University of British Columbia
CPSC 314 Computer Graphics
Jan-Apr 2008

Tamara Munzner

Lighting/Shading III

Week 7, Fri Feb 29

http://www.ugrad.cs.ubc.ca/~cs314/Vjan2008

News

- reminder: extra TA office hours in lab 2-4
  - so no office hours for me today 2-3

Reading for Lighting/Shading

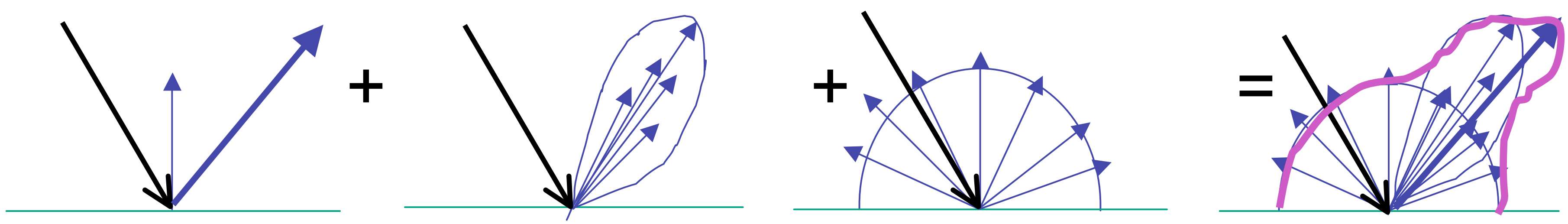
- FCG Chap 9 Surface Shading
- RB Chap Lighting

Review: Light Source Placement

- geometry: positions and directions
  - standard: world coordinate system
    - effect: lights fixed wrt world geometry
  - alternative: camera coordinate system
    - effect: lights attached to camera (car headlights)

Review: Reflectance

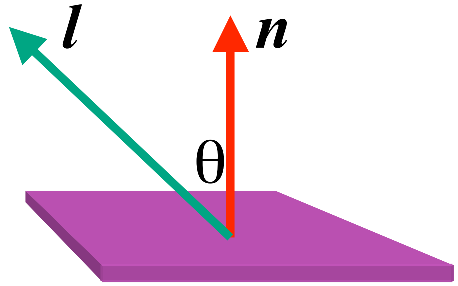
- *specular*: perfect mirror with no scattering
- *gloss*: mixed, partial specularity
- *diffuse*: all directions with equal energy



specular + glossy + diffuse =
reflectance distribution

Review: Diffuse Reflection

$$I_{diffuse} = k_d \, I_{light} \, (n \bullet l)$$

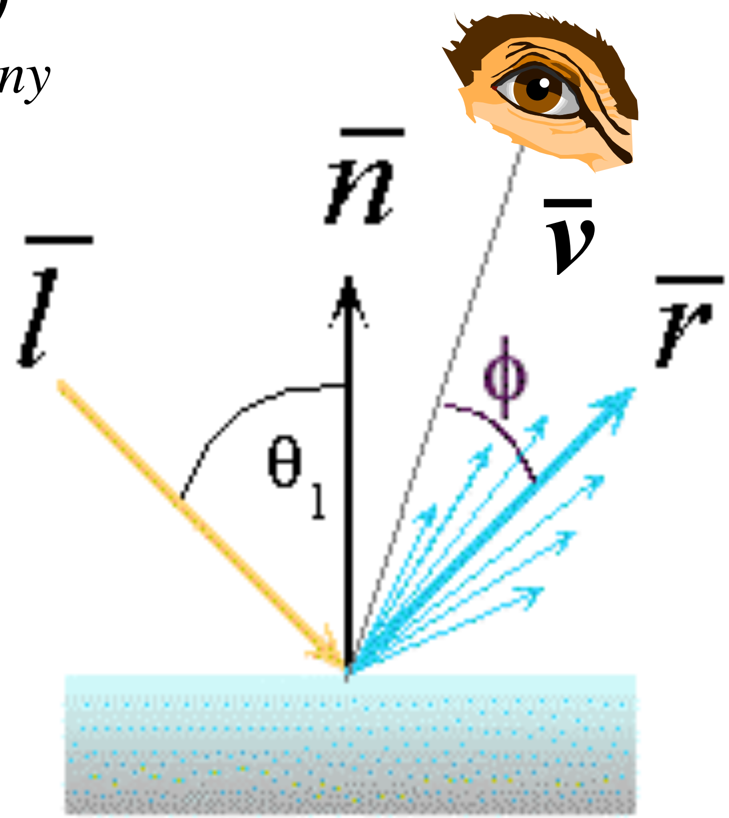


Phong Lighting

- most common lighting model in computer graphics
  - (Phong Bui-Tuong, 1975)

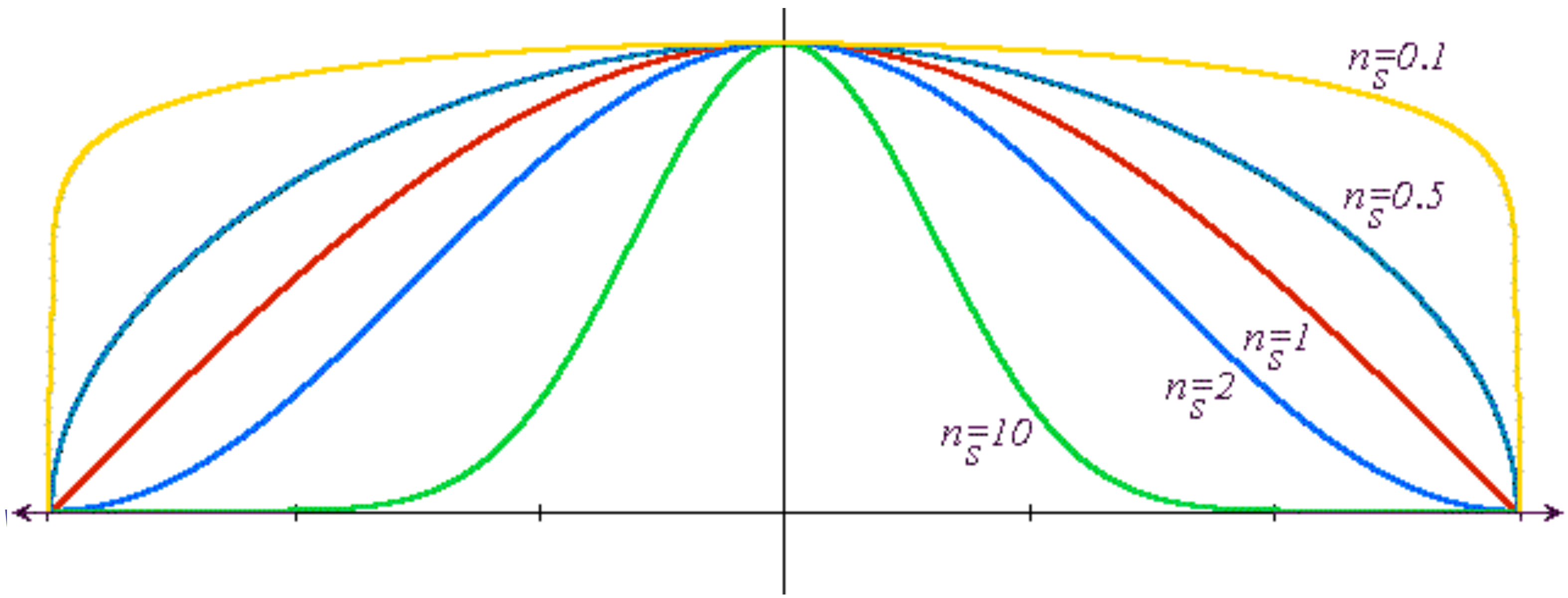
$$\mathbf{I}_{\mathbf{specular}} = \mathbf{k}_{\mathbf{s}}\mathbf{I}_{\mathbf{light}}(\cos\phi)^{n_{shiny}}$$

- n_{shiny} : purely empirical constant, varies rate of falloff
- k_s: specular coefficient, highlight color
- no physical basis, works ok in practice



Phong Lighting: The n_{shiny} Term

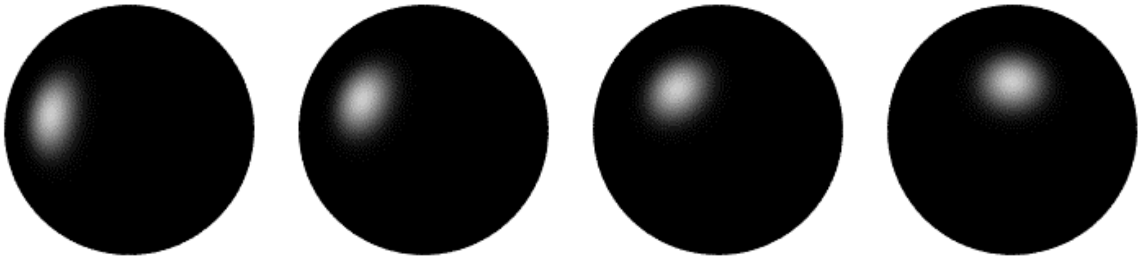
- Phong reflectance term drops off with divergence of viewing angle from ideal reflected ray
-



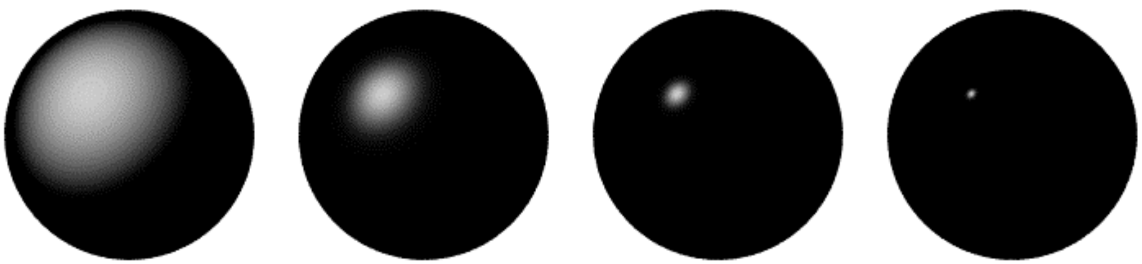
Viewing angle – reflected angle

Phong Examples

varying l



varying n_{shiny}

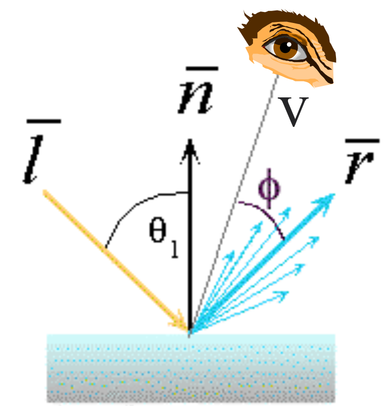


Calculating Phong Lighting

- compute **cosine** term of Phong lighting with vectors

$$\mathbf{I}_{\mathbf{specular}} = \mathbf{k}_{\mathbf{s}} \mathbf{I}_{\mathbf{light}} (\mathbf{v} \bullet \mathbf{r})^{n_{shiny}}$$

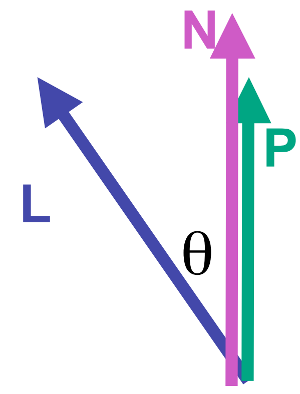
- v: unit vector towards viewer/eye
- r: ideal reflectance direction (unit vector)
- k_s: specular component
  - highlight color
- I_{light}: incoming light intensity



- how to efficiently calculate **r** ?

Calculating R Vector

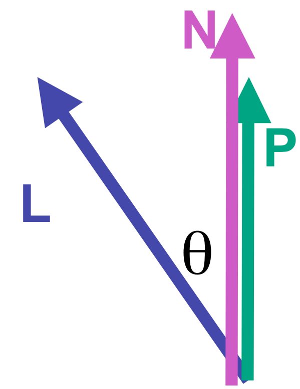
P = **N** cos θ = projection of **L** onto **N**



Calculating R Vector

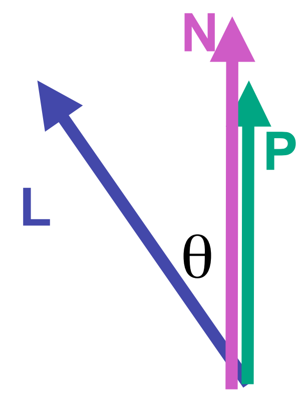
P = **N** cos θ = projection of **L** onto **N**

P = N (N · L)



Calculating R Vector

$\mathbf{P} = \mathbf{N} \cos\theta \, |\mathbf{L}| \, |\mathbf{N}|$	projection of **L** onto **N**
$\mathbf{P} = \mathbf{N} \cos\theta$	**L, N** are unit length
$\mathbf{P} = \mathbf{N} (\mathbf{N} \cdot \mathbf{L})$	



Calculating R Vector

$\mathbf{P} = \mathbf{N} \cos \theta \, |\mathbf{L}| \, |\mathbf{N}|$ projection of **L** onto **N**

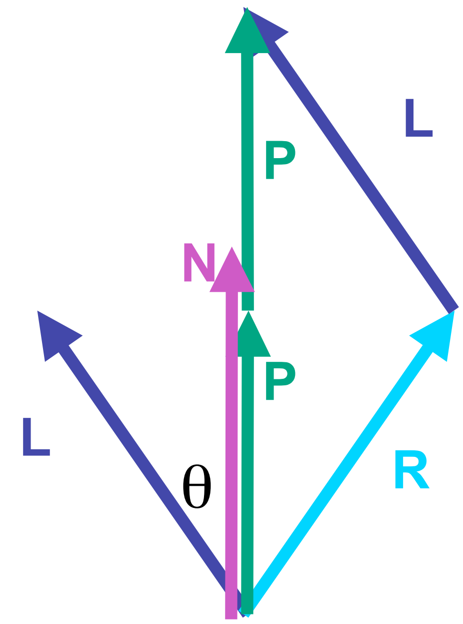
$\mathbf{P} = \mathbf{N} \cos \theta$ **L, N** are unit length

$\mathbf{P} = \mathbf{N} (\mathbf{N} \cdot \mathbf{L})$

$2 \mathbf{P} = \mathbf{R} + \mathbf{L}$

$2 \mathbf{P} - \mathbf{L} = \mathbf{R}$

$2 (\mathbf{N} (\mathbf{N} \cdot \mathbf{L})) - \mathbf{L} = \mathbf{R}$



Phong Lighting Model

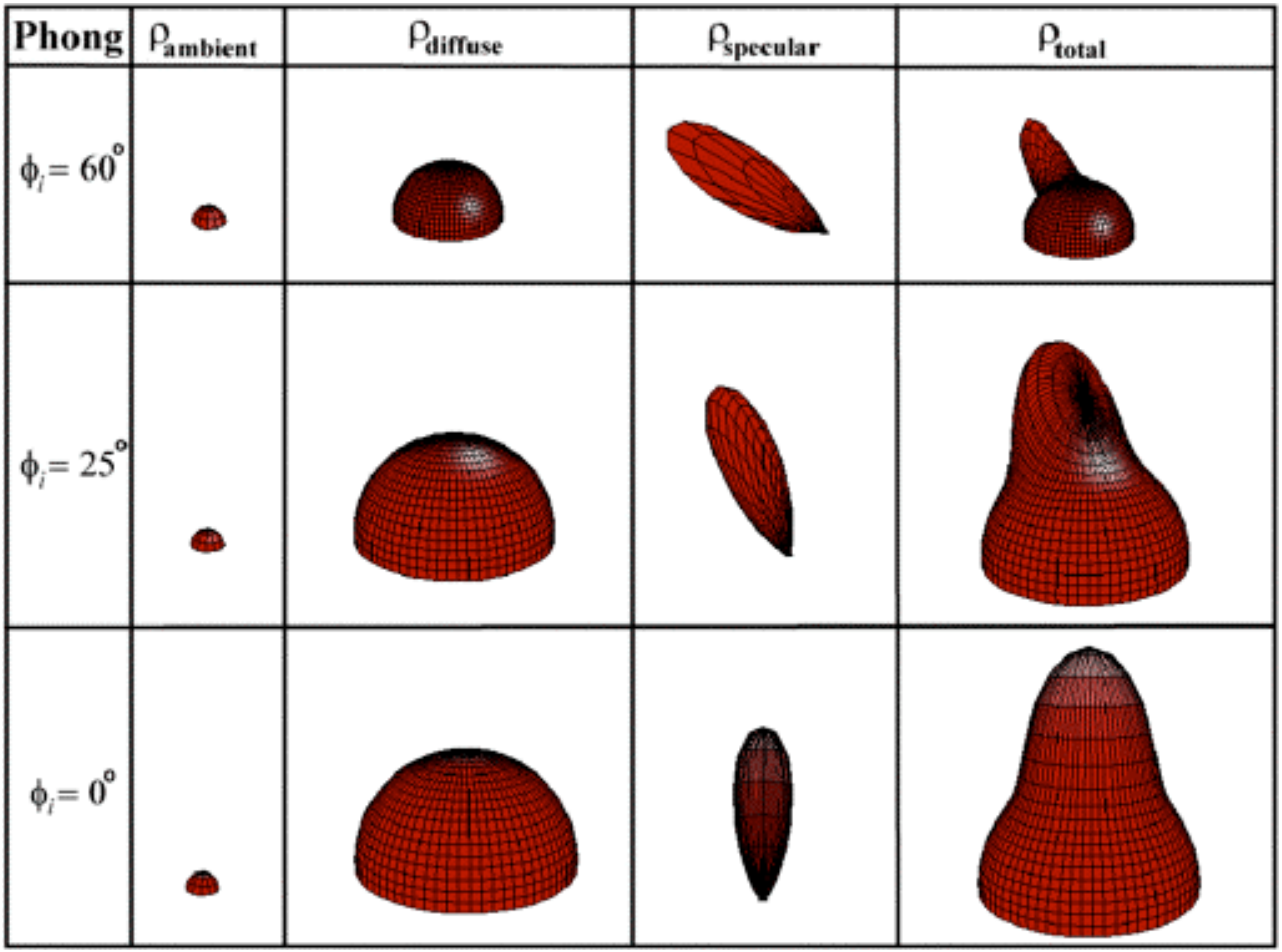
- combine ambient, diffuse, specular components

$$\mathbf{I}_{\mathbf{total}} = \mathbf{k}_{\mathbf{a}}\mathbf{I}_{\mathbf{ambient}} + \sum_{i=1}^{\#lights} \mathbf{I}_{\mathbf{i}}\left(\mathbf{k}_{\mathbf{d}}(\mathbf{n} \bullet \mathbf{l}_{\mathbf{i}}) + \mathbf{k}_{\mathbf{s}}(\mathbf{v} \bullet \mathbf{r}_{\mathbf{i}})^{n_{shiny}}\right)$$

- commonly called *Phong lighting*
  - once per light
  - once per color component

- reminder: normalize your vectors when calculating!

Phong Lighting: Intensity Plots

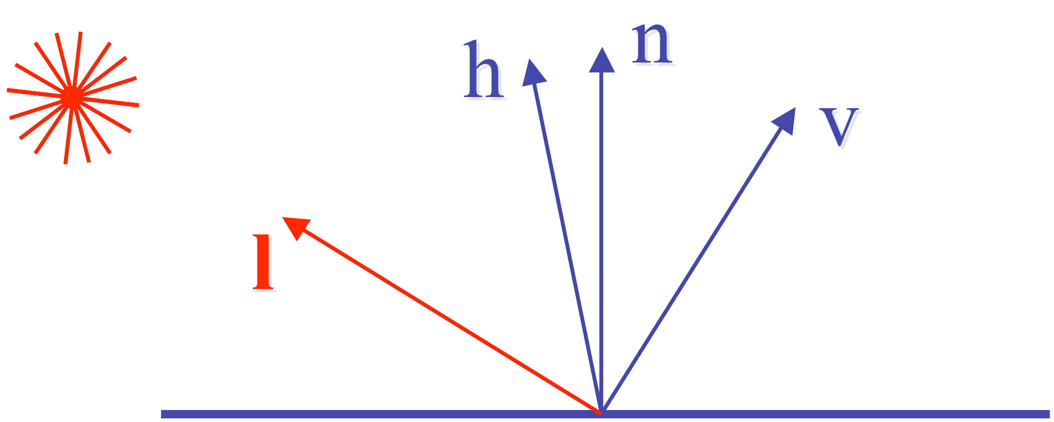


Blinn-Phong Model

- variation with better physical interpretation
  - Jim Blinn, 1977

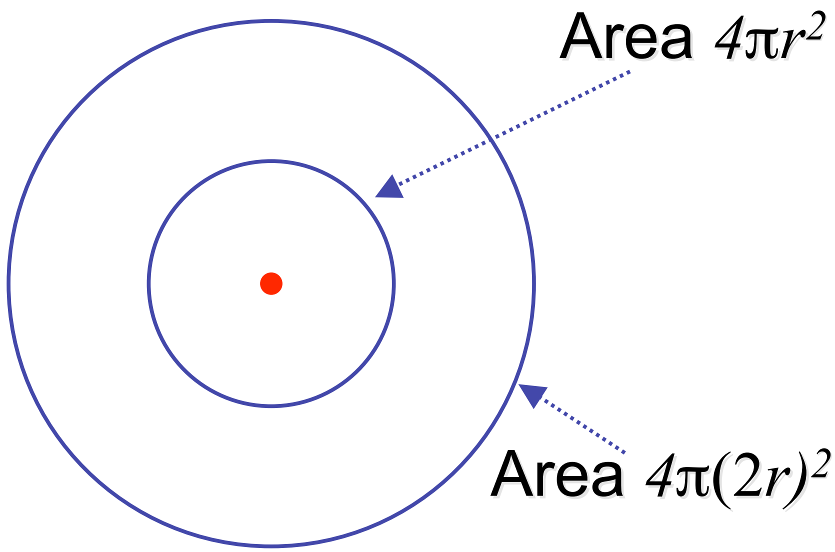
$$I_{out}(\mathbf{x}) = \mathbf{k_s}(\mathbf{h} \bullet \mathbf{n})^{n_{shiny}} \bullet I_{in}(\mathbf{x}); \text{with } \mathbf{h} = (\mathbf{l} + \mathbf{v})/2$$

- ***h***: halfway vector
  - h must also be explicitly normalized: h / |h|
  - highlight occurs when h near n



Light Source Falloff

- quadratic falloff
  - brightness of objects depends on power per unit area that hits the object
  - the power per unit area for a point or spot light decreases quadratically with distance



Light Source Falloff

- non-quadratic falloff
  - many systems allow for other falloffs
  - allows for faking effect of area light sources
  - OpenGL / graphics hardware
    - I_o: intensity of light source
    - $\boldsymbol{x}$: object point
    - r: distance of light from $\boldsymbol{x}$

$$\mathbf{I_{in}}(\mathbf{x}) = \frac{1}{ar^2 + br + c} \cdot \mathbf{I_0}$$

Lighting Review

- lighting models
  - ambient
    - normals don’t matter
  - Lambert/diffuse
    - angle between surface normal and light
  - Phong/specular
    - surface normal, light, and viewpoint

Lighting in OpenGL

- light source: amount of RGB light emitted
  - value represents percentage of full intensity e.g., (1.0,0.5,0.5)
  - every light source emits ambient, diffuse, and specular light
- materials: amount of RGB light reflected
  - value represents percentage reflected e.g., (0.0,1.0,0.5)
- interaction: component-wise multiply
  - red light (1,0,0) x green surface (0,1,0) = black (0,0,0)

Lighting in OpenGL

```
glLightfv(GL_LIGHT0, GL_AMBIENT, amb_light_rgba );
glLightfv(GL_LIGHT0, GL_DIFFUSE, dif_light_rgba );
glLightfv(GL_LIGHT0, GL_SPECULAR, spec_light_rgba );
glLightfv(GL_LIGHT0, GL_POSITION, position);
glEnable(GL_LIGHT0);

glMaterialfv( GL_FRONT, GL_AMBIENT, ambient_rgba );
glMaterialfv( GL_FRONT, GL_DIFFUSE, diffuse_rgba );
glMaterialfv( GL_FRONT, GL_SPECULAR, specular_rgba );
glMaterialfv( GL_FRONT, GL_SHININESS, n );
```

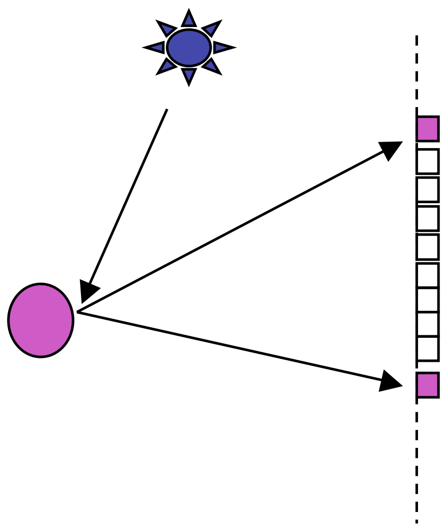
- warning: glMaterial is expensive and tricky
  - use cheap and simple glColor when possible
  - see OpenGL Pitfall #14 from Kilgard's list

  http://www.opengl.org/resources/features/KilgardTechniques/oglpitfall/

Shading

Lighting vs. Shading

- **lighting**
  - process of computing the luminous intensity (i.e., outgoing light) at a particular 3-D point, usually on a surface
- **shading**
  - the process of assigning colors to pixels
  - (why the distinction?)



Applying Illumination

- we now have an illumination model for a point on a surface
- if surface defined as mesh of polygonal facets, *which points should we use?*
  - fairly expensive calculation
  - several possible answers, each with different implications for visual quality of result

Applying Illumination

- polygonal/triangular models
  - each facet has a constant surface normal
  - if light is directional, diffuse reflectance is constant across the facet
  - why?

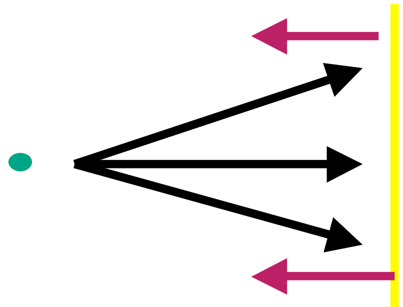
Flat Shading

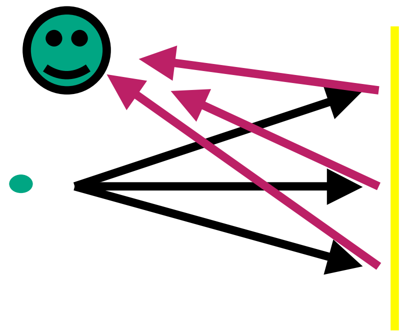
- simplest approach calculates illumination at a single point for each polygon

- obviously inaccurate for smooth surfaces

Flat Shading Approximations

- if an object really <u>is</u> faceted, is this accurate?
- no!
  - for point sources, the direction to light varies across the facet
  - for specular reflectance, direction to eye varies across the facet

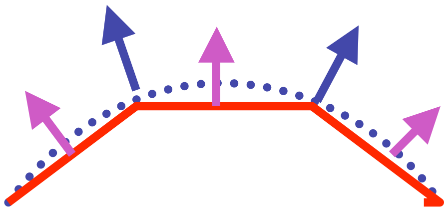




Improving Flat Shading

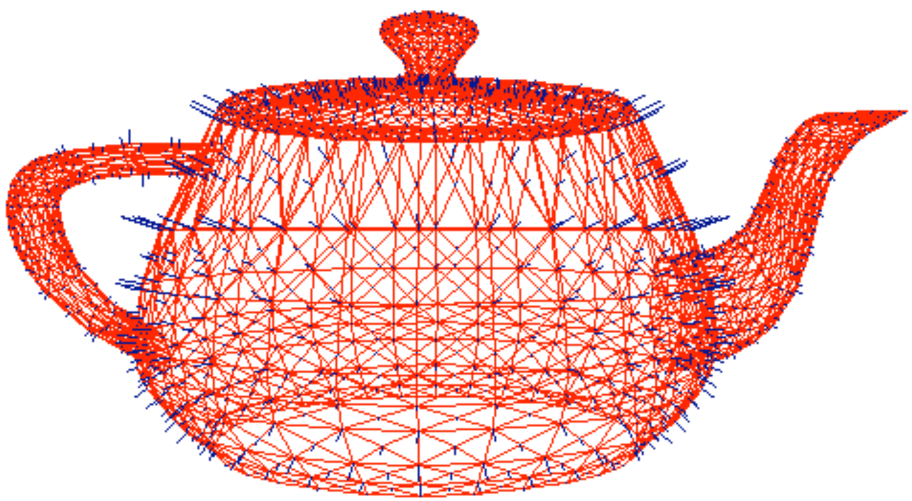
- what if evaluate Phong lighting model at each pixel of the polygon?
  - better, but result still clearly faceted

- for smoother-looking surfaces we introduce *vertex normals* at each vertex
  - usually different from facet normal
  - used *only* for shading
  - think of as a better approximation of the *real* surface that the polygons approximate



Vertex Normals

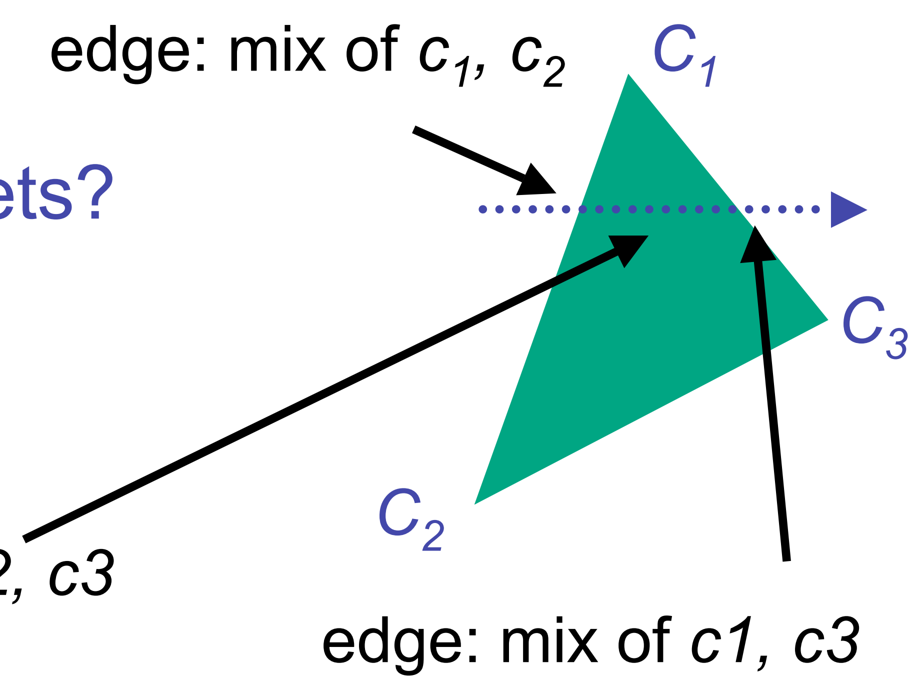
- vertex normals may be
  - provided with the model
  - computed from first principles
  - approximated by averaging the normals of the facets that share the vertex



Gouraud Shading

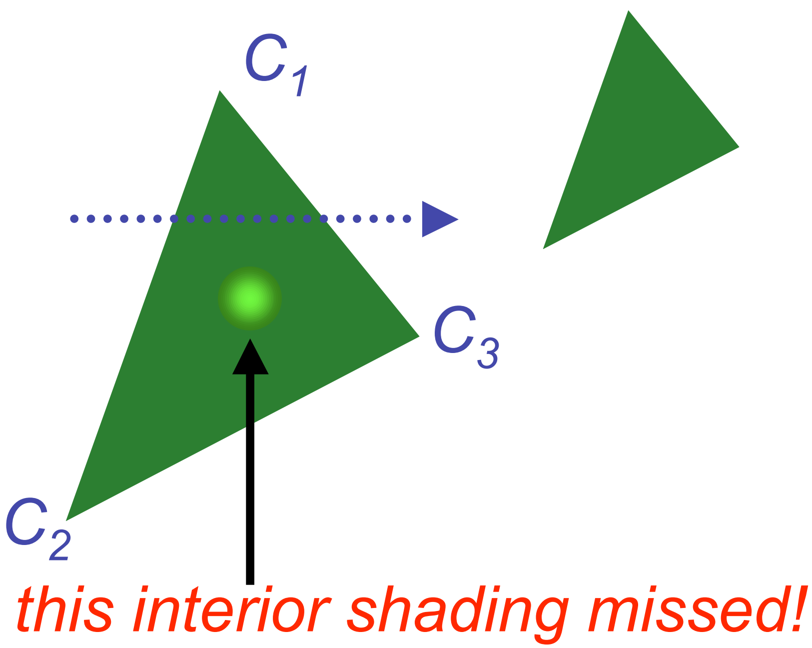
- most common approach, and what OpenGL does
  - perform Phong lighting at the vertices
  - linearly interpolate the resulting colors over faces
    - along edges
    - along scanlines

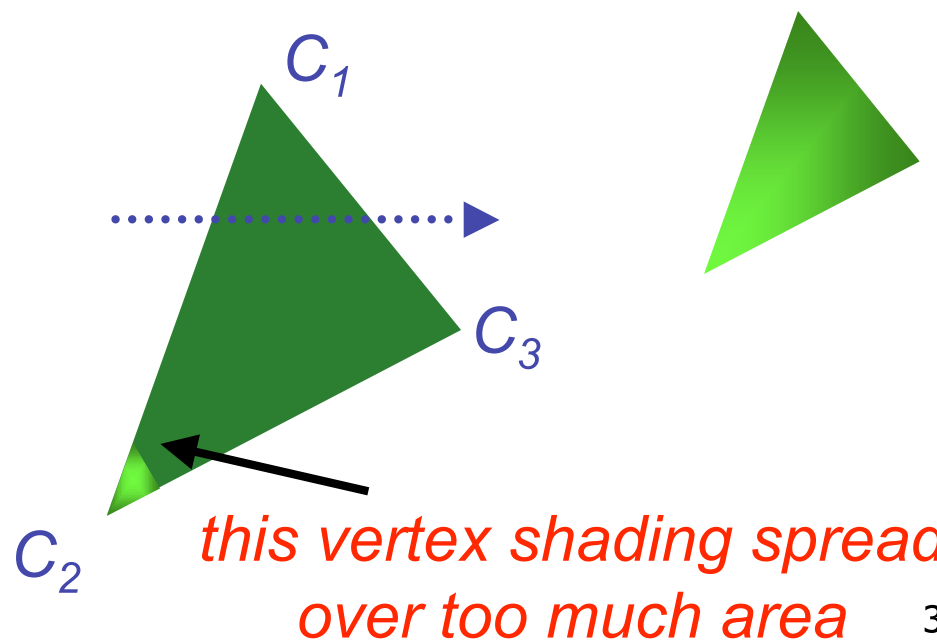
does this eliminate the facets?



Gouraud Shading Artifacts

- often appears dull, chalky
- lacks accurate specular component
  - if included, will be averaged over entire polygon



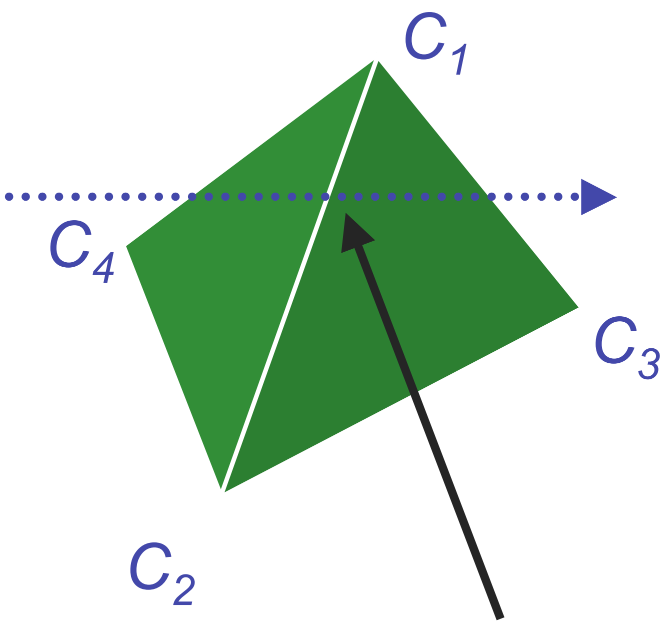


Gouraud Shading Artifacts

- Mach bands
  - eye enhances discontinuity in first derivative
  - very disturbing, especially for highlights

Gouraud Shading Artifacts

- Mach bands



Discontinuity in rate of color change occurs here