Lighting

- So…given a 3-D triangle and a 3-D viewpoint, we can set the right pixels
- But what color should those pixels be?
- If we're attempting to create a realistic image, we need to simulate the *lighting* of the surfaces in the scene
  - Fundamentally simulation of *physics* and *optics*
  - As you'll see, we use a lot of approximations to do this simulation fast enough

Shading Model

The *shading model* calculates the brightness and color to display for a point on a visible surface.

The model is *approximate* – a compromise between accuracy and cost of computing.

Definitions

- **Illumination**: the transport of energy (in particular, the luminous flux of visible light) from light sources to surfaces & points
  - Note: includes *direct* and *indirect* illumination
- **Lighting**: the 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

Definitions

- Illumination models fall into two categories:
  - **Empirical**: simple formulations that approximate observed phenomenon
  - **Physically-based**: models based on the actual physics of light interacting with matter
- We mostly use empirical models in interactive graphics for simplicity
- Increasingly, realistic graphics are using physically-based models

Components of Illumination

- Two components of illumination: light sources and surface properties
- Light sources
  - Optical attributes (i.e, color of the light)
  - Geometric attributes
    - Position
    - Direction
    - Shape
  - Directional attenuation

Components of Illumination

- Surface properties
  - Optical properties (i.e., color of the surface)
  - Geometric attributes
    - Position
    - Orientation
    - Micro-structure
- Common simplifications in interactive graphics
  - Only *direct* illumination from emitters to surfaces
  - Simplify geometry of emitters to trivial cases

Light Sources

1. Light-emitting (light bulbs, sun)
2. Light-reflecting (illuminated surfaces of objects such as walls)

Ambient Light Sources

- Objects not directly lit are typically still visible
  - E.g., the ceiling in this room, undersides of desks
- This is the result of *indirect* illumination from emitters, bouncing off intermediate surfaces
- Too expensive to calculate (in real time), so we use a model called an *ambient* light source
  - No spatial or directional characteristics; illuminates all surfaces equally
  - Amount reflected depends on surface properties

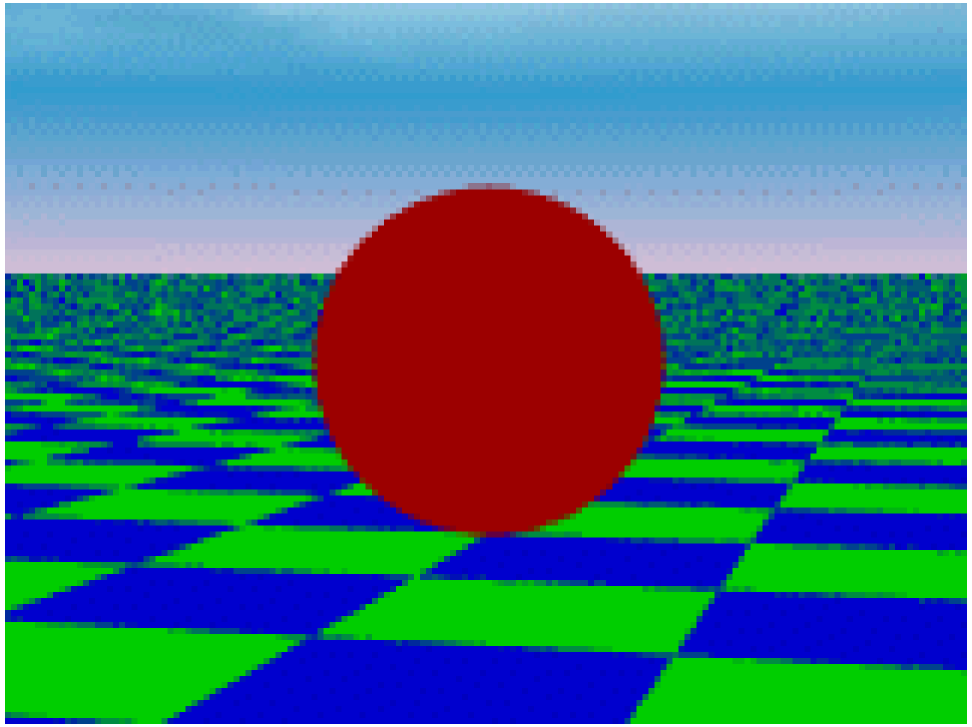
Ambient Light Sources

- For each sampled wavelength, the ambient light reflected from a surface depends on
  - The surface properties, $k_{ambient}$
  - The intensity of the ambient light source (constant for all points on all surfaces)

$$I_{reflected} = k_{ambient} I_{ambient}$$

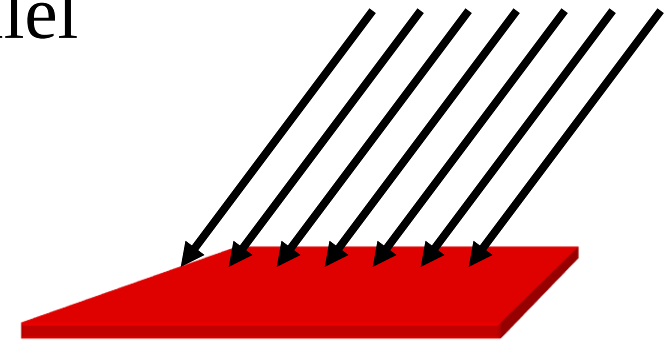
Ambient Light Sources

- A scene lit only with an ambient light source:



Directional Light Sources

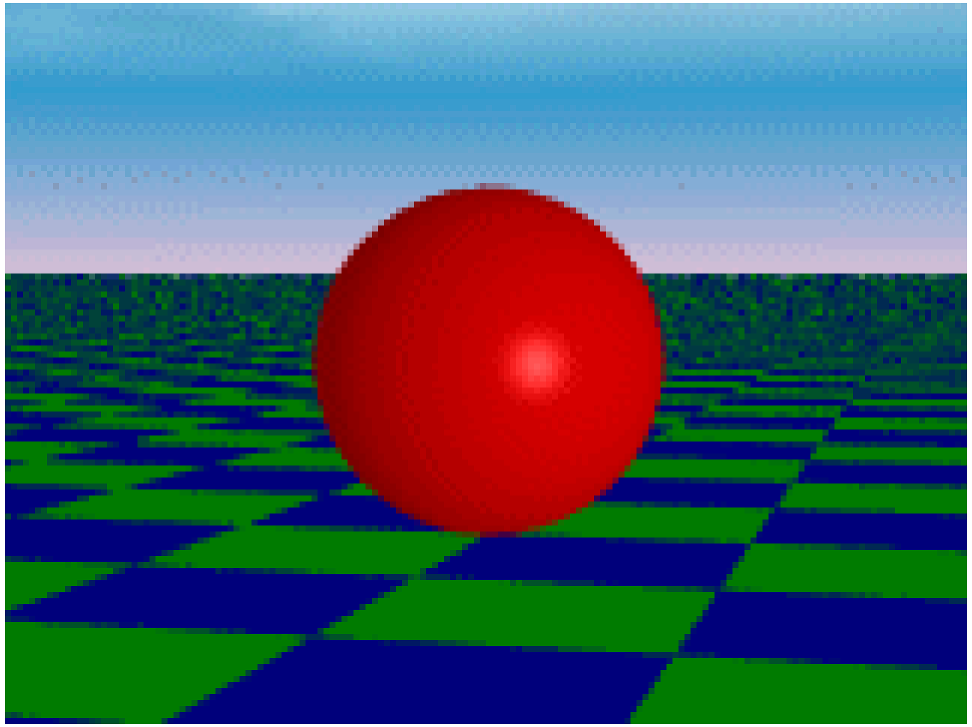
- For a *directional* light source we make the simplifying assumption that all rays of light from the source are parallel
  - As if the source is infinitely far away from the surfaces in the scene
  - A good approximation to sunlight



- The direction from a surface to the light source is important in lighting the surface
- With a directional light source, this direction is constant for all surfaces in the scene

Directional Light Sources

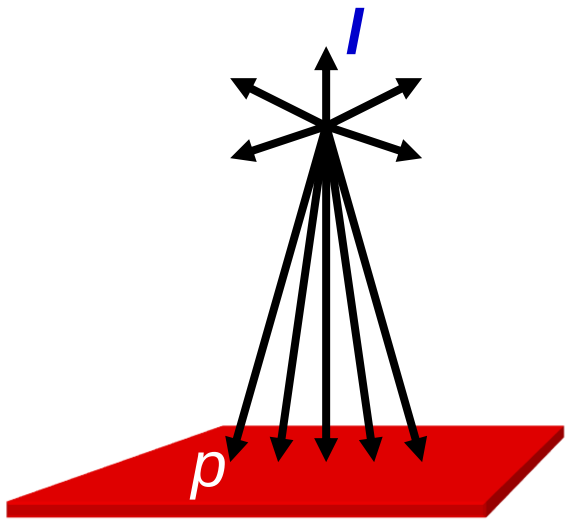
- The same scene lit with a directional and an ambient light source



Point Light Sources

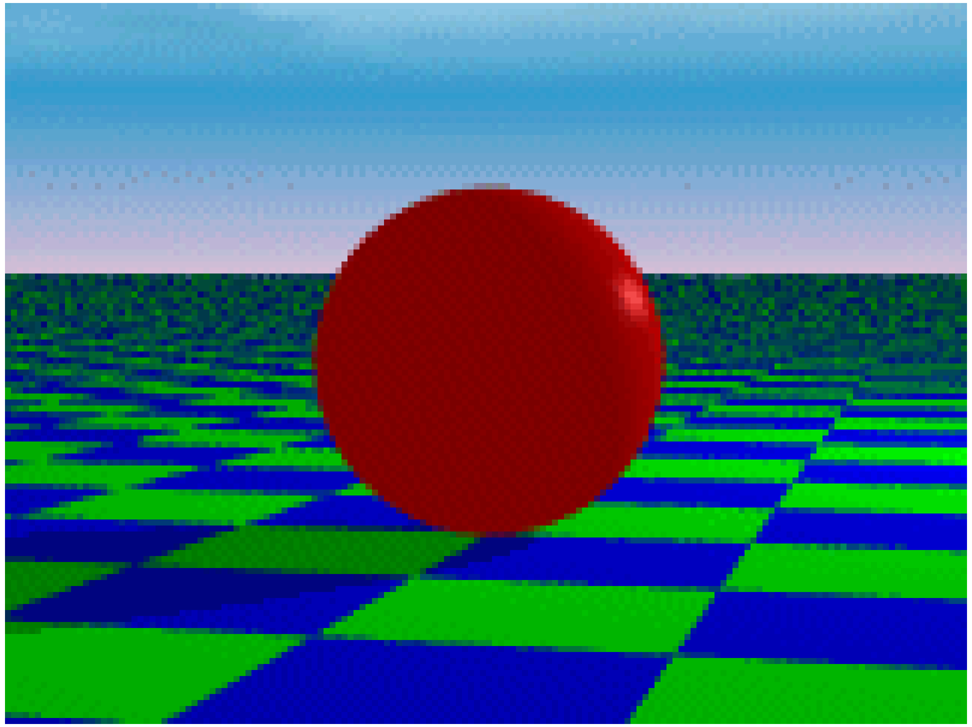
- A *point light source* emits light equally in all directions from a single point
- The direction to the light from a point on a surface thus differs for different points:
  - So we need to calculate a normalized vector to the light source for every point we light:

$$\vec{d} = \frac{\vec{p} - \vec{l}}{\left\| \vec{p} - \vec{l} \right\|}$$



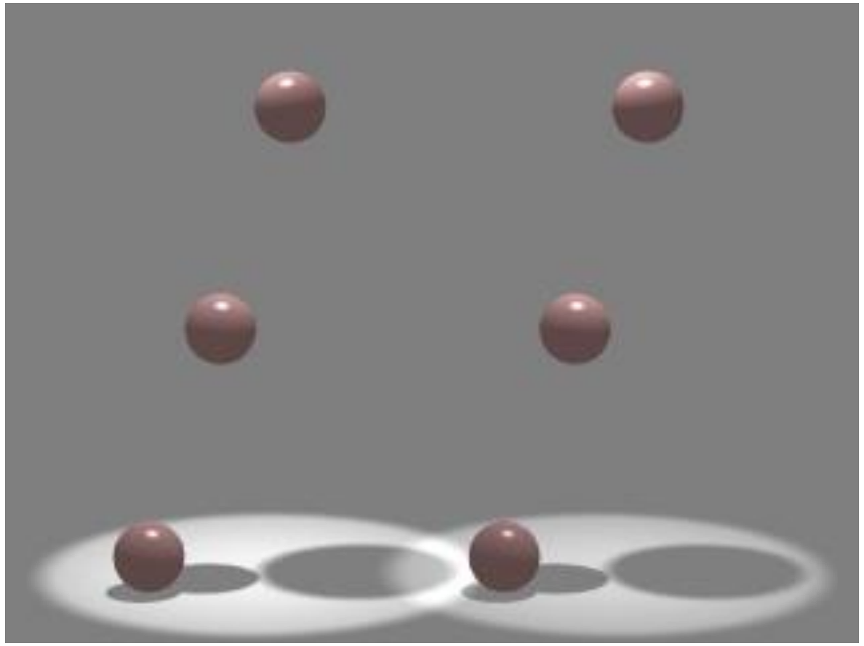
Point Light Sources

- Using an ambient and a point light source:



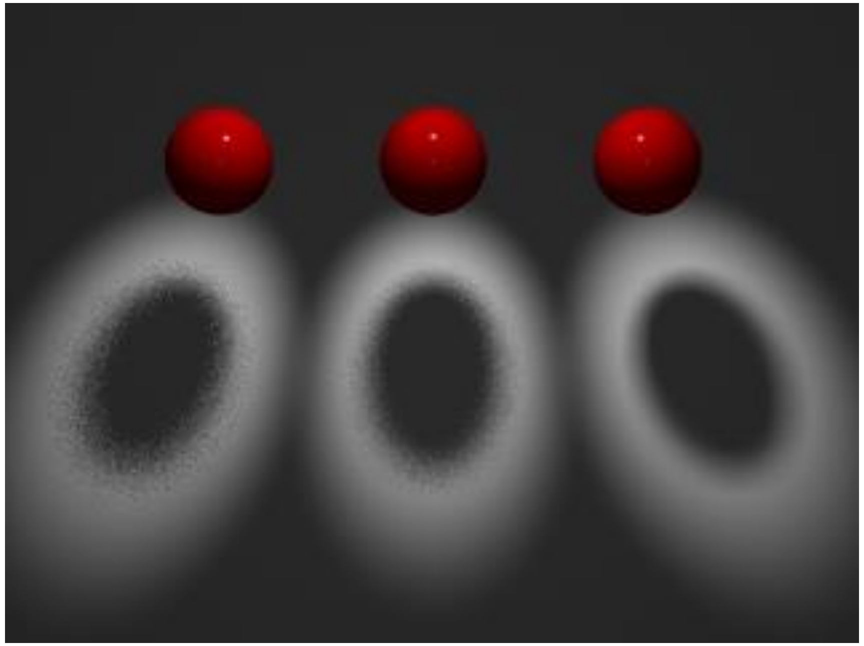
Other Light Sources

- *Spotlights* are point sources whose intensity falls off directionally.
  - Requires color, point direction, falloff parameters
  - Supported by OpenGL



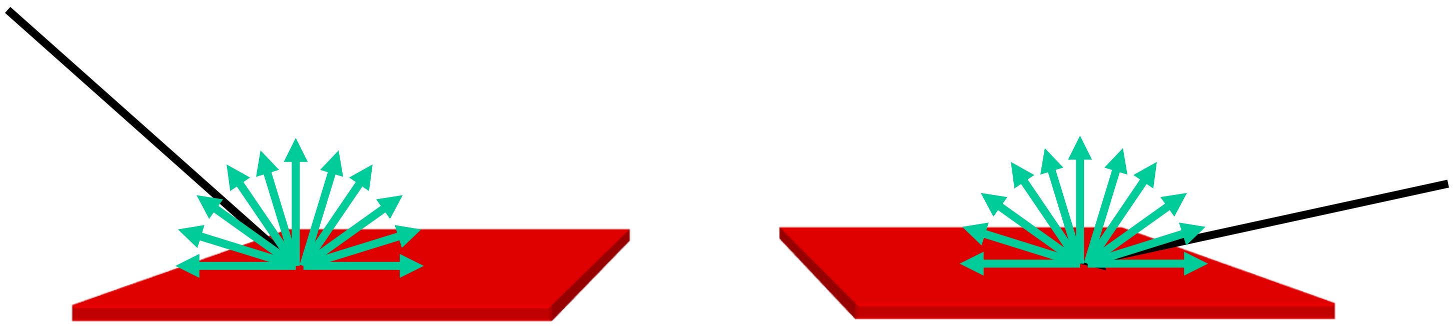
Other Light Sources

- *Area light sources* define a 2-D emissive surface (usually a disc or polygon)
  - Good example: fluorescent light panels
  - Capable of generating *soft shadows* (*why?*)



The Physics of Reflection

- Ideal diffuse reflection
  - An *ideal diffuse reflector*, at the microscopic level, is a very rough surface (real-world example: chalk)
  - Because of these microscopic variations, an incoming ray of light is equally likely to be reflected in any direction over the hemisphere:



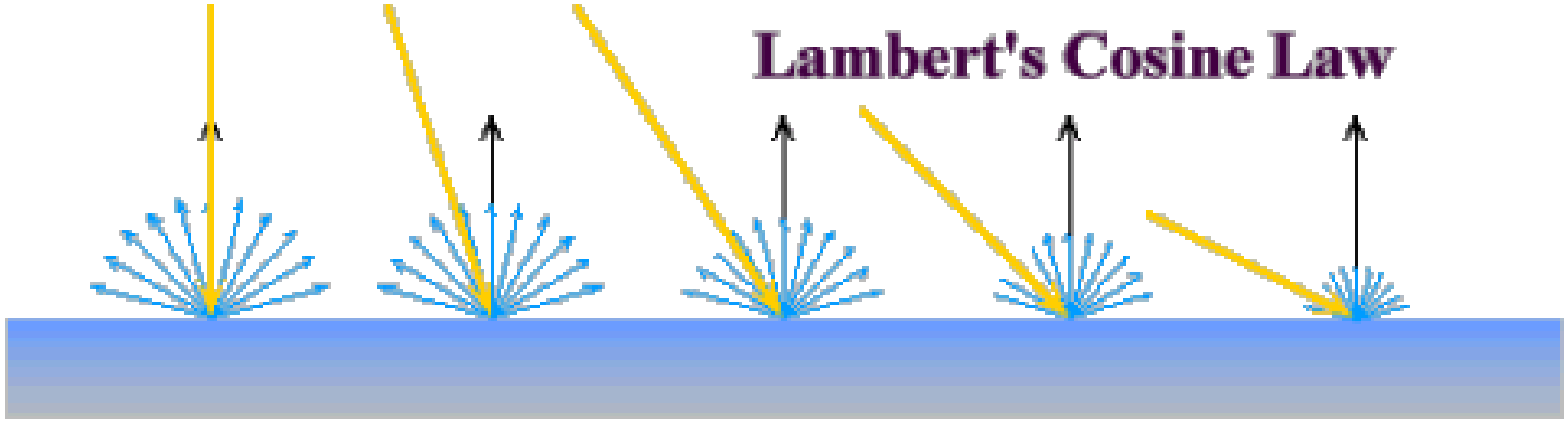
Lambert’s Cosine Law

- Ideal diffuse surfaces reflect according to *Lambert’s cosine law:*

  The energy reflected by a small portion of a surface from a light source in a given direction is proportional to the cosine of the angle between that direction and the surface normal

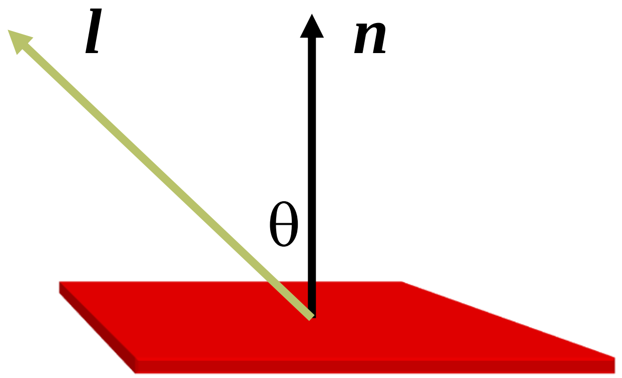
- These are often called *Lambertian surfaces*
- Note that the reflected intensity is independent of the viewing direction, but does depend on the surface orientation with regard to the light source

Lambert’s Law



Computing Diffuse Reflection

- The angle between the surface normal and the incoming light is the *angle of incidence:*



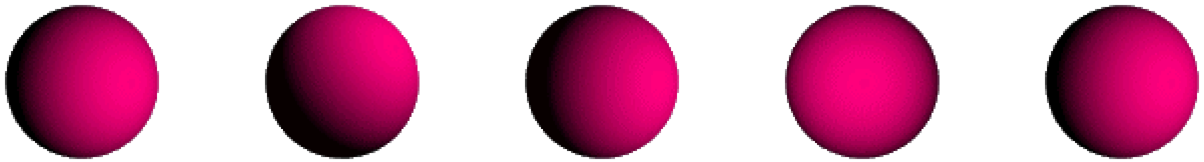
$$I_{diffuse} = k_d \, I_{light} \cos\theta$$

- In practice we use vector arithmetic:

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

Diffuse Lighting Examples

- We need only consider angles from 0° to 90° (*Why?*)
- A Lambertian sphere seen at several different lighting angles:



Attenuation: Distance

- f_{att} models distance from light
  - $I_{diffuse} = k_d f_{att} I_{light} \; (\boldsymbol{n} \bullet \boldsymbol{l})$
- Realistic
  - $f_{\mathrm{att}} = 1/(d_{\mathrm{light}})^2$
- Hard to control, so use
  - $f_{\mathrm{att}} = 1/(c_1 + c_2 d_{\mathrm{light}} + c_3 d_{\mathrm{light}}^2)$

Specular Reflection

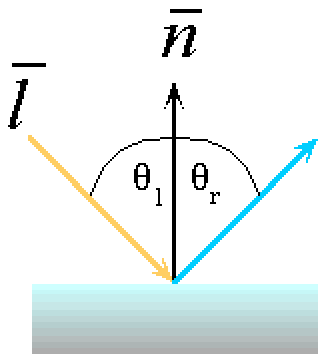
- Shiny surfaces exhibit *specular reflection*
  - Polished metal
  - Glossy car finish
- A light shining on a specular surface causes a bright spot known as a *specular highlight*
- Where these highlights appear is a function of the viewer's position, so specular reflectance is view-dependent

The Physics of Reflection

- At the microscopic level a specular reflecting surface is very smooth
- Thus rays of light are likely to bounce off the microgeometry in a mirror-like fashion
- The smoother the surface, the closer it becomes to a perfect mirror

The Optics of Reflection

- Reflection follows *Snell's Laws:*
  - The incoming ray and reflected ray lie in a plane with the surface normal
  - The angle that the reflected ray forms with the surface normal equals the angle formed by the incoming ray and the surface normal:



$$\theta_{\text{(l)ight}} = \theta_{\text{(r)eflection}}$$

Non-Ideal Specular Reflectance

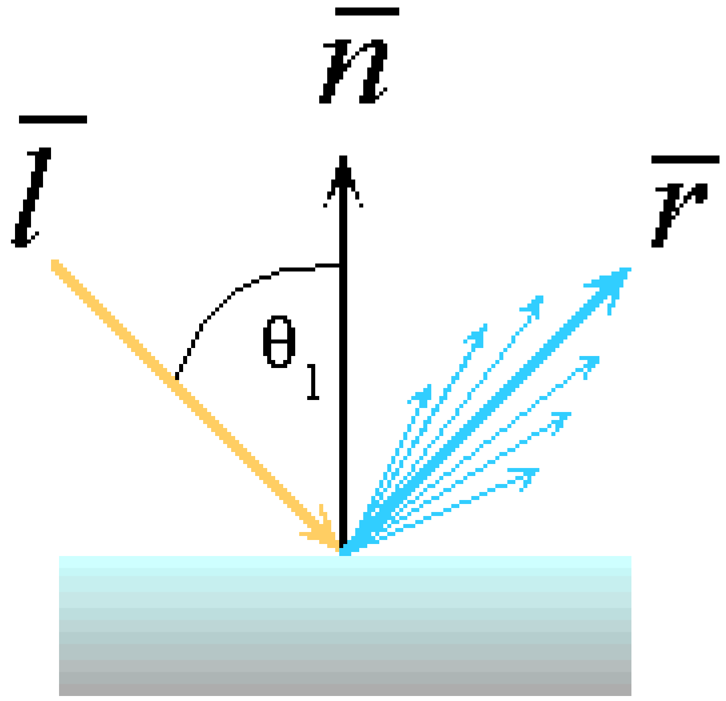
- Snell’s law applies to perfect mirror-like surfaces, but aside from mirrors (and chrome) few surfaces exhibit perfect specularity
- How can we capture the “softer” reflections of surface that are glossy rather than mirror-like?
- One option: model the microgeometry of the surface and explicitly bounce rays off of it
- Or…

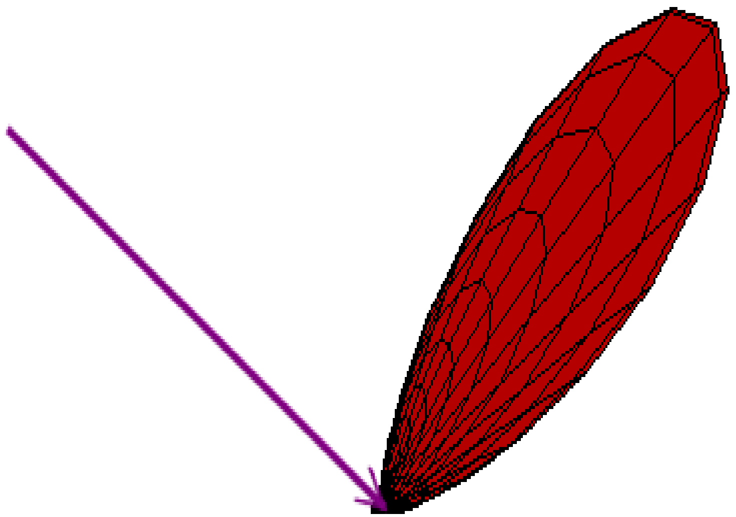
Non-Ideal Specular Reflectance: An Empirical Approximation

- In general, we expect most reflected light to travel in direction predicted by Snell's Law
- But because of microscopic surface variations, some light may be reflected in a direction slightly off the ideal reflected ray
- As the angle from the ideal reflected ray increases, we expect less light to be reflected

Non-Ideal Specular Reflectance: An Empirical Approximation

An illustration of this angular falloff:



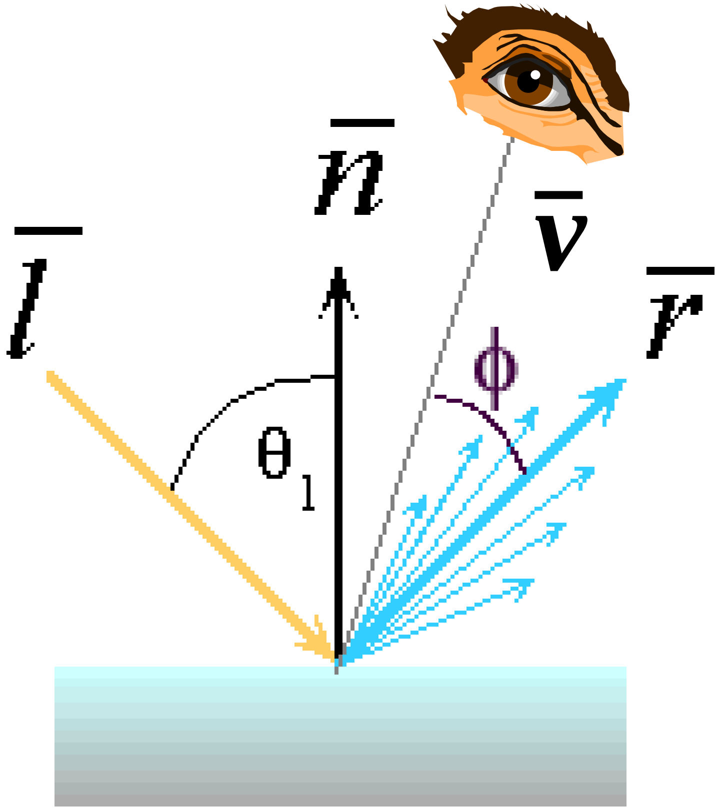


How might we model this falloff?

Phong Lighting

- The most common lighting model in computer graphics was suggested by Phong:

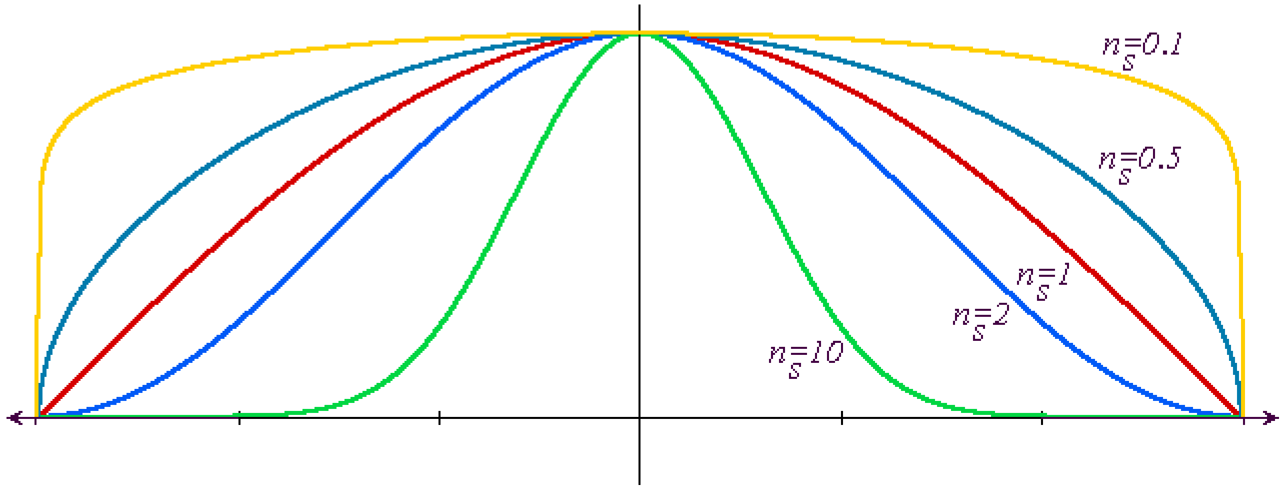
$$I_{specular} = k_s I_{light} \left(\cos\varphi\right)^{n_{shiny}}$$



- The n_{shiny} term is a purely empirical constant that varies the rate of falloff
- Though this model has no physical basis, it works (sort of) in practice

Phong Lighting: The n_{shiny} Term

- This diagram shows how the Phong reflectance term drops off with divergence of the viewing angle from the ideal reflected ray:



- *What does this term control, visually?*

Calculating Phong Lighting

- The **cos** term of Phong lighting can be computed using vector arithmetic:

$$I_{specular} = k_s I_{light} \left(\hat{V} \cdot \hat{R} \right)^{n_{shiny}}$$

  - V is the unit vector towards the viewer
  - R is the ideal reflectance direction

- An aside: we can efficiently calculate R

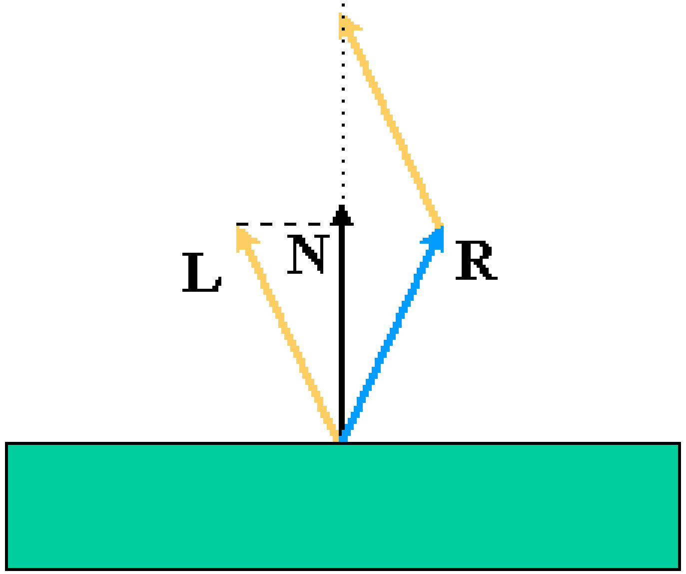
$$\hat{R} = \left(2 \left(\hat{N} \cdot \hat{L} \right) \right) \hat{N} - \hat{L}$$

Calculating The *R* Vector

$$\hat{R} = \left(2\left(\hat{N} \cdot \hat{L}\right)\right)\hat{N} - \hat{L}$$

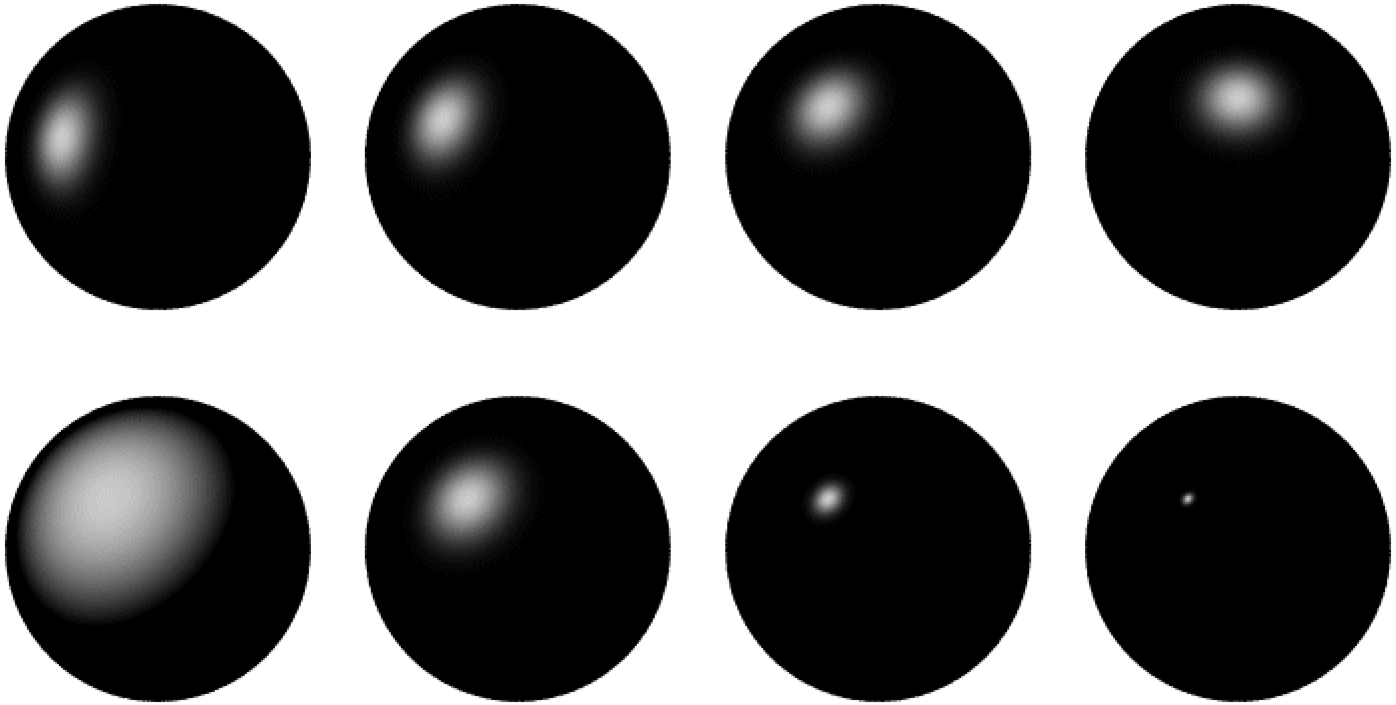
- This is illustrated below:

$$\hat{R} + \hat{L} = \left(2\left(\hat{N} \cdot \hat{L}\right)\right)\hat{N}$$



Phong Examples

- These spheres illustrate the Phong model as L and n_{shiny} are varied:



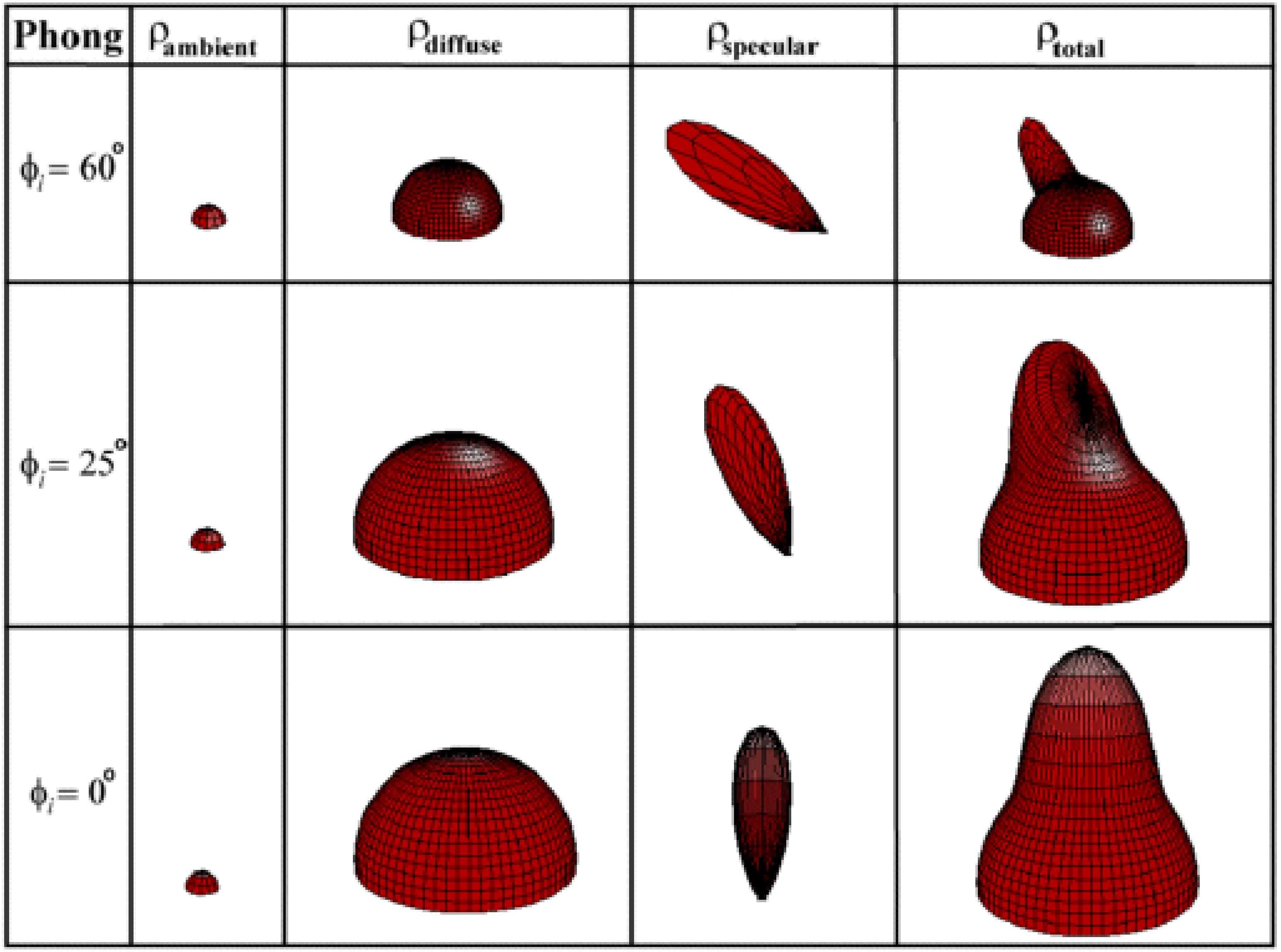
The Phong Lighting Model

- Let's combine ambient, diffuse, and specular components:

$$I_{total} = k_a I_{ambient} + \sum_{i=1}^{\#lights} I_i \left(k_d \left(\hat{N} \cdot \hat{L} \right) + k_s \left(\hat{V} \cdot \hat{R} \right)^{n_{shiny}} \right)$$

- Commonly called *Phong lighting*
  - Note: once per light
  - Note: once per color component
  - *Do k_a, k_d, and k_s vary with color component?*

Phong Lighting: Intensity Plots



Applying Illumination

- We now have an illumination model for a point on a surface
- Assuming that our surface is defined as a mesh of polygonal facets, *which points should we use?*
- Keep in mind:
  – It's a fairly expensive calculation
  – Several possible answers, each with different implications for the visual quality of the result

Applying Illumination

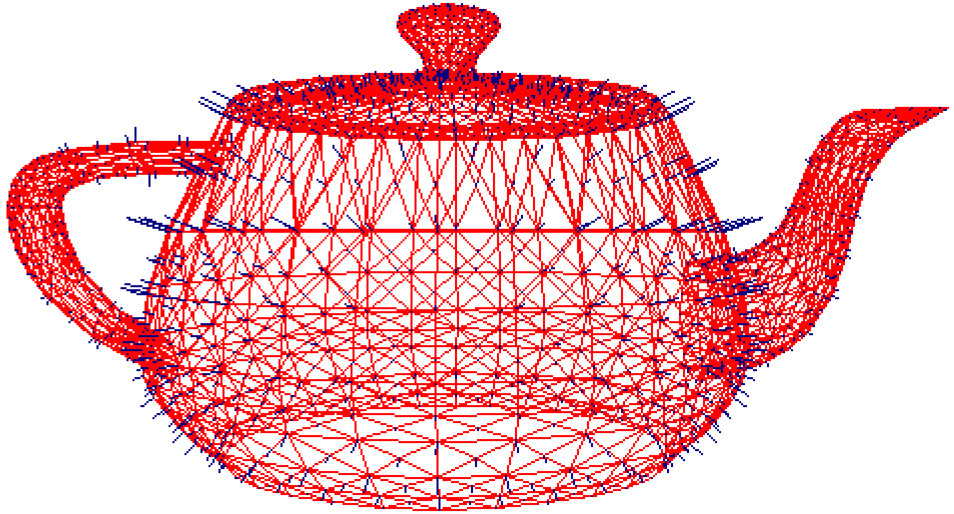
- With polygonal/triangular models:
  - Each facet has a constant surface normal
  - If the light is directional, the diffuse reflectance is constant across the facet

Flat Shading

- We can refine it a bit by evaluating the Phong lighting model at each pixel of each polygon, but the result is still clearly faceted:
- To get 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 (draw it)

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

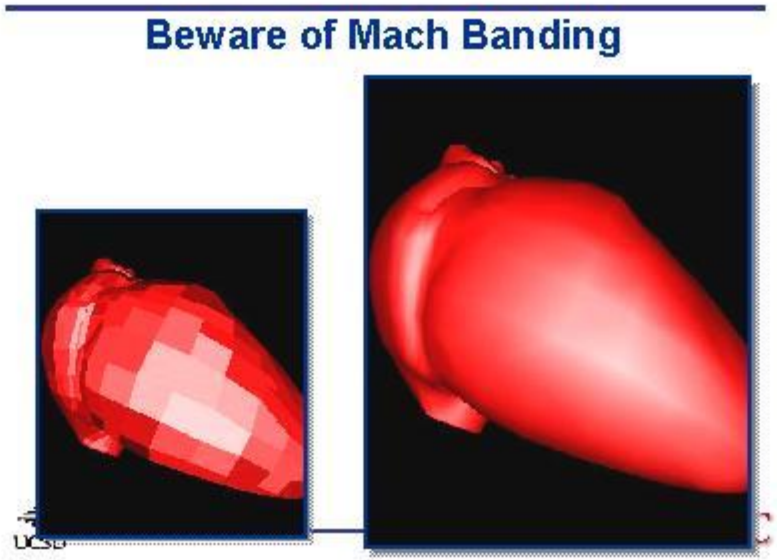


Gouraud Shading

- This is the most common approach
  - Perform Phong lighting at the vertices
  - Linearly interpolate the resulting colors over faces
    - Along edges
    - Along scanlines
  - This is what OpenGL does
- *Does this eliminate the facets?*

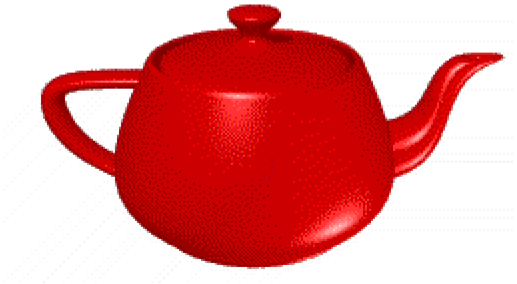
Gouraud Shading

- Artifacts
  - Often appears dull, chalky
  - Lacks accurate specular component
    - If included, will be averaged over entire polygon
  - Mach Banding (draw example)
    - Artifact at discontinuities in intensity or intensity slope



Phong Shading

- *Phong shading* is not the same as Phong lighting, though they are sometimes mixed up
  - Phong lighting: the empirical model we've been discussing to calculate illumination at a point on a surface
  - Phong shading: linearly interpolating the surface normal across the facet, applying the Phong lighting model at every pixel
    - Same input as Gouraud shading
    - Usually very smooth-looking results:
    - But, considerably more expensive



Phong Shading

- Linearly interpolate the vertex normals
  - Compute lighting equations at each pixel
  - Can use specular component

Shortcomings of Shading

- Polygonal silhouettes remain
- Perspective distortion not captured in interpolation down scanlines
- Interpolation dependent on polygon orientation
- Shared vertices
- Bad averaging to compute vertex normals