

Computer Graphics (Fall 2005)

COMS 4160, Lecture 16: Illumination and Shading 1

<http://www.cs.columbia.edu/~cs4160>

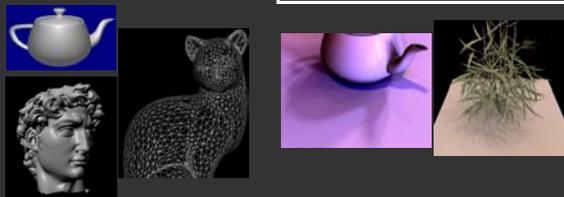
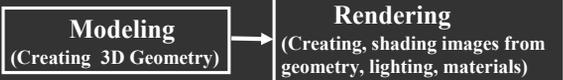
To Do

- Work on HW 3, do well
- Start early on HW 4

- Discussion of midterm
 - But remember HW 3, HW 4 more important

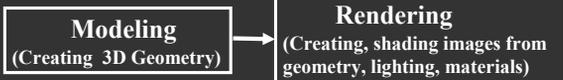
Course Outline

- 3D Graphics Pipeline



Course Outline

- 3D Graphics Pipeline



Unit 1: Transformations

Weeks 1,2. Ass 1 due Sep 22

Unit 3: OpenGL

Weeks 5-7.

Ass 3 due Nov 10

Unit 2: Spline Curves

Weeks 3,4. Ass 2 due Oct 7

Unit 4: Lighting, Shading

Weeks 8,9.

Written Ass 1 due Nov 16

Midterm on units 1-3: Oct 26

Ass 4: Interactive 3D Video Game (final project) due Dec 13

Rendering: 1960s (visibility)

- Roberts (1963), Appel (1967) - hidden-line algorithms
- Warnock (1969), Watkins (1970) - hidden-surface
- Sutherland (1974) - visibility = sorting



Images from FvDFH, Pixar's Shutterbug
Slide ideas for history of Rendering courtesy Marc Levoy

Rendering: 1970s (lighting)

1970s - raster graphics

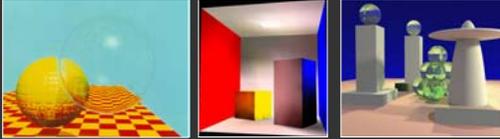
- Gouraud (1971) - diffuse lighting, Phong (1974) - specular lighting
- Blinn (1974) - curved surfaces, texture
- Catmull (1974) - Z-buffer hidden-surface algorithm



Rendering (1980s, 90s: Global Illumination)

early 1980s - global illumination

- Whitted (1980) - ray tracing
- Goral, Torrance et al. (1984) radiosity
- Kajiyu (1986) - the rendering equation



Outline

- Preliminaries
- Basic diffuse and Phong shading
- Gouraud, Phong interpolation, smooth shading
- Formal reflection equation (next lecture)
- Texture mapping (in one week)
- Global illumination (next unit)

For today's lecture, slides and chapter 9 in textbook

Motivation

- Objects not flat color, perceive shape with appearance
- Materials interact with lighting
- Compute correct shading pattern based on lighting
 - This is not the same as shadows (separate topic)
- Some of today's lecture review of last OpenGL lec.
 - Idea is to discuss illumination, shading independ. OpenGL
- Today, initial hacks (1970-1980)
 - Next lecture: formal notation and physics

Linear Relationship of Light

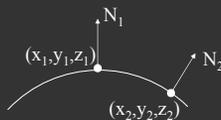
- Light energy is simply sum of all contributions

$$I = \sum_k I_k$$

- Terms can be calculated separately and later added together:
 - multiple light sources
 - multiple interactions (diffuse, specular, more later)
 - multiple colors (R-G-B, or per wavelength)

General Considerations

Surfaces are described as having a position, and a normal at every point.

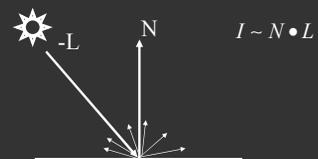


Other vectors used

- L = vector to the light source
light position minus surface point position
- E = vector to the viewer (eye)
viewer position minus surface point position

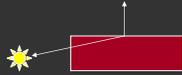
Diffuse Lambertian Term

- Rough matte (technically Lambertian) surfaces
 - Not shiny: matte paint, unfinished wood, paper, ...
- Light reflects equally in all directions
- Obey Lambert's cosine law
 - Not exactly obeyed by real materials

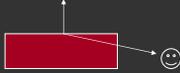


Meaning of negative dot products

- If $(N \cdot L)$ is negative, then the light is behind the surface, and cannot illuminate it.



- If $(N \cdot E)$ is negative, then the viewer is looking at the underside of the surface and cannot see its front-face.



- In both cases, I is clamped to Zero.

Phong Illumination Model

- Specular or glossy materials: highlights
 - Polished floors, glossy paint, whiteboards
 - For plastics highlight is color of light source (not object)
 - For metals, highlight depends on surface color
- Really, (blurred) reflections of light source



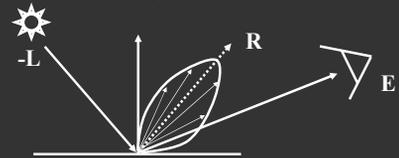
Roughness →

Idea of Phong Illumination

- Find a simple way to create highlights that are view-dependent and happen at about the right place
- Not physically based
- Use dot product (cosine) of eye and reflection of light direction about surface normal
- Alternatively, dot product of half angle and normal
- Raise cosine lobe to some power to control sharpness or roughness

Phong Formula

$$I \sim (R \cdot E)^p$$



$$R = ?$$

$$R = -L + 2(L \cdot N)N$$

Alternative: Half-Angle (Blinn-Phong)

$$I \sim (N \cdot H)^p$$



- In practice, both diffuse and specular components for most materials

Outline

- Preliminaries
- Basic diffuse and Phong shading
- *Gouraud, Phong interpolation, smooth shading*
- Formal reflection equation (next lecture)
- Texture mapping (in one week)
- Global illumination (next unit)

Not in text. If interested, look at FvDFH pp 736-738

Triangle Meshes as Approximations

- Most geometric models are large collections of triangles.
- Triangles have 3 vertices, each with a position, color, normal, and other parameters (such as n for Phong reflection).
- The triangles are an approximation to the actual surface of the object.



Coloring Between the Lines

- We know how to calculate the light intensity given:
 - surface position
 - normal
 - viewer position
 - light source position (or direction)
- How do we shade a triangle between its vertices, where we aren't given the normal?

Flat vs. Gouraud Shading



`glShadeModel(GL_FLAT)`



`glShadeModel(GL_SMOOTH)`

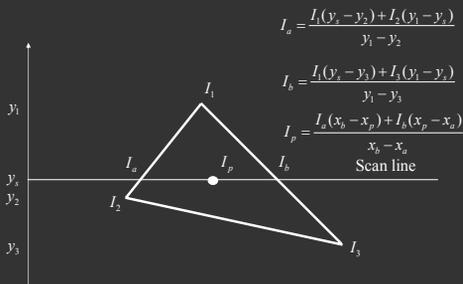
Flat - Determine that each face has a single normal, and color the entire face a single value, based on that normal.

Gouraud - Determine the color at each vertex, using the normal at that vertex, and interpolate linearly for the pixels between the vertex locations.

Gouraud Shading - Details 1

- Inter-vertex interpolation can be done in object space (along the face), but it is simpler to do it in image space (along the screen).
- 2 ways for a vertex to get its normal:
 - given when the vertex is defined.
 - take all the normals from faces that share the vertex, and average them.

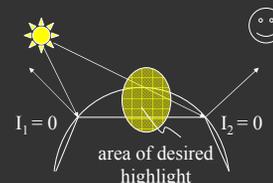
Gouraud Shading - Details 2



Actual implementation efficient: difference equations while scan converting

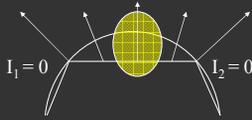
Gouraud and Errors

- $I_1 = 0$ because $(N \cdot E)$ is negative.
- $I_2 = 0$ because $(N \cdot L)$ is negative.
- Any interpolation of I_1 and I_2 will be 0.



2 Phongs make a Highlight

- Besides the Phong Reflectance model (\cos^n), there is a Phong Shading model.
- Phong Shading: Instead of interpolating the intensities between vertices, interpolate the *normals*.
- The entire lighting calculation is performed for each pixel, based on the interpolated normal. (OpenGL doesn't do this, but you can with current programmable shaders)



Problems with Interpolated Shading

- Silhouettes are still polygonal
- Interpolation in screen, not object space: perspective distortion
- Not rotation or orientation-independent
- How to compute vertex normals for sharply curving surfaces?
- But at end of day, polygons is much easier than explicitly representing curved objects like spline patches for rendering