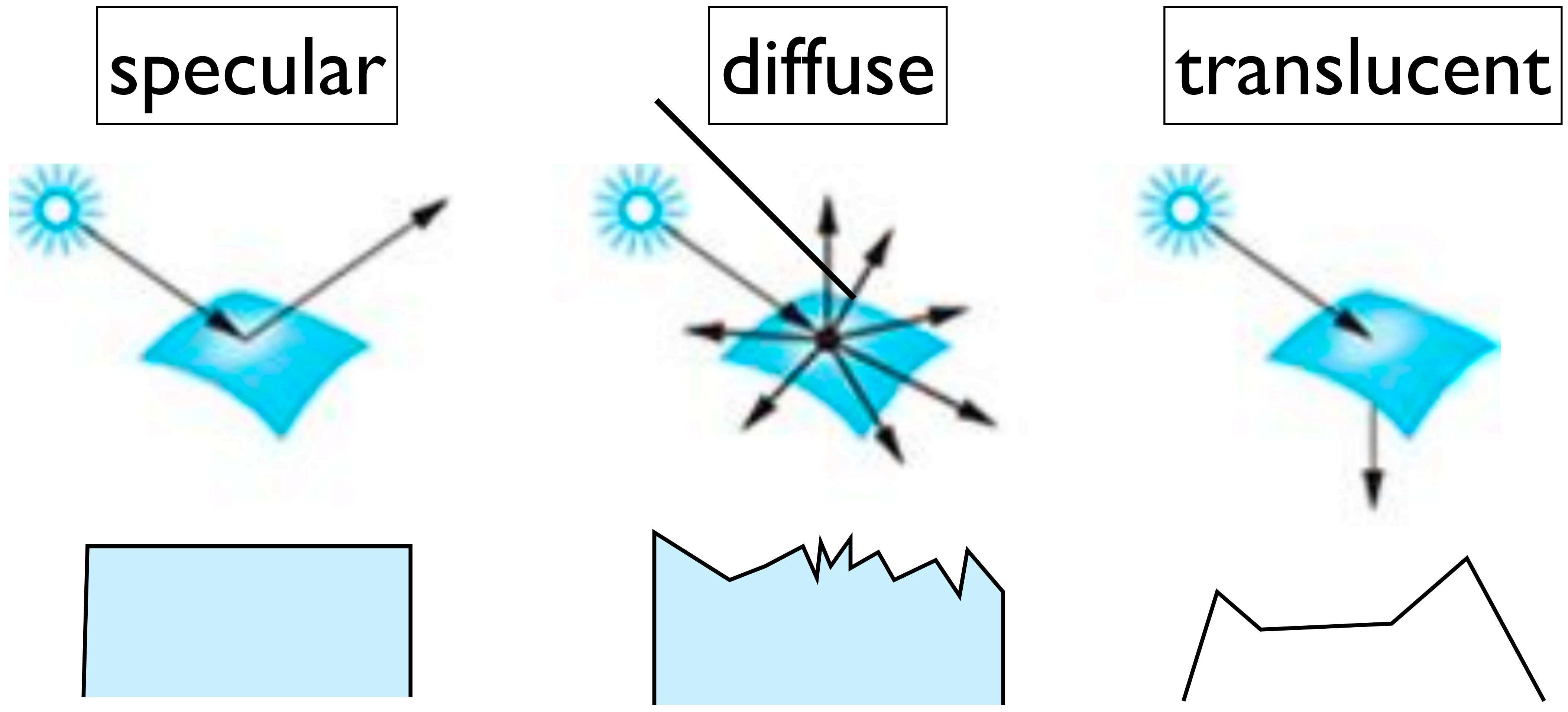
direct light is the color of the light source
reflected light is the color of the light light reflected from the object surface
for rendering, color of light source and reflected light determines the colors of pixels in the frame buffer
only need to consider the rays that leave the source and reach the viewers eye

Global Effects



Light-material interactions

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

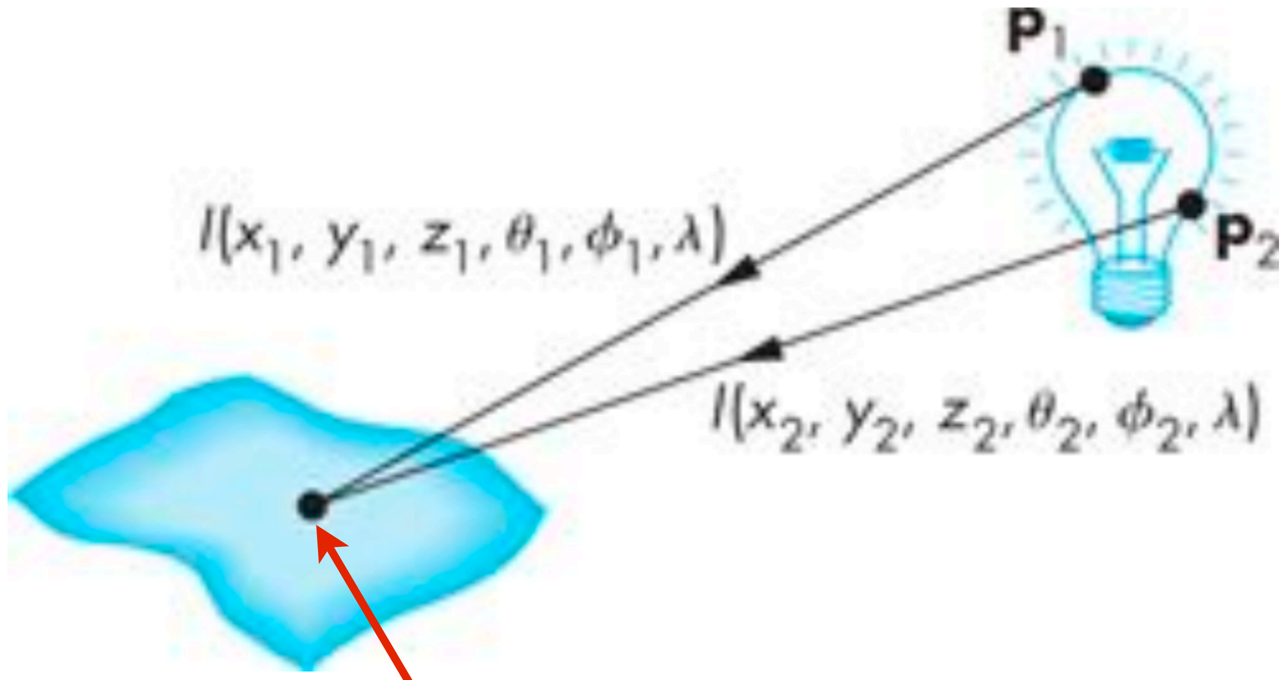


specular: shiny, smooth surface. light scattered in narrow range close to angle of reflection
e.g., mirror is perfectly specular
diffuse: matte, rough surface. light scattered in all directions
translucent: allows some light to pass through object. refraction: e.g., glass or water

General light source

Illumination function:

$$I(\mathbf{x}, \omega, \lambda)$$



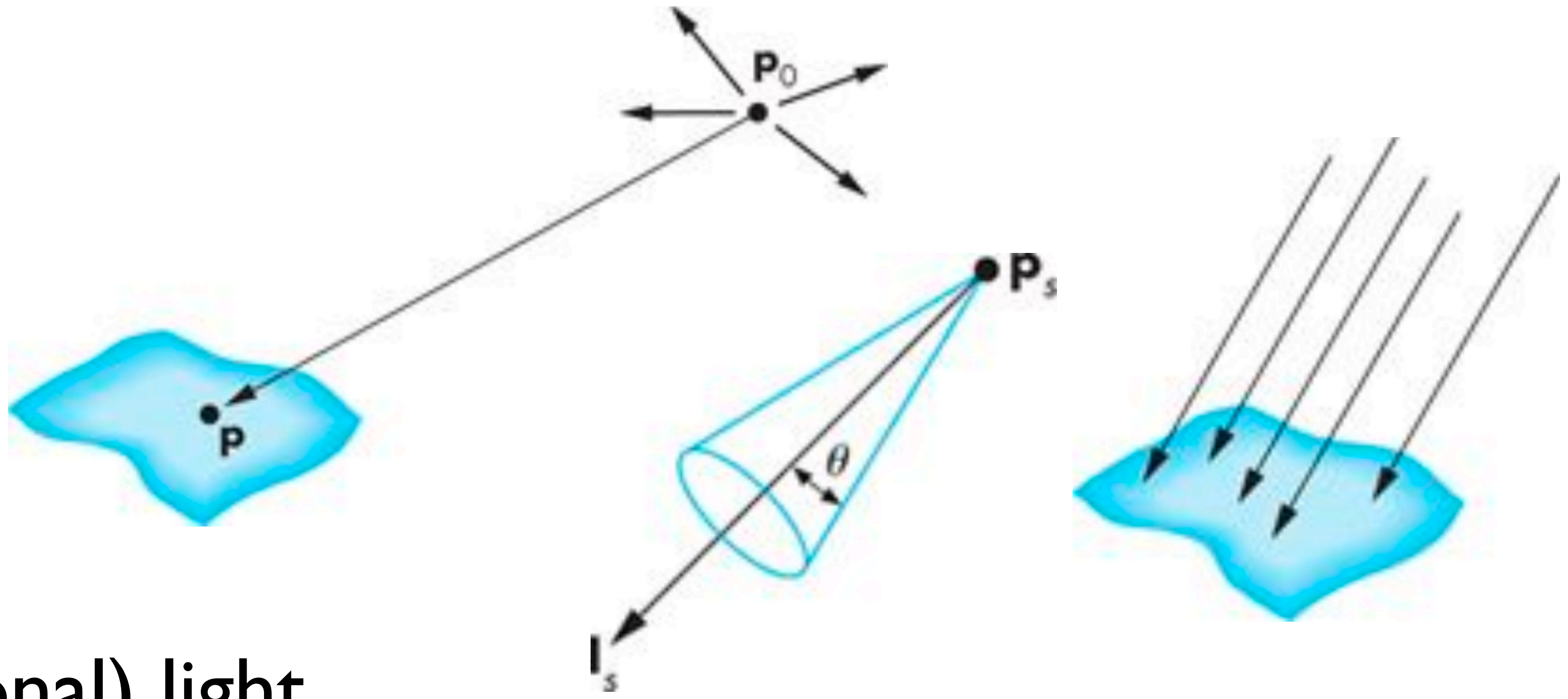
P_1
P_2
$I(x_1, y_1, z_1, \theta_1, \phi_1, \lambda)$
$I(x_2, y_2, z_2, \theta_2, \phi_2, \lambda)$

integrate contributions from all sources to shade the point

\vec{x} = (x,y,z)
\vec{omega} = theta, phi

Idealized light sources

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



luminance: $$\mathbf{I} = \begin{bmatrix} I_r \\ I_g \\ I_b \end{bmatrix}$$

source will be described through three component intensity or **luminance**
decompose into red, green, blue channels
e.g., use the red component of source to calculate red component of image
use a single scalar equations - each equation applied independently to each channel

Ambient light source

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

$$\mathbf{I}_a = \begin{bmatrix} I_{ar} \\ I_{ag} \\ I_{ab} \end{bmatrix}$$

$$I_a$$

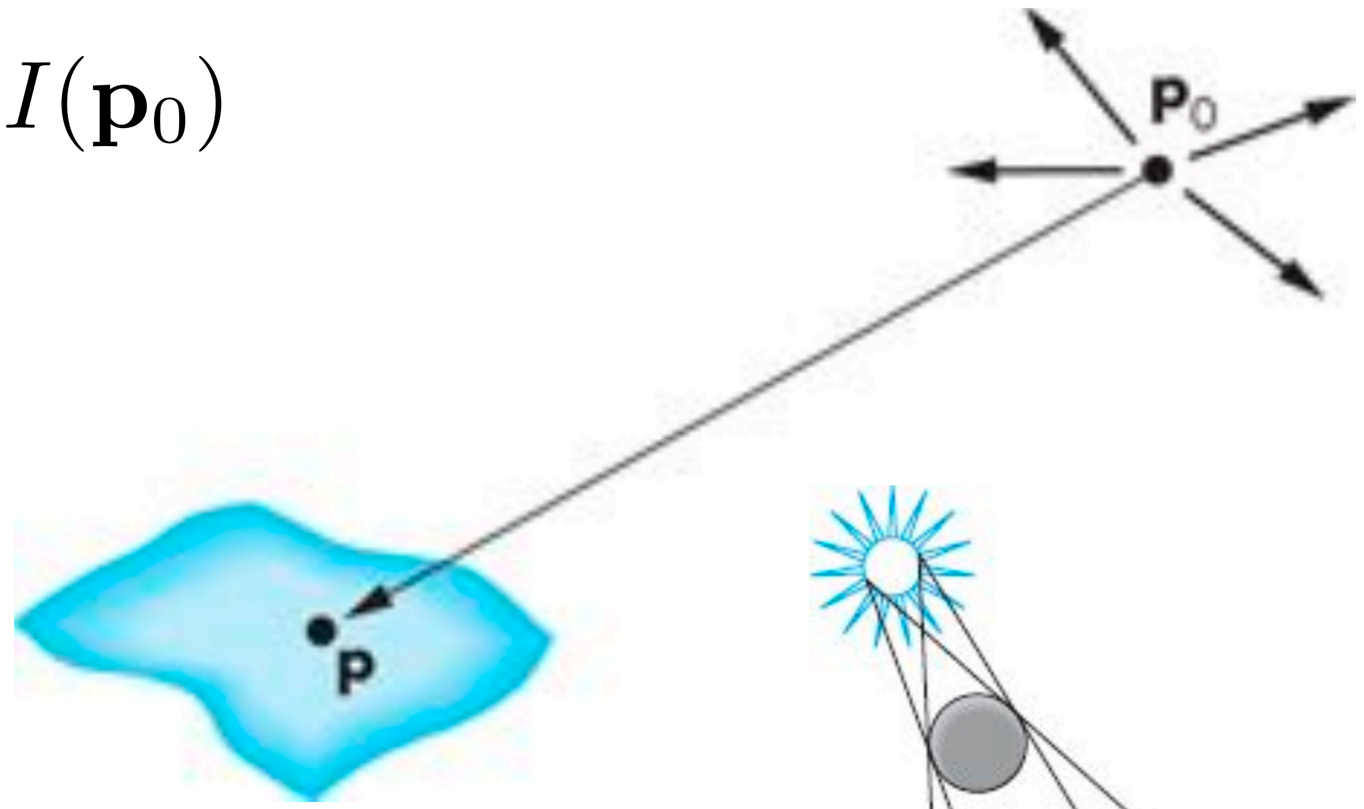
use scalar I_a to denote any component of \vec{I}_a
ambient light is the same everywhere
but different surfaces will **reflect** it differently

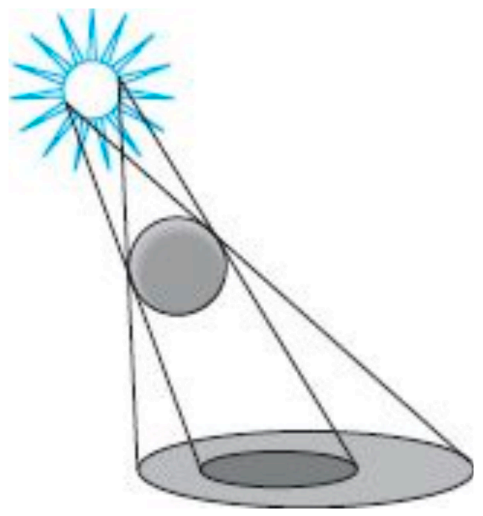
Point light source

$$\mathbf{I}(\mathbf{p}_0) = \begin{bmatrix} I_r(\mathbf{p}_0) \\ I_g(\mathbf{p}_0) \\ I_b(\mathbf{p}_0) \end{bmatrix} \qquad I(\mathbf{p}_0)$$

illumination intensity at **p**:

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

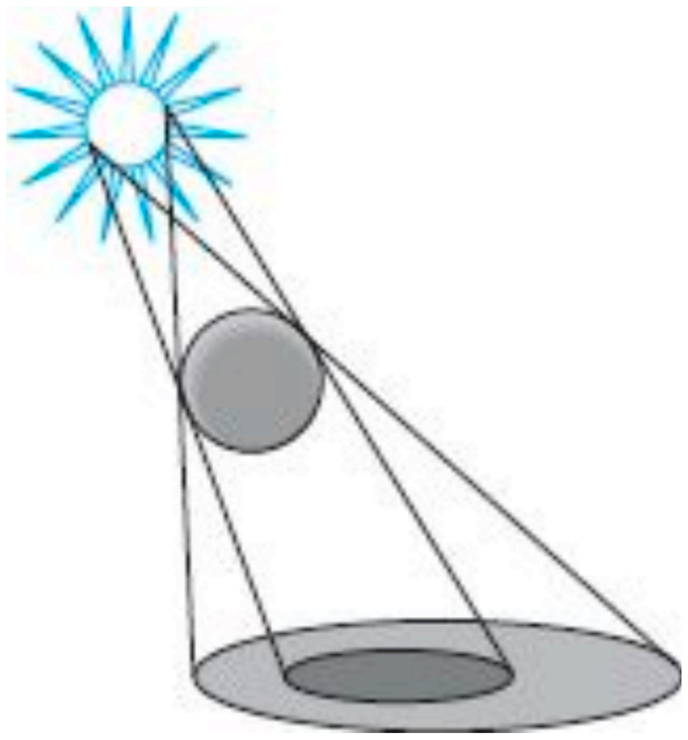




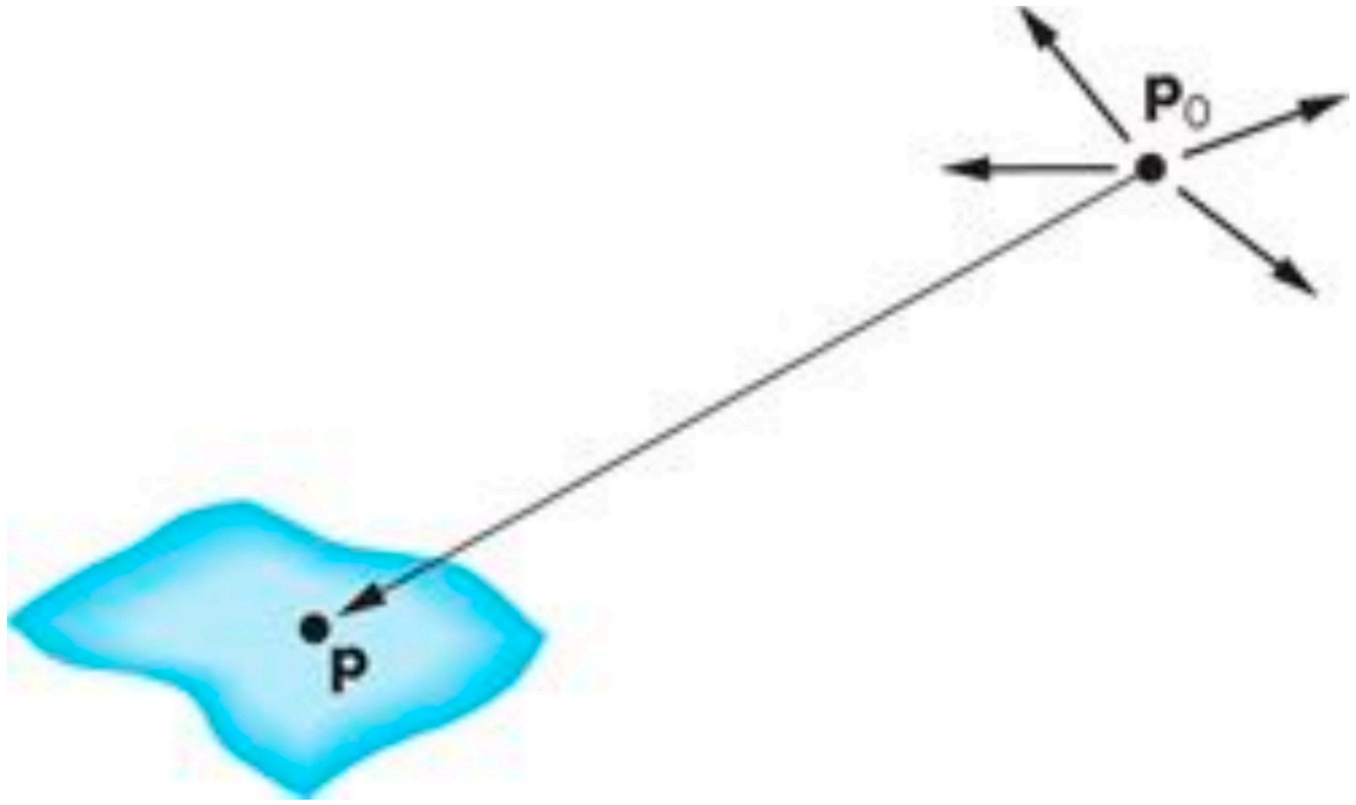
- use scalar I(\vec{p}_0) to denote any of three components
- points sources alone aren't too realistic looking -- tend to be high contrast
- most real-world scenes have large light sources
- add ambient light to mitigate high contrast

Point light source

Most real-world scenes have large light sources



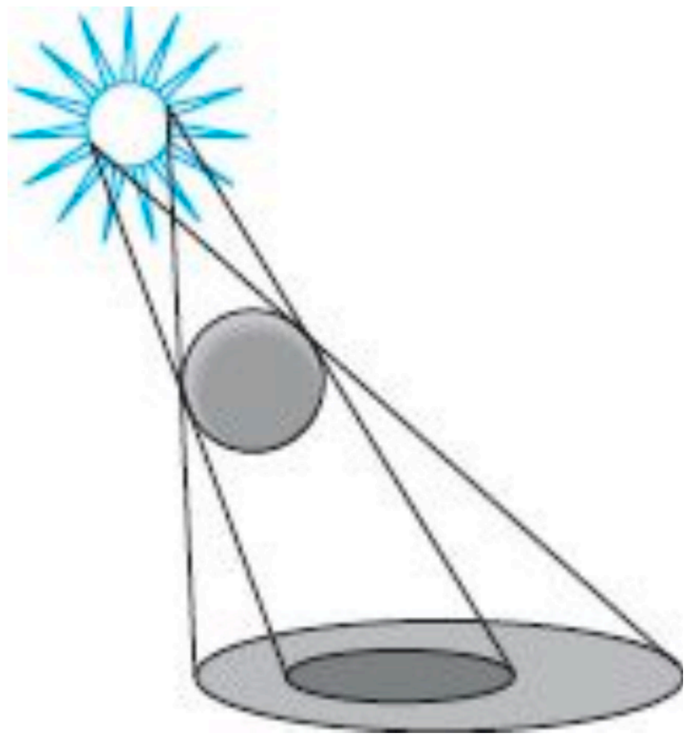
Point light sources alone aren't too realistic
- **add ambient light to mitigate high contrast**



- **umbra** is fully in shadow, **penumbra** is partially in shadow

Point light source

Most real-world scenes
have large light sources



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

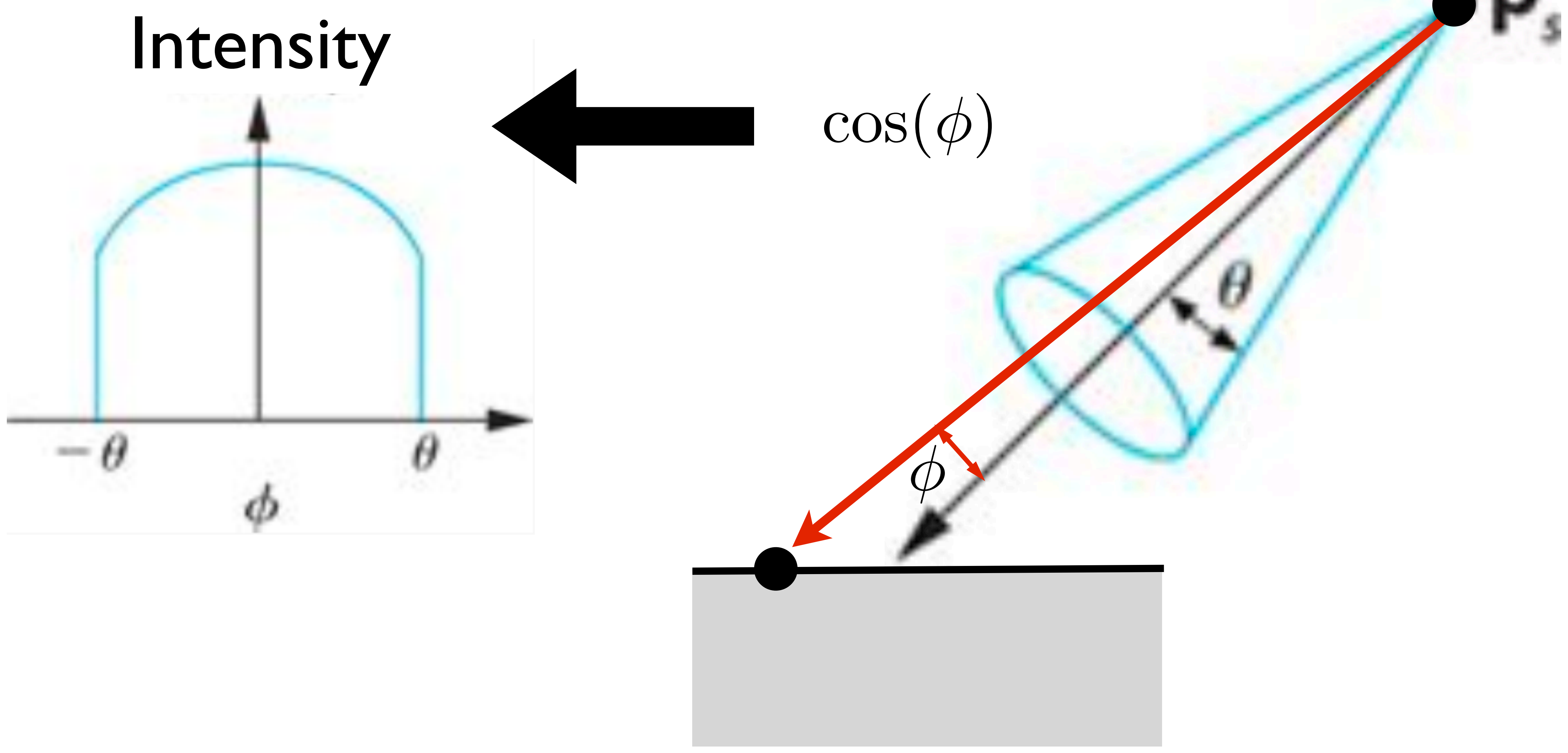
$$i(\mathbf{p}, \mathbf{p}_0) = \frac{1}{d^2}\mathbf{I}(\mathbf{p}_0)$$

$$\Downarrow$$

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

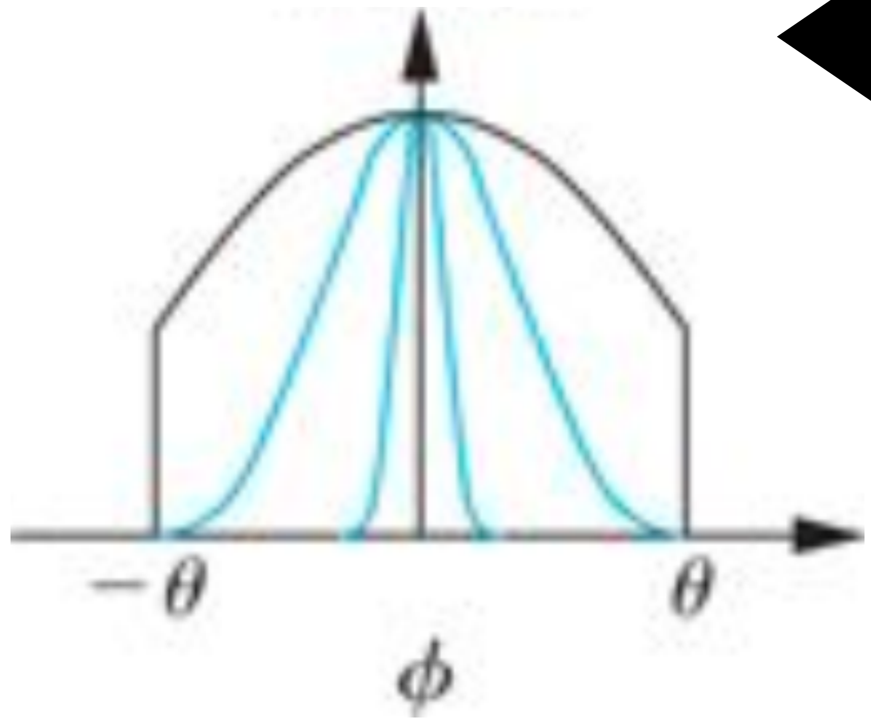
In practice, we also replace the 1/d^2 term by something that falls off more slowly

Spotlights

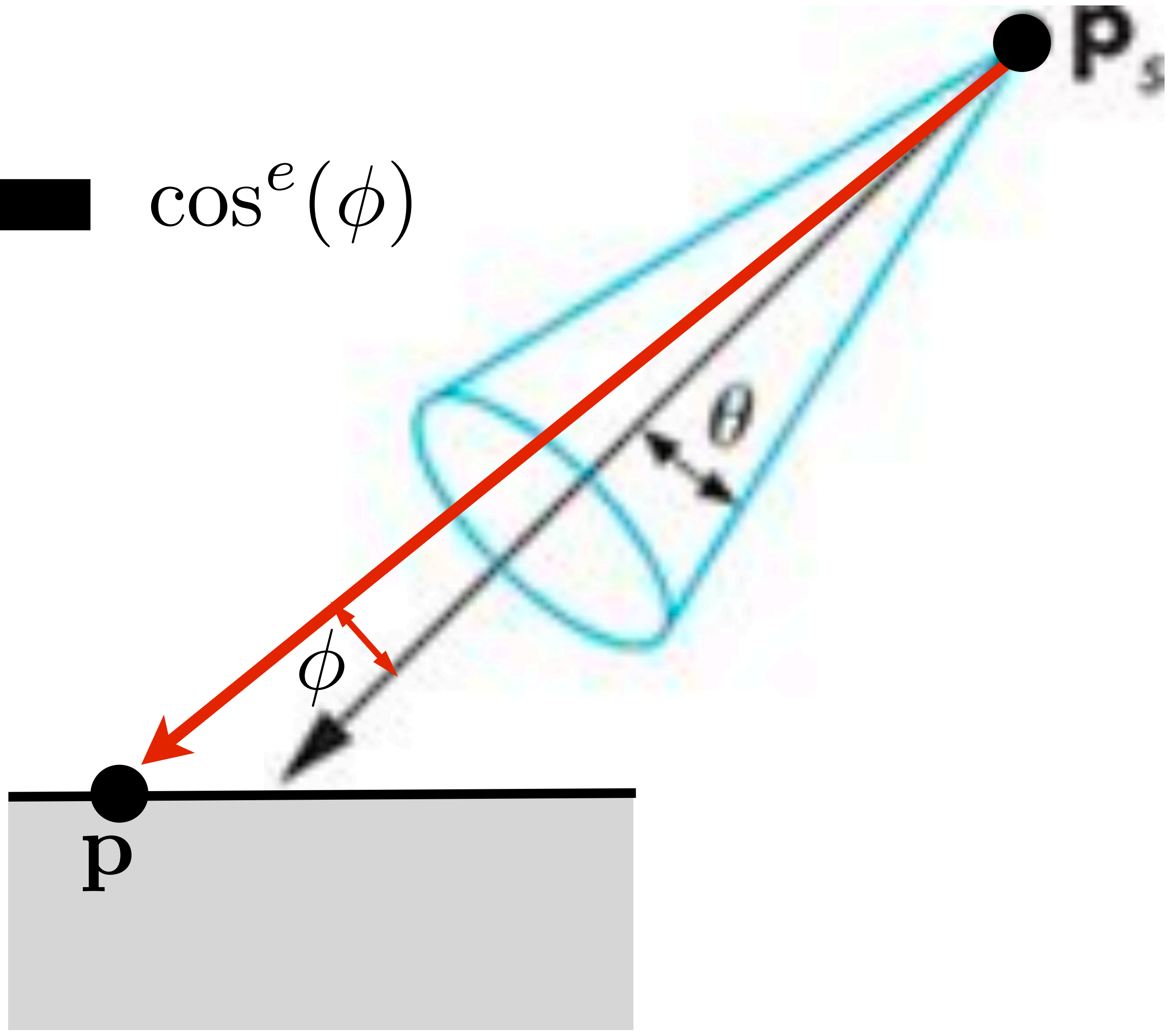


Spotlights

Intensity



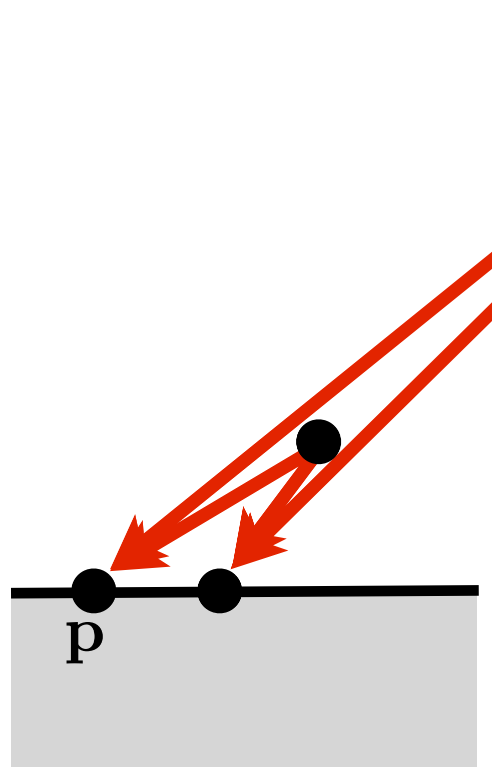
$\cos^e(\phi)$



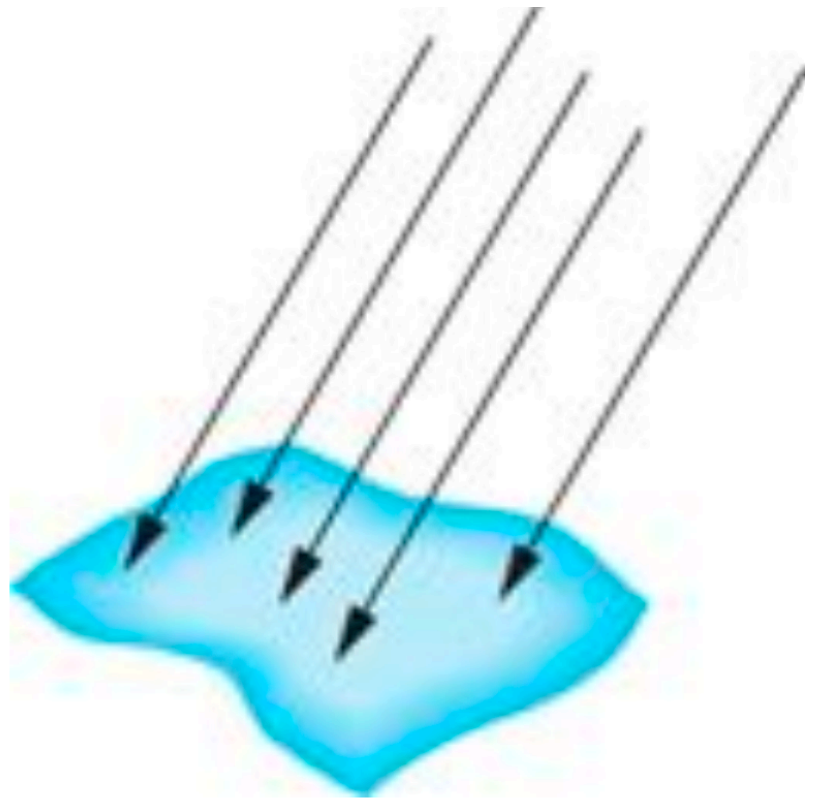
$\cos^e(\phi)\ i(\mathbf{p}, \mathbf{p}_s)$

add an exponent for greater control
final result is like point light but modified by this cone

Distant light source

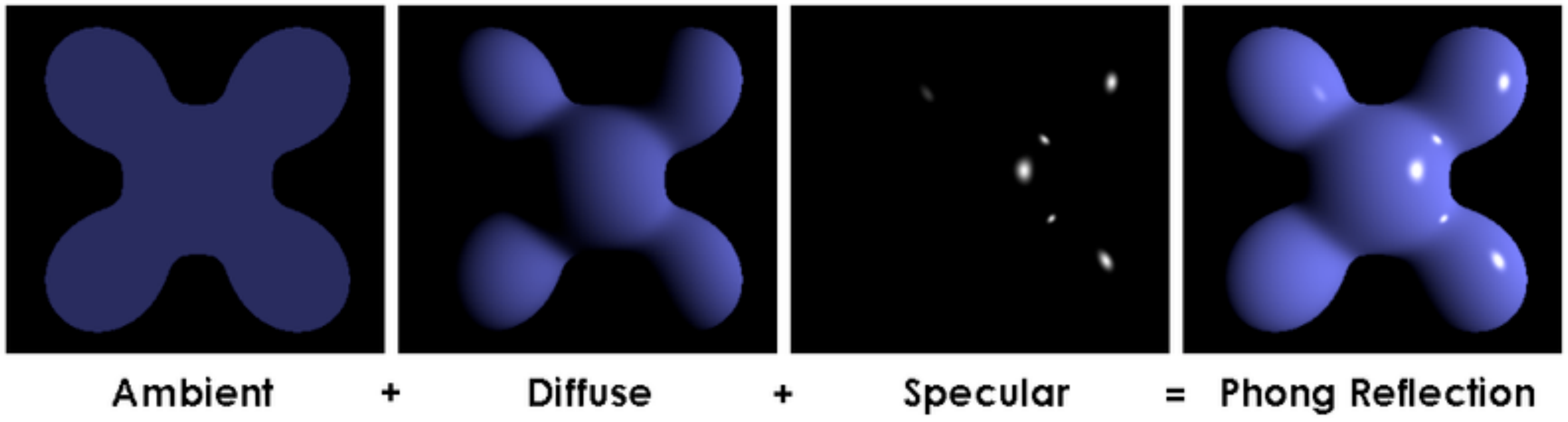


characterized by direction



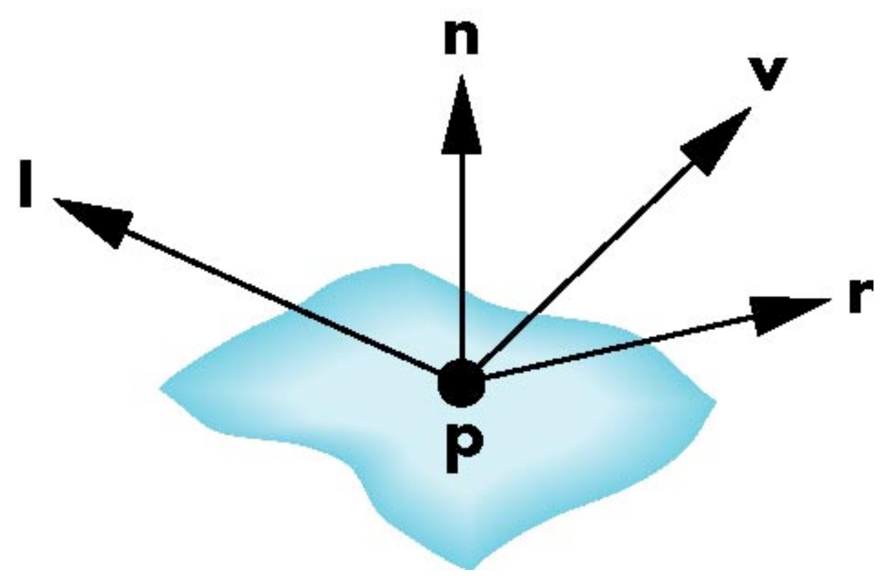
most shading calculations require direction from the surface point to the light source position
if the light source is very far, the direction vectors don't change
e.g., sun
characterized by direction rather than position

Phong Reflection Model



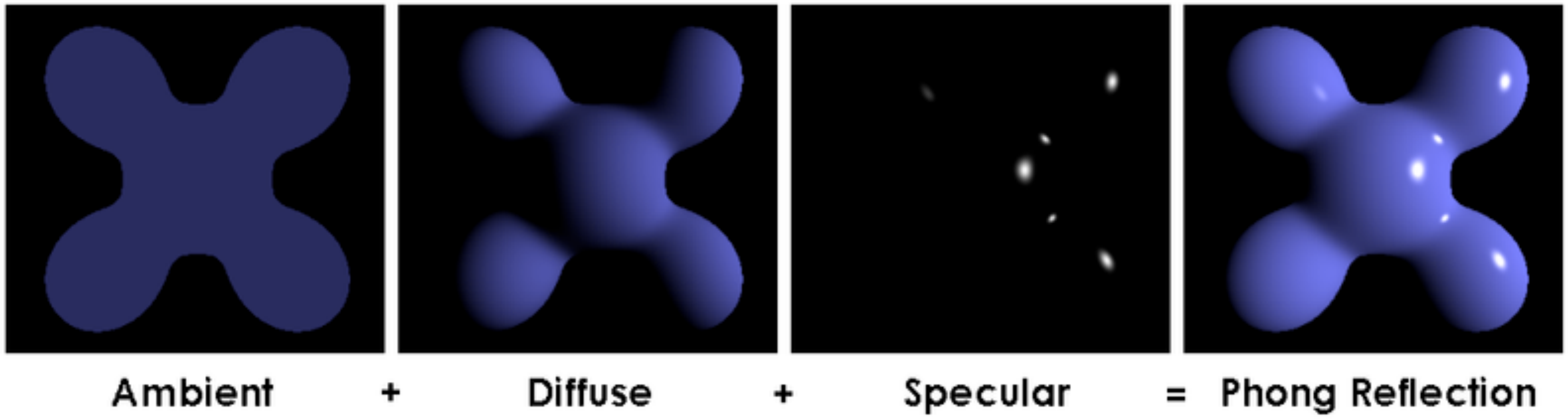
Brad Smith, Wikimedia Commons

- •efficient, reasonably realistic
- •3 components
- •4 vectors



- **l** to light source
- **n** surface normal
- **v** to viewer
- **r** perfect reflector (function of **n** and **l**)

Phong Reflection Model

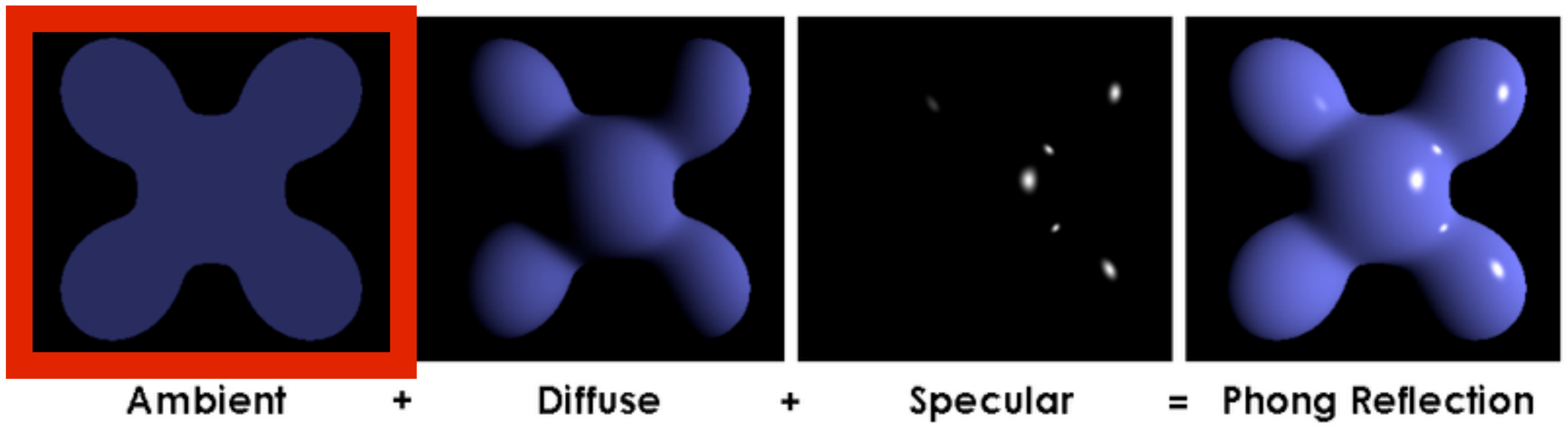


Brad Smith, Wikimedia Commons

for each *light source* and each *color channel*:

$$\underbrace{I}_{\boxed{\text{color intensity}}} = I_a + I_d + I_s = \underbrace{L_a}_{\boxed{\text{illumination}}}\underbrace{R_a}_{\boxed{\text{reflectance}}} + L_d R_d + L_s R_s$$

Ambient reflection



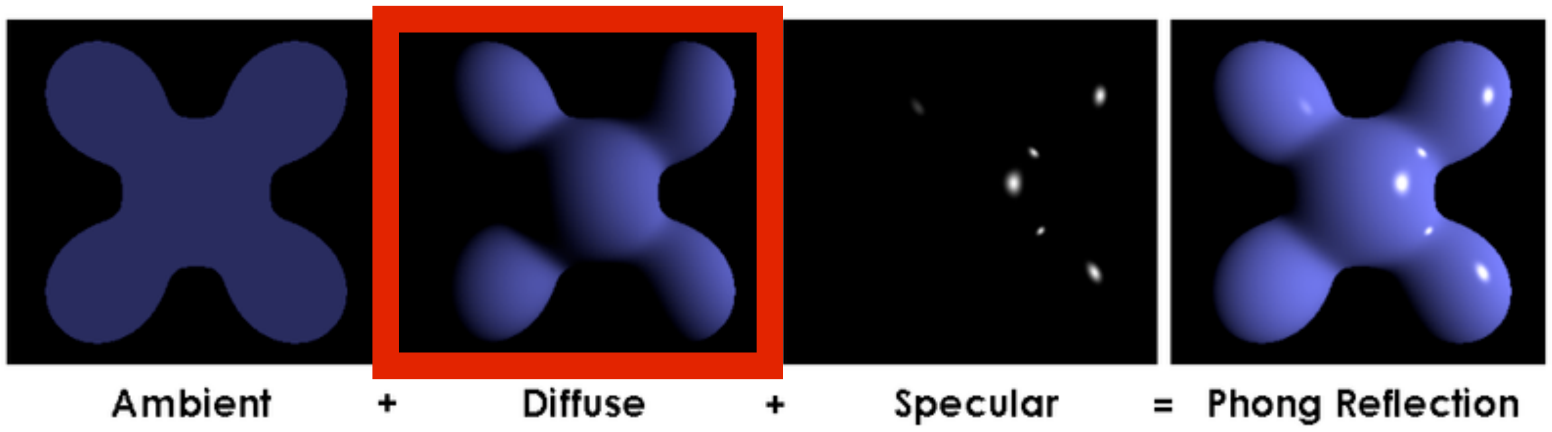
different ambient coefficients for different colors

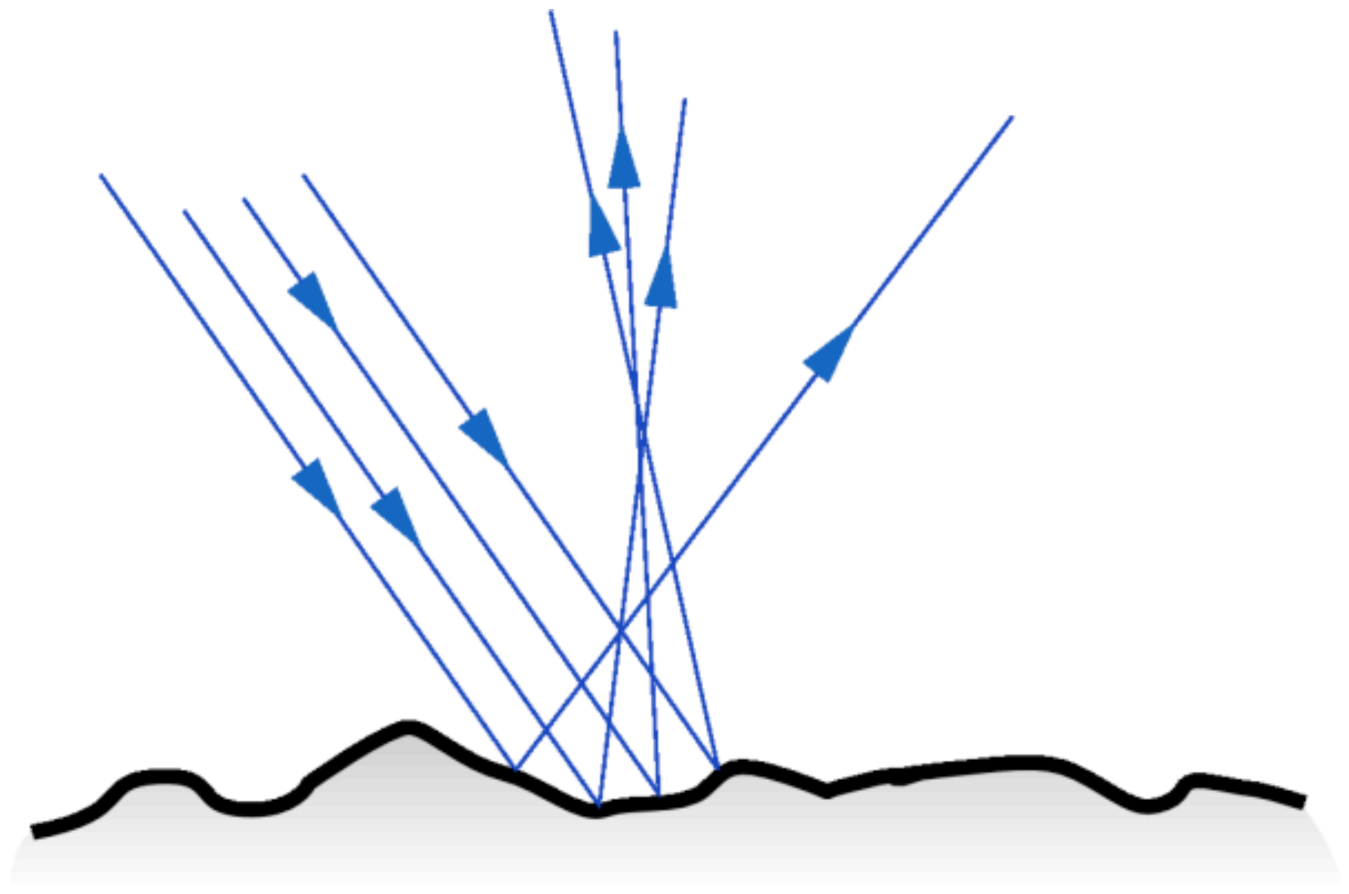
$$I_a = k_a L_a \qquad 0 \leq k_a \leq 1$$

ambient reflection coefficient

e.g., white light shining on the object will be reflected differently in red, green, blue channels
e.g., more red and blue reflection here

Diffuse reflection

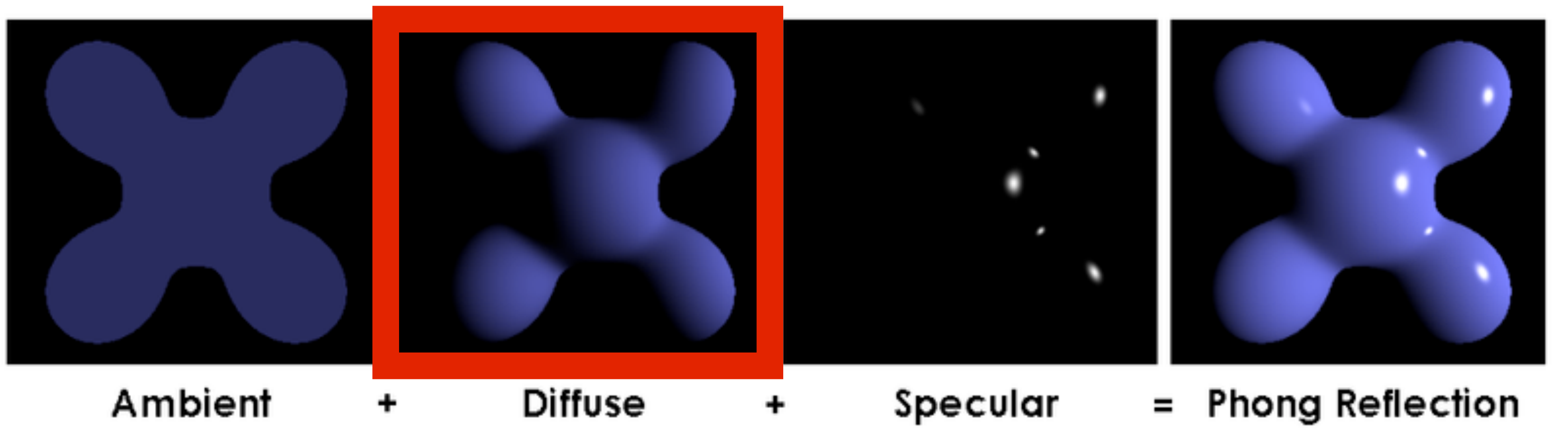




http://emmybella.wordpress.com/the-laws-of-reflection/

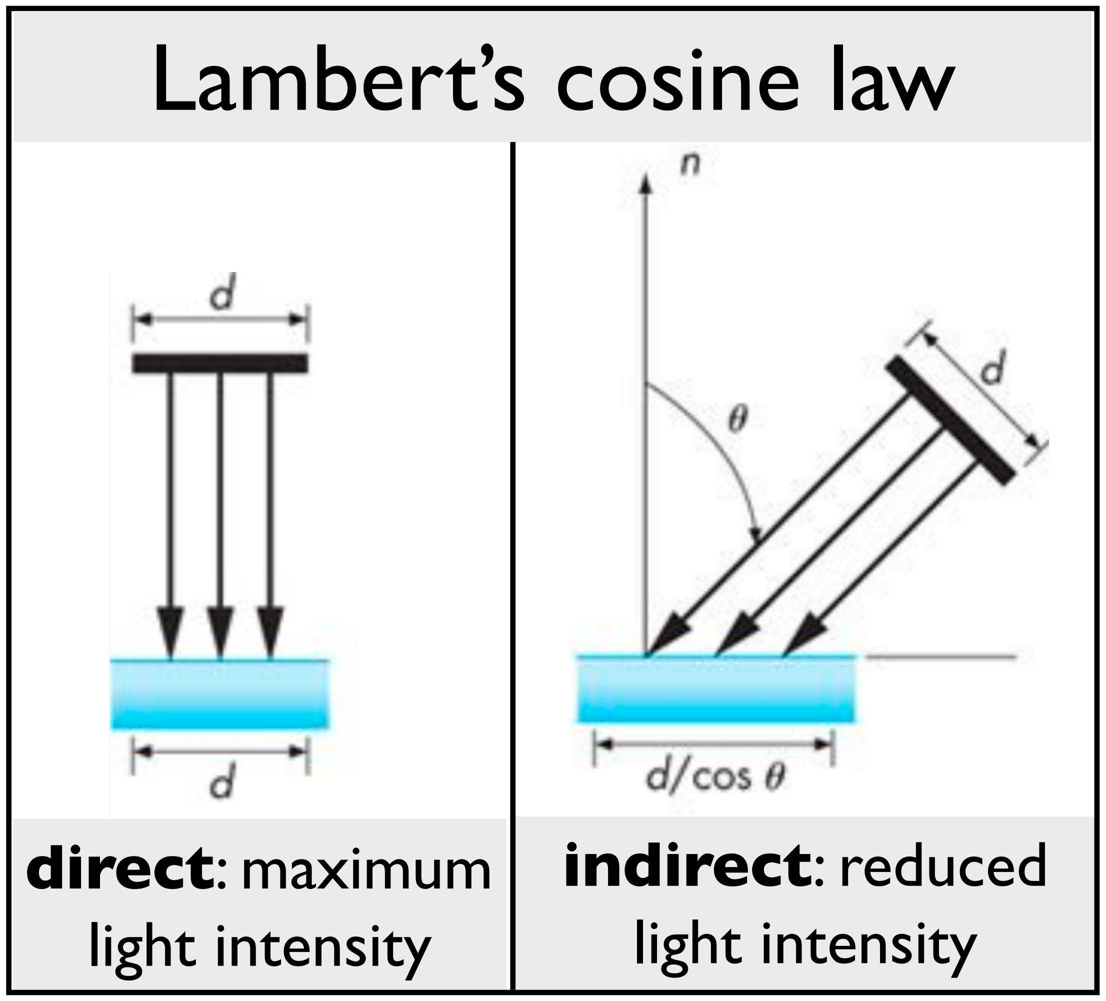
e.g., paper, unfinished wood, unpolished stone

Diffuse reflection



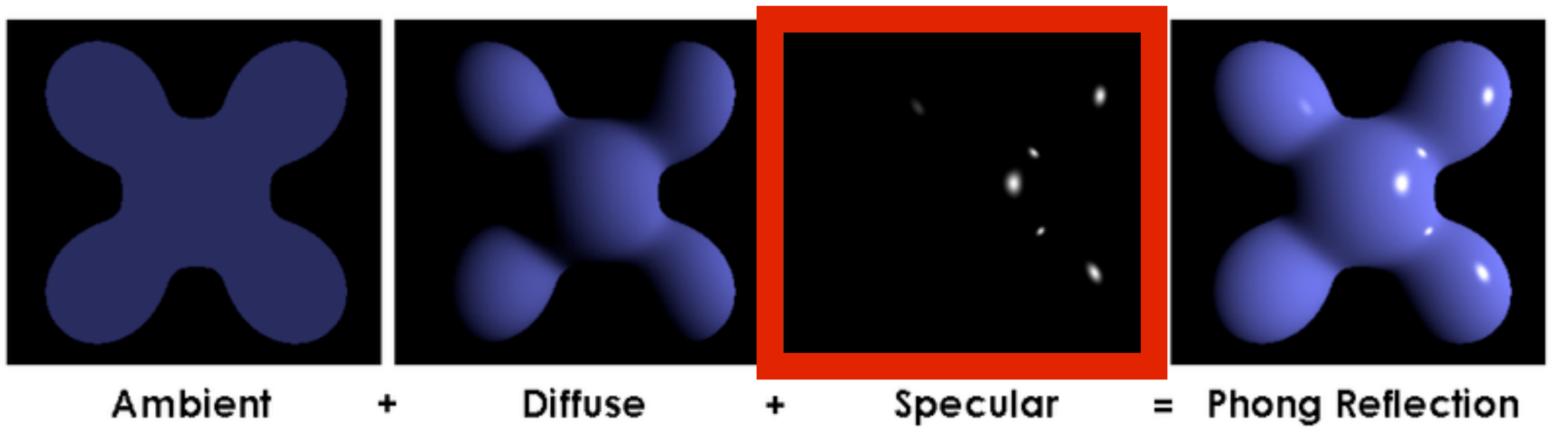
$$I_d = k_d \max(0, \mathbf{l} \cdot \mathbf{n}) L_d$$

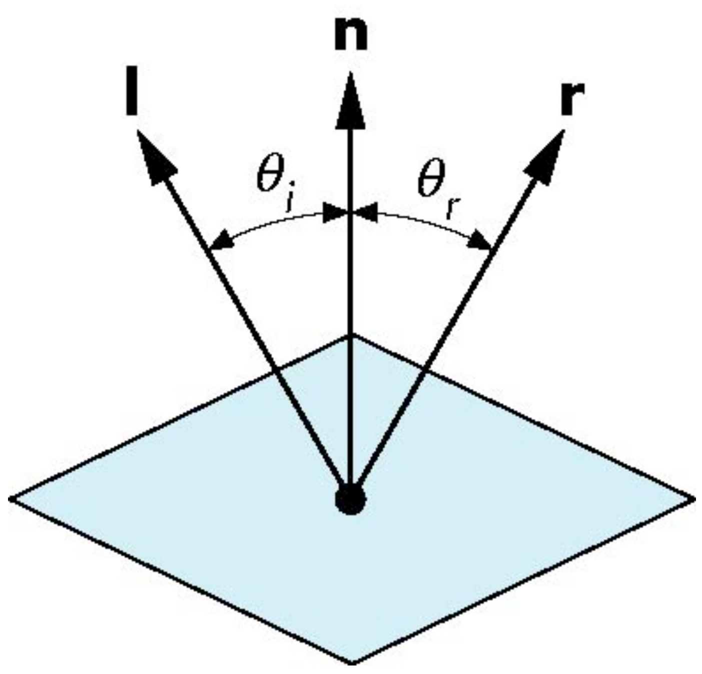
diffuse reflection coefficient


Lambert's cosine law
d
d
n
θ
d
d/cos θ
direct: maximum light intensity
indirect: reduced light intensity

- the light is reduced by cos of angle
  - this is because same amount of light is spread over larger area when light comes in at an angle

Specular reflection



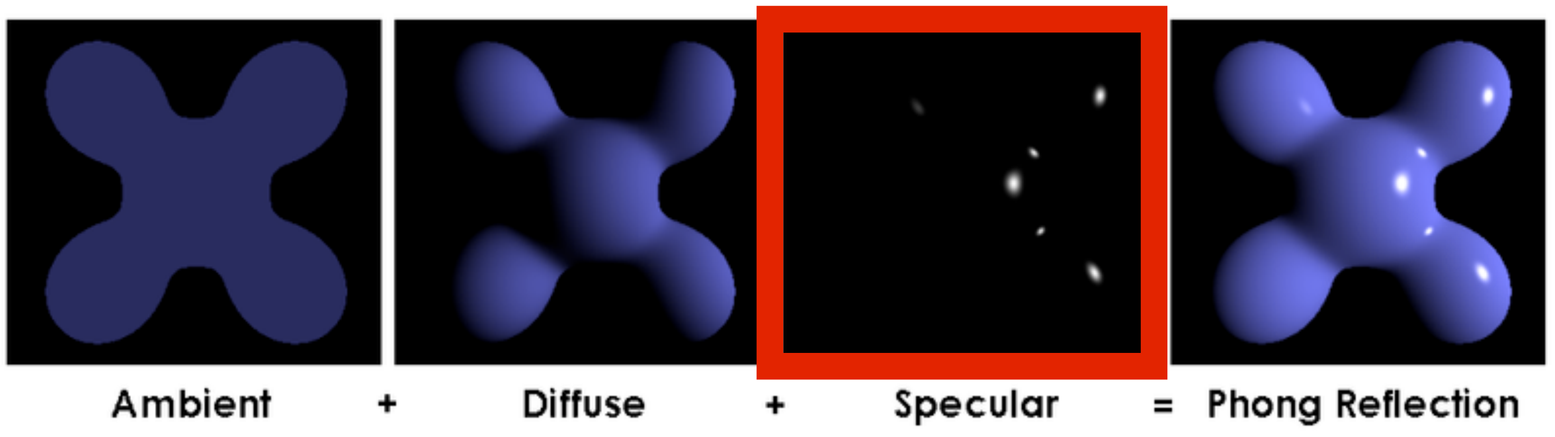


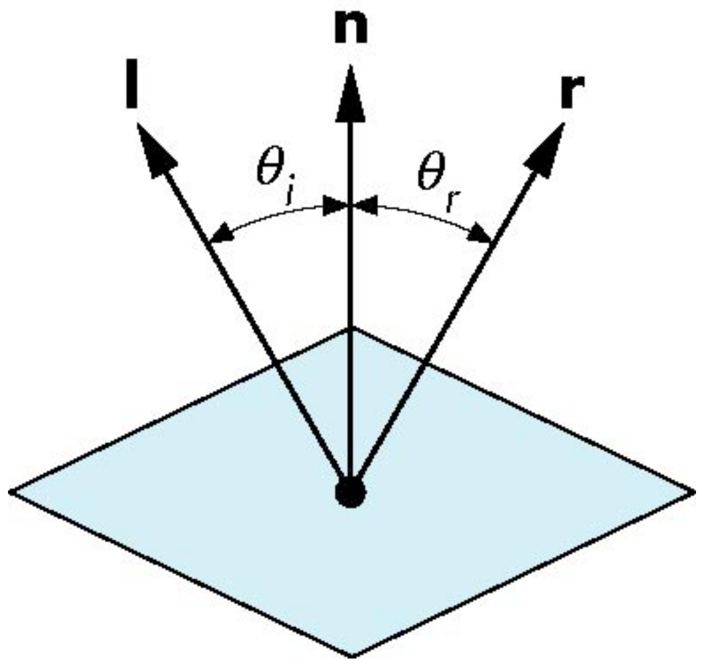
Ideal reflector:

- angle of incidence= angle of reflection
- viewer position matters

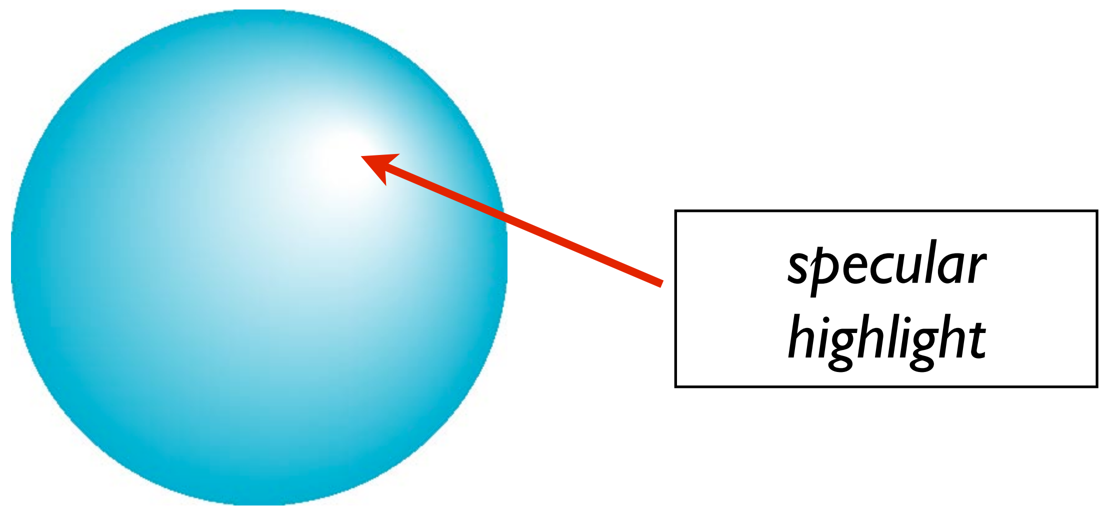
e.g., white light shining on the object will be reflected differently in red, green, blue channels
e.g., more red and blue reflection here

Specular reflection





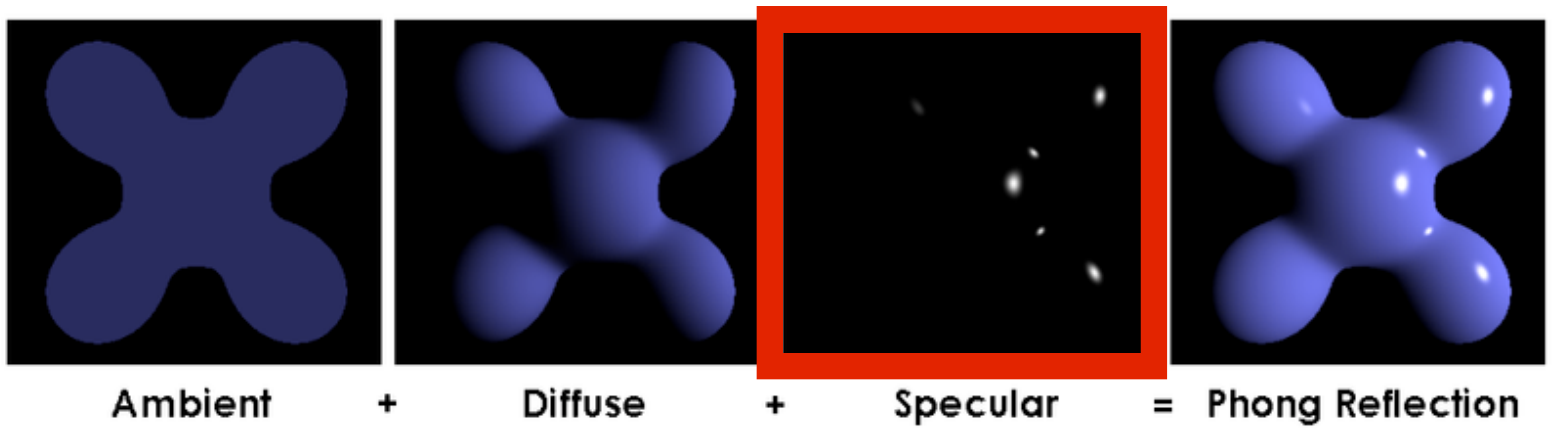
Specular surface:

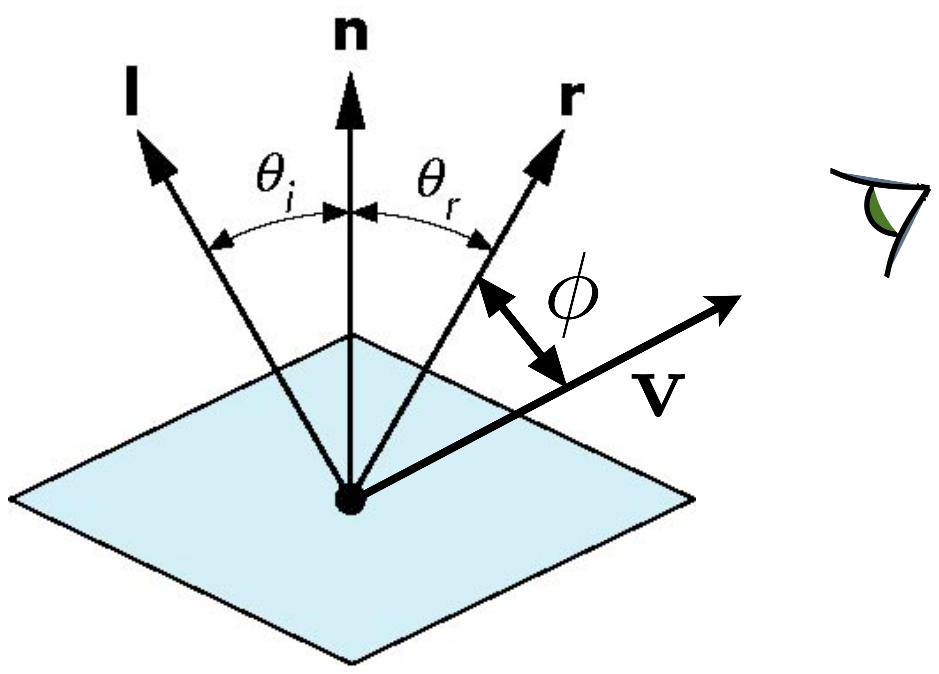


specular reflection is strongest in mirror reflection direction

area of specular highlight depends on how smooth the surface is

Specular reflection





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

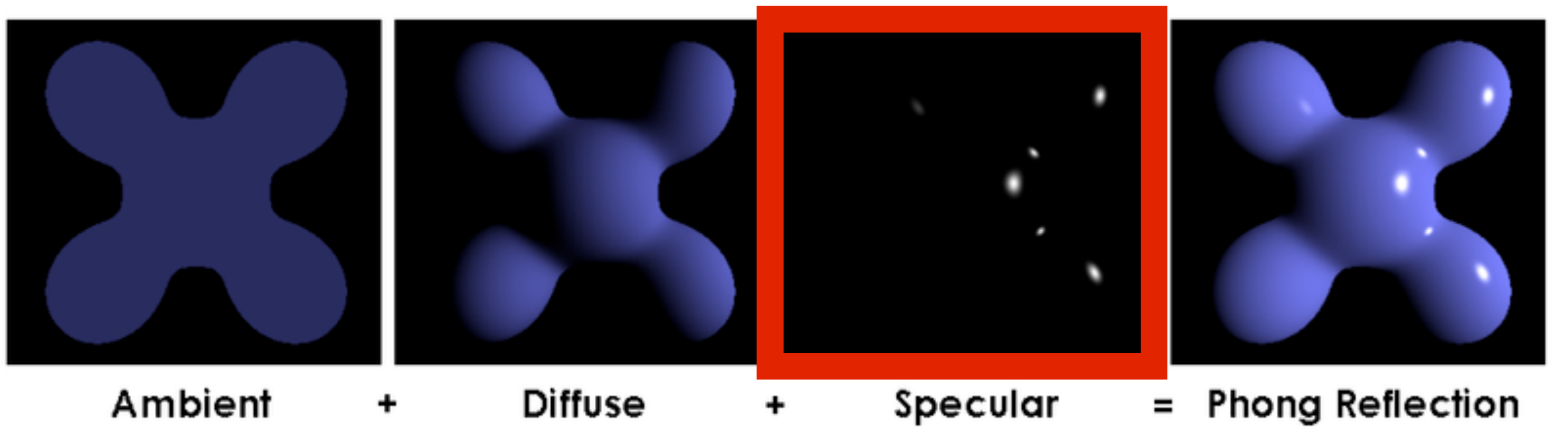
specular reflection coefficient

shininess coefficient

specular reflection is strongest in mirror reflection direction
drops off with increasing angle ϕ

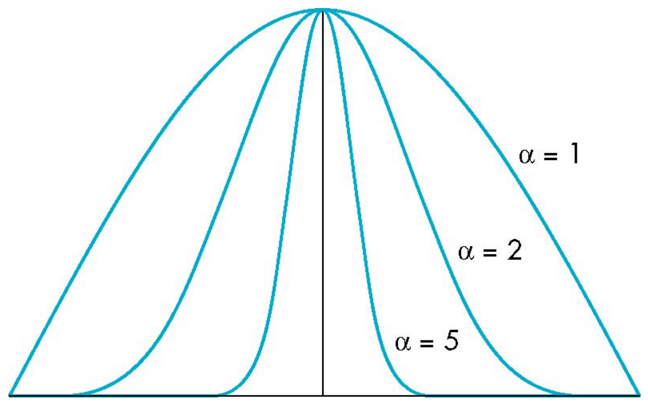
Phong proposed this model

Specular reflection



$$I_s = k_s L_s \max(0, \cos\phi)^\alpha$$

shininess coefficient

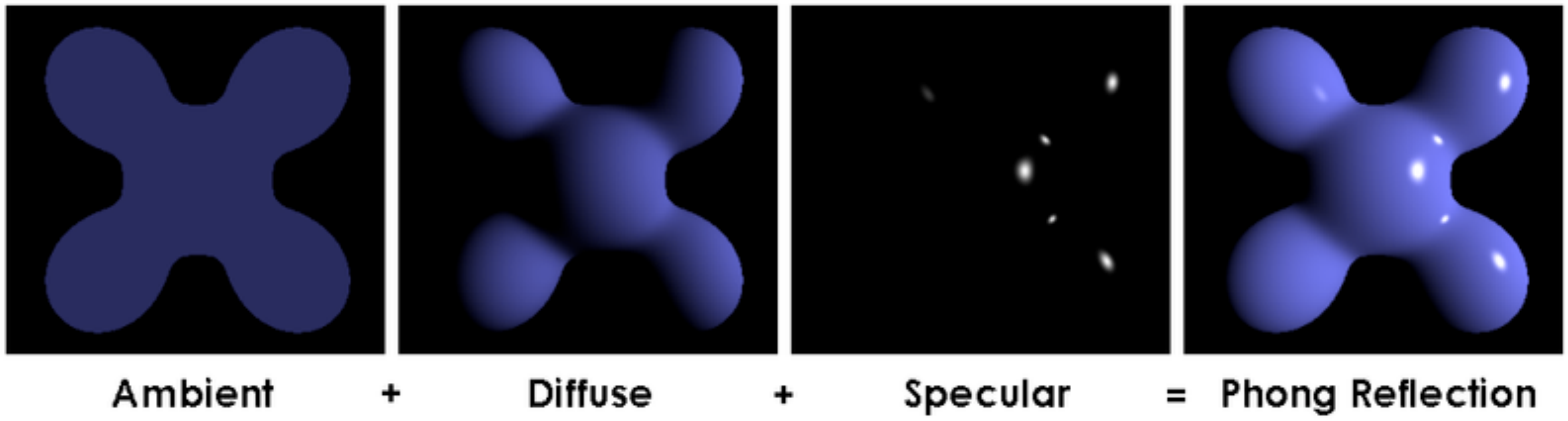


$\alpha = 5..10$ plastic

$\alpha = 100..200$ metal

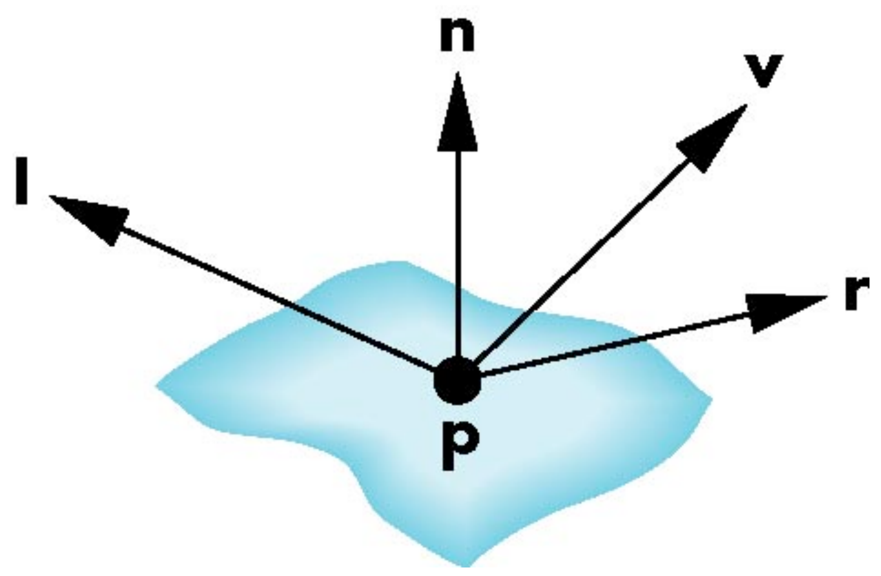
Phong proposed this model
clamp to 0 -- avoid negative values
the fuzzy highlight was too big without an exponent

Phong Reflection Model

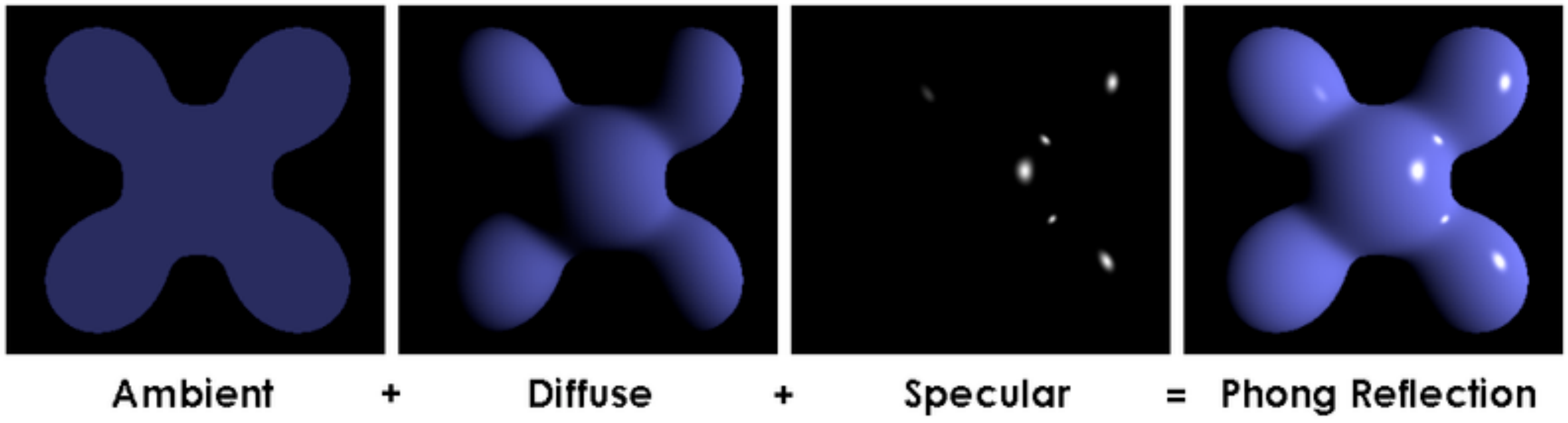


Brad Smith, Wikimedia Commons

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

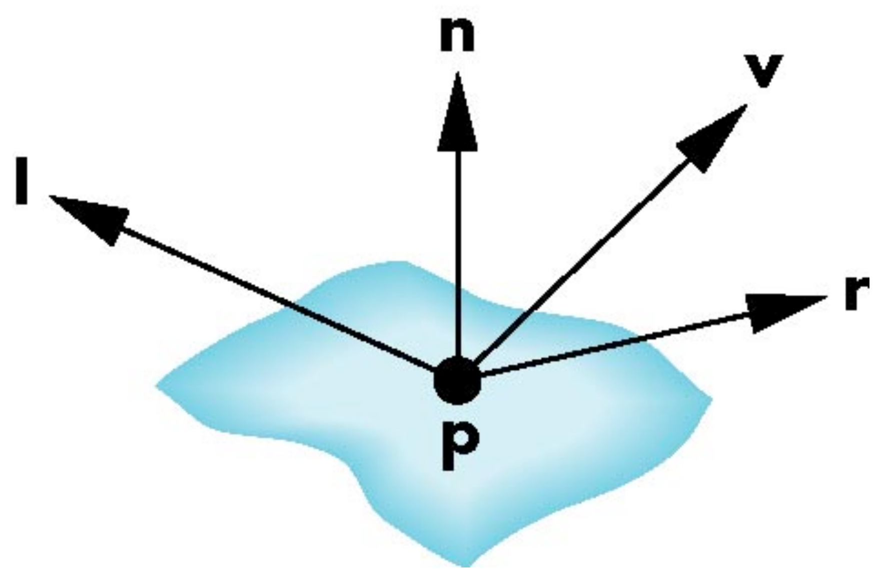


Phong Reflection Model

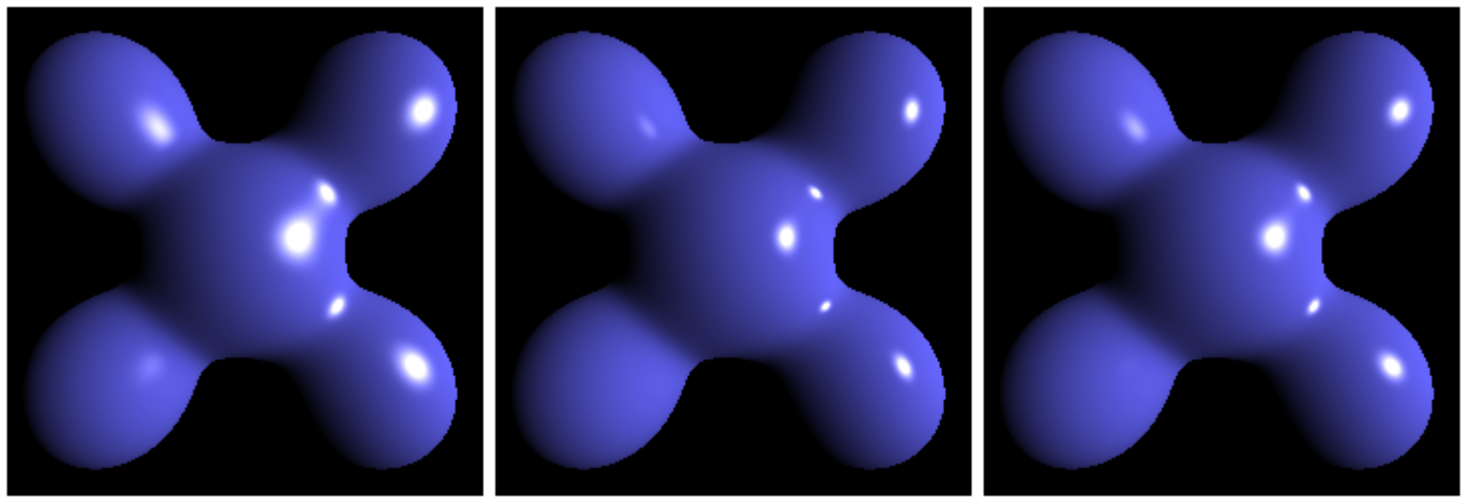


Brad Smith, Wikimedia Commons

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



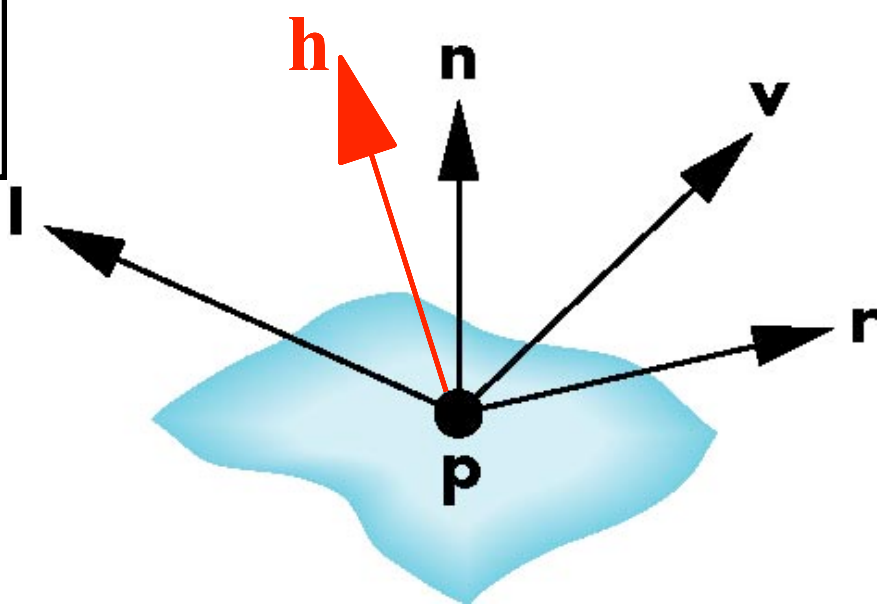
Alternative: Blinn-Phong Model



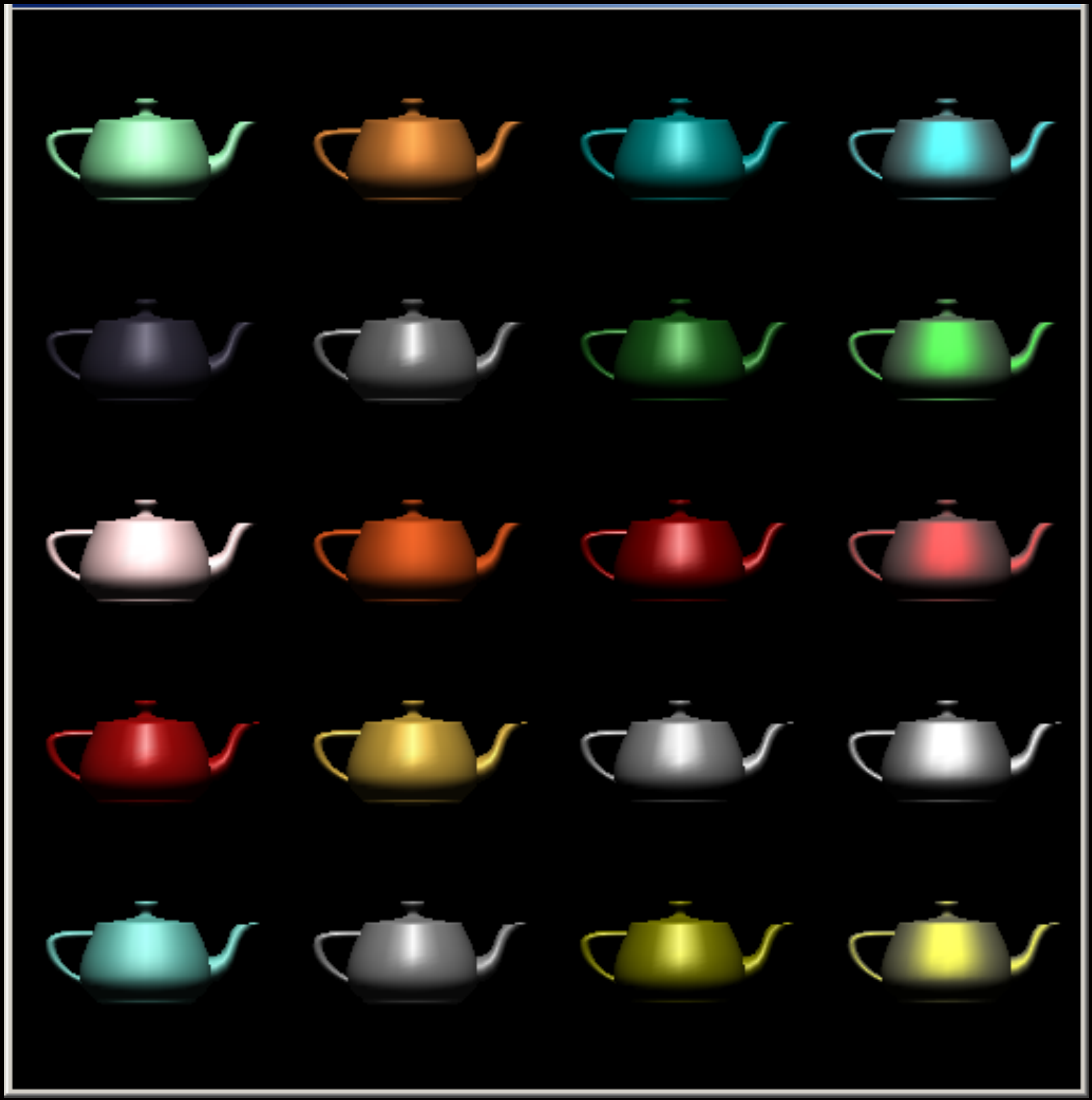
halfway vector

$$\boxed{\mathbf{h} = \frac{\mathbf{l}+\mathbf{v}}{|\mathbf{l}+\mathbf{v}|}}$$

$$\begin{aligned} I &= I_a + I_d + I_s \\ &= k_a L_a + k_d L_d \max(0, \mathbf{l}\cdot\mathbf{n}) \\ &\quad + k_s L_s \max(0, \mathbf{h}\cdot\mathbf{n})^{\alpha} \end{aligned}$$



replace **v.r** with **h.n**
this way we don’t have to recompute **r**, which depends on **n**
h does not depend on **n**
saves a lot especially for directional lights and constant viewing direction



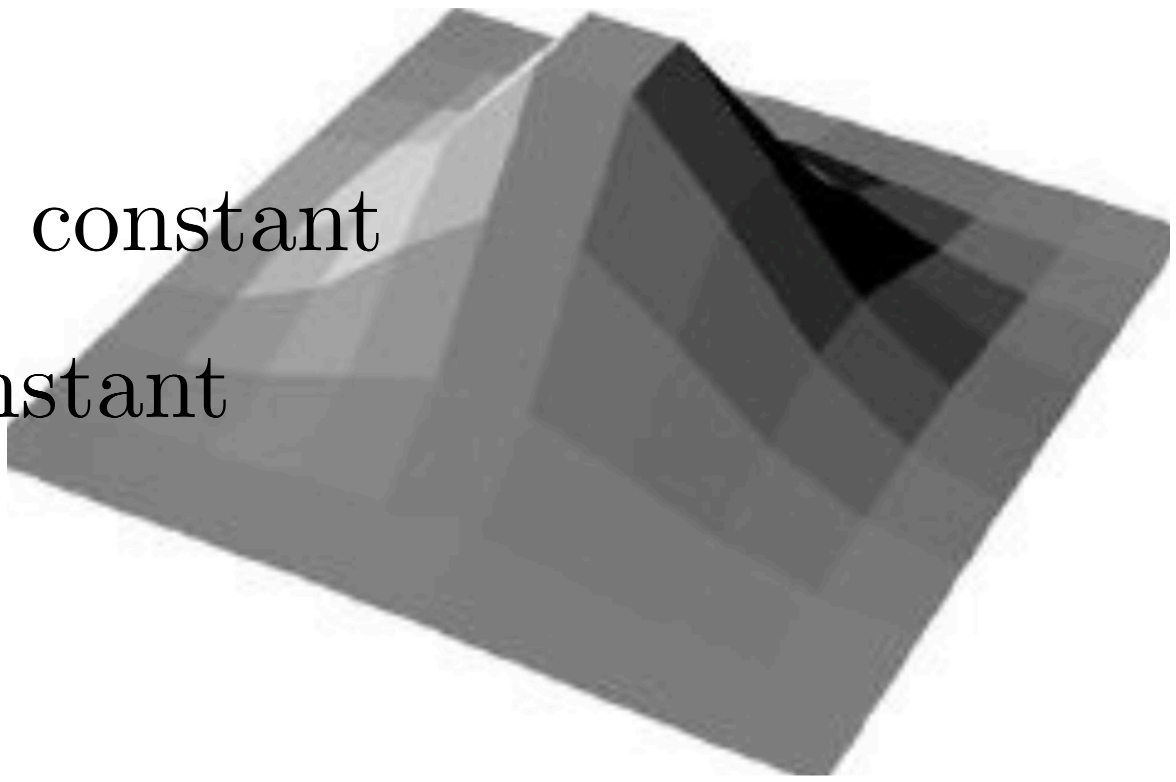
p

10: eggshell
100: shiny
1000: glossy
10000: mirror-like

Shading Polygonal Geomtery

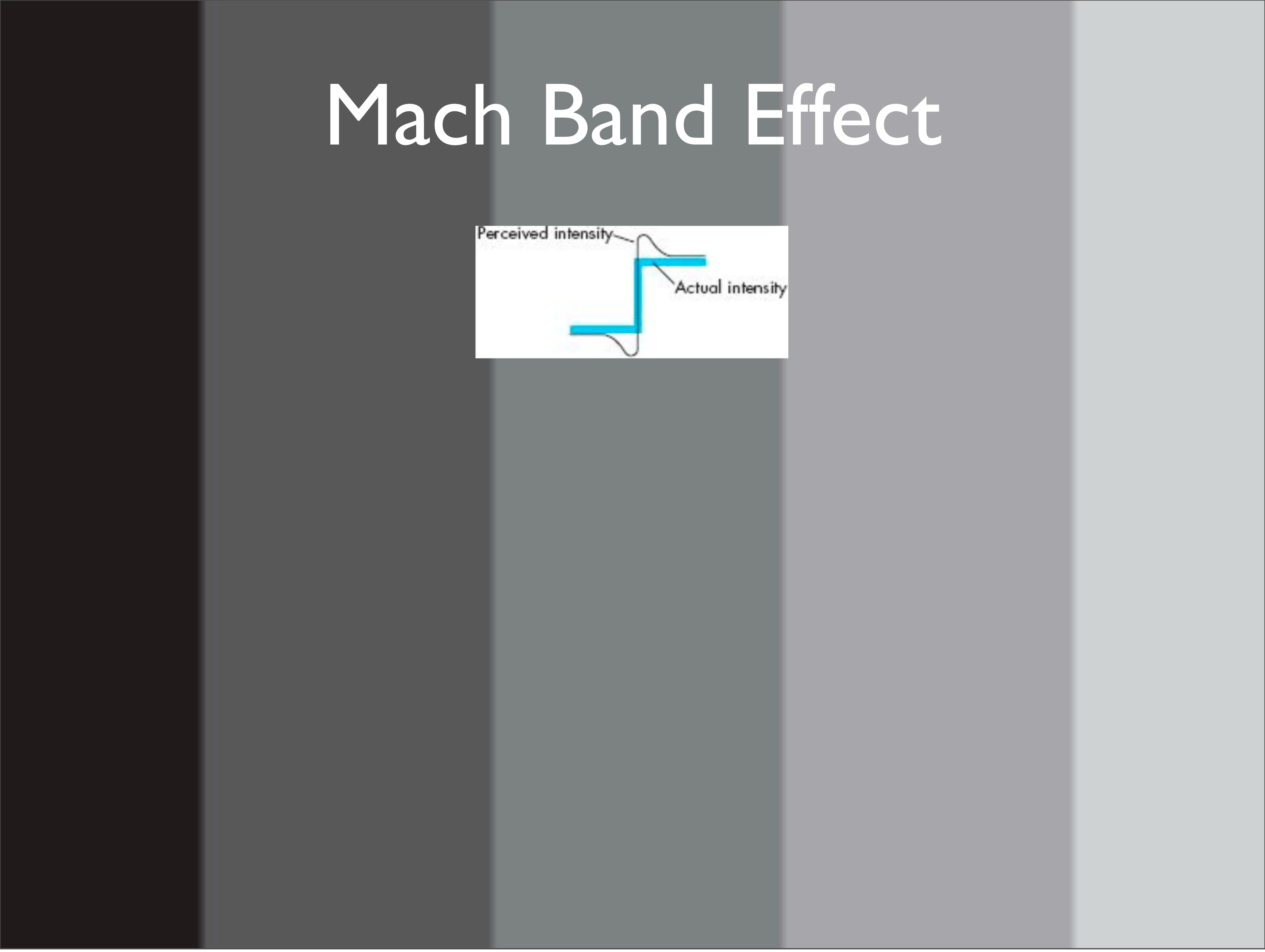
Constant ("Flat") Shading

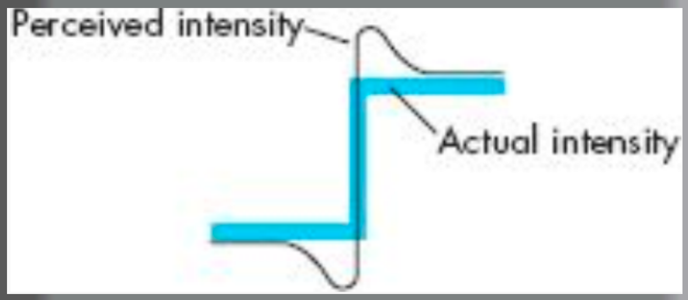
- Simplest approach
- apply illumination model **once** for each polygon
- valid when:
  - light source at ∞ $\mathbf{l} \cdot \mathbf{n} = \text{constant}$
  - viewer at ∞ $\mathbf{v} \cdot \mathbf{n} = \text{constant}$
  - object is actually faceted



If light source or viewer is not at infty, need heuristic for picking color – e.g., first vertex, or polygon center
– does not produce variations in gradation

Mach Band Effect

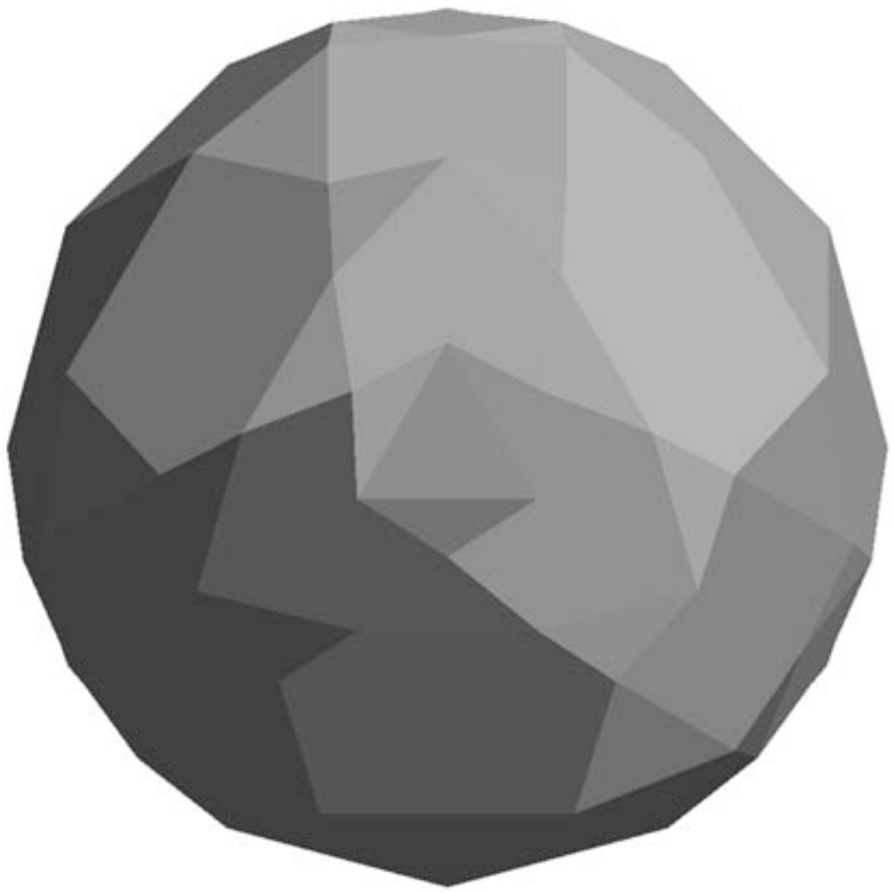




This effect makes flat shading seem even worse

Shading Polygons

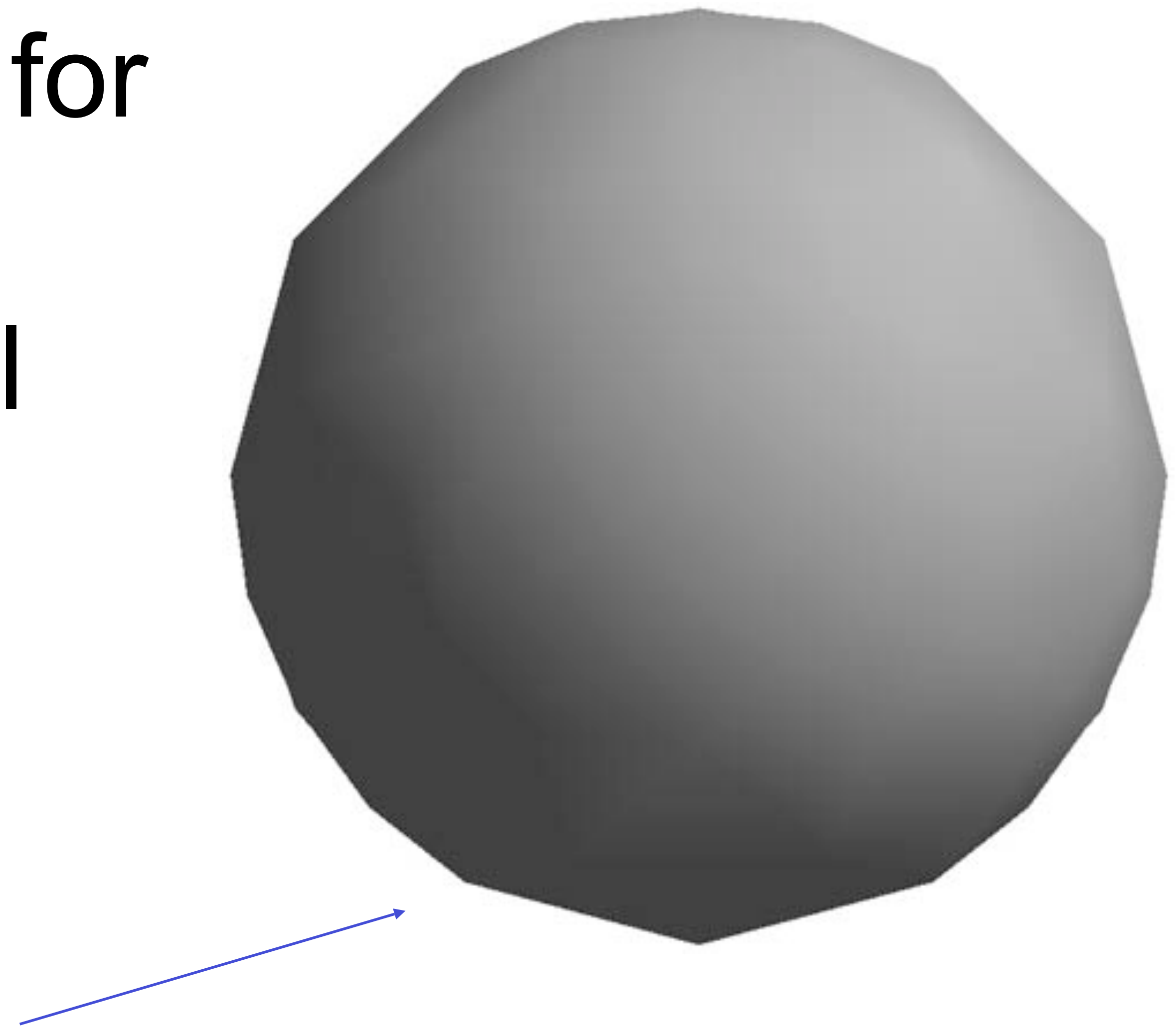
- Polygons often approximate curve surfaces but are inherently flat
- Consider polygonal ‘sphere’
- Want to smooth the rough face of each surface facet
- How do we fix this?



even if we applied the lighting model at each pixel, we would still get faceted appearance

Smooth Shading

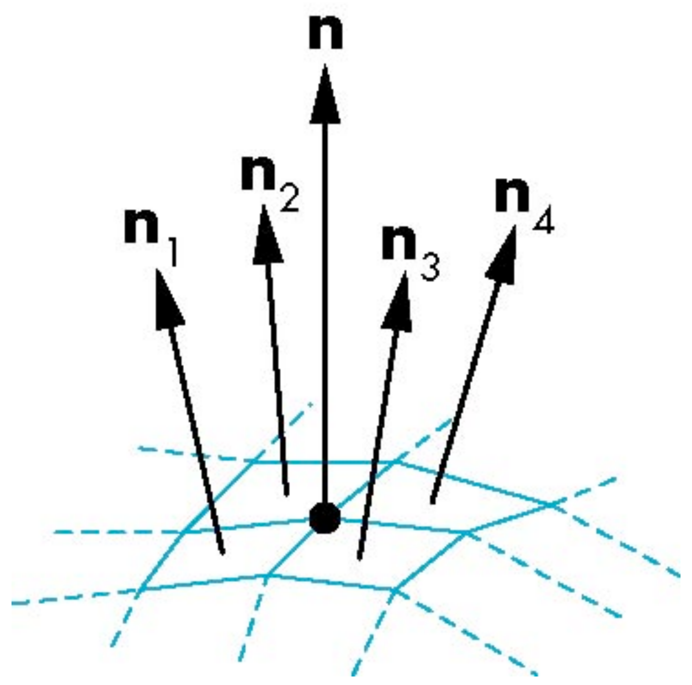
- We can simply find a new normal at each vertex for a sphere
- Easy for sphere model
  - If centered at origin $\mathbf{n} = \mathbf{p}$
- Results in smoother shading
- Note *silhouette edge*



Vertex Normals

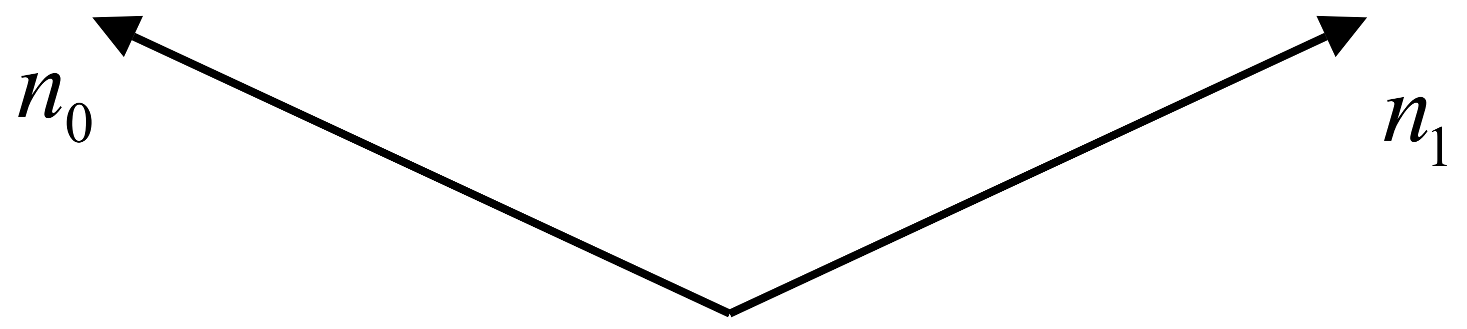
- The sphere example is not general because we knew the normal at each vertex analytically
- For polygonal meshes, Gouraud 1971 proposed we use the average of normals around a mesh vertex

$$\mathbf{n} = \frac{\sum \mathbf{n}_i}{|\sum \mathbf{n}_i|}$$



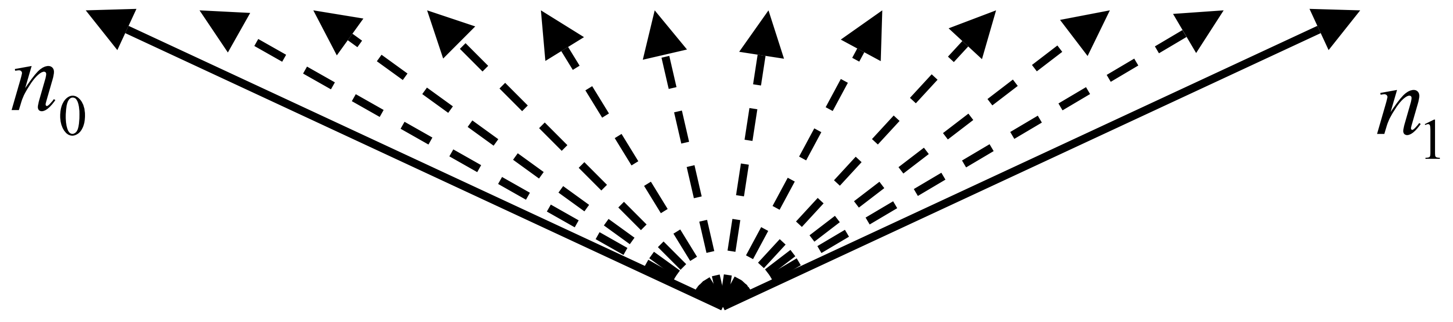
Interpolating Normals

- Must renormalize



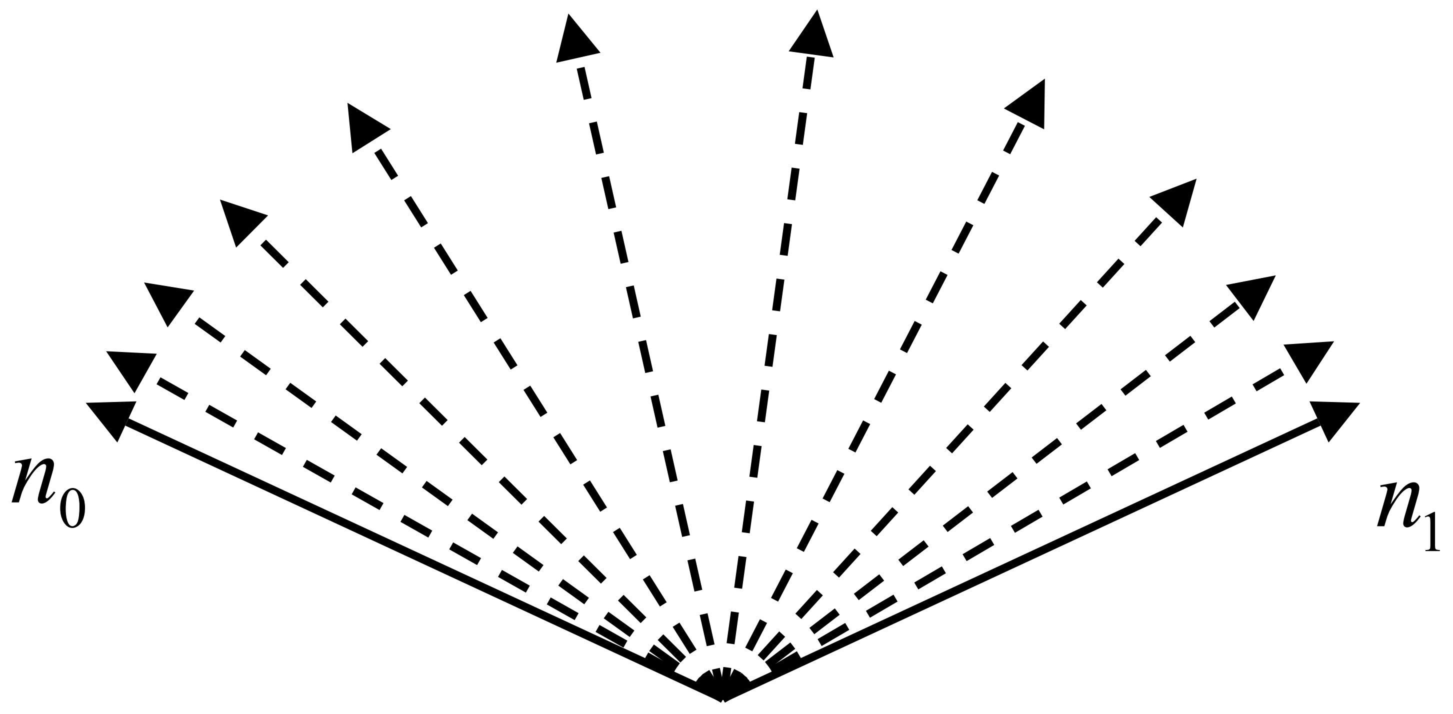
Interpolating Normals

- Must renormalize



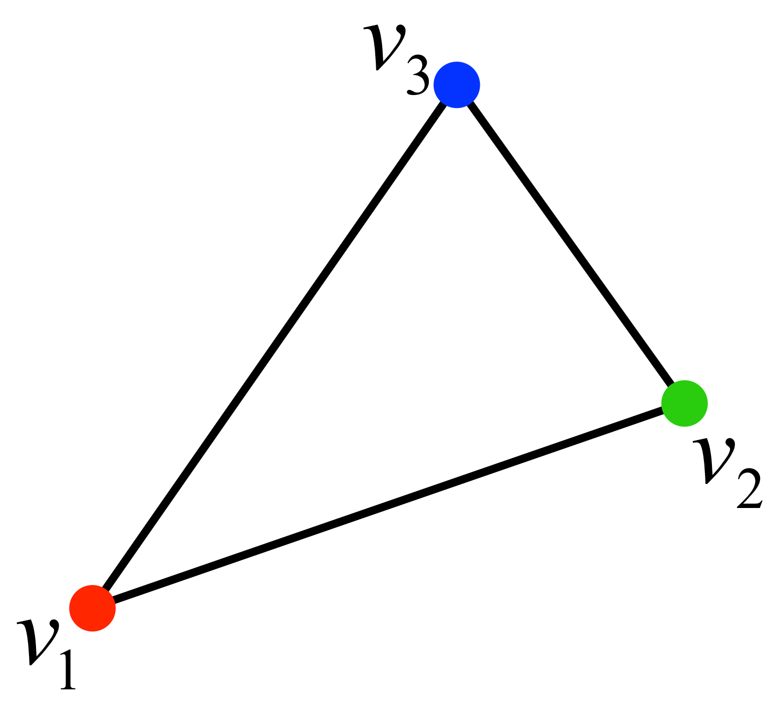
Interpolating Normals

- Must renormalize



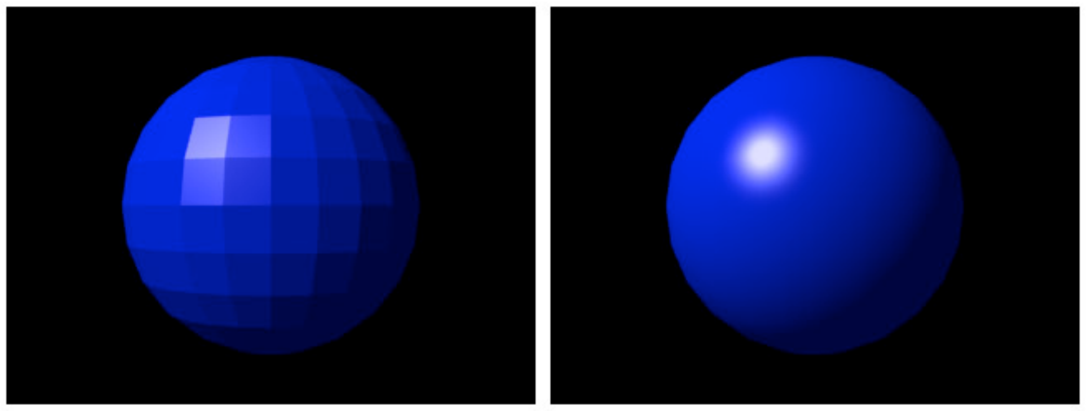
Gouraud Smoothing

- AKA: *intensity interpolation* shading
- Used in OpenGL
- Find vertex normals
- Apply Phong light model at each vertex
- Interpolate vertex shades across each polygon

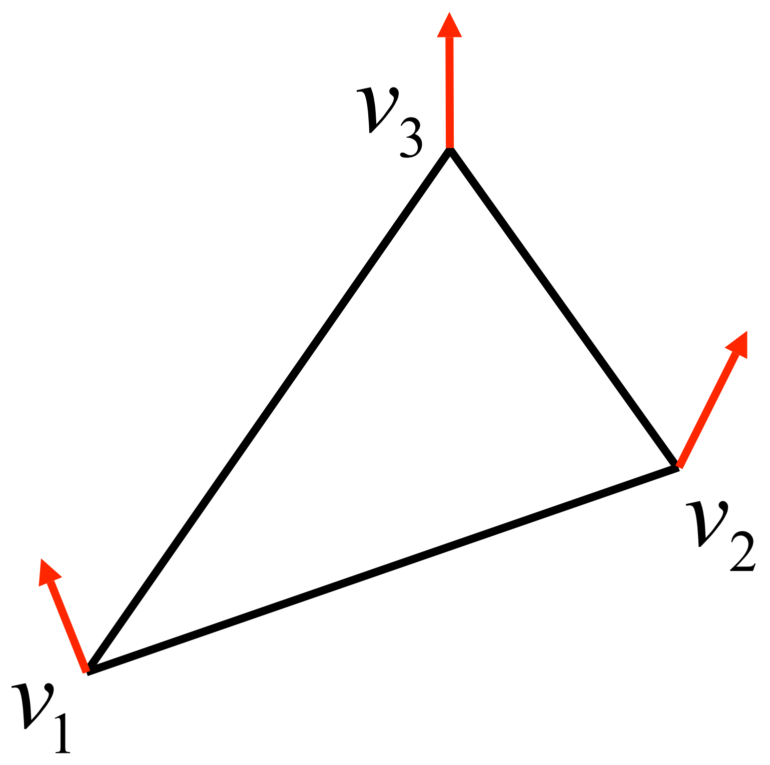


Phong Smoothing

- Not the Phong lighting model!
- AKA: *normal-vector interpolation* shading
- Find vertex normals
- Interpolate vertex normals across edges and then across polygon
- Find shades using normals across polygons

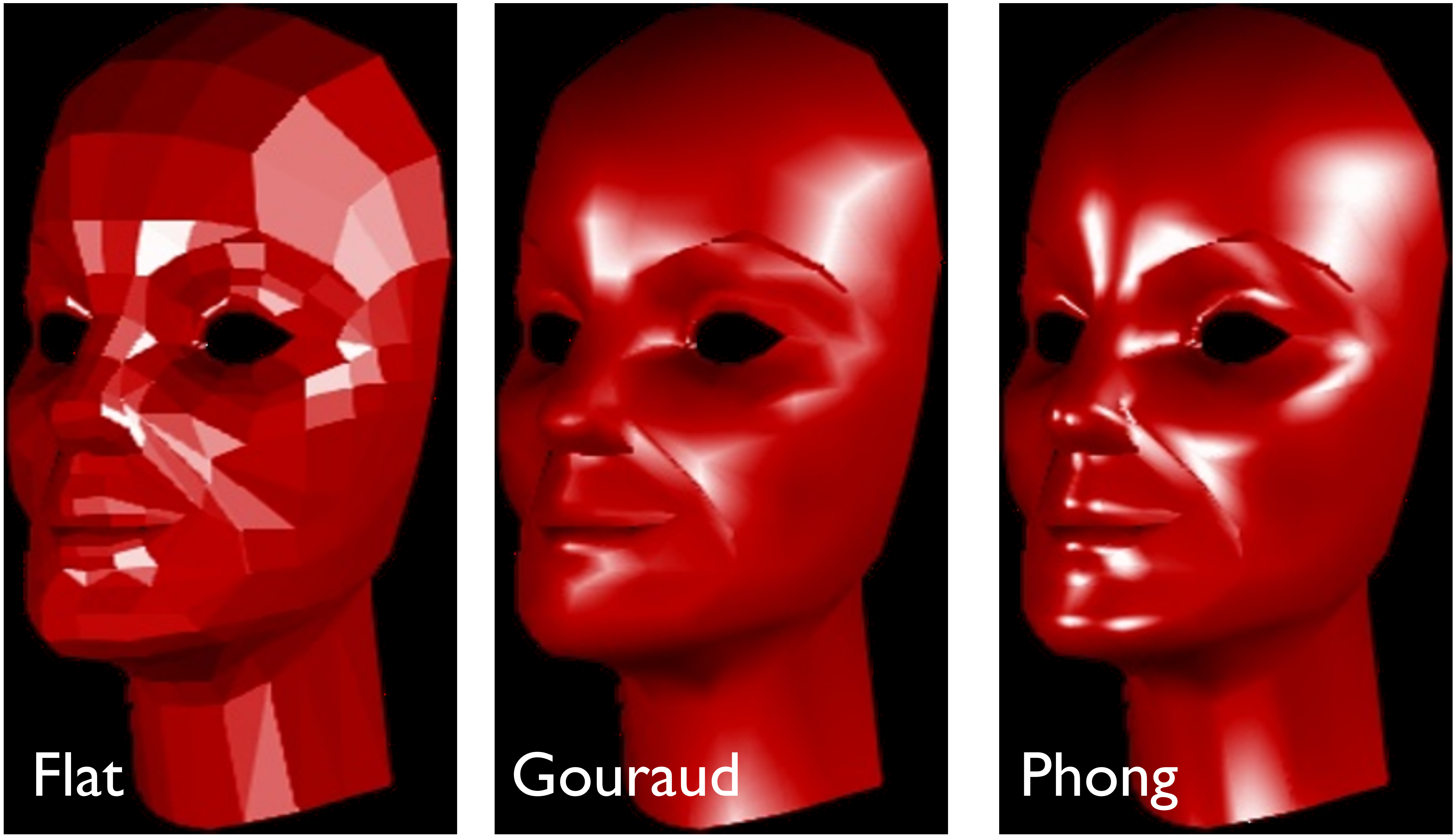


Wikimedia Commons



interpolate normals and make an independent shading calculation
Note: Phong shading requires that the lighting model be applied to each fragment - hence, **per-fragment shading**

Comparison



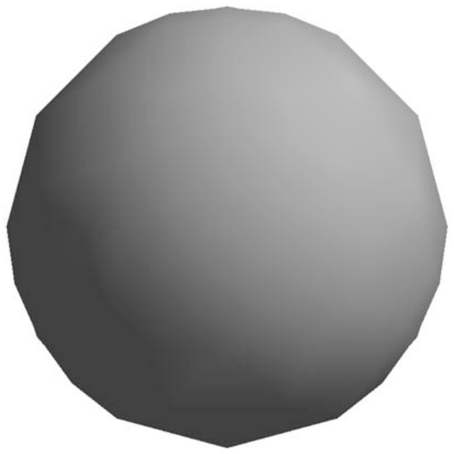
- Phong interpolation looks smoother -- can see edges on the Gouraud model
- but Phong is a lot more work
- both Phong and Gouraud require vertex normals
- both Phong and Gouraud leave silhouettes

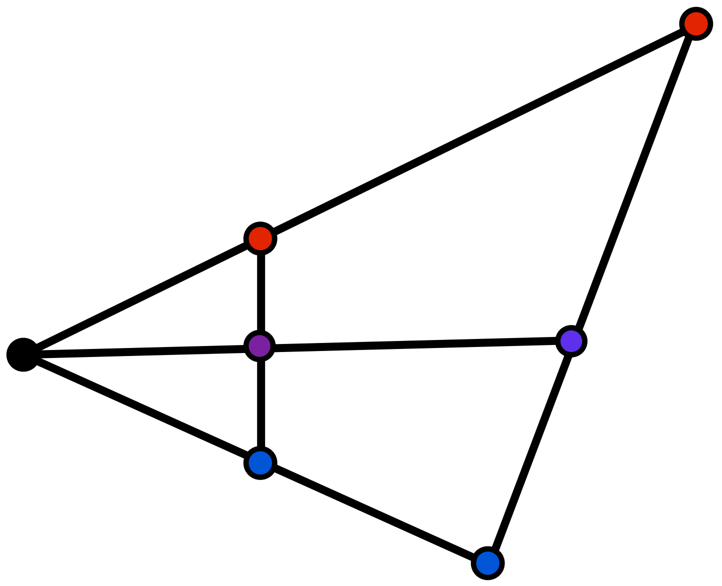
Comparison

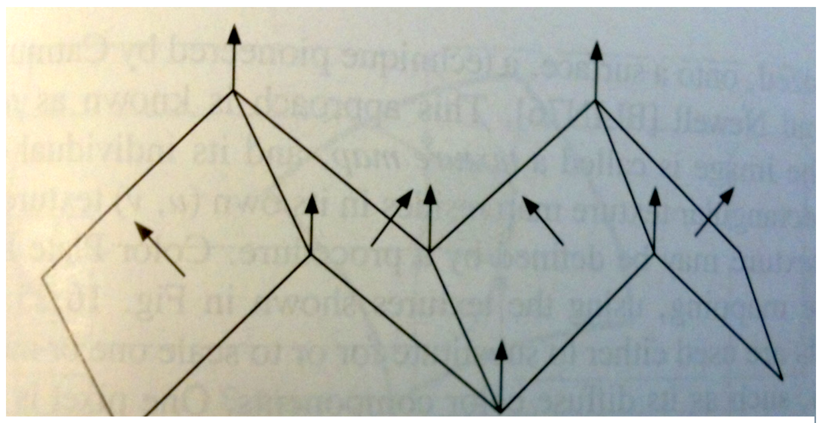
- If the polygon mesh approximates surfaces with a high curvatures, Phong smoothing may look smooth when Gouraud shows edges
- Phong smoothing requires much more work than Gouraud smoothing
- Both need data structures to represent meshes so we can obtain vertex normals
- Both leave the silhouette jagged

Problems with Interpolated Shading

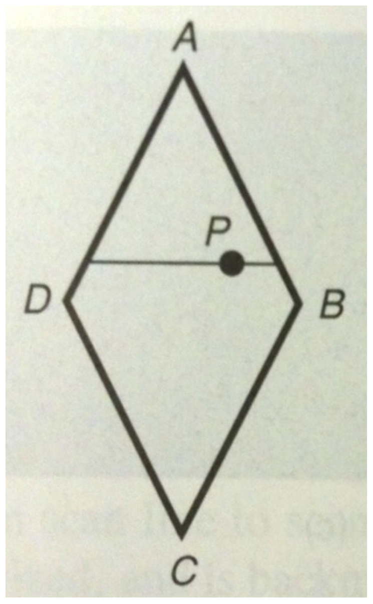
- Polygonal silhouette
- Perspective distortion
- Orientation dependence
- Unrepresentative surface normals

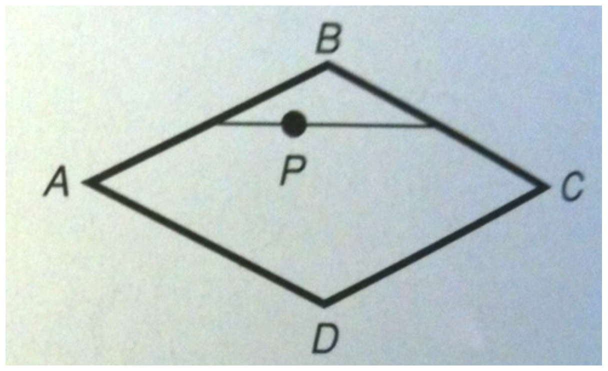






Foley, van Dam, Feiner, Hughes



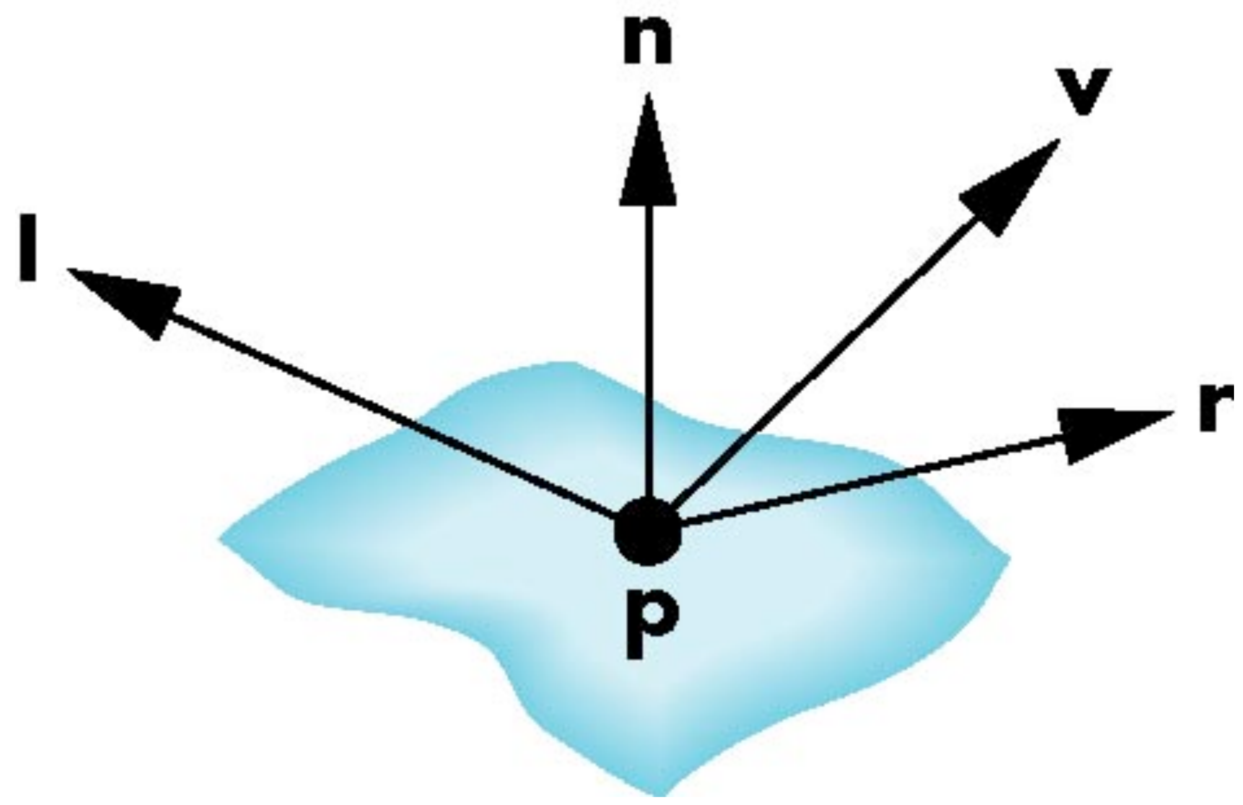


Foley, van Dam, Feiner, Hughes

Shading and Lighting in OpenGL

Material/Shading

- A simple model that can be computed rapidly
- Includes three components
  - Diffuse
  - Specular
  - Ambient
- Uses four vectors
  - To source, l
  - To viewer, v
  - Normal, n
  - Perfect reflector, r



Steps in OpenGL shading

1. Enable shading and select model
2. Specify lights
3. Specify material properties
4. Specify normals

Enabling Shading

- Shading calculations are enabled by
  - `glEnable(GL_LIGHTING)`
  - Once lighting is enabled, glColor() ignored
- Polygon shading is turned on as:
  - `glShadeModel(GL_SMOOTH) or`
  - `glShadeModel(GL_FLAT)`
- Must enable light sources individually
  - `glEnable(GL_LIGHTi)` i=0,1…..
- Choose lighting parameters

Defining a Light Source

- For each light source, we can set an RGB for the diffuse, specular, and ambient parts, and the position

```
   GLfloat diffuse0[]={1.0, 0.0, 0.0, 1.0};
   GLfloat ambient0[]={1.0, 0.0, 0.0, 1.0};
  GLfloat specular0[]={1.0, 0.0, 0.0, 1.0};
 GLfloat light0_pos[]={1.0, 2.0, 3,0, 1.0};

             glEnable(GL_LIGHTING);
              glEnable(GL_LIGHT0);
glLightv(GL_LIGHT0, GL_POSITION, light0_pos);
 glLightv(GL_LIGHT0, GL_AMBIENT, ambient0);
 glLightv(GL_LIGHT0, GL_DIFFUSE, diffuse0);
glLightv(GL_LIGHT0, GL_SPECULAR, specular0);
```

Material Properties

- Material properties are also part of the OpenGL state and match the terms in the Phong model
- Set by **`glMaterialv()`**

```
  GLfloat ambient[] = {0.2, 0.2, 0.2, 1.0};
  GLfloat diffuse[] = {1.0, 0.8, 0.0, 1.0};
 GLfloat specular[] = {1.0, 1.0, 1.0, 1.0};

 glMaterialf(GL_FRONT, GL_AMBIENT, ambient);
 glMaterialf(GL_FRONT, GL_DIFFUSE, diffuse);
glMaterialf(GL_FRONT, GL_SPECULAR, specular);
```