CSE 167: Introduction to Computer Graphics Lecture #5: Illumination

Jürgen P. Schulze, Ph.D.
University of California, San Diego
Fall Quarter 2019

Announcements

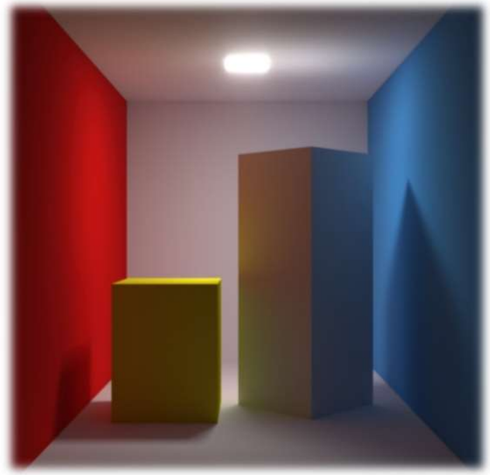
- Project 2 due this Friday at 2pm
  - Grading in basement labs B260 and B270 (others ok)
  - Upload code to Canvas by 2pm
- Midterm #1: next Thursday in lieu of lecture

Shading

Shading

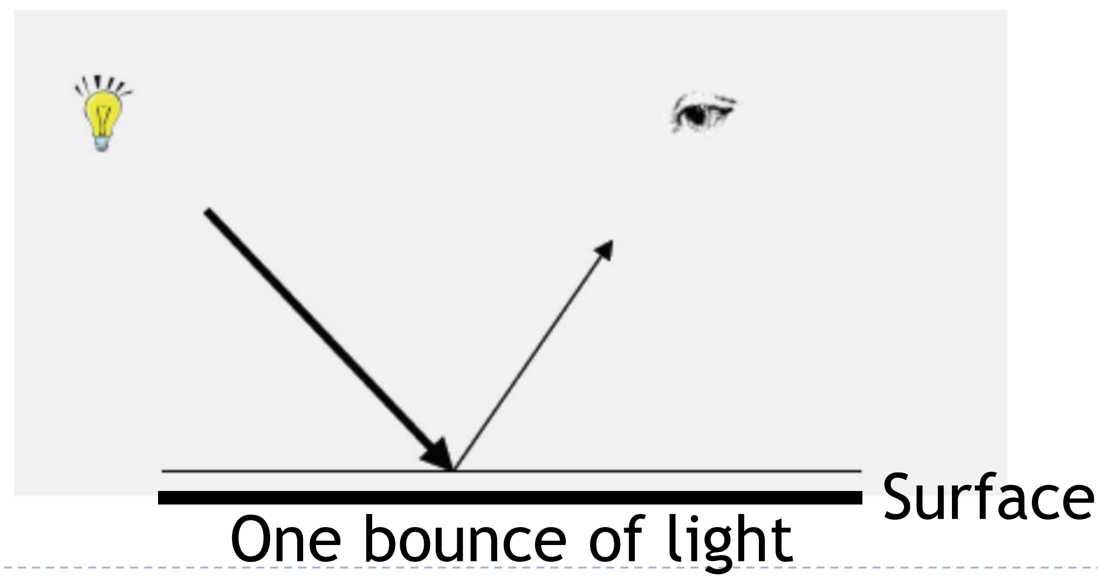
- Compute interaction of light with surfaces
- Requires simulation of physics
- "Global illumination"
  - Multiple bounces of light
  - Computationally expensive, minutes per image
  - Used in movies, architectural design, etc.
- Appearance = Material Definition + Light Sources

Global Illumination



Interactive Applications

- No physics-based simulation
- Simplified models
- Reproduce perceptually most important effects
- Local illumination
  - Only one bounce of light between light source and viewer



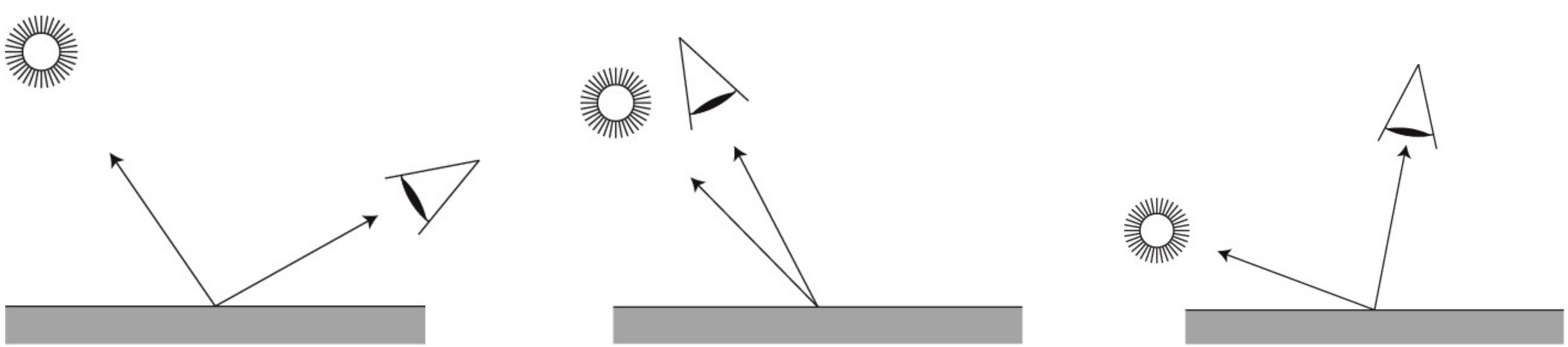
UCSD

Local Illumination

- Gives material its color
- Light can be reflected by
  - Mirror
  - White wall
  - Glossy metal
  - etc.

Local Illumination

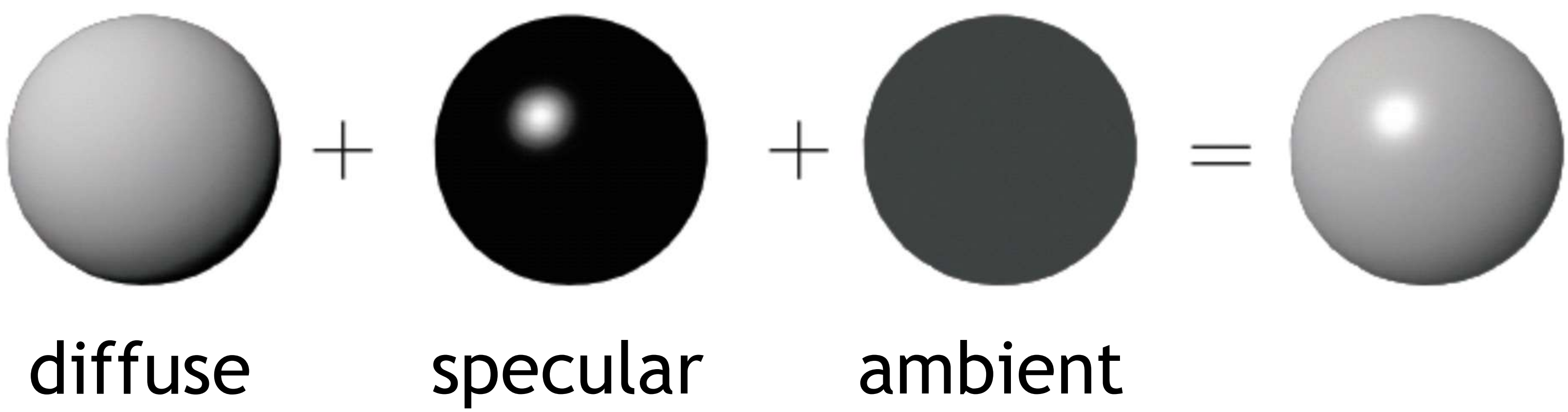
- Model reflection of light at surfaces
  - Assumption: no subsurface scattering
- Bidirectional reflectance distribution function (BRDF)
  - Given light direction, viewing direction, how much light is reflected towards the viewer
  - For any pair of light/viewing directions!



Local Illumination

Simplified model

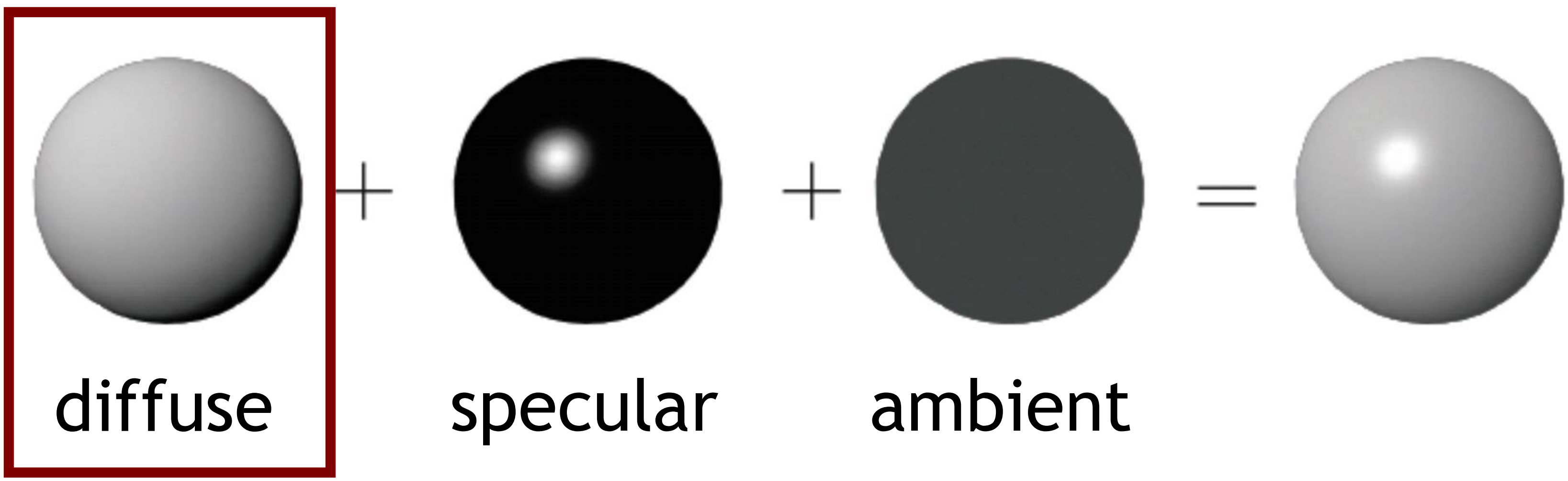
- Sum of 3 components
- Covers a large class of real surfaces



Local Illumination

Simplified model

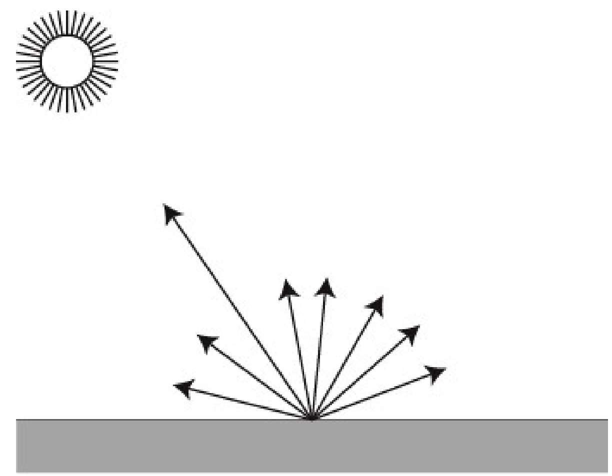
- Sum of 3 components
- Covers a large class of real surfaces

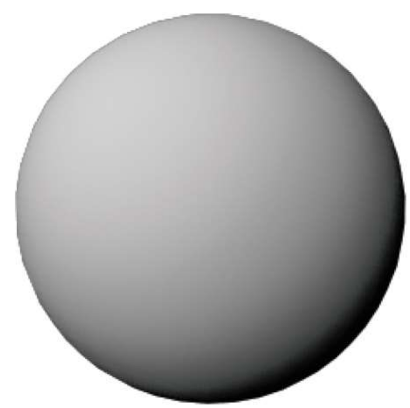


UCSD

Diffuse Reflection

- Ideal diffuse material reflects light equally in all directions
- View-independent
- Matte, not shiny materials
  - Paper
  - Unfinished wood
  - Unpolished stone

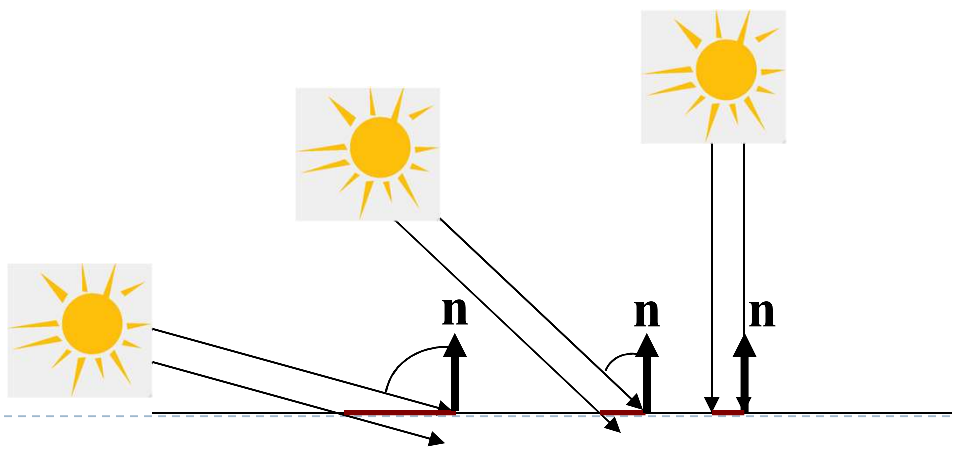


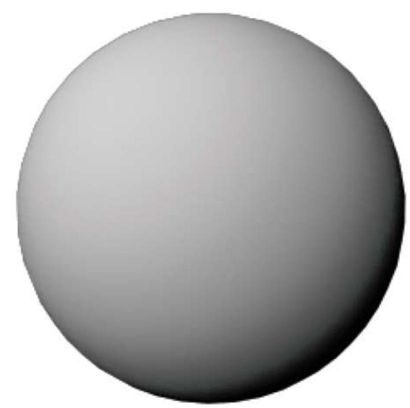


UCSD

Diffuse Reflection

- Beam of parallel rays shining on a surface
  - Area covered by beam varies with the angle between the beam and the normal
  - The larger the area, the less incident light per area
  - Incident light per unit area is proportional to the cosine of the angle between the normal and the light rays
- Object darkens as normal turns away from light
- Lambert's cosine law (Johann Heinrich Lambert, 1760)
- Diffuse surfaces are also called Lambertian surfaces





UCSD

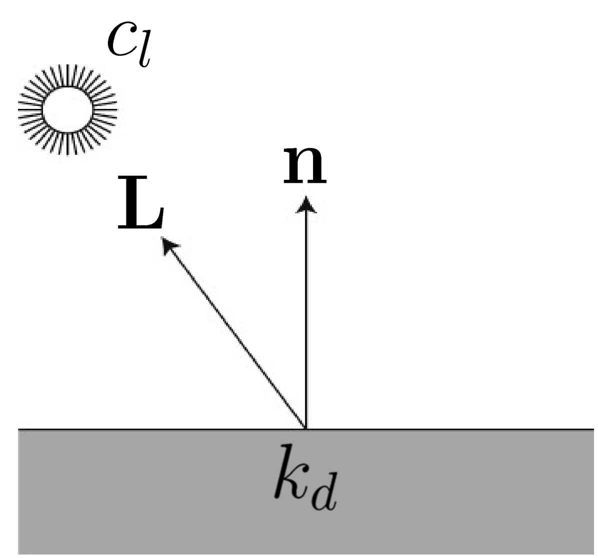
Diffuse Reflection

- Given
  - Unit (normalized!) surface normal **n**
  - Unit (normalized!) light direction **L**
  - Material diffuse reflectance (material color) k_d
  - Light color (intensity) c_l

- Diffuse color c_d is:

$$c_d = c_l k_d \underbrace{(\mathbf{n} \cdot \mathbf{L})}$$

Proportional to cosine between normal and light



UCSD

Diffuse Reflection

Notes

- Parameters k_d, c_l are r,g,b vectors
- Need to compute r,g,b values of diffuse color c_d separately
- Parameters in this model have no precise physical meaning
  - c_l: strength, color of light source
  - k_d: fraction of reflected light, material color

UCSD

Diffuse Reflection

- Provides visual cues
  - Surface curvature
  - Depth variation

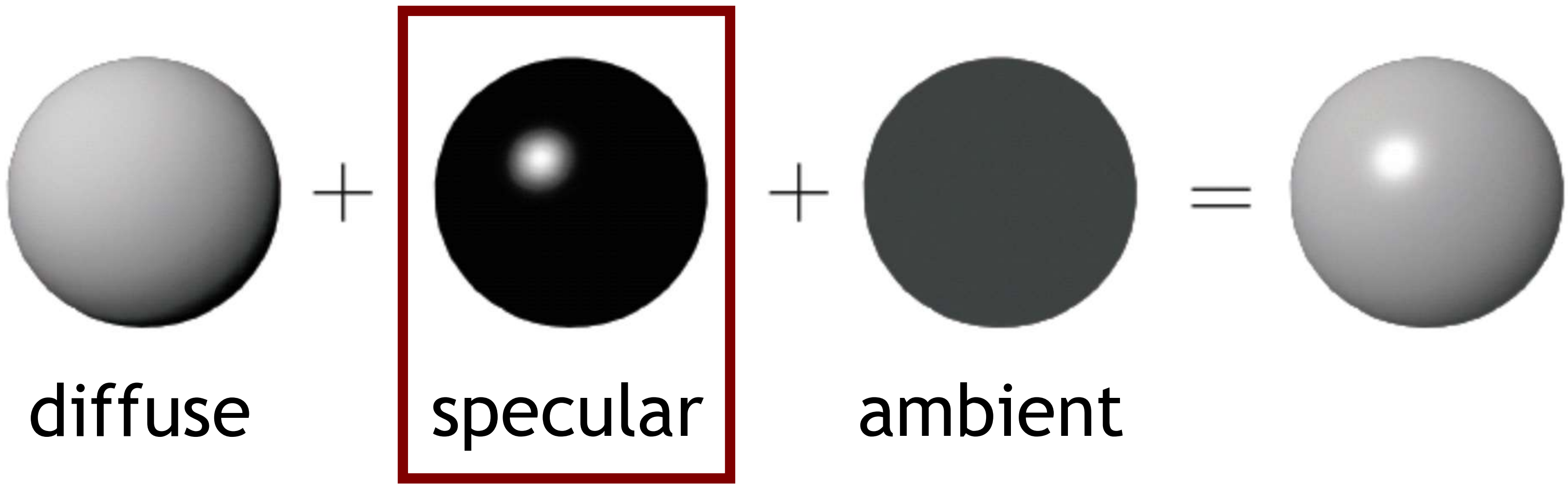
Lambertian (diffuse) sphere under different lighting directions

UCSD

Local Illumination

Simplified model

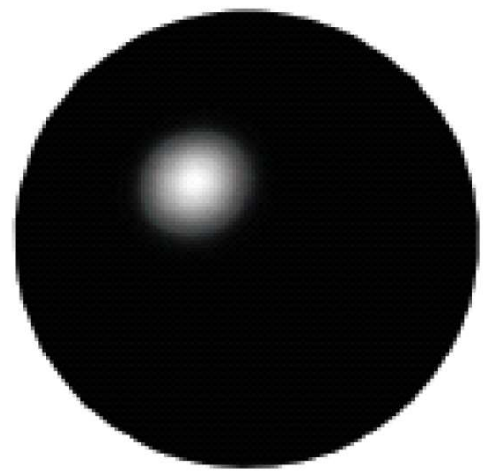
- Sum of 3 components
- Covers a large class of real surfaces



UCSD

Specular Reflection

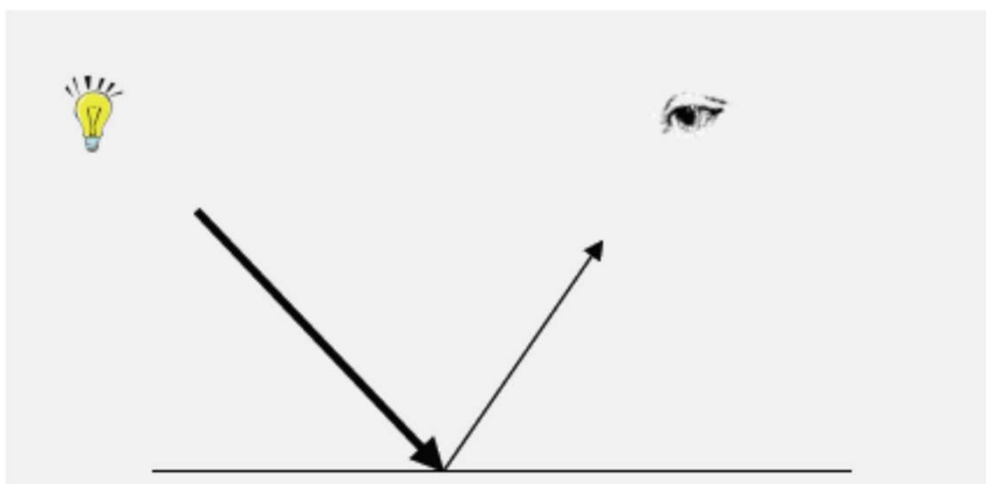
- Shiny surfaces
  - Polished metal
  - Glossy car finish
  - Plastics
- Specular highlight
  - Blurred reflection of the light source
  - Position of highlight depends on viewing direction



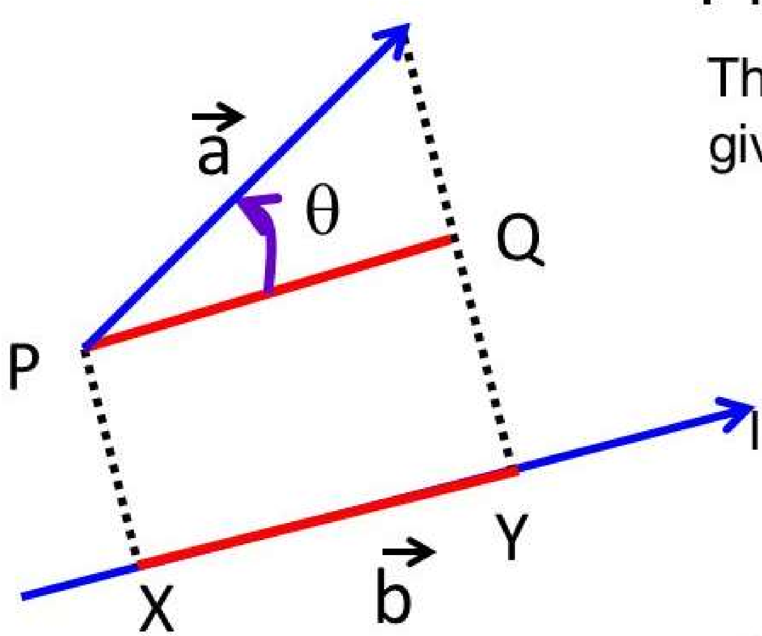
Specular highlight

Specular Reflection

- Ideal specular reflection is mirror reflection
  - Perfectly smooth surface
  - Incoming light ray is bounced in single direction
  - Angle of incidence equals angle of reflection



Projection of vector on another vector



Projection of $\vec{a}$ on $\vec{b}$ is XY

The projection of **a** onto **b** will be given by:

$$proj_b a = |a| \cos\theta \, \frac{\mathrm{b}}{|\mathrm{b}|}$$

In summary, the $proj_a$ b has length $|a| \cos\theta$, and direction $\frac{b}{|b|}$

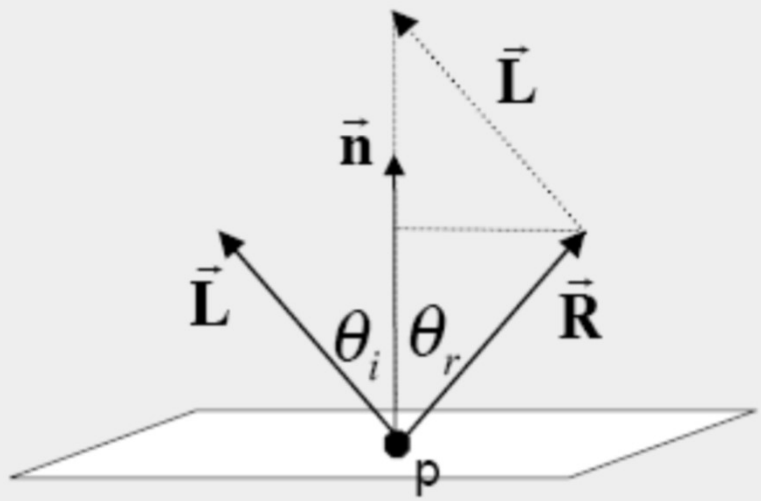
It is called the scalar component of **a** in the direction of **b**

Law of Reflection

- Angle of incidence equals angle of reflection

$$\vec{\mathbf{R}} + \vec{\mathbf{L}} = 2\cos\theta\ \vec{\mathbf{n}} = 2\left(\vec{\mathbf{L}} \cdot \vec{\mathbf{n}}\right)\vec{\mathbf{n}}$$

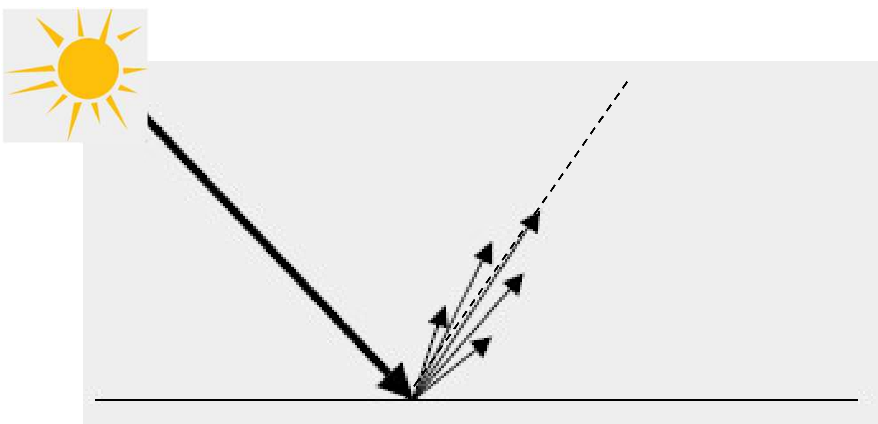
$$\vec{\mathbf{R}} = 2\left(\vec{\mathbf{L}} \cdot \vec{\mathbf{n}}\right)\vec{\mathbf{n}} - \vec{\mathbf{L}}$$



$$\theta_r = \theta_i$$

Specular Reflection

- Many materials are not perfect mirrors
  - Glossy materials



Glossy teapot

UCSD

Glossy Materials

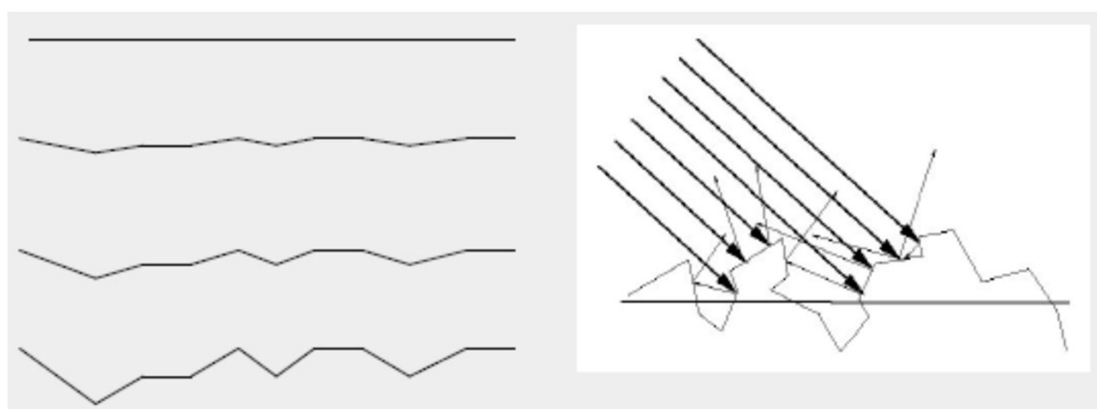
- Assume surface composed of small mirrors with random orientation (micro-facets)
- Smooth surfaces
  - Micro-facet normals close to surface normal
  - Sharp highlights
- Rough surfaces
  - Micro-facet normals vary strongly
  - Blurry highlight

Polished

Smooth

Rough

Very rough

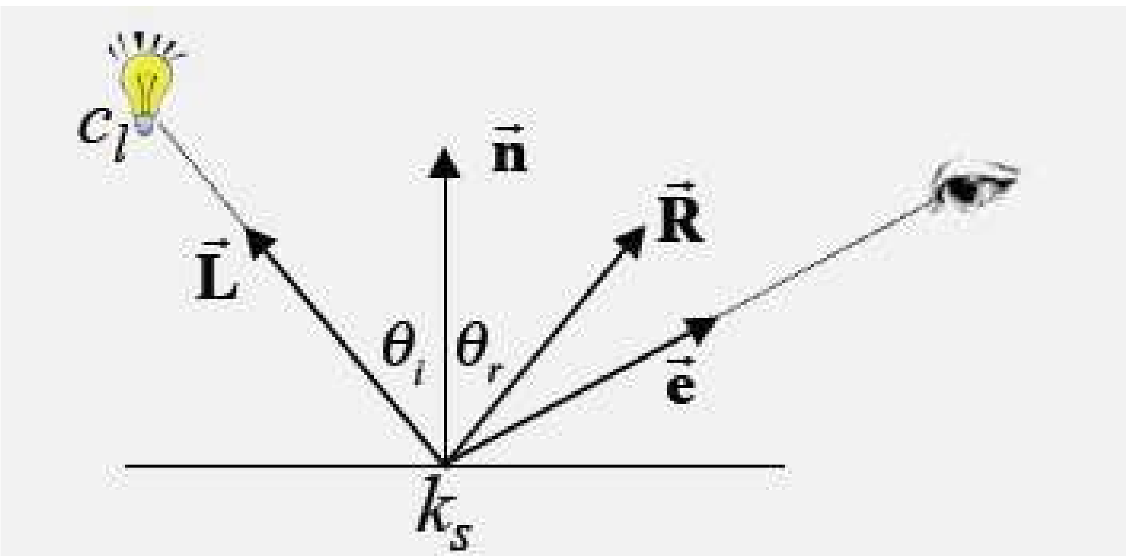


Glossy Surfaces

- Expect most light to be reflected in mirror direction
- Because of micro-facets, some light is reflected slightly off ideal reflection direction
- Reflection
  - Brightest when view vector is aligned with reflection
  - Decreases as angle between view vector and reflection direction increases

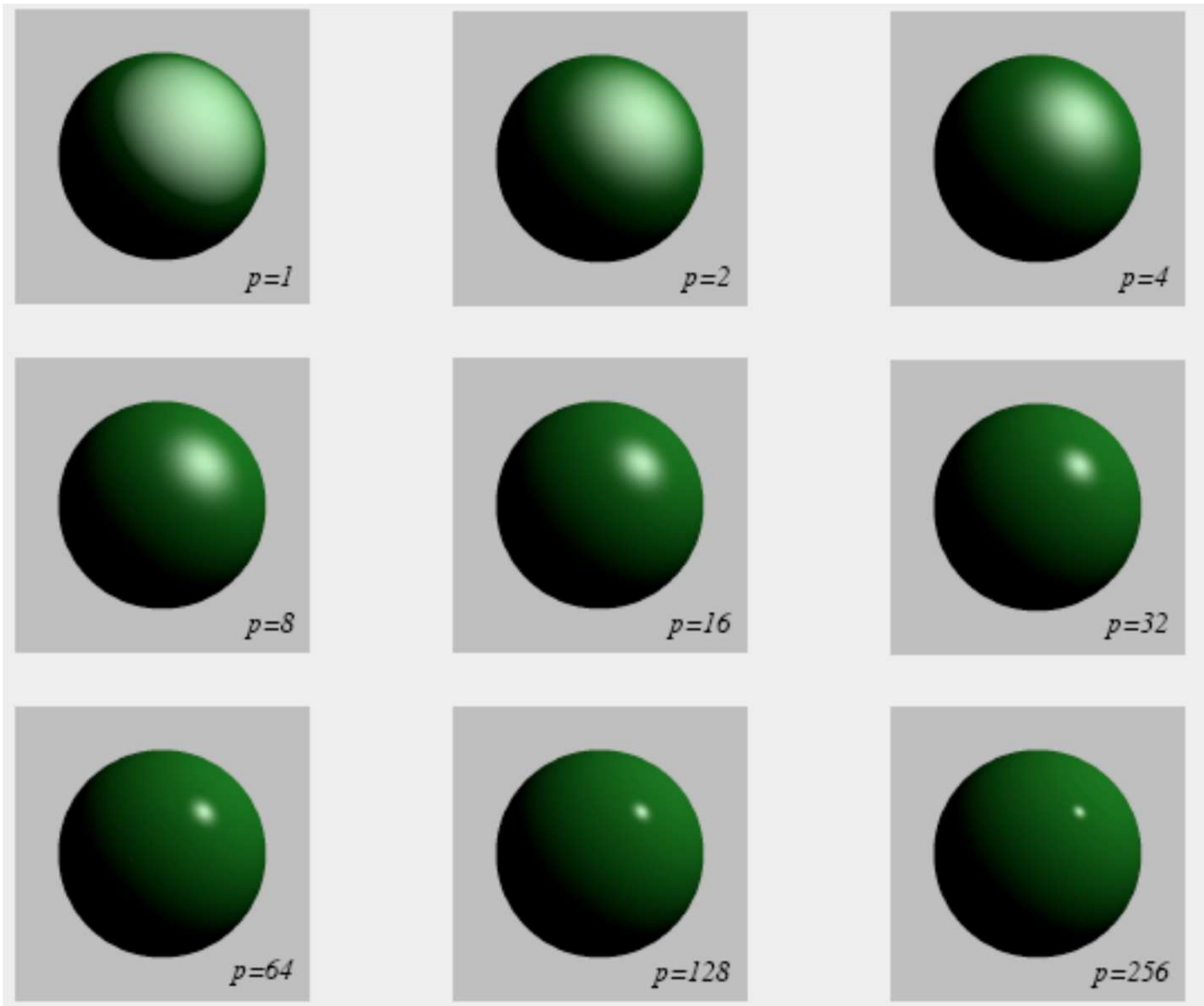
Phong Shading Model

- Developed by Bui Tuong Phong in1973
- Specular reflectance coefficient k_s
- Phong exponent p
  - Greater p means smaller (sharper) highlight



$$c = k_s c_l \left(\mathbf{R} \cdot \mathbf{e}\right)^p$$

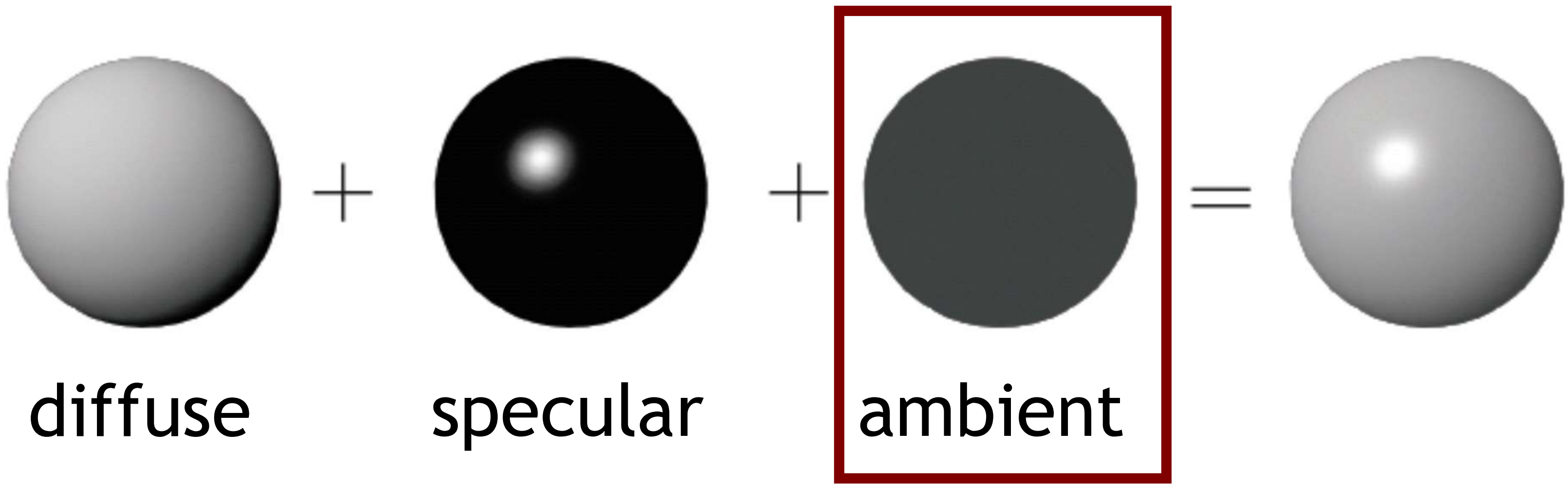
Phong Shading Model



Local Illumination

Simplified model

- Sum of 3 components
- Covers a large class of real surfaces



Ambient Light

- In real world, light is bounced all around scene
- Could use global illumination techniques to simulate
- Simple approximation
  - Add constant ambient light at each point: $k_a c_a$
  - Ambient light color: c_a
  - Ambient reflection coefficient: k_a
- Areas with no direct illumination are not completely dark

Complete Phong Shading Model

- Phong model supports multiple light sources
- All light colors *c* and material coefficients *k* are 3-component vectors for red, green, blue

$$c = \sum_i c_{l_i}(k_d(L_i \cdot n) + k_s(R \cdot e)^p + k_a)$$

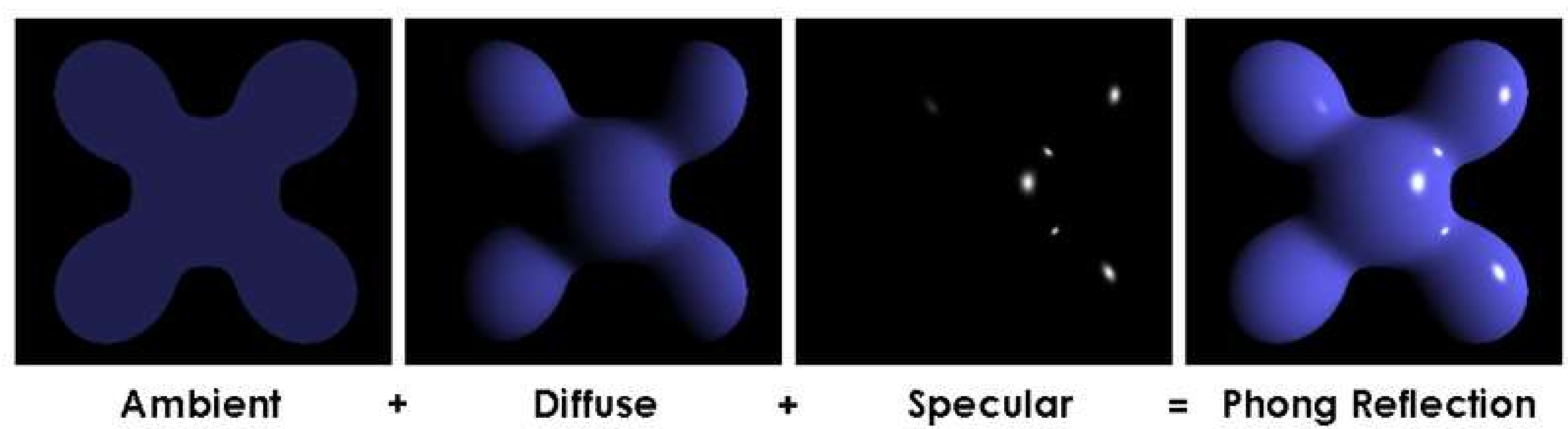


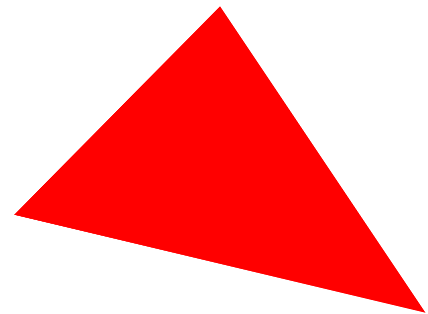
Image by Brad Smith

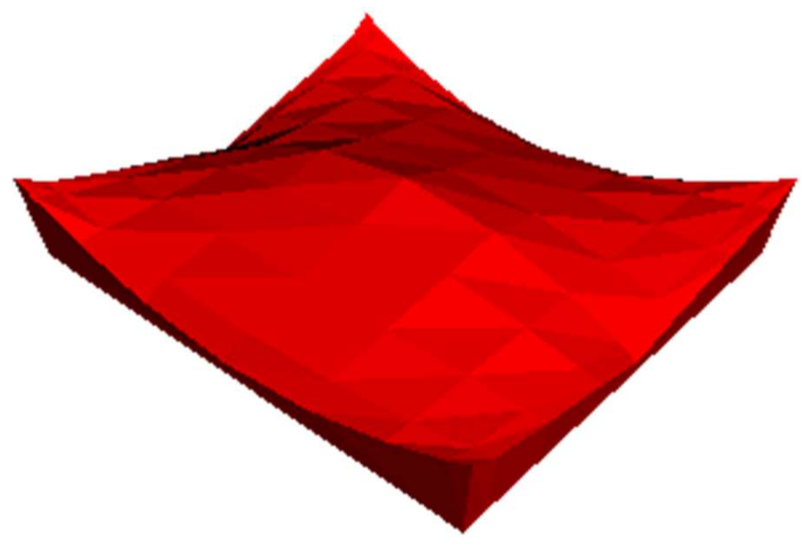
Types of Shading

- Per-triangle
- Per-vertex
- Per-pixel

Per-Triangle Shading

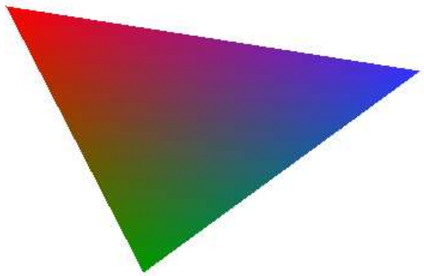
- A.k.a. *flat shading*
- Evaluate shading once per triangle, based on normal vector
- Advantage
  - Fast
- Disadvantage
  - Faceted appearance

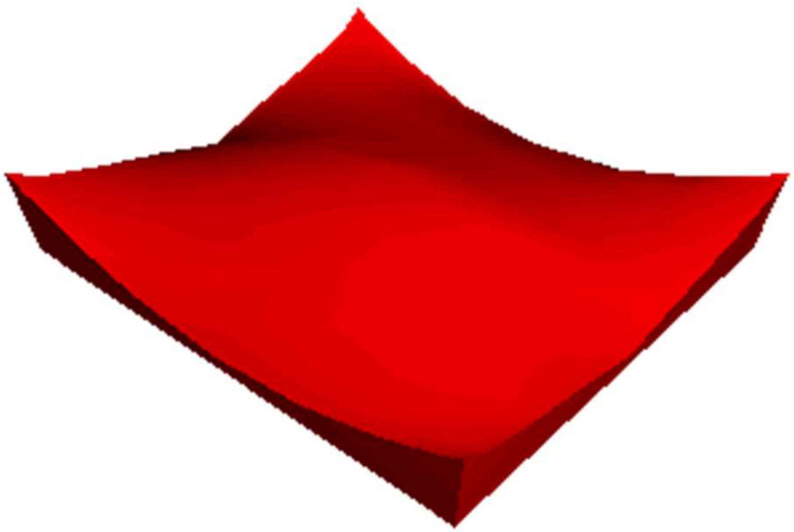




Per-Vertex Shading

- Known as *Gouraud shading* (→ Henri Gouraud, 1971)
- Interpolates vertex colors across triangles
- Advantages
  - Fast (no less work in fragment shader)
  - Smoother surface appearance than with flat shading
- Disadvantage
  - Problems with small highlights

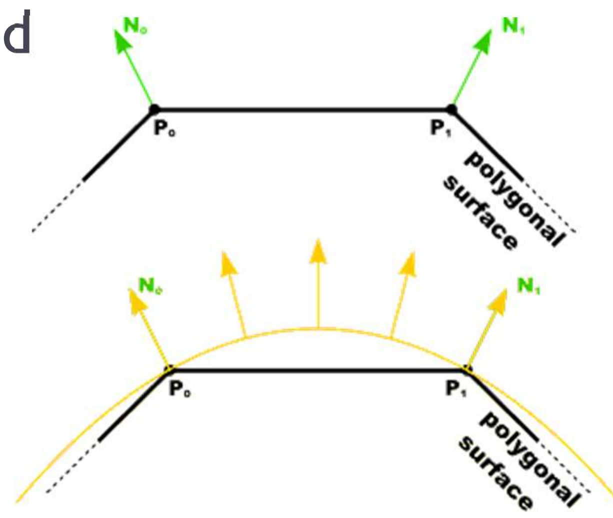




UCSD

Per-Pixel Shading

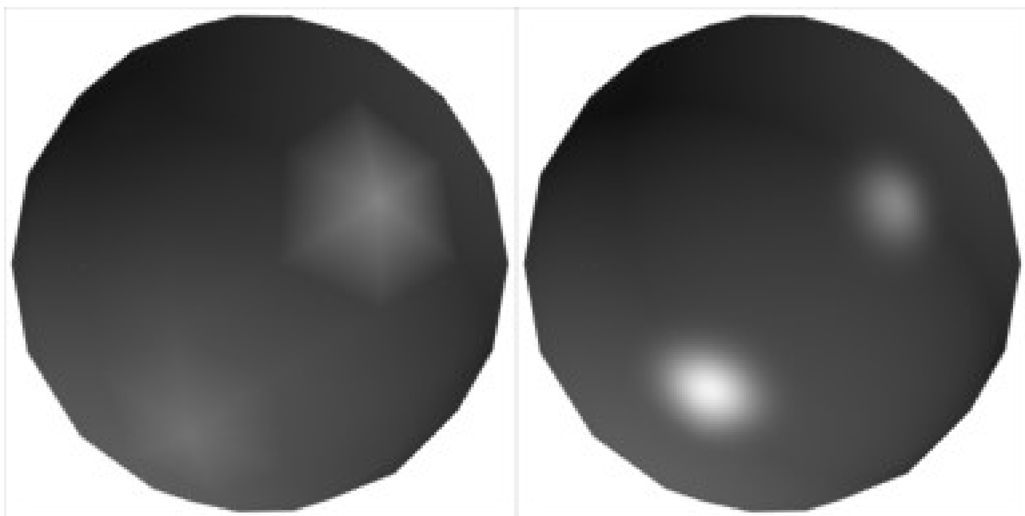
- A.k.a. *Phong Interpolation* (not to be confused with *Phong Illumination Model*)
  - Rasterizer interpolates <u>normals</u> (instead of colors) across triangles
  - Illumination model is evaluated at each pixel
  - Simulates shading with normals of a curved surface
- Advantage
  - Highest rendering quality
- Disadvantage
  - Slow



Source: Penny Rheingans, UMBC

Gouraud vs. Per-Pixel Shading

- Gouraud shading has problems with highlights when polygons are large
- More triangles improve the result, but reduce frame rate
- Video: https://www.youtube.com/watch?v=FI5i-UnlQps&feature=youtu.be



Gouraud   Per-Pixel

UCSD

Summary

- Per-pixel shading looks best and is only slightly more computationally expensive
- On slower GPUs Gouraud shading may make sense (e.g., in OpenGL ES on older mobile devices)
- In CSE 167 we always use per-pixel shading

Lights

Light Sources

- Real light sources can have complex properties
  - Geometric area over which light is produced
  - Anisotropy (directionally dependent)
  - Reflective surfaces act as light sources (indirect light)

- Need to use simplified model for real-time rendering

Types of Light Sources

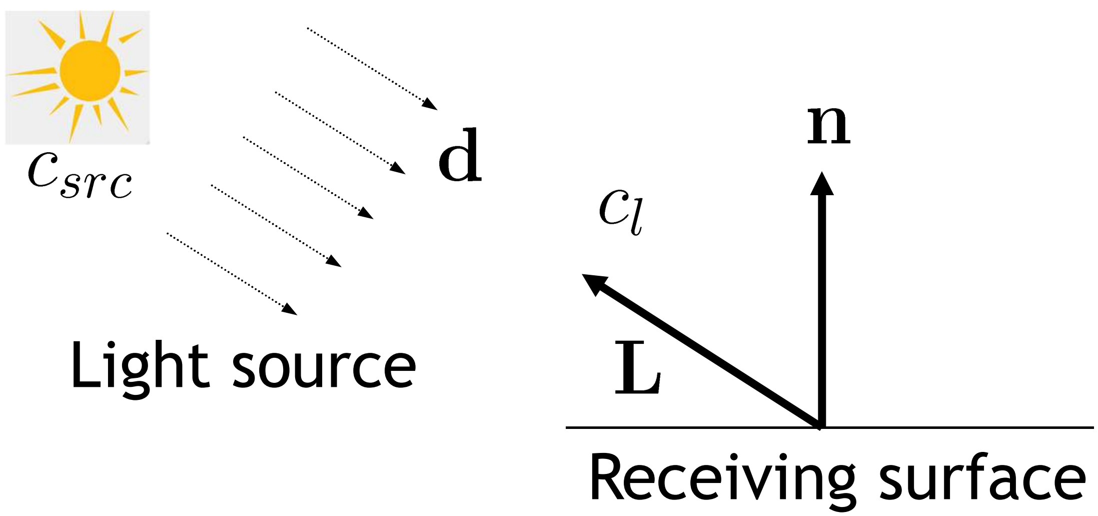
- At each point on surfaces we need to know
  - Direction of incoming light (the $\mathbf{L}$ vector)
  - Intensity of incoming light (the c_l values)
- Three light types:
  - Directional: from a specific direction
  - Point light: from a specific point
  - Spotlight: from a specific point with intensity that depends on direction

Lecture Overview

- Light Sources
  - Directional Lights
  - Point Lights
  - Spot Lights

Directional Light

- Light from a distant source
  - Light rays are parallel
  - Direction and intensity are the same everywhere
  - As if the source were infinitely far away
  - Good approximation of sunlight
- Specified by a unit length direction vector, and a color



$$\mathbf{L} = -\mathbf{d}$$

$$c_l = c_{src}$$

Lecture Overview

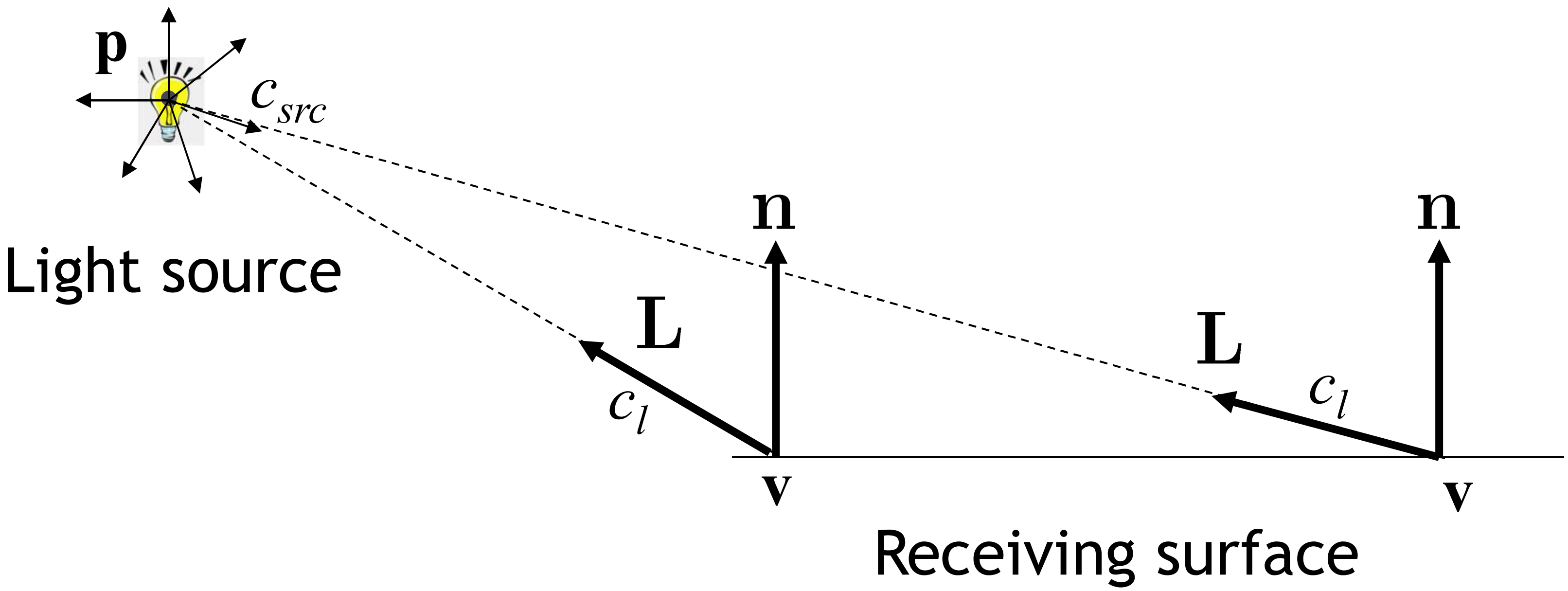
- Light Sources
  - Directional Lights
  - Point Lights
  - Spot Lights

Point Lights

- Similar to light bulbs
- Infinitely small point radiates light equally in all directions
  - Light vector varies across receiving surface
  - What is light intensity over distance proportional to?
  - Intensity drops off proportionally to the inverse square of the distance from the light
    - Reason for inverse square falloff: Surface area A of sphere:
      $A = 4 \pi r^2$

UCSD

Point Light Math



At any point v on the surface:

$$\mathbf{L} = \frac{\mathbf{p} - \mathbf{v}}{\|\mathbf{p} - \mathbf{v}\|}$$

Attenuation: $$c_l = \frac{c_{src}}{\|\mathbf{p} - \mathbf{v}\|^2}$$

Light Attenuation

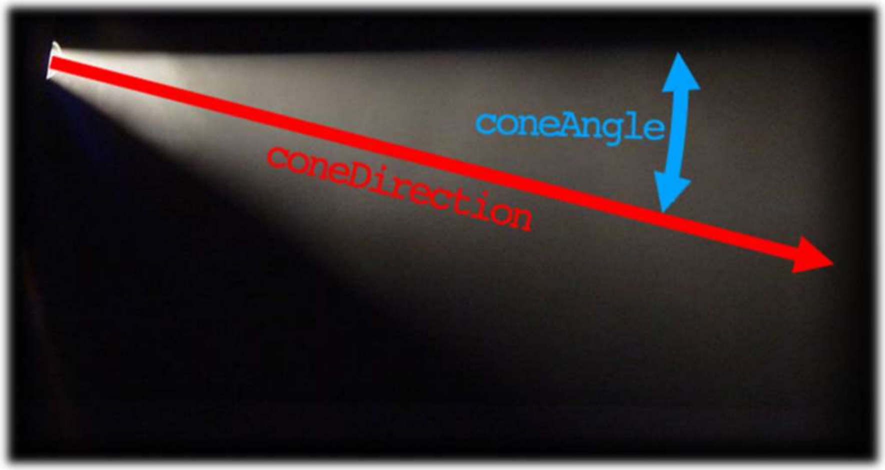
- Adding constant factor k to denominator for better control
- Quadratic attenuation: $k*(p-v)^2$
  - Most computationally expensive, most physically correct
- Linear attenuation: $k*(p-v)$
  - Less expensive, less accurate
- Constant attenuation: k
  - Fastest computation, least accurate

Lecture Overview

- Light Sources
  - Directional Lights
  - Point Lights
  - Spot Lights

Spotlights

- Like point light, but intensity depends on direction

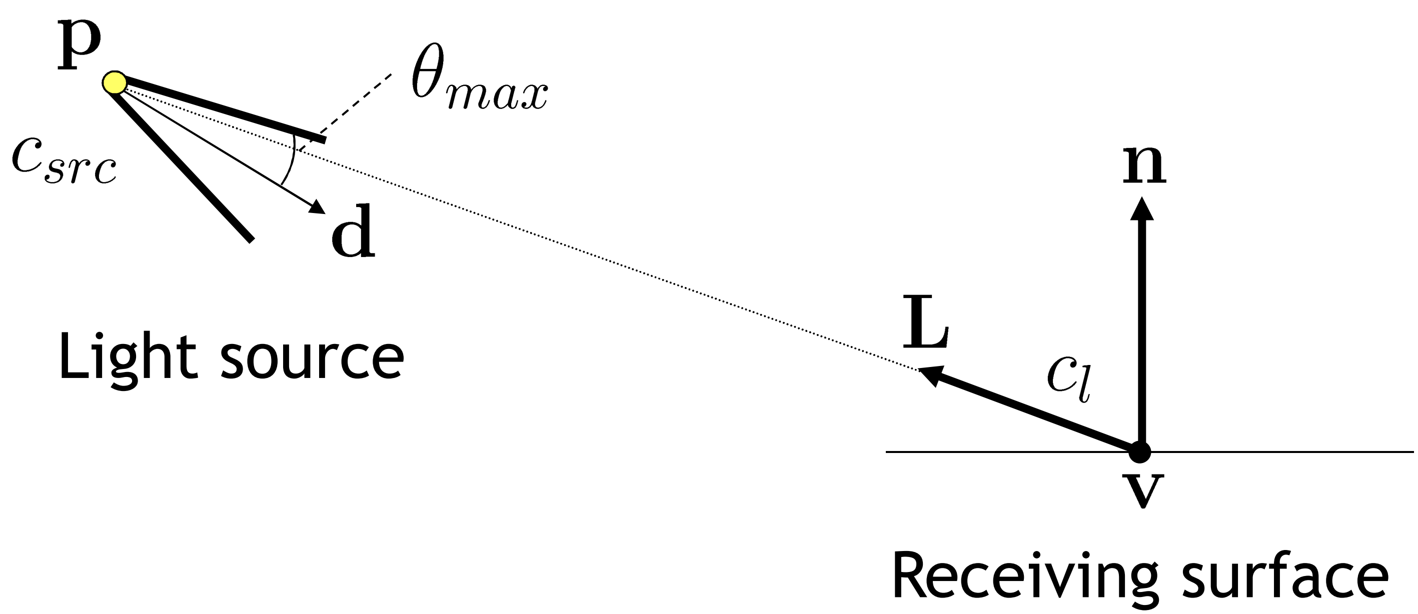


Parameters

- Position: location of light source
- Cone direction *d*: center axis of light source
- Intensity falloff:
  - Beam width (cone angle θ_{max})
  - The way the light tapers off at the edges of the beam (cosine exponent *f*)

UCSD

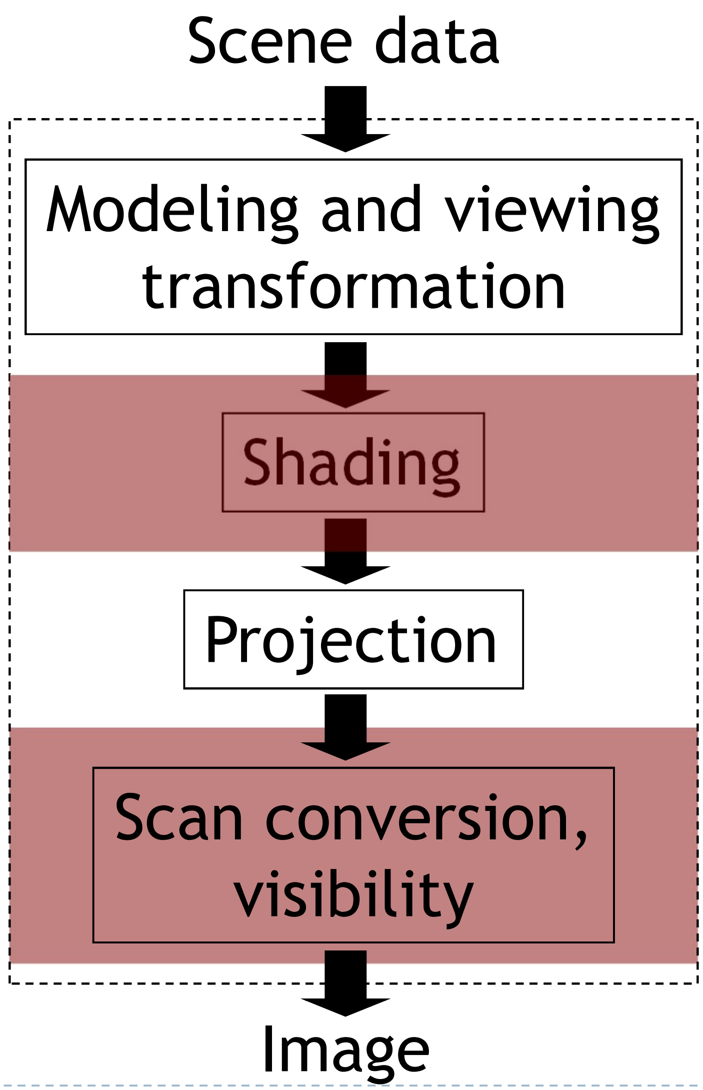
Spotlights



$$\mathbf{L} = \frac{\mathbf{p} - \mathbf{v}}{\|\mathbf{p} - \mathbf{v}\|}$$

$$c_l = \begin{cases} 0 & \text{if} \quad -\mathbf{L} \cdot \mathbf{d} \leq \cos(\theta_{max}) \\ c_{src}\,(-\mathbf{L} \cdot \mathbf{d})^f & \text{otherwise} \end{cases}$$

Rendering Pipeline



- Place object in 3D space
- Determine colors of vertices
  - Per vertex shading
- Map triangles to 2D
- Draw triangles
  - Per pixel shading

UCSD

Vertex Shader

```
#version 150

uniform mat4 camera;
uniform mat4 model;

in vec3 vert;
in vec2 vertTexCoord;
in vec3 vertNormal;

out vec3 fragVert;
out vec2 fragTexCoord;
out vec3 fragNormal;

void main()
{
    // Pass some variables to the fragment shader
    fragTexCoord = vertTexCoord;
    fragNormal = vertNormal;
    fragVert = vert;

    // Apply all matrix transformations to vert
    gl_Position = camera * model * vec4(vert, 1);
}
```


Source: Tom Dallling's OpenGL Tutorial

Fragment Shader for Diffuse Reflection

```
#version 150

uniform mat4 model;
uniform sampler2D tex;

uniform struct Light
{
   vec4 position; // if w component=0 it's directional
   vec3 intensities; // a.k.a the color of the light
   float attenuation; // only needed for point and spotlights
   float ambientCoefficient;
   float coneAngle;  // only needed for spotlights
   vec3 coneDirection; // only needed for spotlights
   float exponent;  // cosine exponent for how light tapers off
} light;

in vec2 fragTexCoord;
in vec3 fragNormal;
in vec3 fragVert;

out vec4 finalColor;
```

Source: Tom Dallling's OpenGL Tutorial

Fragment Shader Part 2

```
void main()
{
    // calculate normal in world coordinates
    mat3 normalMatrix = transpose(inverse(mat3(model)));
    vec3 normal = normalize(normalMatrix * fragNormal);

    // calculate the location of this fragment (pixel) in world coordinates
    vec3 fragPosition = vec3(model * vec4(fragVert, 1));

    // calculate the vector from this pixels surface to the light source
    vec3 surfaceToLight = light.position - fragPosition;

    // calculate the cosine of the angle of incidence
    float brightness = dot(normal, surfaceToLight) / (length(surfaceToLight) * length(normal));
    brightness = clamp(brightness, 0, 1);

    // calculate final color of the pixel, based on:
    // 1. The angle of incidence: brightness
    // 2. The color/intensities of the light: light.intensities
    // 3. The texture and texture coord: texture(tex, fragTexCoord)
    vec4 surfaceColor = texture(tex, fragTexCoord);
    finalColor = vec4(brightness * light.intensities * surfaceColor.rgb, surfaceColor.a);
}
```


Source: Tom Dallling's OpenGL Tutorial