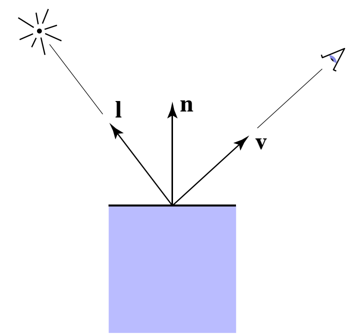
Shading Basics

CS 4620

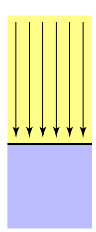
Shading

- Compute light reflected toward camera
- Inputs:
  - eye direction
  - light direction (for each of many lights)
  - surface normal
  - surface parameters (color, shininess, …)

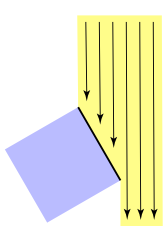


Diffuse reflection

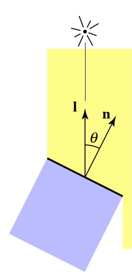
- Light is scattered uniformly in all directions
  - the surface color is the same for all viewing directions
- Lambert's cosine law



Top face of cube receives a certain amount of light



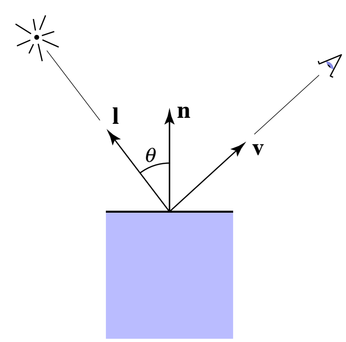
Top face of 60° rotated cube intercepts half the light



In general, light per unit area is proportional to cos θ = **l • n**

Lambertian shading

- Shading independent of view direction



illumination from source

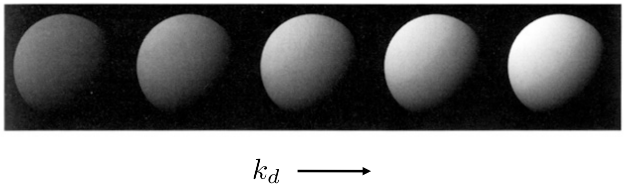
$$L_d = k_d\, I \max(0, \mathbf{n} \cdot \mathbf{l})$$

diffuse coefficient

diffusely reflected light

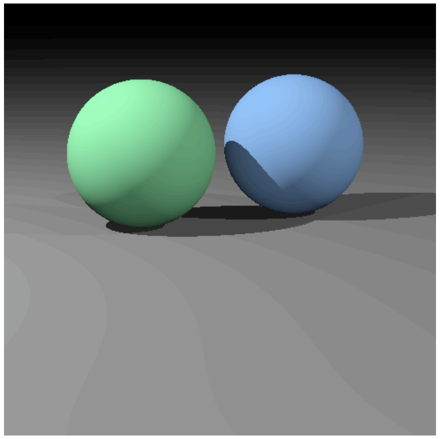
Lambertian shading

- Produces matte appearance



[Foley et al.]

Diffuse shading



Shadows

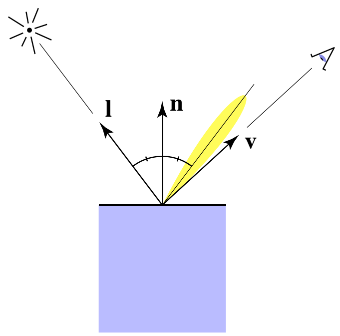
- Surface is only illuminated if nothing blocks its view of the light.
- With ray tracing it's easy to check
  - Just intersect a ray with the scene!
- Not so easy for rasterization pipeline
  - Since each triangle is processed separately
  - "Shadow Maps" [Williams] are raster-based alternative

Multiple lights

- Important to fill in black shadows
- Just loop over lights, add contributions
- Ambient shading
  - black shadows are not really right
  - one solution: dim light at camera
  - alternative: add a constant "ambient" color to the shading…

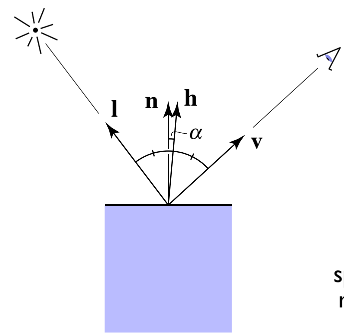
Specular shading (Blinn-Phong)

- Intensity depends on view direction
  - bright near mirror configuration



Specular shading (Blinn-Phong)

- Close to mirror ⇔ half vector near normal
  - Measure "near" by dot product of unit vectors



$$\mathbf{h} = \text{bisector}(\mathbf{v}, \mathbf{l}) = \frac{\mathbf{v} + \mathbf{l}}{\|\mathbf{v} + \mathbf{l}\|}$$

$$L_s = k_s\, I \max(0, \cos\alpha)^p = k_s\, I \max(0, \mathbf{n} \cdot \mathbf{h})^p$$

specularly reflected light (L_s)

specular coefficient (k_s)

Phong model—plots

- Increasing *n* narrows the lobe

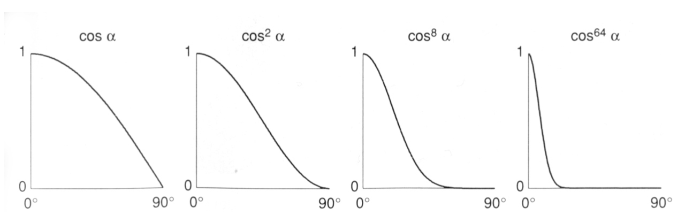
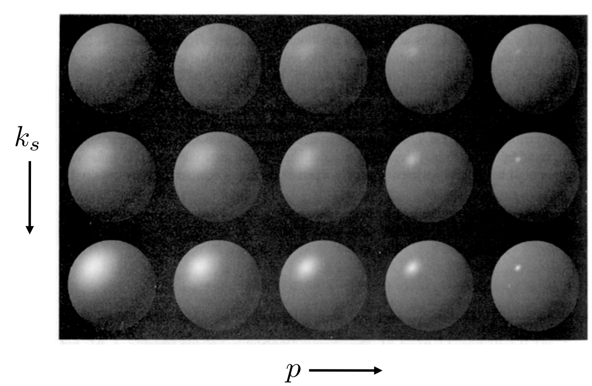


Fig. 16.9 Different values of $\cos^n \alpha$ used in the Phong illumination model.

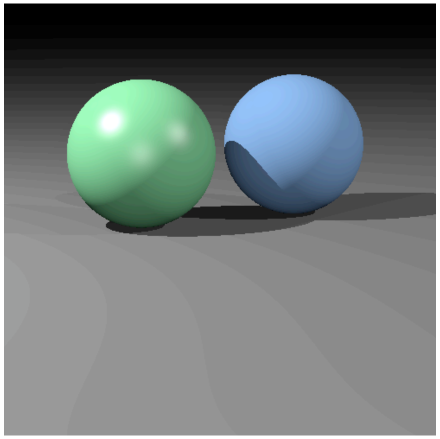
[Foley et al.]

Specular shading



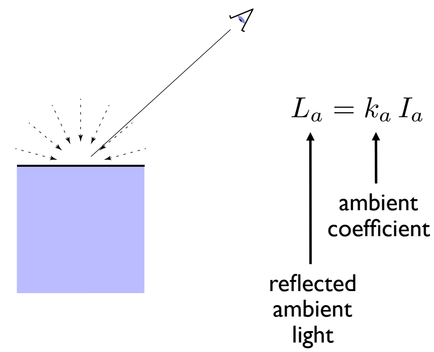
[Foley et al.]

Diffuse + Phong shading



Ambient shading

- Shading that does not depend on anything
  - add constant color to account for disregarded illumination and fill in black shadows



Putting it together

- Usually include ambient, diffuse, Phong in one model

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

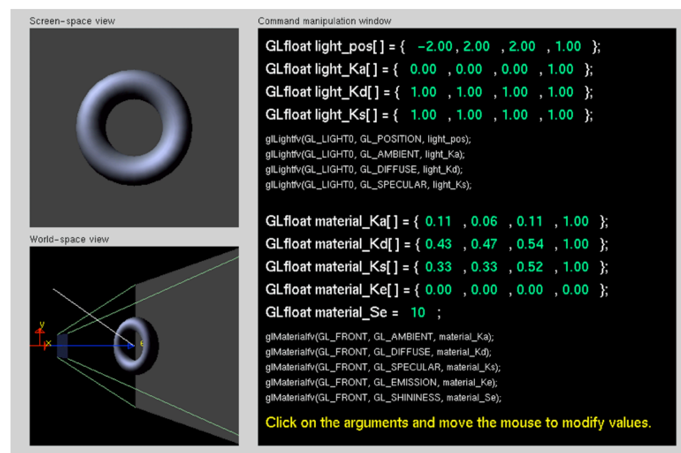
- The final result is the sum over many lights

$$
L = L_a + \sum_{i=1}^{N} [(L_d)_i + (L_s)_i]
$$

$$
L = k_a\, I_a + \sum_{i=1}^{N} [k_d\, I_i \max(0, \mathbf{n} \cdot \mathbf{l}_i) + k_s\, I_i \max(0, \mathbf{n} \cdot \mathbf{h}_i)^p]
$$

LightMaterial Demo

OpenGL Tutors program by Nate Robins
http://www.xmission.com/~nate/tutors.html



Mirror reflection

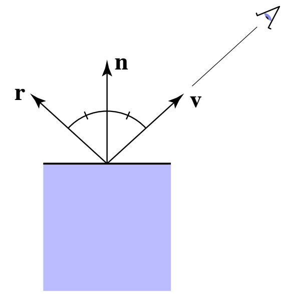
- Consider perfectly shiny surface
  - there isn't a highlight
  - instead there's a reflection of other objects
- Can render this using recursive ray tracing
  - to find out mirror reflection color, ask what color is seen from surface point in reflection direction
  - already computing reflection direction for Phong…
- "Glazed" material has mirror reflection and diffuse

$$L = L_a + L_d + L_m$$

  - where L_m is evaluated by tracing a new ray

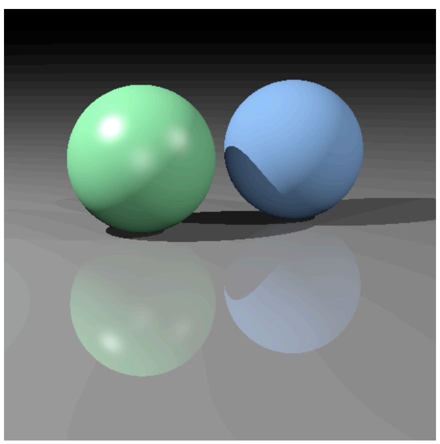
Mirror reflection

- Intensity depends on view direction
  - reflects incident light from mirror direction



$$\begin{aligned}\mathbf{r} &= \mathbf{v} + 2((\mathbf{n}\cdot\mathbf{v})\mathbf{n} - \mathbf{v}) \\ &= 2(\mathbf{n}\cdot\mathbf{v})\mathbf{n} - \mathbf{v}\end{aligned}$$

Diffuse + mirror reflection (glazed)



(glazed material on floor)