Announcements

- Programming Assignment 1 is out today
- Due next Thursday by 11:59pm
- ACM Programming Contest
- Meeting tonight at 7pm in 011

Last Time

- Reviewed Homework 1
- Assigned / Demoed Programming Assignment 1
- Due next Thursday (9/27) by 11:59pm
- Discussed different ways of representing geometric objects for computer graphics
  - Procedural
  - Tessellated polygons

Today

- Programming assignment 1 is out
- Any questions?
- Talking about lighting and shading
- Focusing on OpenGL

Light and Matter

- Review:
  - Materials do NOT have color
  - Light does
    - Material objects appear to have color because they reflect only certain wavelengths of light
- How does light interact with matter?
**Light and Matter**

- Specular reflection

- Diffuse reflection

**Light and Matter**

- Diffuse Reflection

- Translucency / Transparency

**Lights**

- So what are some properties of lights?
  - Wavelength (Color)
  - Position
  - Size
  - Intensity
  - Distribution of light

**Point Lights**

- Emit light evenly in all directions
  - That is, photons (rays) from a point light all originate from the same point, and have directions evenly distributed over the sphere
### Directional Lights
- Point light sources at an infinite (or near infinite) distance
- Can assume that all rays are parallel

### Spot Lights
- Similar to point lights, but intensity of emitted light varies by direction
- Can think of it as only emitting rays over a patch on the sphere

### Area Lights
- Emits light in every direction from a surface
- Can think of it as a set of point lights, or a patch on which every point is a point light

### The Rendering Equation
- In short, the light out from a point in a specific direction depends on:
  - The light it emits in that direction
  - The light incident on that point from every direction, affected by
    - How reflective the material is for that pair of directions
    - How close the incoming direction is to the surface normal

---

**Jim Kajiya, 1986**

- This is a theoretical model of light transport
- It's not actually solvable by conventional means
- Radiosity and various ray-tracing methods attempt to approximate its actual solution in various ways
- OpenGL uses a simplified model
Light Simplifications

- We must simplify lights for real-time rendering:
  - Single RGB color instead of wavelength distribution (denoted $L_m$)
  - Intensity is rolled into $L_m$
  - No area lights
  - Distribution of light is uniform
    - Except for spot lights

- No indirect light
- Use an ambient light term
- Really just a hack
- Shadows are hard shadows, or are not included at all

Phong Reflection Model

- A simplification of the rendering equation
- Divided into 3 parts
  - Ambient
  - Diffuse
  - Specular
- The sum of these components describes the color at a point

\[
I = I_a(R_a, L_a) + I_d(n, l, R_d, L_d, a, b, c, d) + I_s(r, v, R_s, L_s, n, a, b, c, d)
\]

- $R_{something}$ represents how reflective the surface is
- $L_{something}$ represents the intensity of the light
- In practice, these are each 3-vectors
- One each for R, G, and B

Phong Reflection Model: Ambient Term

- Assume that ambient light is the same everywhere
- Is this generally true?
- $I_a(R_a, L_a) = R_a \cdot L_a$
- The contribution of ambient light at a point is just the intensity of the light modulated by how reflective the surface is (for that color)
Phong Reflection Model: Diffuse Term

- \( I_d(n, l, R_d, L_d, a, b, c, d) = \frac{R_d}{(a + bd + cd^2)} \cdot \max(l \cdot n, 0) \cdot L_d \)
- \( a, b, c \): user-defined constants
- \( d \): distance from the point to the light
- Let's consider these parts

Lambert's Cosine Law

- The incident angle of the incoming light affects its apparent intensity
- Does the sun seem brighter at noon or 6pm?
- Why?

Phong Reflection Model: Diffuse Term

- We already know how to get the cosine between the light direction and the normal
- \( n \cdot l \)
- What happens if the surface is facing away from the light?
- That's why we use \( \max(n \cdot l, 0) \)
- Why not just take \( |n \cdot l| \)?

Phong Reflection Model: Specular Term

- \( I_s(r, v, R_s, L_s, a, b, c, d) = \frac{R_s}{(a + bd + cd^2)} \cdot \max(r \cdot v, 0)^n \cdot L_s \)
- \( n \cdot v \)?
- Reflection is strongest in the direction of the reflection vector
- \( r \cdot v \) is maximized when the viewpoint vector (or really the vector to the viewpoint) is in the same direction as \( r \)
- What is \( n \)?

Phong Reflection Model: Specular Term

- In the real world, lights seem to get dimmer as they get further away
- Intensity decreases with distance
- We can simulate that by adding an attenuation term
- \( (R_s / (a + bd + cd^2)) \)
- User can choose the \( a,b,c \) constants to achieve the desired “look”
Phong Reflection Model

- So how do we compute these vectors?
  - Normal (\(n\))
  - Light (\(l\))
  - Viewer (\(v\))
  - Reflection (\(r\))

Where do vectors come from?

- So how do we compute these vectors?
  - Viewer (\(v\)) = normalize(eyePos - p)
  - Light (\(l\)) = normalize(lightPos - p)
  - Normal (\(n\))
  - Reflection (\(r\))

Reflection Vector

- A reflection of \(l\) across \(n\)
- Must be in the same plane as \(n\) and \(l\)

Where do vectors come from?

- So how do we compute these vectors?
  - Viewer (\(v\)) = normalize(eyePos - p)
  - Light (\(l\)) = normalize(lightPos - p)
  - Normal (\(n\))
  - Reflection (\(r\)) = \(-l + 2 \cdot (n \cdot l) \cdot n\)

Normal Vector

- Can be stored with the model
- More likely (at least with OpenGL), we're dealing with a model made up of triangles

Where do vectors come from?

- So how do we compute these vectors?
  - Viewer (\(v\)) = normalize(eyePos - p)
  - Light (\(l\)) = normalize(lightPos - p)
  - Normal (\(n\)) = normalize(cross(v2 - v1, v3 - v1))
  - Reflection (\(r\)) = \(-l + 2 \cdot (n \cdot l) \cdot n\)
**Summing up Lighting**

- So that’s how we think about lighting
  - Computing the color of a single point on a surface
  - Now we’re going to talk about shading
  - Not shadows
  - Graphics term: Filling in a polygon with color

**Types of Shading**

- There are several well-known / commonly-used shading methods
  - Flat shading
  - Gouraud shading
  - Phong shading

---

**Flat Shading**

- Simplest type of shading
  - Treat the entire polygon as one point (usually the center)
  - Solve the Phong lighting equations once
  - Fill in the whole polygon with that color

---

**Gouraud Shading**

- A bit more complicated than flat shading
  - Compute normals at each vertex
  - Solve the Phong lighting equations at each vertex
  - Linearly interpolate color inside the triangle
  - Gouraud shading is the default in OpenGL
  - Flat shading is also built in

---

**Gouraud vs. Phong**

- Gouraud shading is:
  - Still very fast
  - MUCH nicer looking than flat shading
  - However:
    - Specular highlights brighten/fade near vertices
    - The change in linear function is noticeable at triangle edges

---

**Phong Shading**

- Not the same as Phong lighting
  - Compute normals at each vertex
  - Linearly interpolate *normals* at each point inside the triangle
  - Solve the Phong lighting equations at each fragment (pixel)
  - Not built into OpenGL
  - Can now be done in real-time with programmable shaders
Shading Comparison

Lighting in OpenGL
- Need to enable
  GL_LIGHTING
  GL_LIGHTN // (N = 0, 1, 2, 3, ...)
- Can use:
  - Point lights
  - Spot lights
  - Directional lights
  - Ambient lights

Lighting in OpenGL
- Can choose your shading model:
  - Flat (GL_FLAT)
  - Gouraud (GL_SMOOTH)
- Other functions you might need:
  - glLight{fv}
  - glLightModel
  - glMaterial
  - glColorMaterial
  - glNormal3f

Next Time
- Continuing our discussion of lighting and shading
- OpenGL demo of lighting/shading functions
- Introducing stylized shading techniques