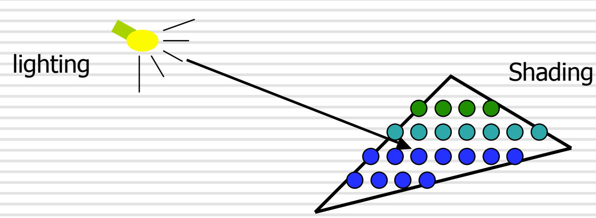
IMGD 1001 - The Game Development Process: Illumination

by
Robert W. Lindeman (gogo@wpi.edu)
Kent Quirk (kent_quirk@cognitoy.com)
(with lots of input from Mark Claypool!)

Illumination and Shading

- Problem: Model light/surface point interactions to determine final color and brightness
- Apply the lighting model at a set of points across the entire surface

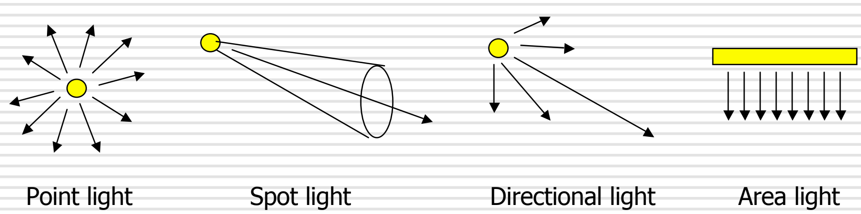


Illumination Model

- The governing principles for computing the illumination
- An illumination model usually considers
  - Light attributes (intensity, color, position, direction, shape)
  - Object surface attributes (color, reflectivity, transparency, *etc.*)
  - Interaction among lights and objects

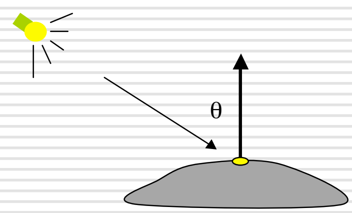
Basic Light Sources

- Light intensity can be independent or dependent of the distance between object and the light source



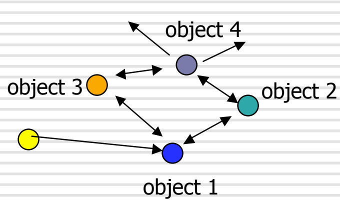
Local Illumination

- Only consider the light, the observer position, and the object material properties



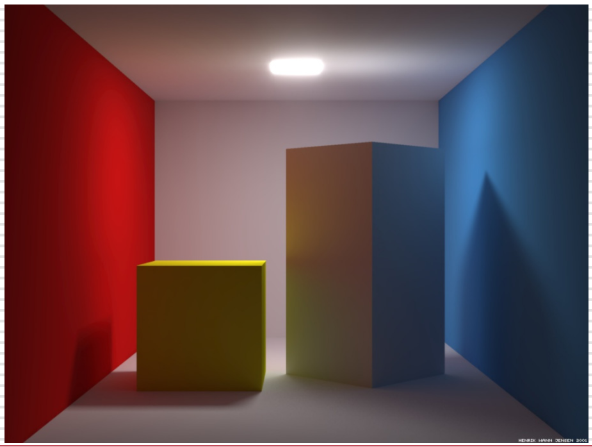
Global Illumination

- Take into account the interaction of light from all the surfaces in the scene
- Example:
  - Ray Tracing



Global Illumination (cont.)

- Radiosity: View independent

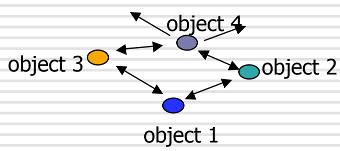


Simple Local Illumination

- Reduce the complex workings of light to three components
  - Ambient
  - Diffuse
  - Specular
- Final illumination at a point (vertex) = ambient + diffuse + specular
- Materials reflect each component differently
  - Use different material reflection coefficients
    - K_a, K_d, K_s

Ambient Light Contribution

- Ambient light = background light
- Light that is scattered by the environment
  - It's just there
- **Frequently assumed to be constant**
- Very simple approximation of global illumination
- No direction: independent of light position, object orientation, observer's position/orientation

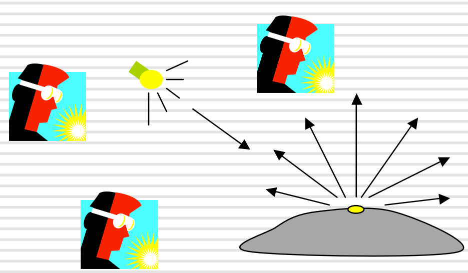


constant

Ambient = I x K_a

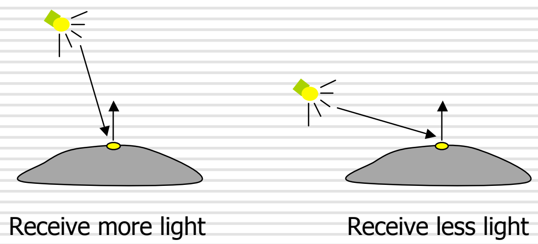
Diffuse Light Contribution

- Diffuse light: The illumination that a surface receives from a light source that reflects equally in all direction
  - Eye point does not matter



Diffuse Light Calculation

- Need to decide how much light the object point receives from the light source
  - Based on Lambert's Law



Diffuse Light Calculation (cont.)

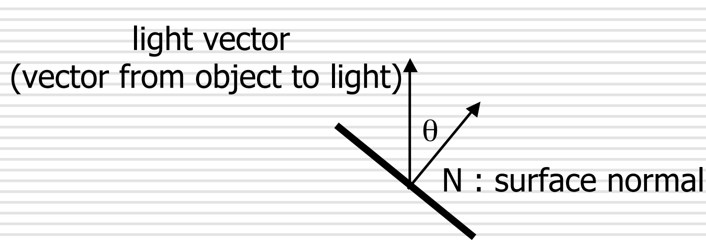
- Lambert's law: the radiant energy D that a small surface patch receives from a light source is:

$$\text{Diffuse} = K_d \times I \times \cos(\theta)$$

K_d: diffuse reflection coefficient

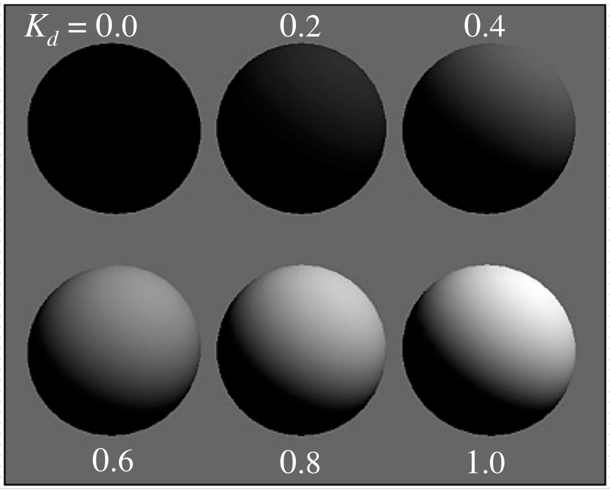
I: light intensity

θ: angle between the light vector and the surface normal



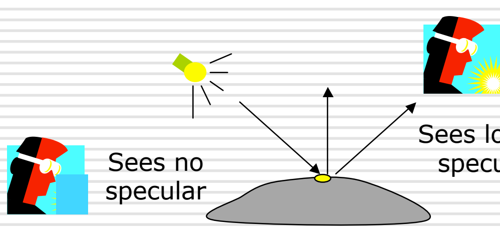
Diffuse Light Examples

I = 1.0



Specular Light Contribution

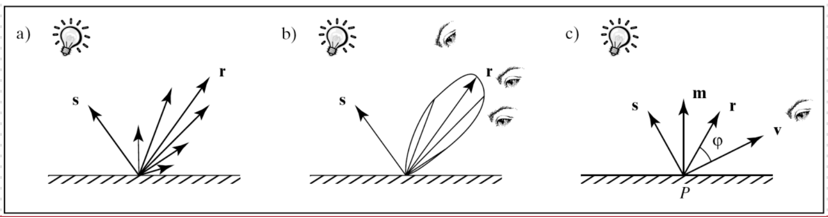
- The bright spot on the object
- The result of total reflection of the incident light in a concentrate region



Specular Light Calculation

- How much reflection you can see depends on where you are
  - But for non-perfect surface you will still see specular highlight when you move a little bit away from the ideal reflection direction

  Φ is deviation of view angle from mirror direction
  - When ϕ is small, you see more specular highlight



Specular Light Calculation (cont.)

- Phong lighting model
  - Not Phong *shading* model

$$\boxed{\text{Specular} = K_s \times I \times \cos^{f}(\phi)}$$

- The effect of 'f' in the Phong model

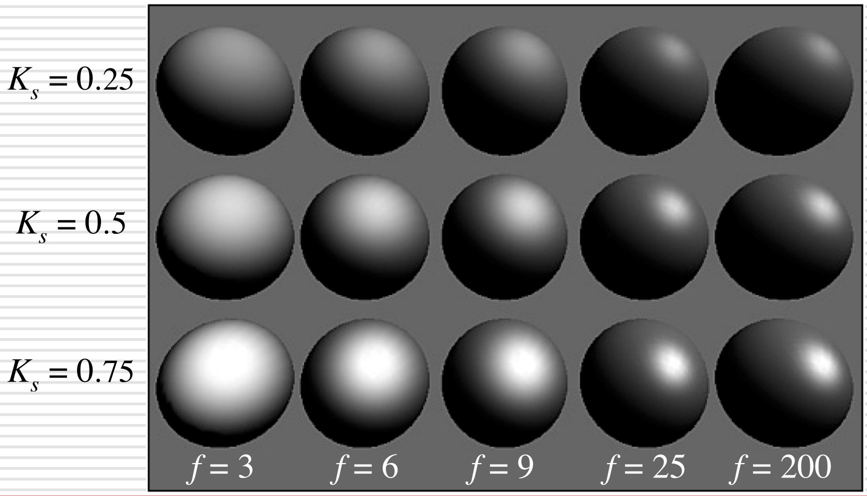
f = 10

f = 30

f = 90

f = 270

Specular Light Examples



Putting It All Together

- Illumination from a light

  Illum = ambient + diffuse + specular

  = K_a x I + K_d x I x cos(θ) + K_s x I x $\cos^f(\phi)$

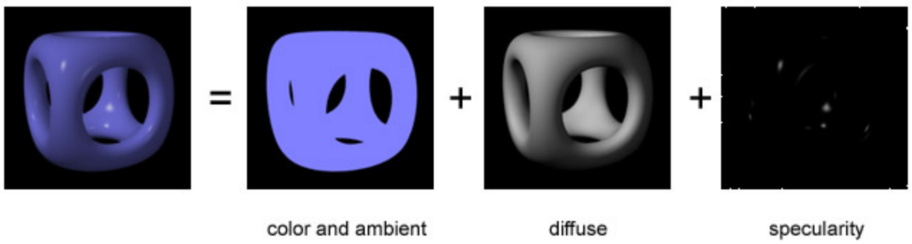
- If there are N lights

  Total illumination for a point P = Σ (Illum)

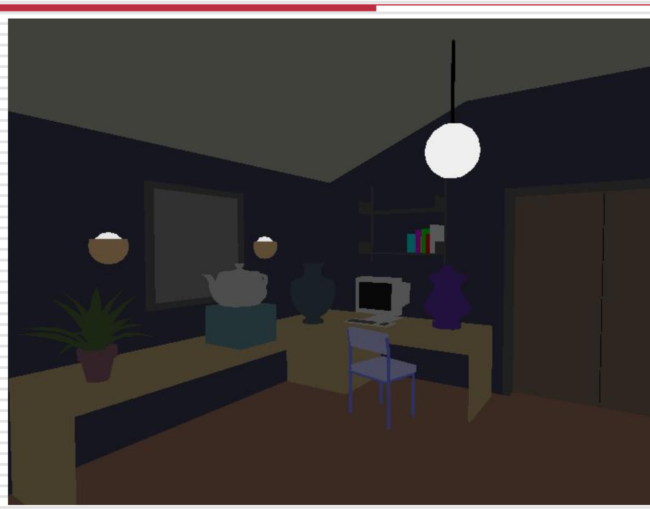
- Some more terms to be added
  - Self emission
  - Global ambient
  - Light distance attenuation and spot light effect

Putting It All Together (cont.)

- **Illum = ambient + diffuse + specular**



Ambient Lighting Example



Diffuse Lighting Example

Lindeman & Quirk (& Claypool) - WPI Dept. of Computer Science 21

Specular Lighting Example

Lindeman & Quirk (& Claypool) - WPI Dept. of Computer Science 22

Adding Color

- Sometimes light or surfaces are colored
- Treat R, G and B components separately
  - *i.e.*, can specify different RGB values for either light or material
- Illumination equation goes from

Illum = ambient + diffuse + specular

$$= K_a \times I + K_d \times I \times \cos(\theta) + K_s \times I \times \cos^f(\phi)$$

To:

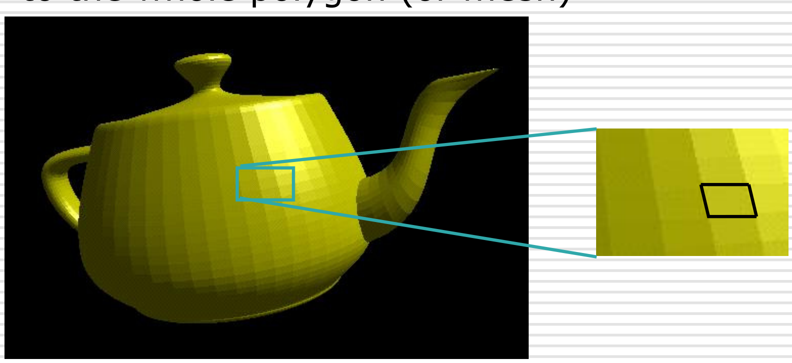
$$Illum_r = K_{ar} \times I_r + K_{dr} \times I_r \times \cos(\theta) + K_{sr} \times I_r \times \cos^f(\phi)$$

$$Illum_g = K_{ag} \times I_g + K_{dg} \times I_g \times \cos(\theta) + K_{sg} \times I_g \times \cos^f(\phi)$$

$$Illum_b = K_{ab} \times I_b + K_{db} \times I_b \times \cos(\theta) + K_{sb} \times I_b \times \cos^f(\phi)$$

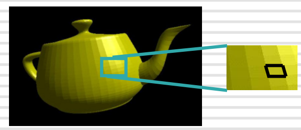
Polygon Shading Models

- Flat shading
  - Compute lighting once and assign the color to the whole polygon (or mesh)



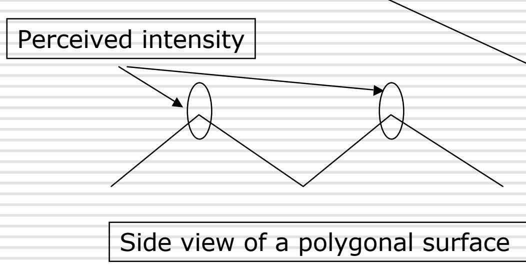
Flat Shading

- Only use one vertex normal and material property to compute the color for the polygon
- Benefit: fast to compute
- Used when
  - Polygon is small enough
  - Light source is far away (why?)
  - Eye is very far away (why?)



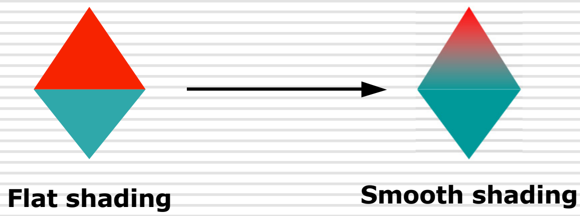
Mach-Band Effect

- Flat shading suffers from "mach banding"
  - Human eyes accentuate discontinuities at boundaries



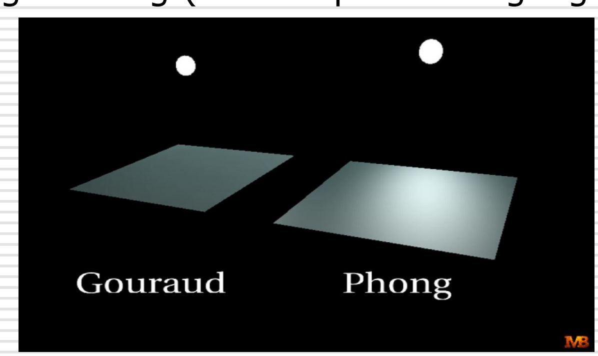
Smooth Shading

- Fix the mach banding
  - Remove edge discontinuities
- Compute lighting for more points on each face



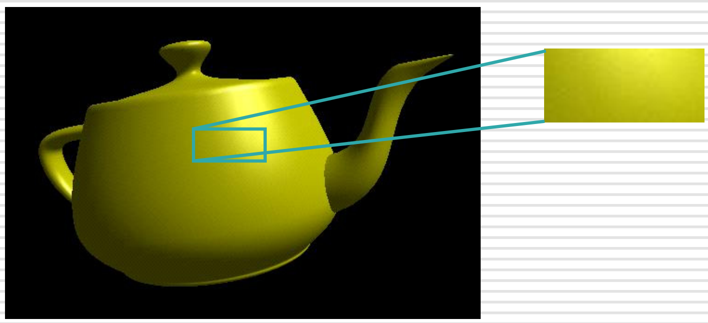
Smooth Shading (cont.)

- Two popular methods
  - Gouraud shading
  - Phong shading (better specular highlight)



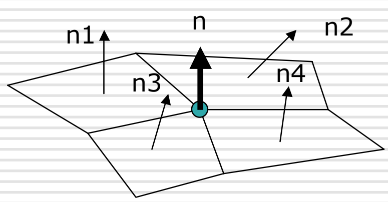
Gouraud Shading

- Lighting is calculated for each of the polygon vertices
- Colors are interpolated for interior pixels



Gouraud Shading (cont.)

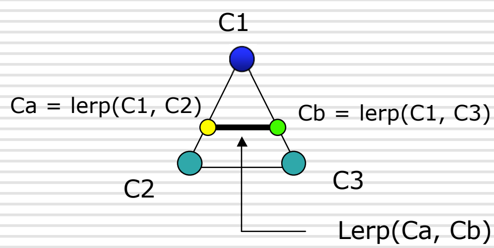
- Per-vertex lighting calculation
- Normal is needed for each vertex
- Per-vertex normal can be computed by averaging the adjacent face normals

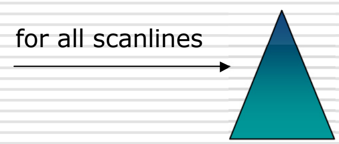


n = (n1 + n2 + n3 + n4) / 4.0

Gouraud Shading (cont.)

- Compute vertex illumination (color) before the projection transformation
- Shade interior pixels: color interpolation (normals are not needed)

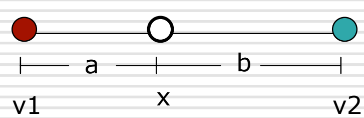




* lerp: linear interpolation

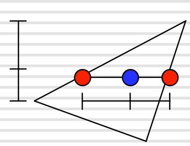
Gouraud Shading (cont.)

- Linear interpolation



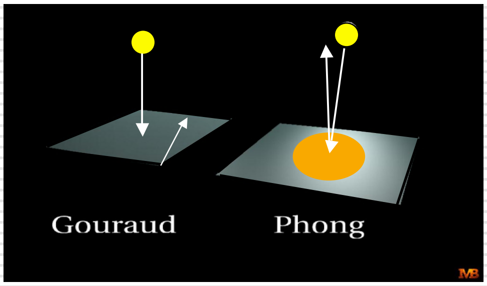
x = b / (a+b) * v1 + a/(a+b) * v2

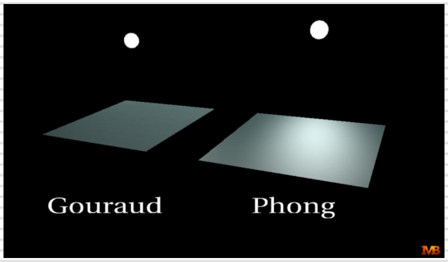
- Interpolate triangle color: use y distance to interpolate the two end points in the scanline, and use x distance to interpolate interior pixel colors



Gouraud Shading Problem

- Lighting in the polygon interior can be inaccurate



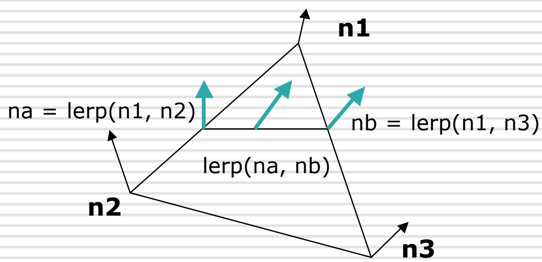


Phong Shading

- Instead of interpolation, we calculate lighting for each pixel inside the polygon (per-pixel lighting)
- Need normals for all the pixels
  - Not provided by user!
- Phong shading algorithm
  - Interpolate the normals across polygon
  - Compute lighting during rasterization
    - Need to map the normal back to world or eye space though

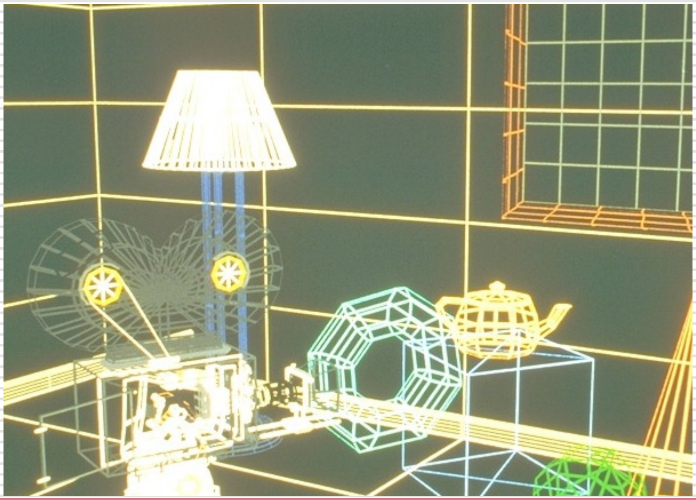
Phong Shading (cont.)

- Normal interpolation

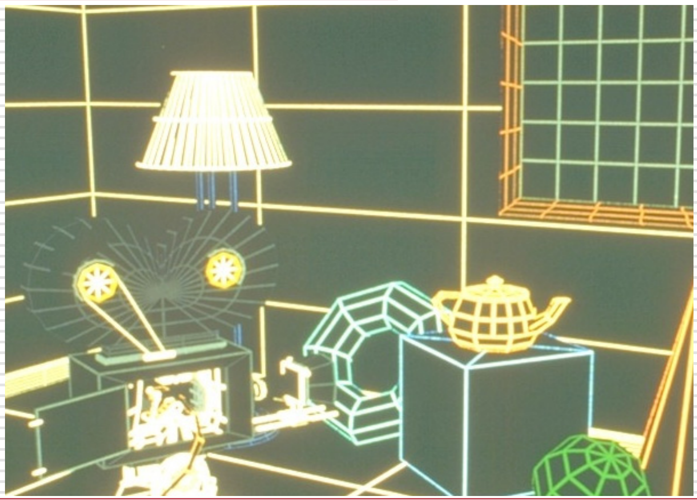


- Slow
  - Not supported by OpenGL and most graphics hardware

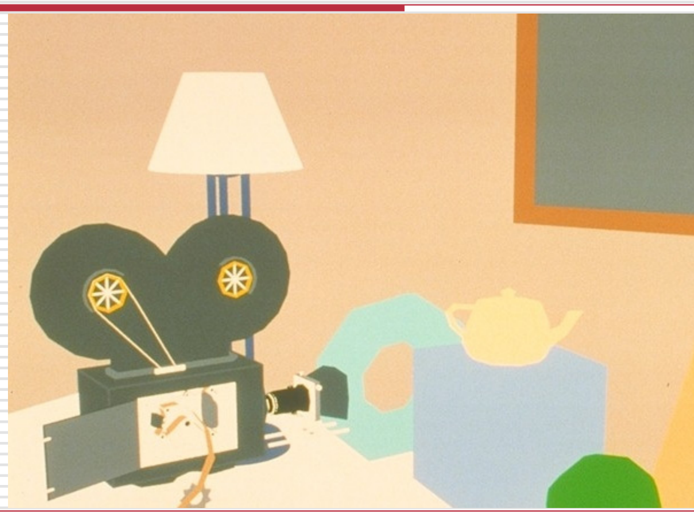
Colored Wireframe



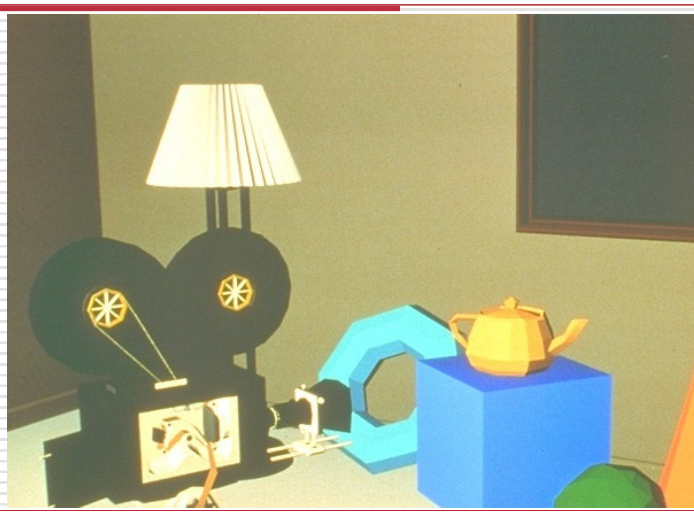
Colored Hidden-Line Removal



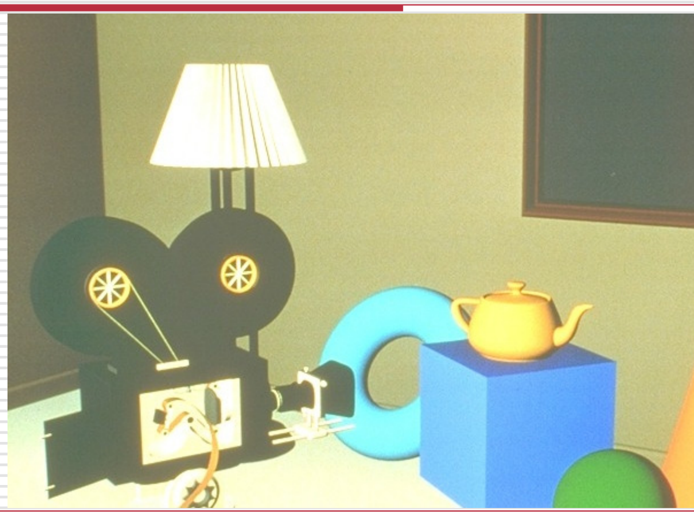
Ambient Term Only



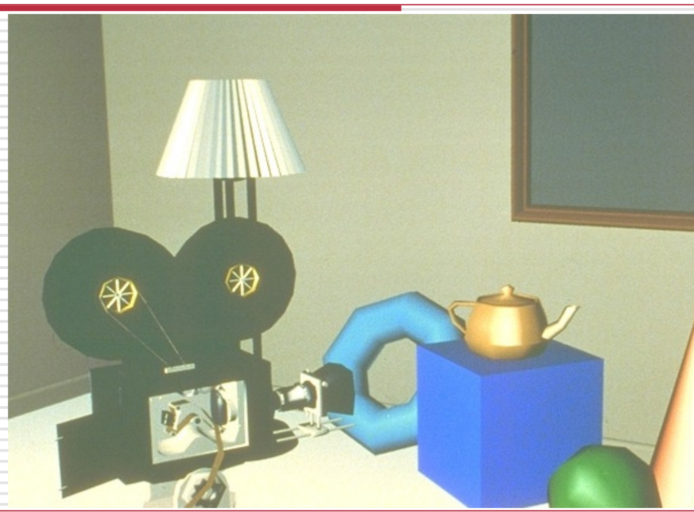
Flat Shading



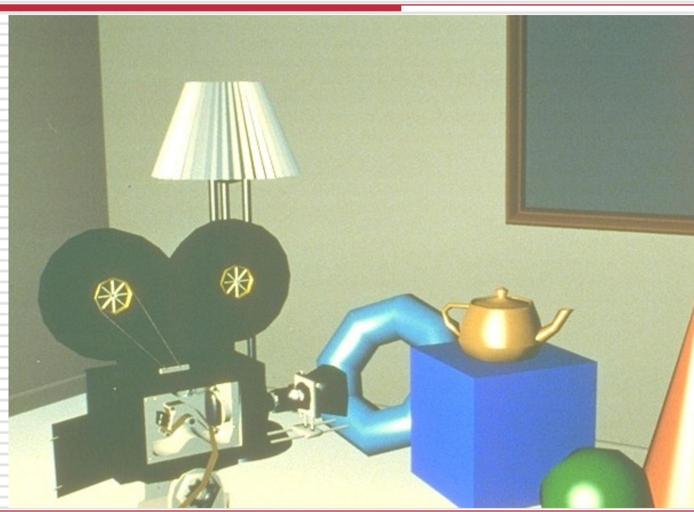
Diffuse Shading + Interp. Normals



Gouraud Shading



Ambient + Diffuse + Specular



Ambient + Diffuse + Specular + Interpolated Normals

Radiosity

Texture Mapping

Texture Mapping + Ray Tracing