UNIVERSITÉ Concordia UNIVERSITY
FACULTY OF ENGINEERING AND COMPUTER SCIENCE
Department of Computer Science and Software Engineering

# COMP371
# COMPUTER GRAPHICS

## LECTURE 10
## LIGHTING AND SHADING

# Lecture Overview

- Review of last class
- Global and Local Illumination
- Normal Vectors
- Light Sources
- Phong Illumination Model

# Global Illumination

- Ray tracing
- Radiosity
- Photon Mapping
- Follow light rays through a scene
- Accurate, but expensive (off-line)

# Raytracing example

Martin Moeck,
Siemens Lighting

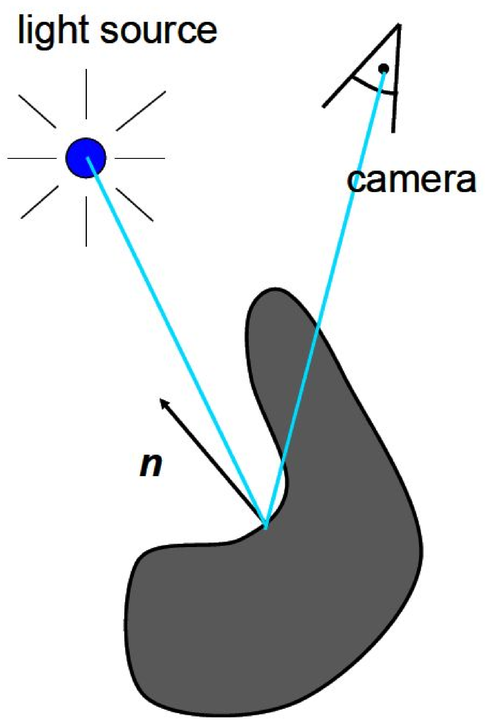
# Radiosity Example

Restaurant Interior.

Guillermo Leal, Evolucion Visual

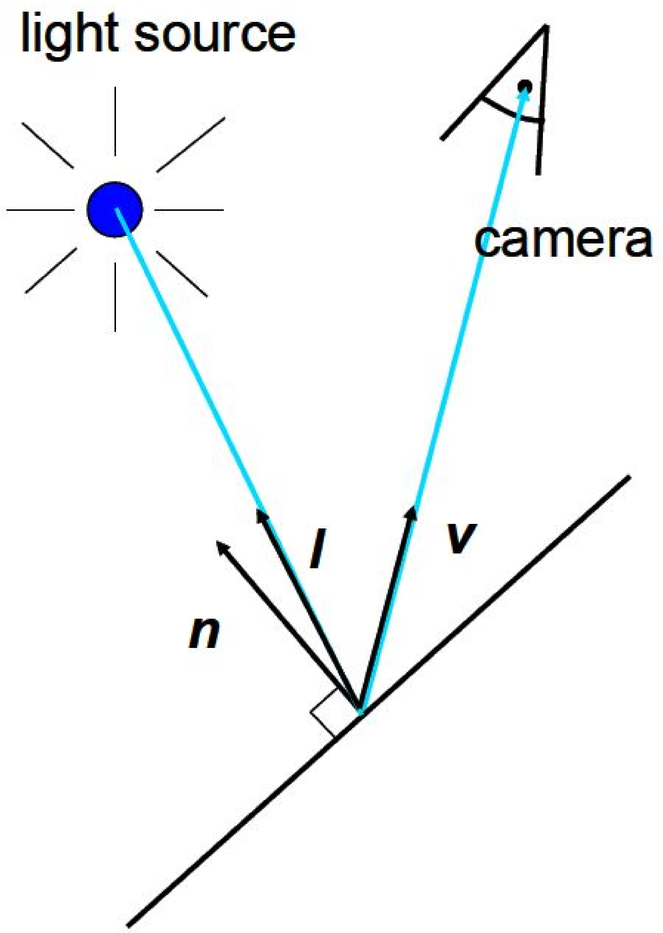
# Local Illumination

- Approximate model
- Local interaction between light, surface, viewer
- Phong model (this lecture):
  - fast, supported in OpenGL
- GPU shaders



# Local Illumination cont'd

- Color determined only
  based on surface normal,
  relative camera position
  and relative light position
- What effects does this ignore?



# Normal Vectors

- Must calculate and specify the normal vector
  - Even in OpenGL!

- Example: plane

# Method 1: Normal of a plane

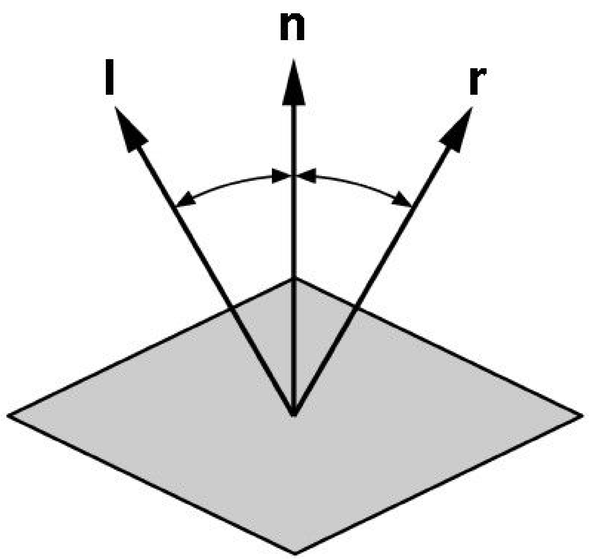
- Method I: given by $f(p) = ax + by + cz + d = 0$
- Let $p_0$ be a known point on the plane
- Let p be an arbitrary point on the plane
- Recall: $u \bullet v = 0$ if and only if u orthogonal to v
- $n \bullet (p - p_0) = n \bullet p - n \bullet p_0 = 0$
- Consequently $n_0 = [a\ b\ c]^T$
- Normalize to $n = n_0/|n_0|$

# Method 2: Normal of a plane

- Method II: plane given by $p_0$, $p_1$, $p_2$
  - Points must not be collinear → cross product defined as 0
  - Recall: u **x** v orthogonal to u and v
  - $n_0 = (p_1 – p_0)$ x $(p_2 – p_0)$
  - Order of cross product determines orientation
  - Normalize to $n = n_0/|n_0|$

# Reflected Vector

- Perfect reflection: angle of incidence equals angle of reflection
- Also: *l, n, and r lie in the same plane* →
- Assume |l| = |n| = 1, guarantee |r| = 1




$l.n = \cos\theta = n.r$

$r = al + bn$

Can write r as a linear combination of l and n

(1) $n.r = al.n + b = l.n$

(2) $1 = r.r = a^2+2abl.n+b^2$

Solution: $a = -1$ and $b = 2(l-n)$ → $\mathbf{r = 2(l.n)n-l}$

# Light Sources and Material Properties

- Appearance depends on
  - Light sources, their locations and properties
  - Material (surface) properties
  - Viewer position

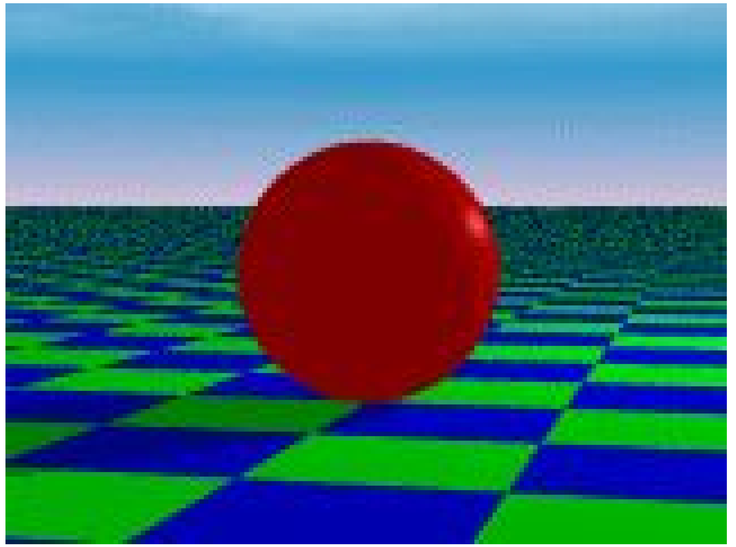
# Types of Light Sources

- Ambient light: no identifiable source or direction
- Point source: given only by point
- Directional light: given only by direction
- Spotlight: from source in direction
  - Cut-off angle defines a cone of light
  - Attenuation function (brighter in center)

# Point Source

- Given by a point $p_0$
- Light emitted equally in all directions
- Intensity decreases with square of distance

$$I \propto \frac{1}{|p - p_0|^2}$$

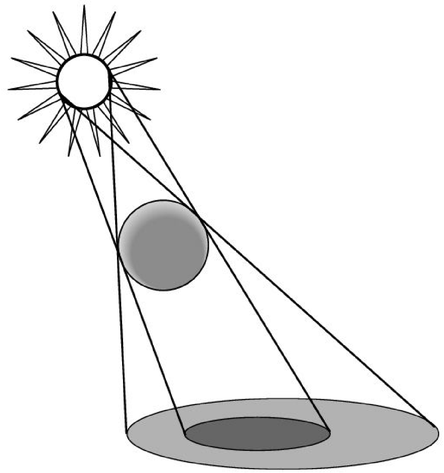


# Limitations of Point Sources

- Shading and shadows inaccurate
- Example: penumbra (partial “soft” shadow)
- Similar problems with highlights
- Compensate with attenuation
- Softens lighting
- Better with ray tracing
- Better with radiosity

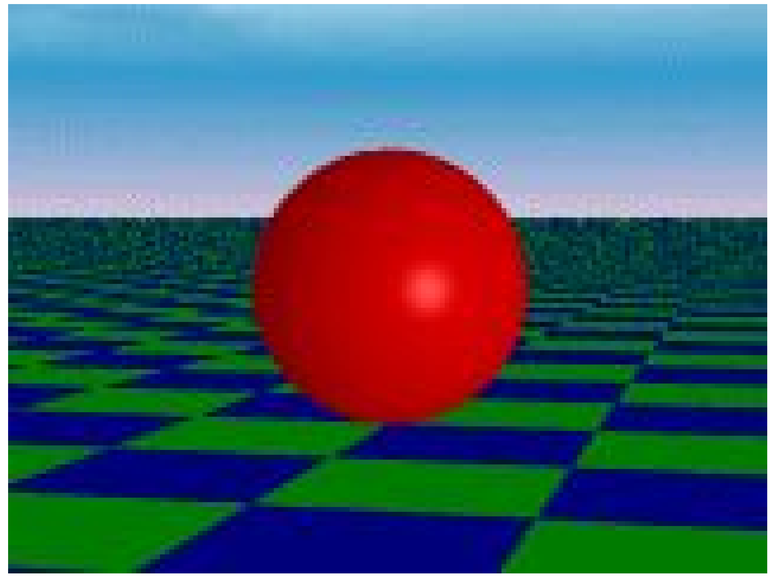
$$\frac{1}{a + bq + cq^2}$$

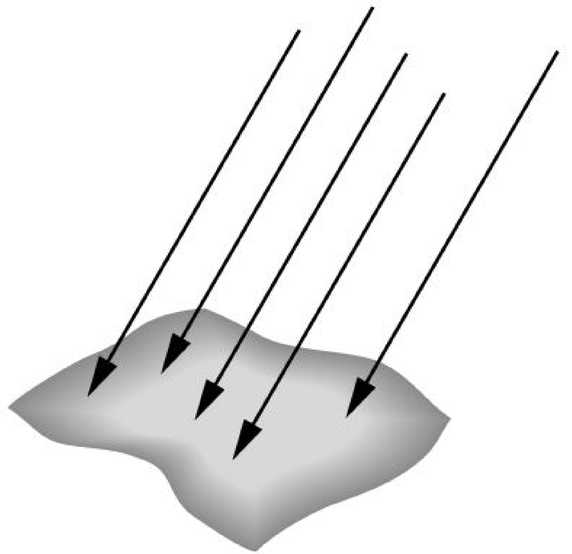
q = distance |p – p0|
a, b, c constants



# Directional Light Source

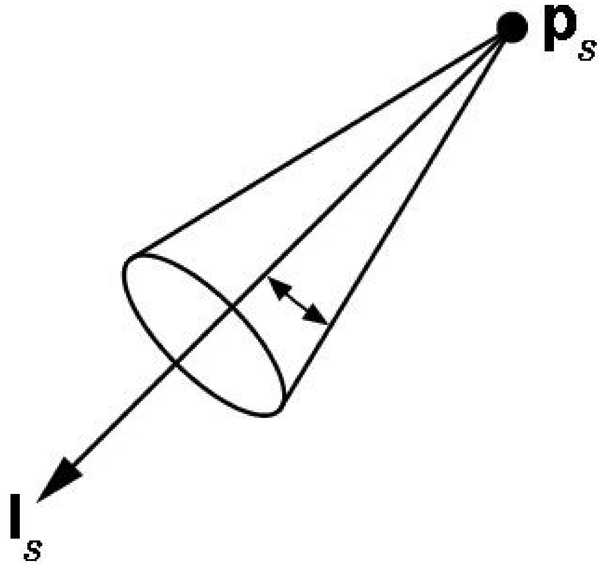
- Given by a direction vector
- Simplifies some calculations
- In OpenGL:
- Point source $[x\ y\ z\ 1]^T$
- Directional source $[x\ y\ z\ 0]^T$





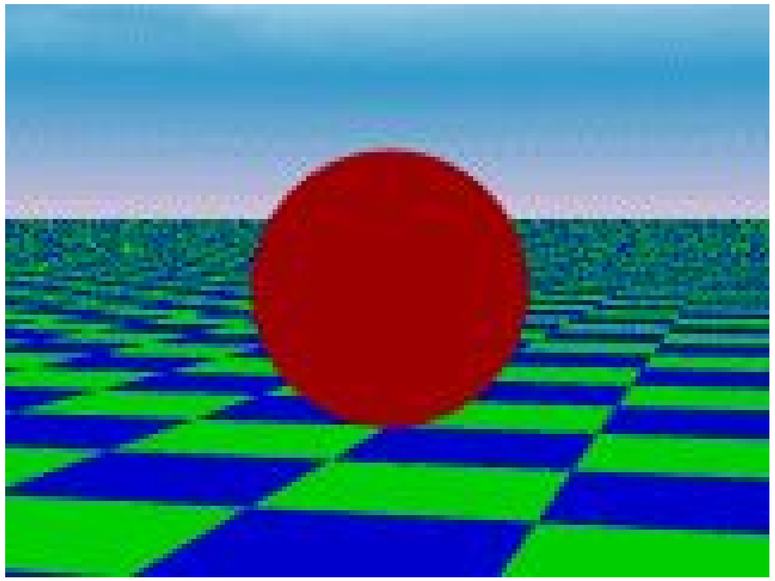
# Spotlight

- Most complex light source in OpenGL
- Light still emanates from point
- Cut-off by cone determined by angle θ



# Global Ambient Light

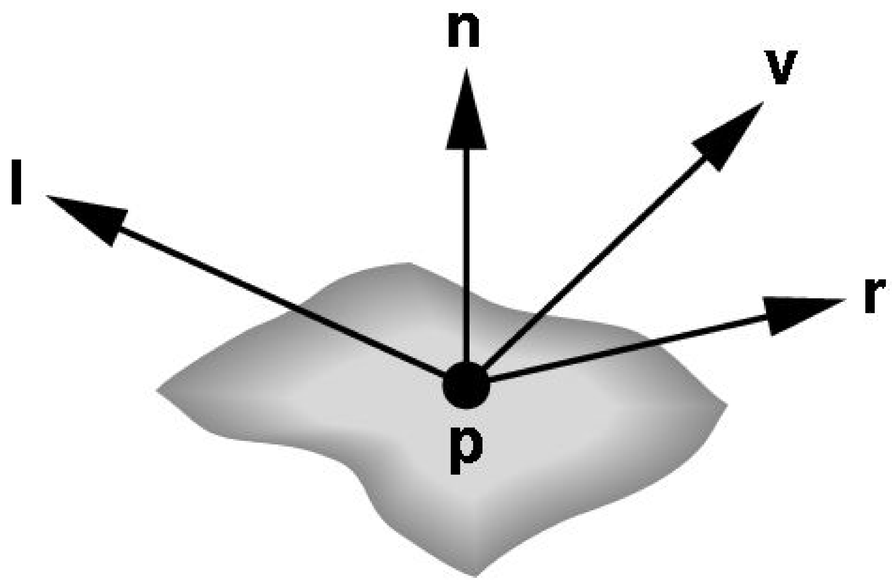
- Independent of light source
- Lights entire scene
- Computationally inexpensive
- Simply add [$G_R$ $G_G$ $G_B$] to every pixel on every object
- Not very interesting on its own.
  - A cheap hack to make the scene brighter.



# Phong Illumination Model

- Calculate color for arbitrary point on surface
- Compromise between realism and efficiency
- Local computation (no visibility calculations)
- Basic inputs are material properties and l, n, v:

l = unit vector to light source
n = surface normal
v = unit vector to viewer
r = reflection of l at p
(determined by l and n)



# Phong Illumination Overview

1. Start with global ambient light [$G_R$ $G_G$ $G_B$]
2. Add contributions from each light source
3. Clamp the final result to [0, 1]

- Calculate each color channel (R,G,B) separately
- Light source contributions decomposed into
  - Ambient reflection
  - Diffuse reflection
  - Specular reflection
- Based on ambient, diffuse, and specular *lighting and material* properties

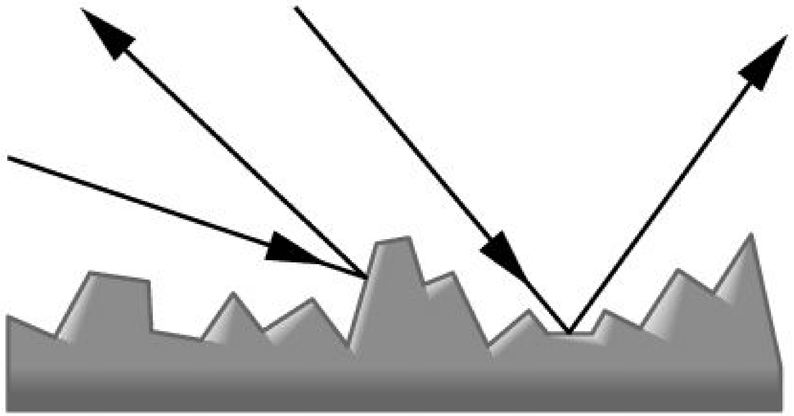
# Ambient Reflection

$I_a = k_a L_a$

- Intensity of ambient light is uniform at every point
- Ambient reflection coefficient $k_a$, $0 \leq k_a \leq 1$
- May be different for every surface and r,g,b
- Determines reflected fraction of ambient light
- $L_a$ = ambient component of light source (can be set to different value for each light source)
- Note: $L_a$ is not a physically meaningful quantity

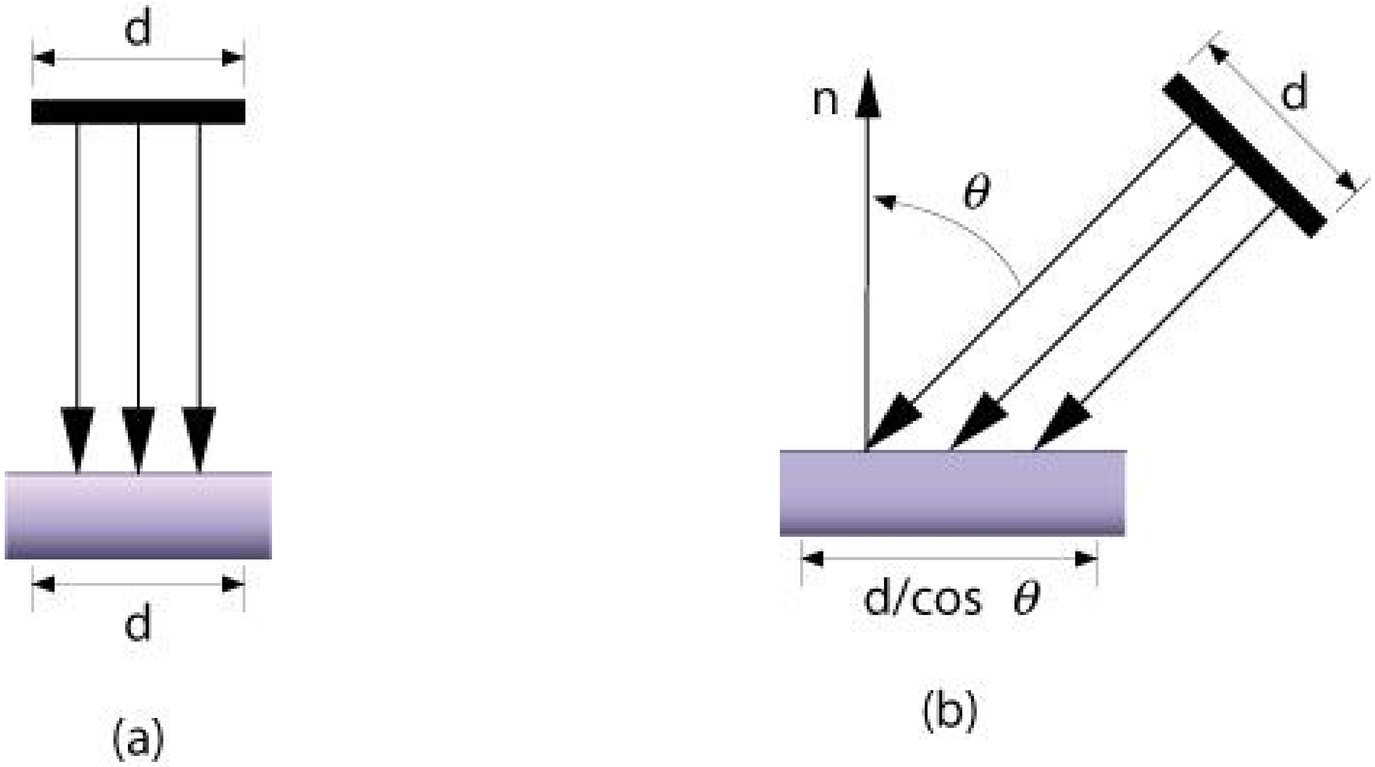
# Diffuse Reflection

- Diffuse reflector scatters light equally to all directions
- Called *Lambertian* surface
- Diffuse reflection coefficient $k_d$, $0 \leq k_d \leq 1$
- Only the angle of incoming light is important



# Lambert's Law

Intensity depends on angle of incoming light



# Diffuse Light Intensity Depends on Angle of Incoming Light

- Recall

l = unit vector to light

n = unit surface normal

θ = angle to normal

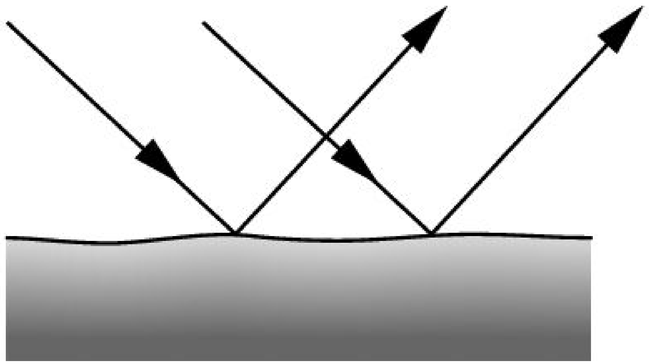
- $\cos \theta = l \cdot n$
- $I_d = k_d L_d (l \cdot n)$
- With attenuation: $$I_d = \frac{k_d L_d}{a + bq + cq^2}(l \cdot n)$$

q = distance to light source,

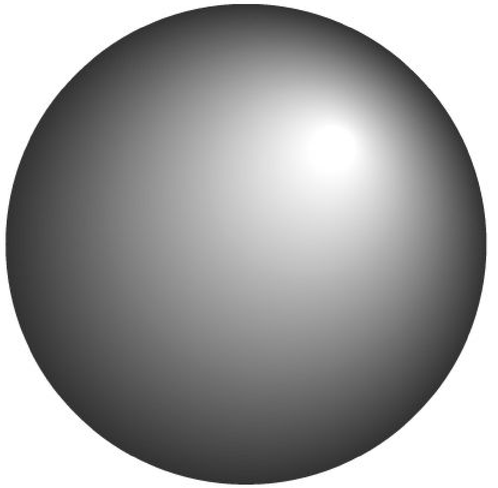
$L_d$ = diffuse component of light

# Specular Reflection

- Specular reflection coefficient $k_s$, $0 \leq k_s \leq 1$
- Shiny surfaces have high specular coefficient
- Used to model specular highlights
- Does not give mirror effect (need other techniques)



Specular reflection



Specular highlights

# Specular Reflection

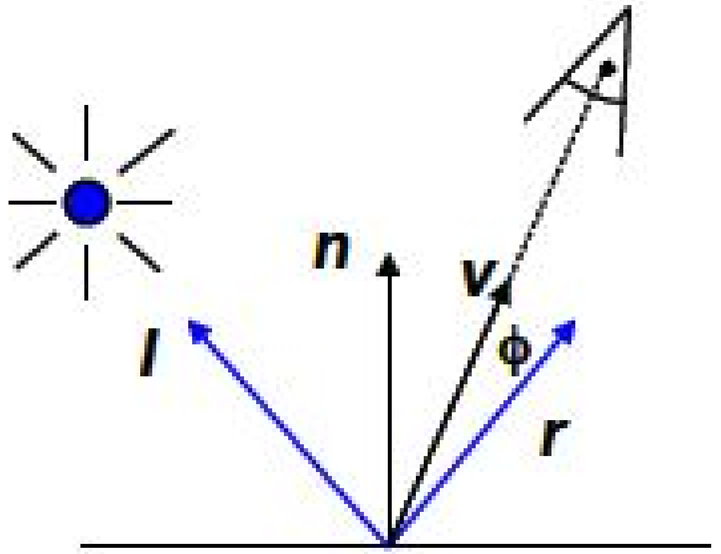
- Recall

  v = unit vector to camera

  r = unit reflected vector

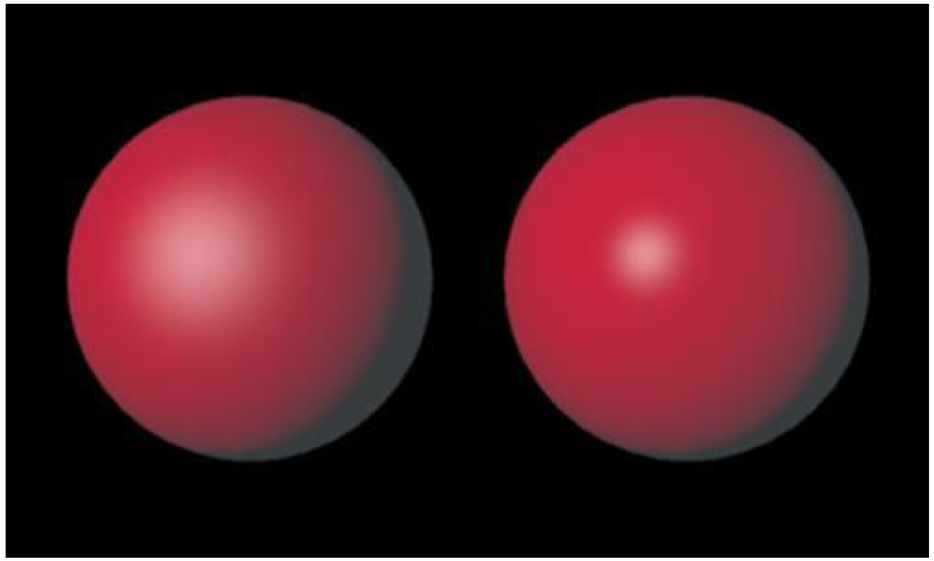
  φ = angle between v and r

- $\cos \varphi = v \bullet r$
- $I_s = k_s L_s (\cos \varphi)^{\alpha}$
- $L_s$ is specular component of light
- $\alpha$ is shininess coefficient
- Can add distance term as well

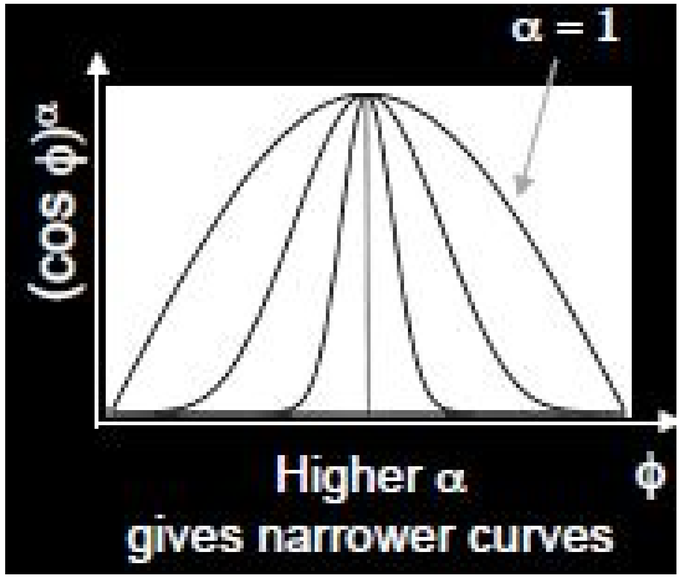


# Shininess Coefficient

- $I_s = k_s L_s (\cos \varphi)^{\alpha}$
  - α is the shininess coefficient



low α high α



# Summary of Phong Model

- Light components for each color:
  - Ambient ($L_a$), diffuse ($L_d$), specular ($L_s$)
- Material coefficients for each color:
  - Ambient ($k_a$), diffuse ($k_d$), specular ($k_s$)
- Distance q for surface point from light source

l = unit vector to light

n = surface normal

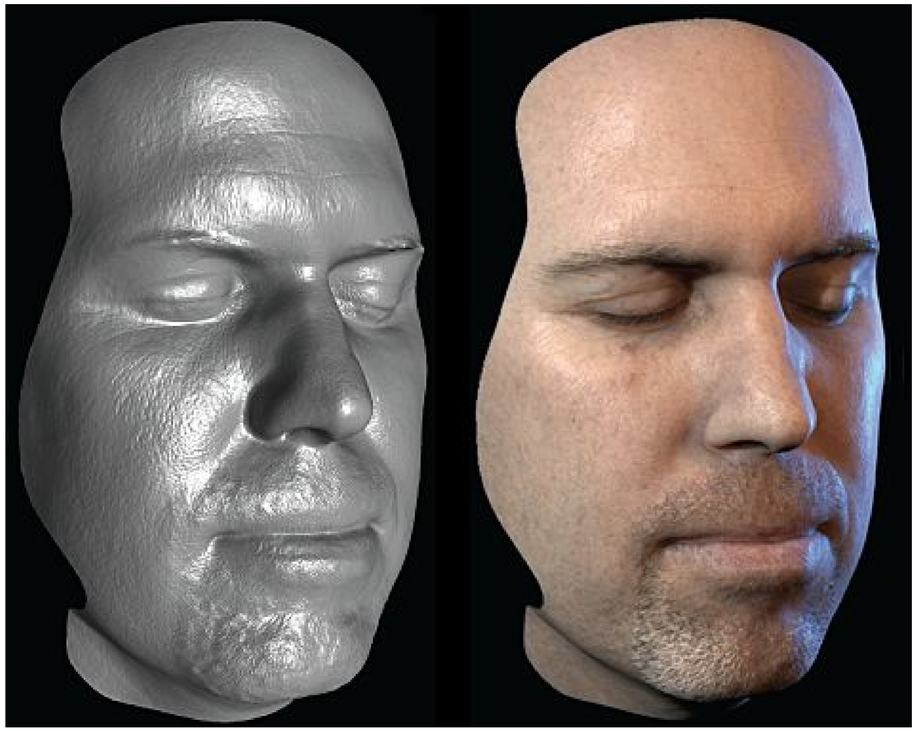
r = l reflected about n

v = vector to viewer

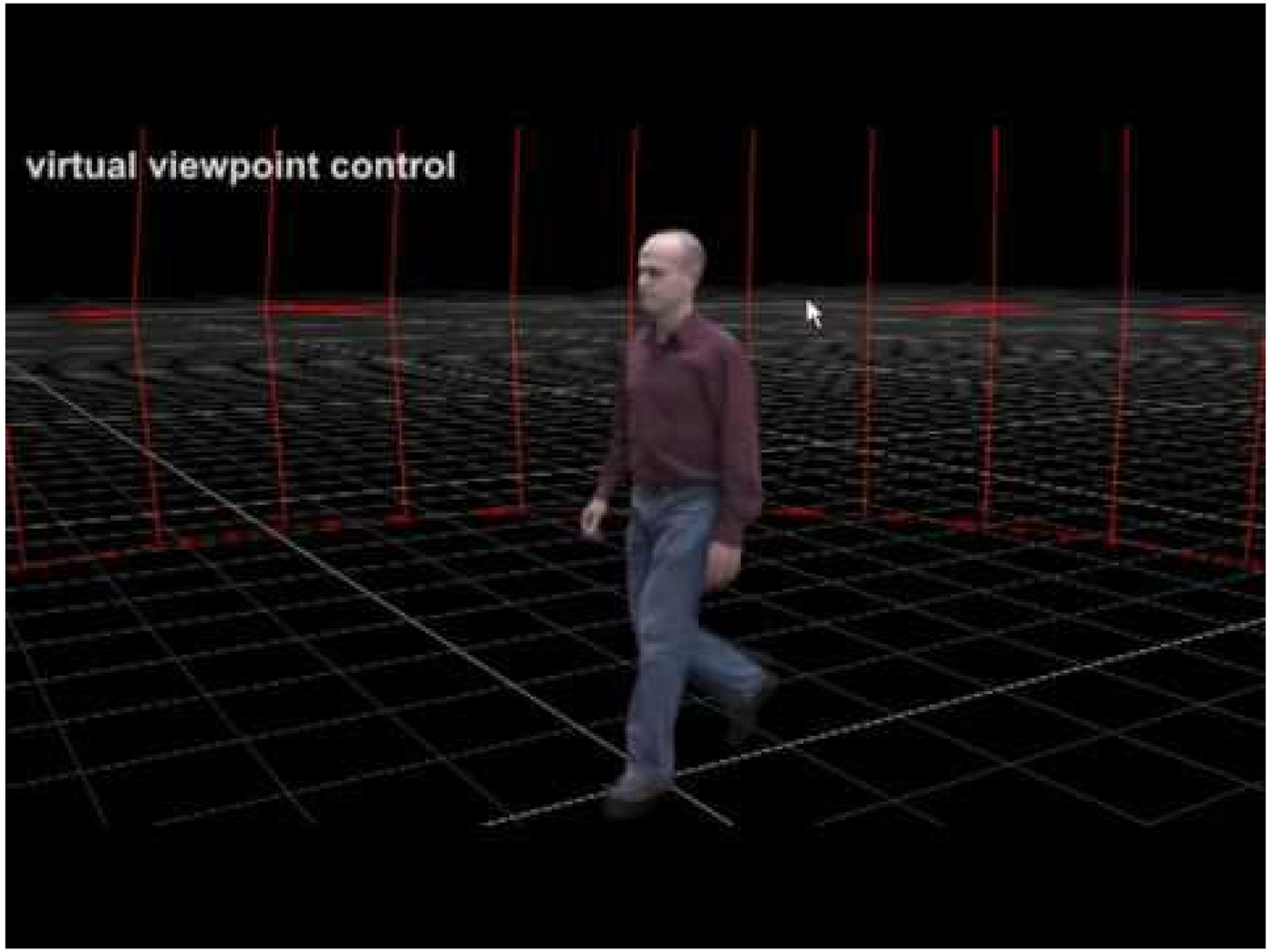
$$I = \frac{1}{a + bq + cq^2}\left(k_d L_d (l \cdot n) + k_s L_s (r \cdot v)^{\alpha}\right) + k_a L_a$$

# BRDF

- Bidirectional Reflection Distribution Function
- Must measure for real materials
- Isotropic vs. anisotropic
- Mathematically complex
- Programmable pixel shading



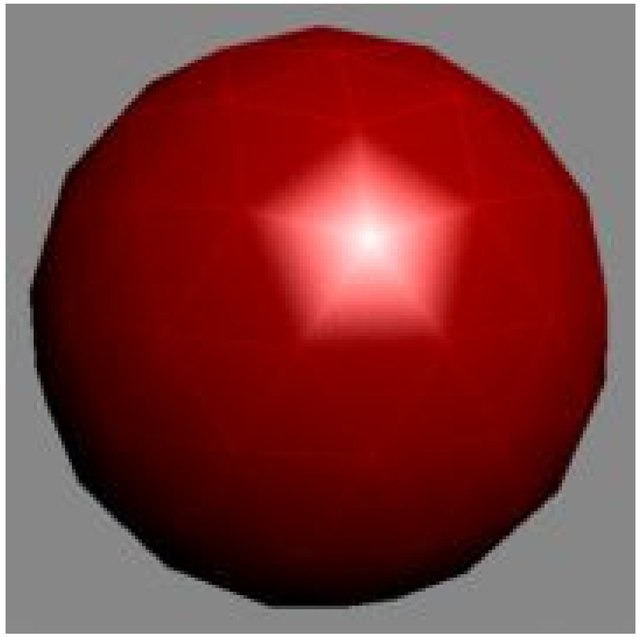
Lighting properties of a human face were captured and face re-rendered;
Institute for Creative Technologies



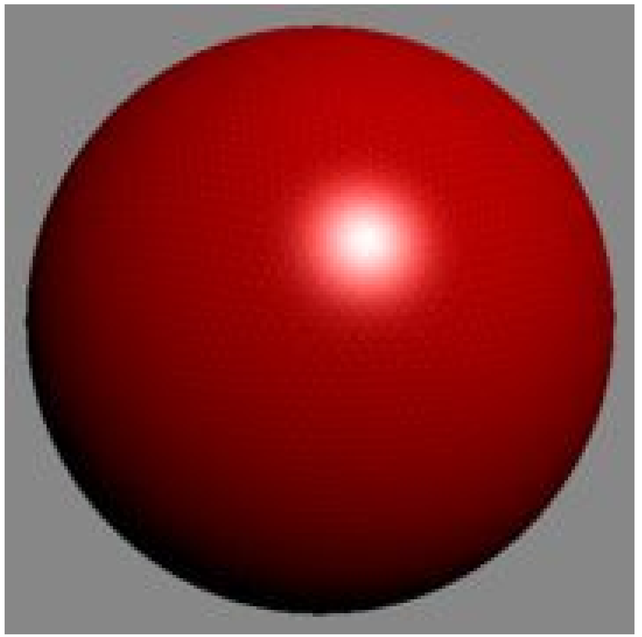
# Gouraud Shading

- “Fixed Pipeline” OpenGL implements Gouraud Shading
- Gouraud Shading is an implementation of the Phong Shading Model where Lighting is computed per-vertex
- The result is then interpolated on the whole Triangle
- Specular highlights are typically blurry, a high tessellation is required for good lighting
- It can be completely implemented in a Vertex Shader

# Gouraud Shading



**Low Poly Count**



**High Poly Count**

# Phong Shading

- Is typically implemented per Fragment (pixel)
- It provides “sharp” specular highlights even with low polygon count since lighting is calculated for each pixel individually.
- It is implemented on both the Vertex and Fragment Shader
- The Vertex Shader Transforms the Lighting Data to View Space
- The Fragment Shader calculates the Phong Shading Equation (Ambient + Diffuse + Specular)

# Phong Model Implementation with GLSL

- Light Properties and Material Properties are implemented as Shader Uniform Variables
- Be careful from which Coordinate System the Light Positions are sent to the Shaders
- Typically, lighting is computed in View Space
  - Normals must be converted
  - Eye, Light vectors must be converted

# Phong Shading in GLSL

**Vertex Shader**

- Transform vertex positions to Clip Space
- Transform lighting data to View Space

**Fragment Shader**

- Calculate and add ambient, diffuse and specular components

# Phong Shading in GLSL

- Vertex Shader
- Uniforms
  - Light position
  - Transformation Matrices
- Input
  - Vertex Data
    (pos, normal, color)
- Output
  - Normal (N)
  - Light Vector (L)
  - Eye Vector (V)
  - Vertex Color

- Fragment Shader
- Uniforms
  - Material Coefficients
    (ka, kd, ks, n)
  - Light Coefficients
    (Color, Attenuation const)
- Inputs in View Space
  - Normal (N)
  - Light Vector (L)
  - Eye Vector (V)
- Output
  - Fragment Color

# Phong Shading in GLSL Multiple Light Sources

- Use arrays for light parameters in your Shaders to support multiple light sources.

# Review

- Global and Local Illumination
- Normal Vectors
- Light Sources
- Phong Illumination Model
- Gouraud and Phong Shading

# Next Lecture

- OpenGL shading

CONCORDIA.CA

# Studying Support Slides