CS 432 – Interactive Computer Graphics

Lecture 6 – Part 1

Lighting and Shading Math

Topics

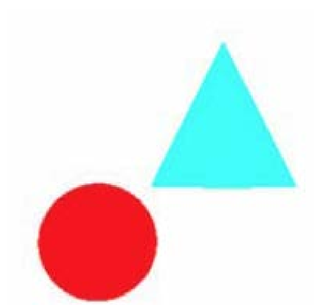
- Light and Shading

Reading

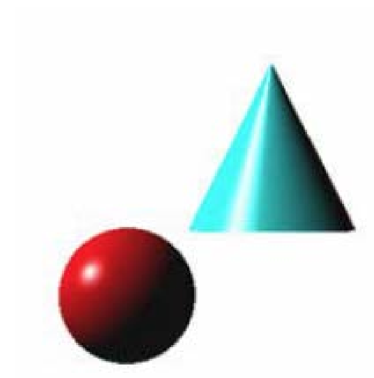
- Angel
  - Chapter 5
- Red Book
  - Chapter 7

Shading

- Shading helps to provide realism and better understanding of depth and scene composition



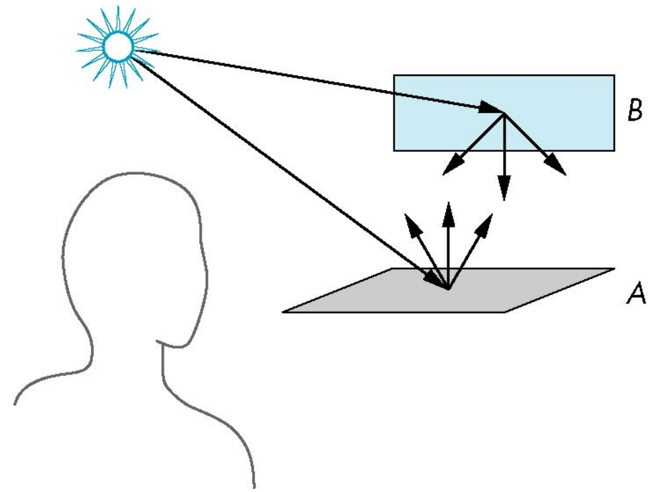
vs



Shading

- Why do things look like this in the real world?
  - Light material interactions causes each point to have a different color and shade
- Need to consider:
  - Light sources
  - Material properties
  - Location of viewer
  - Surface orientation

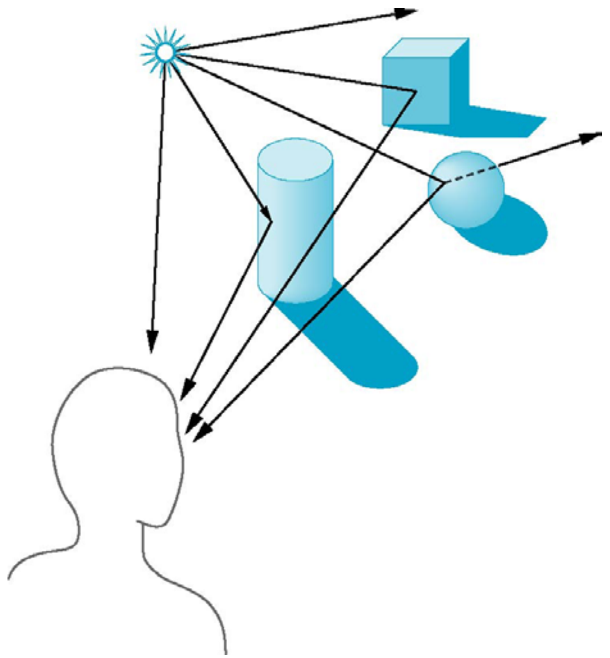
Light and Matter



- Rendering based on physics.
- Components:
  - Light sources
  - Material characteristics
- Light strikes A
  - Some scattered
  - Some absorbed
- Some of the scattered light strikes B
  - Some scattered
  - Some absorbed
- Some of this scattered light strikes A
  - Etc. Etc…

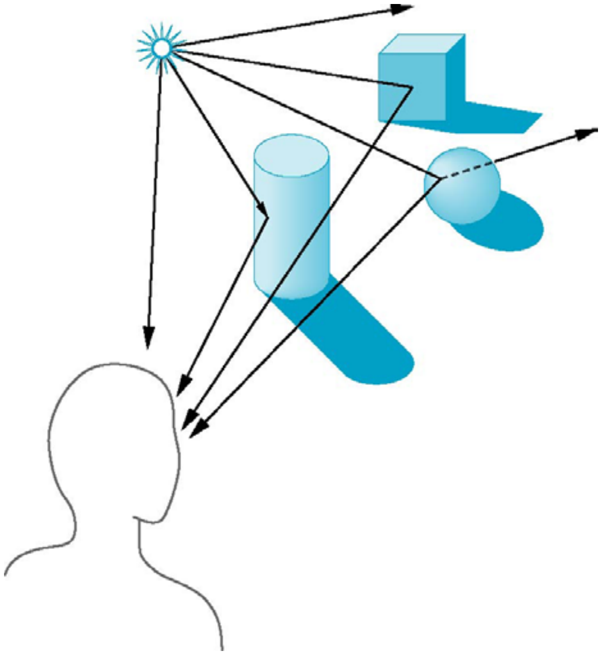
Global vs Local Shading

- This infinite scattering and absorption of light can be performed by *ray tracing*
    - This is a whole different course!
- Such *global* rendering requires all objects and lights sources
- So often too slow
- But there are ways to approximate global effects that look right (**local** methods)



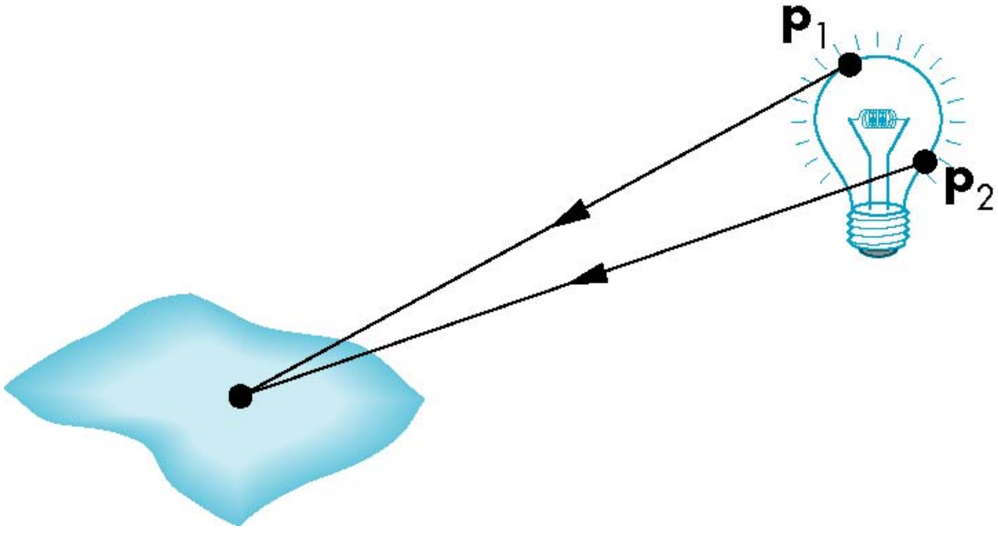
Global vs Local Shading

- The simplest (and most common) local shading assumes independence of all polygons.
- When computing the shade of a pixel on a polygon only take into account
  - The vertices of the polygon
  - The orientation of the polygon
  - The location of the viewer
  - The light sources
  - Material properties of the polygon
- Lets first talk about the different types of light sources.



Light Sources

- *General* light sources are difficult to work with since we must integrate lights coming from all points on the source



Simple Light Sources

- Instead there are four simple types of light sources that are sufficient for rendering most scenes

1. **Point Sources**
   - Model with position and color
2. **Spotlight**
   - Restrict light from ideal point source with a cone
3. **Distance light**
   - Source is at infinity and act as a directional light instead of point source
4. **Ambient light**
   - Same amount of light everywhere in scene
   - Can model contribution of many sources and reflecting surfaces
     - Simulates indirect lighting

Point Sources

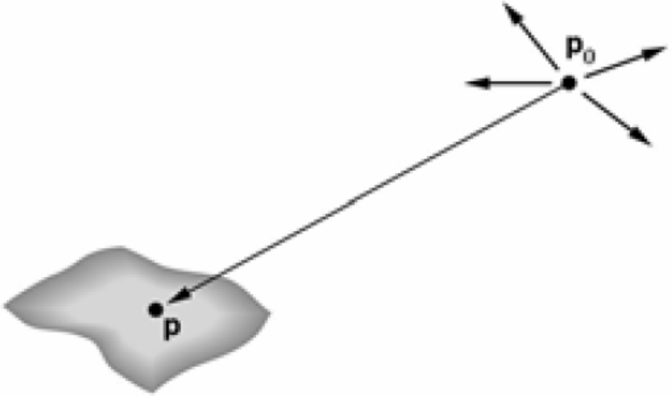
- An ideal point source emits light equally in all directions.
- Characterize a point source located at point p_0 using its RGB colors:

$$I(p_0) = \begin{bmatrix} I_r(p_0) \\ I_g(p_0) \\ I_b(p_0) \end{bmatrix}$$

Point Sources

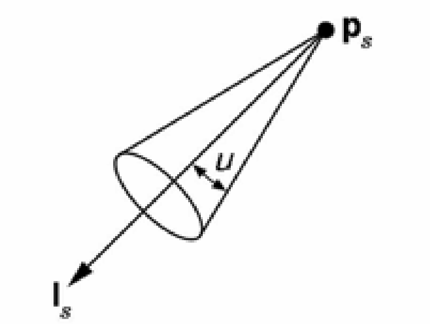
- The intensity of illumination received from a point source is proportional to the inverse square of the distance between the source and the surface
- At point p, the *intensity of light* received from a point source is given by:

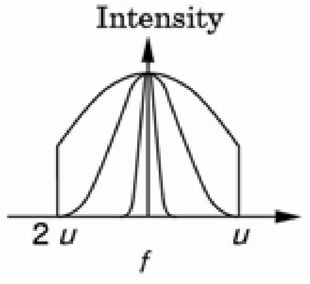
$$I(p, p_0) = \frac{1}{|p - p_0|^2} I(p_0)$$



Spotlights

- Spotlights are characterized by a narrow range of angles through which light is emitted.
- Given a point source, **limit the angle** at which light can be seen.
- Use a *cone* whose apex is at p_s which points in the direction l_s and width is determined by an angle u

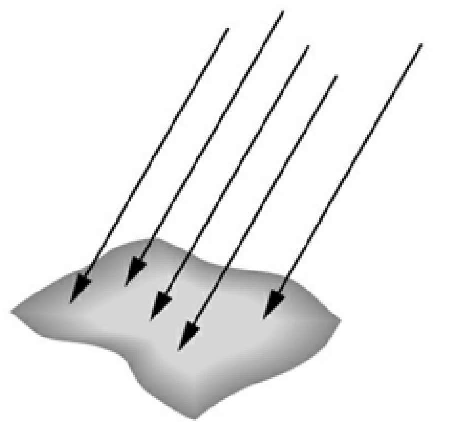




Distant Light

- Source is at infinity
- Replace a point source with a *parallel source* that illuminates objects with parallel rays of light
- Replace location of the light source with **direction** of the light source
- Intensity **doesn't** attenuate with distance

$$I(p, p_0) = I(p_0)$$



Ambient Light

- Ambient light is the uniform lighting in the environment
  - The "background glow"
- Ambient illumination is characterize by an intensity, I_a, that is identical at every point in the scene

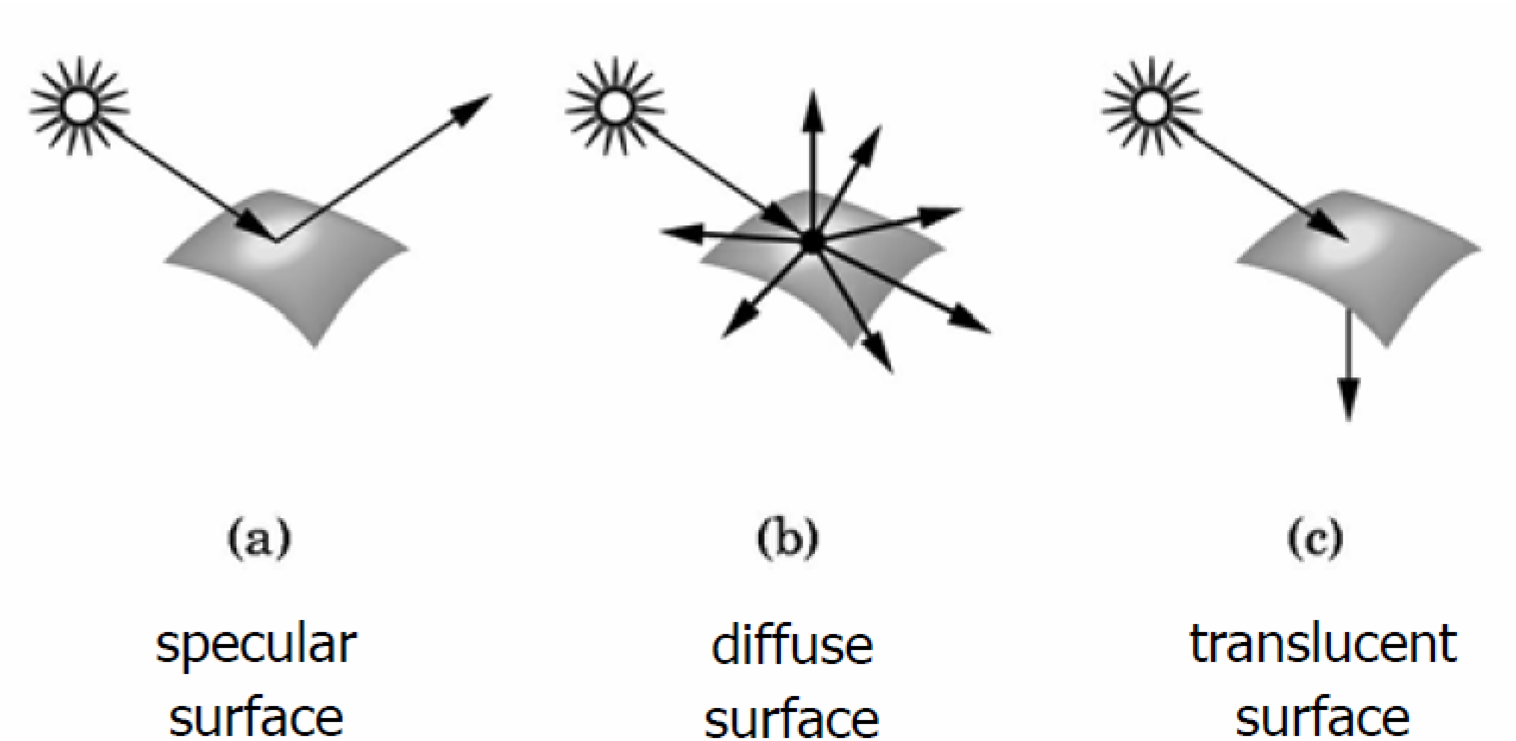
Light-Material Interaction

- Ok so that's the intensity of the light that reaches a point.
- But what happens when light hits a surface?
- Several things!
  - Some light is absorbed
  - Some light is reflected
  - Some light is transmitted (transparency?)
- How much and what color of each of these depends not only on the light but also on the *material properties* of the surface.
- So next let's look at light-material interaction.

Light-Material Interaction

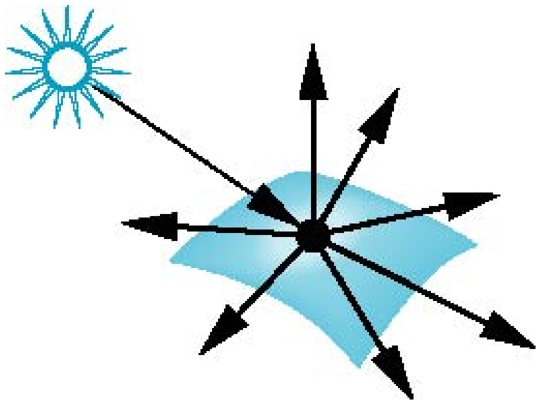
- The color of a surface is determined by its
  - Orientation
  - Material
  - Parameters of the light sources
  - The lighting model
  - Ambient color
- Material is specified as a combination of material parameters:
  - Speculativeness
  - Shininess
  - Diffusiveness
  - Emmisiveness

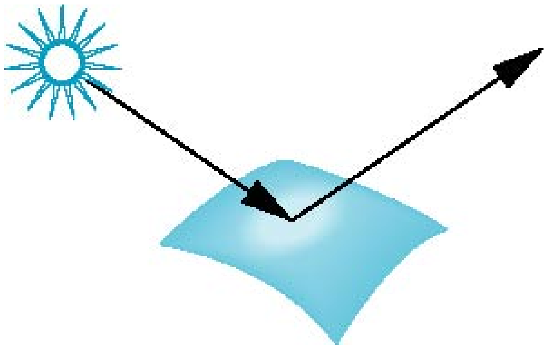
Material Properties



Surface Types

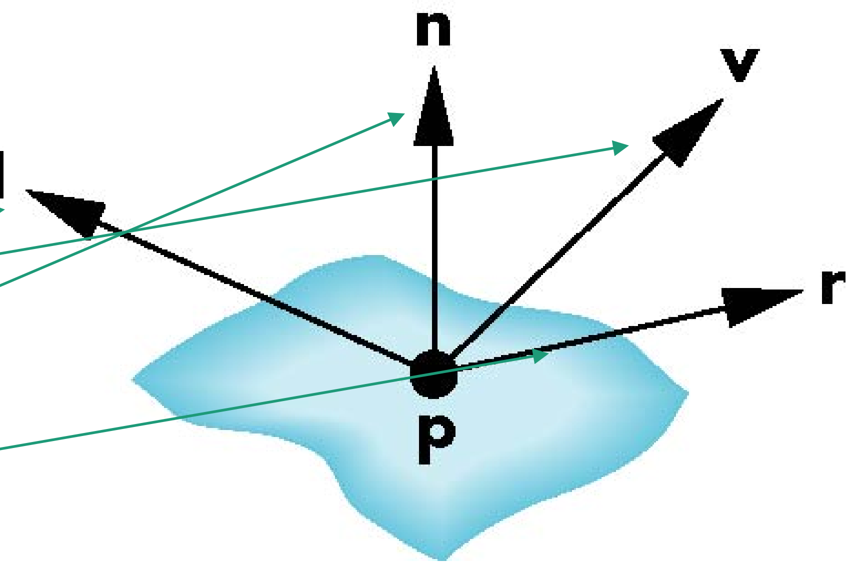
- **Diffuse surfaces** are characterized by reflected light being scattered in all directions
  - Common for rough surfaces
- **Specular surfaces** appear shiny because most of the light is scattered in a narrow range of angles close to the angle of reflection
  - Mirrors are perfect specular surfaces
- **Translucent surfaces** allow some light to penetrate the surface and to emerge from another location on the object in a process called *refraction*.





Phong Model

- A simple model that can be computed rapidly
- Provides a close approximation to physical reality
- Has three components
  - Diffuse
  - Specular
  - Ambient
- Uses four vectors
  - To light source
  - To viewer
  - Surface Normal
  - Perfect reflector



Phong Reflective Model

- The total intensity at point p due to a light is a combination of both the illumination and the reflection (as contributed by the ambient, diffuse, and specular)
- We can the get the total intensity by adding the contribution of all sources and a global ambient term.

Global Ambient

$$I = \sum_i (I_{ia} + I_{id} + I_{is}) + I_a$$

- Ok so how do we compute each of the components?
    - First we need material properties

Model Properties

RGB Colors

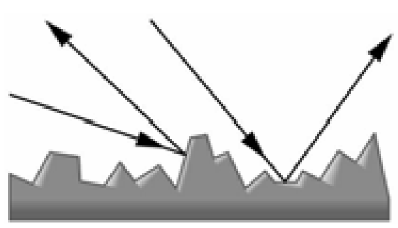
- Material Properties
  - Diffuse Coefficients: k_{dr}, k_{dg}, k_{db}
  - Specular Coefficients: k_{sr}, k_{sg}, k_{sb}
    - Also need a shininess coefficient: α
  - Ambient Coefficients: k_{ar}, k_{ag}, k_{ab}
- Light Properties (there may be several of them)
  - Diffuse Coefficients: L_{dr}, L_{dg}, L_{db}
  - Specular Coefficients: L_{sr}, L_{sg}, L_{sb}
  - Ambient Coefficients: L_{ar}, L_{ag}, L_{ab}

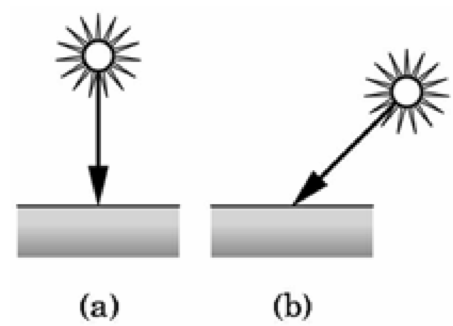
Model Properties

- The intensity due to specularity and diffusion depends not only on material and light properties but
  - Point/Light Position
  - Viewing angle
  - Surface orientation
- Let R_s, R_d be the scaling due to these factors.
- Then our final illumination computation for a particular light source is:
  - $I = R_s L_s k_s + R_d L_d k_d + L_a k_a$
- Obviously we must now define R_s, R_d

Diffuse Reflection

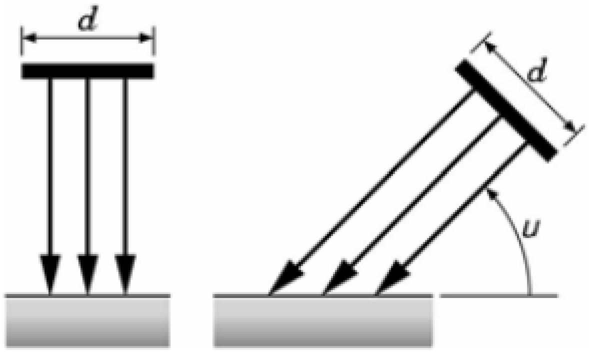
- A *perfectly diffuse reflector* scatters light equally in all directions
- The amount of light reflected depends on the material and the orientation of the light source





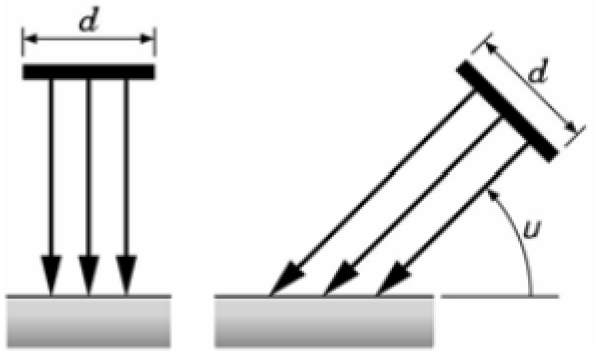
Diffuse Reflection

- The relationship between brightness and surface orientation is called *Lambert's Law*
- So the larger u is the more "concentrated" the light is
  - Spread over a smaller area



Diffuse Reflection

- We usually have (or can compute) the surface normal.
- Therefore we can compute the intensity/concentration of the light at point p as $R_d \propto \cos\theta$ where θ is the angle between the normal n and the direction **to** the light source l.
- If these are unit vectors, then
  - $\cos\theta = l \cdot n$ (the dot product of the vectors)
- Therefore $R_d = l \cdot n$

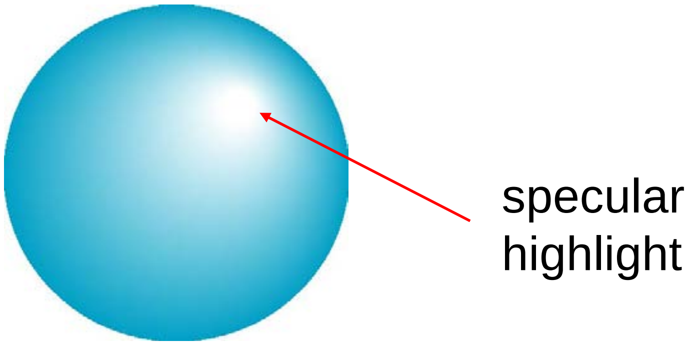


Diffuse Example

- Given:
  - A light source at $L = (3,2,0)$ with diffuse coefficients $L_d = [0.8,0.20,1]$
  - A point on a surface at $P = (1,1,1)$ with normal $N = (1,1,0)$ and diffuse coefficients $k_d = [1,1,0]$
- What is the resulting diffuse color intensities?
  - The direction to the light (from the point) is
    $$l = L - P = [2,1,-1]$$
  - Normalizing each vector we get $l \approx [0.8,0.4,-0.4]$ and $N \approx [0.7, 0.7, 0]$
  - Therefore the intensity of each channel is proportional to
    $$\mathrm{R_d} = l \cdot N \approx 0.866$$
  - The diffuse color is then:
    $$\begin{aligned} I_d &= L_d k_d R_d \\ &= [0.8 \cdot 1 \cdot 0.866, 0.2 \cdot 1 \cdot 0.866, 0 \cdot 0 \cdot 0.866] \\ &= [0.6928, 0.1732,0] \end{aligned}$$

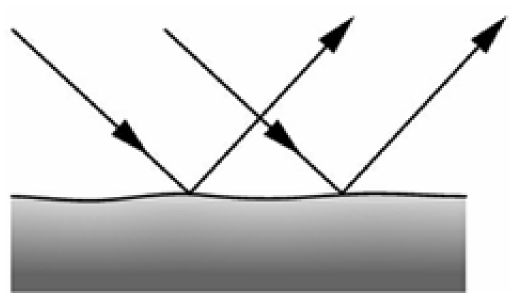
Specular Surfaces

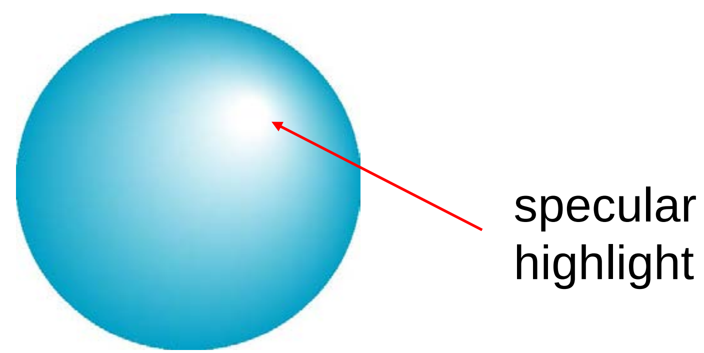
- Most surfaces are neither ideal diffusers nor perfectly specular (ideal reflectors).
- Smooth surfaces show specular highlights due to incoming light being reflected in directions concentrated close to the direction of a perfect reflection
- Color is different from the color of the reflected ambient and diffuse light



Specular Reflection

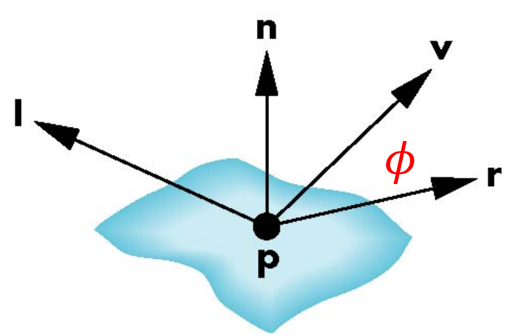
- The smoother the surface, the more concentrated the light is in a smaller range of angles



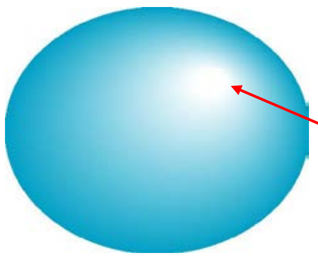


Specular Reflection

- Phong proposed using a term that dropped off as the angle between the viewer and the ideal reflection increased
$$R_s = cos^{\alpha}\emptyset$$
  - Where ∅ is the angle between the direction of the *perfect reflector,* r, and the direction of the *viewer*, v**,** and α is a *shininess coefficient)*
- If we normalize r and v we can again use the dot product to compute the *specular reflection term:*
$$R_s = (r \cdot v)^{\alpha}$$



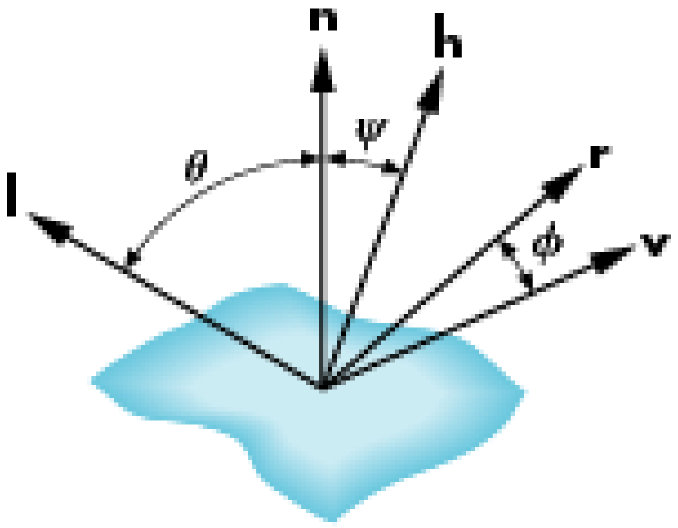
As ϕ goes to zero (v=r), cos(ϕ) goes towards one



specular highlight

Modified Specular reflection

- The specular term is problematic because it requires the calculation of a new refection vector and view vector for each vertex
- Blinn suggested an approximation using the *halfway* vector that is more efficient

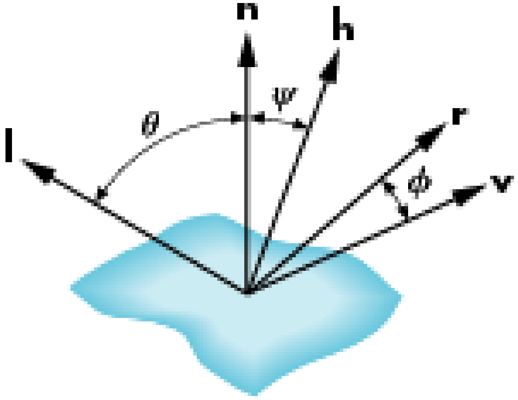


The Halfway Vector

- h is normalized vector halfway between l (the light to vector) and v (the viewer vector)

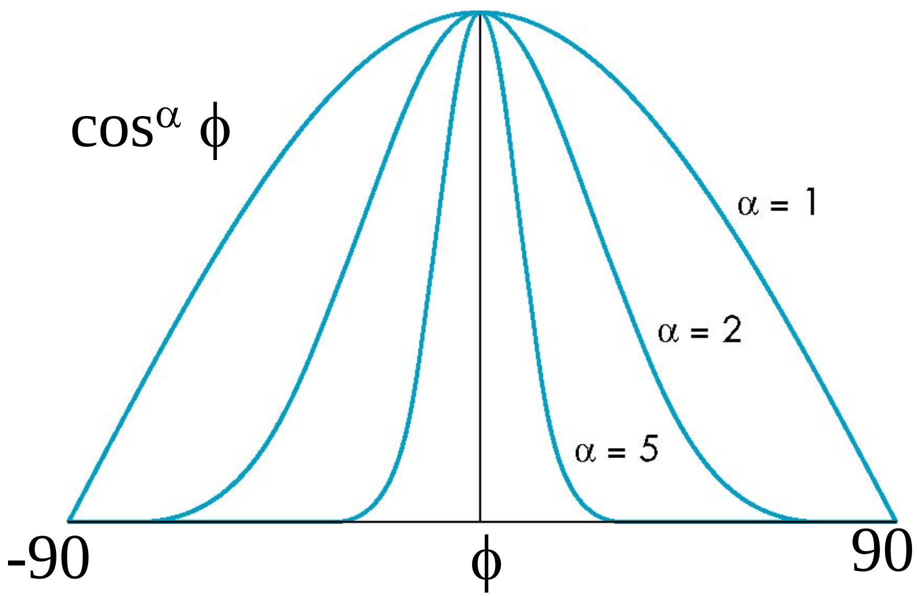
$$h = \frac{l + v}{|l + v|}$$

  - Where $|l + v|$ is the length of the vector $|l + v|$ (in order to make h unit length)
- We can then replace $(v \cdot r)^{\alpha}$ by $(n \cdot h)^{\alpha}$
- So at least we don't need to calculate r



The Shininess Coefficient

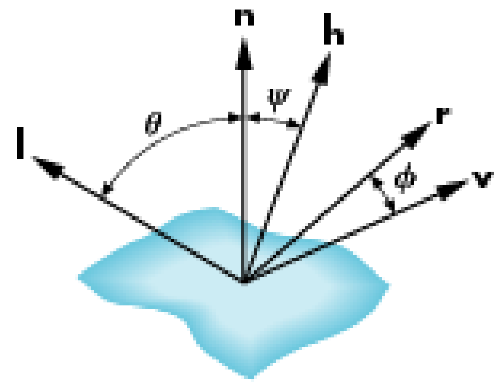
- Values of α between 100 and 200 correspond to metals
- Values between 5 and 10 give surfaces a plastic look



Specular Example

- Given:
  - A light source at $L = (3,2,0)$ with specular coefficients $L_s = [1,1,1]$ and shininess $\alpha = 10$
  - A point on a surface at $P = (1,1,1)$ with normal $N = (1,1,0)$ and specular coefficients $k_s = [0.2,1,1]$
  - The viewer is at location $M = (5,2,2)$
- What is the resulting specular color intensities?
  - The direction to the light (from the point) is
$$l = L - P = [2,1,-1]$$
  - The direction to the viewer is
$$v = M - P = [5,2,2] - [1,1,1] = [4,1,1]$$
  - Normalizing each vector we get $l \approx [0.8,0.4,-0.4]$, $N \approx [0.7, 0.7, 0]$, and $v \approx [0.94, 0.24, 0.24]$
  - The halfway vector is then
$$h \approx \frac{[1.76, 0.64, -0.17]}{1.9} \approx [0.94, 0.34, -0.09]$$
  - And $R_s \propto (n \cdot h)^\alpha = 0.36$
  - The specular color is then:
$$\begin{aligned} I_s &= k_s R_s L_s \\ &= [0.2 \cdot 0.36 \cdot 1, 1 \cdot 0.36 \cdot 1, 1 \cdot 0.36 \cdot 1] \\ &= [0.072, 0.36, 0.36] \end{aligned}$$

Putting It All Together…

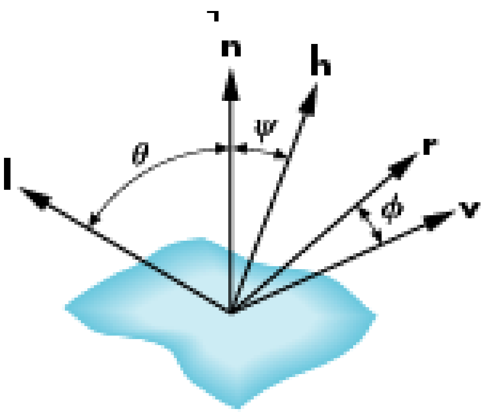


- So given
  - Light location
  - Light properties
  - Material properties
  - Point location
- We can compute the intensity I at that point as
  - $I = R_s L_s k_s + R_d L_d k_d + L_a k_d$
  - $I = (n \cdot h)^{\alpha} L_s k_s + (l \cdot n) L_d k_d + L_a k_a$
- But we still need to take into account the *type* of light
  - Point
  - Distant
  - Spot

Point Source Lighting Model

- Recall for point sources we must take the distance of the point from the light into account:
  - $\frac{1}{|p-p_0|^2}$ is the *distance attenuation factor* due to the distance between the point p and the light source p_0
- Then our final intensity becomes:

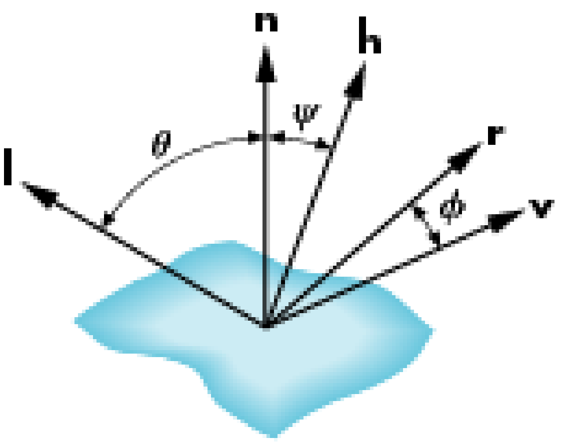
$$I = \frac{1}{|p-p_0|^2}\left[(n \cdot h)^{\alpha} L_s k_s + (l \cdot n) L_d k_d + L_a k_a\right]$$



Distant Source Lighting Model

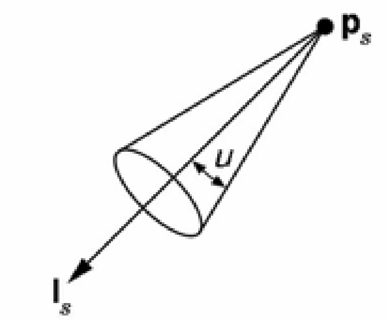
- For distance sources we're given l directly and don't need to compute it.
- Furthermore, the intensity doesn't attenuate.
- Therefore we just have

$$I = (n \cdot h)^{\alpha} L_s k_s + (l \cdot n) L_d k_d + L_a k_a$$



Spotlight Lighting Model

- Similar to point sources, but we only want it to be seen (the light that is) if we're within a certain angle of direction of the spotlight.
  - Therefore we must be given the light location, orientation and how fast the intensity should fall off as we go away from the direction
  - Sometimes we're also given a maximum angle, u



Spotlight Lighting Model

- Given the ray from the light source to the point, l_s (computed using the light position and the point) and the direction of the light l, we can compute the angle using the dot product (as long as the vectors are normalized) as

$$l \cdot l_s$$

- If our intensity is suppose to fall off as we get away from the direction of the light, we have another attenuation factor

$$c_{spot} = \max(l \cdot l_s, 0)^{s_{exp}}$$

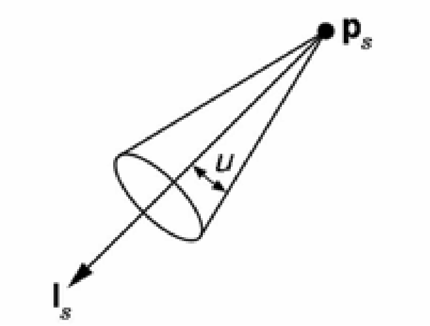
where s_{exp} controls how quickly the intensity falls off from the center of the source

Spotlight Lighting Model

- If we want to cut-off the light when we go beyond θ_{max} we can compute the angle between the light-to-point and the light's direction as:

$$\theta = \cos^{-1}(l \cdot l_s)$$

- And if $|\theta| > \theta_{max}$ then $c_{spot} = 0$

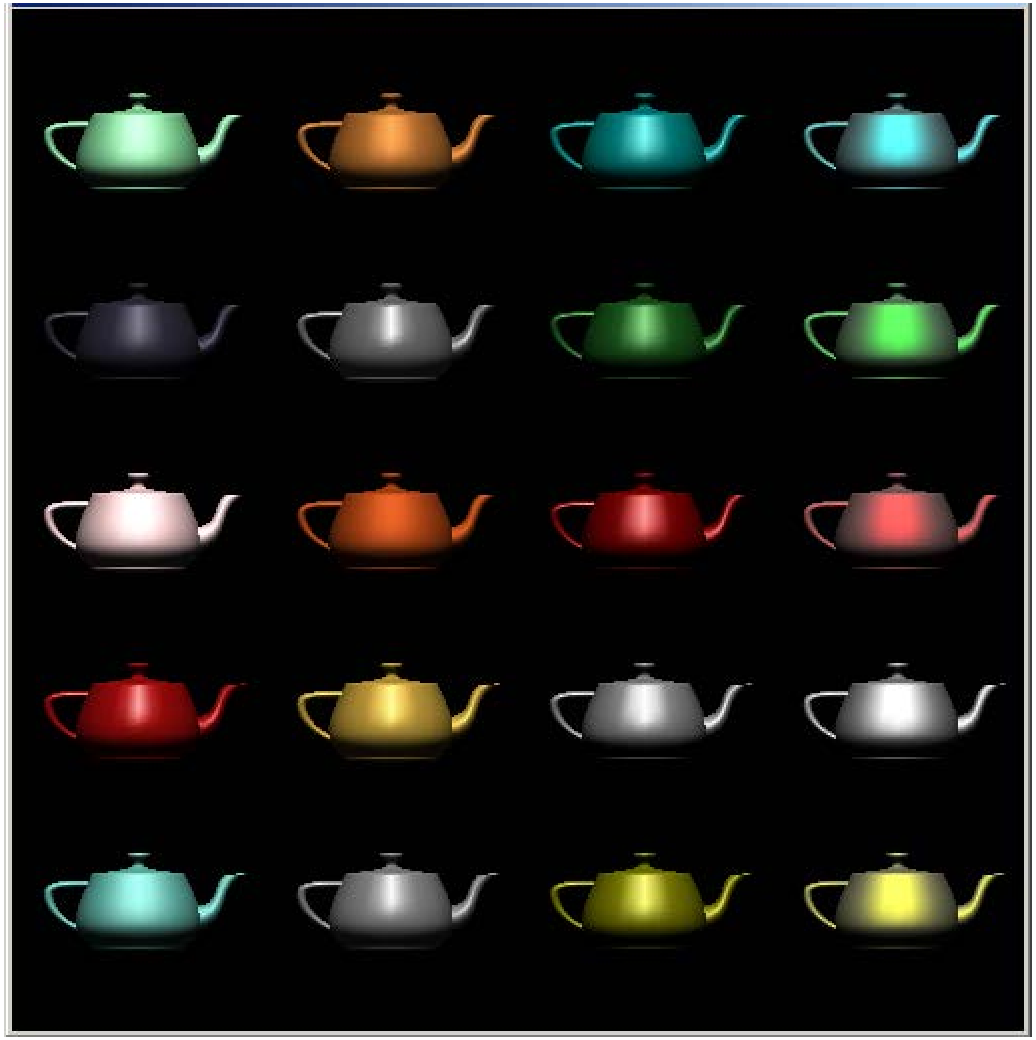


Spotlight Lighting Model

- From point lights we have
$$I = \frac{1}{|p - p_0|^2}\left[\left(n \cdot h\right)^{\alpha} L_s k_s + \left(l \cdot n\right) L_d k_d + L_a k_a\right]$$
- Using the previous equation $c_{spot} = \max(l \cdot l_s, 0)^{s_{exp}}$ (and/or taking max angle u into account) we get the final intensity value of:
$$I = c_{spot} \frac{1}{|p - p_0|^2}\left[(n \cdot h)^{\alpha} L_s k_s + (l \cdot n) L_d k_d + L_a k_a\right]$$

Example

- Only differences in these teapots are the parameters in the modified Phong model



Computation of Vectors

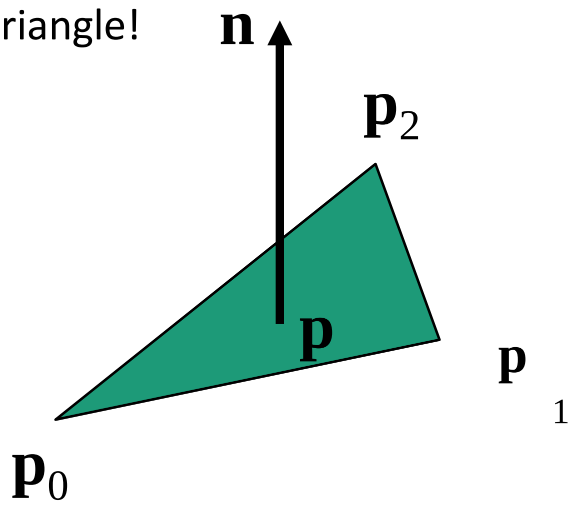
- So to do our computations we need a bunch of vectors
  - $\boldsymbol{l}$= direction of light
  - $\boldsymbol{v}$ = direction from point to viewer
  - $\boldsymbol{n}$ = normal of surface
  - $\boldsymbol{r}$ = reflection vector (don't need if we use half-way vector h)
- $\boldsymbol{l}$ and $\boldsymbol{v}$ are specified by the application
- Can compute h from l and v
- How do we get the normals, n?

Normals

- If we have a mathematical representation of the surface, we can compute a normal at every single point and apply shading to each of those
  - May be prohibitively expensive
  - Plus we might not have a mathematical model form.
  - But this can greatly improve the rendering quality.
- Often instead we compute a single normal per polygon (triangle)
  - If we want finer shading, we can just represent our surface with more, smaller polygons

Normal for Triangle

- Given: Three vertices specifying a triangle: p_0, p_1, p_2
- We can then compute two non-collinear vectors on the triangle:
  $$(p_1 - p_0), (\mathrm{p}_2 - \mathrm{p}_0)$$
- Taking the cross product of these we get the normal of the triangle!
  $$n = (p_2 - p_0) \times (p_1 - p_0)$$
- Which we should then normalize:
  $$n = \frac{n}{|n|}$$



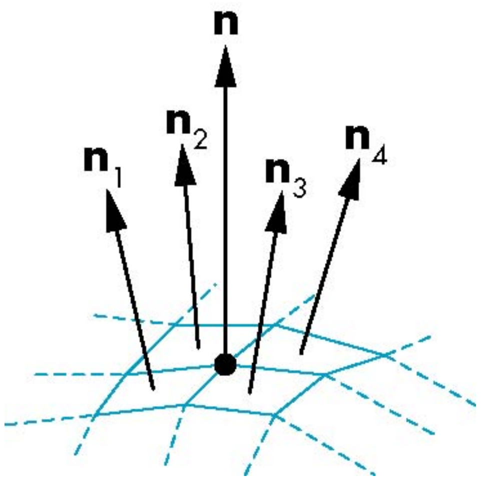
- NOTES:
  - The right-hand-rule (RHR) determines outward face
  - Both Angel (for the CPU) and GLSL (for the GPU shaders) provide a *normalize* function
    - This can especially be useful when doing cross projects

Normals

- But we want to supply a normal for each *vertex* not each triangle.
- To compute the normals we can
  - Process each polygonal face, computing the normal for a vertex of it and apply that normal to all vertices pertaining to that face
    - NOTE: Later polygons will overwrite vertex normals of shared vertices
  - Do the same thing as above but for the vertex's final normal use the *average* of it's proposed normals
    - Extra computation (not too bad) but often may result in much smoother shading
      - This is called Gouraud Shading
- And now of course the normals for each vertex must be placed in a buffer to be accessed by the shaders.

Gouraud Shading

- For polygonal models, Gouraud proposed we use the *average* of the normal around a mesh vertex for the vertex's actual normal
  - $\mathbf{n} = (\mathbf{n}_1+\mathbf{n}_2+\mathbf{n}_3+\mathbf{n}_4) / |\mathbf{n}_1+\mathbf{n}_2+\mathbf{n}_3+\mathbf{n}_4|$
- This provides for smooth shading



Normals

- The reflection direction obviously depends on the normal, so we must first compute them
- Surfaces are approximated by a set of flat polygons
  - To approximate smooth/curved surfaces we will have many small polygons
- Vectors perpendicular to these polygons can be used as the surface normal
  - However, these normal are discontinuous across polygon boundaries
- If a mathematical description of the surface exists (equation), then we can compute a normal at every point (true normals)
  - This can greatly improve the rendering quality.