Computer Graphics

Prof. Feng Liu

Fall 2015

http://www.cs.pdx.edu/~fliu/courses/cs447/

11/12/2015
Last time

- Hidden Surface Removal
Today

☐ Lighting and Shading
☐ Project 2
☐ Will publicize several times in the final week of classes when you can get your project graded
  ■ Demo your program to the instructor in person
  ☐ Bring your own laptop or on a CS Windows Lab Machine
  ■ Latest time to grade
  ☐ 5:00 pm, Friday, December 4, 2015
  ■ No late submission!
Where We Stand

☐ So far we know how to:
  ■ Transform between spaces
  ■ Draw polygons
  ■ Decide what’s in front

☐ Next
  ■ Deciding a pixel’s intensity and color
Normal Vectors

- The intensity of a surface depends on its orientation with respect to the light and the viewer.
- The *surface normal vector* describes the orientation of the surface at a point.
  - Mathematically: Vector that is perpendicular to the tangent plane of the surface.
  - Just “the normal vector” or “the normal.”
  - Will use $\mathbf{n}$ or $\mathbf{N}$ to denote.
- Normals are either supplied by the user or automatically computed.
Transforming Normal Vectors

- Normal vectors are *directions*
  - Normal vectors are perpendicular to tangent vectors: \( \mathbf{n} \cdot (\mathbf{x} - \mathbf{p}) = 0 \)
  - There is a matrix form of this: \( \mathbf{n}'(\mathbf{x} - \mathbf{p}) = 0 \)
  - Consider the equation with a transformed tangent: \( \mathbf{n}'T^{-1}T(\mathbf{x} - \mathbf{p}) = 0 \)
  - The right hand half is the transformed point.
  - The new transpose normal must be equal to: \( \mathbf{n}'T^{-1} \)
  - The new normal must then be: \( (\mathbf{n}'T^{-1})' = (T^{-1})'\mathbf{n} \)

- To transform a normal, multiply it by the inverse transpose of the transformation matrix

- Recall, rotation matrices are their own inverse transpose

- Don’t include the translation! Use \((n_x, n_y, n_z, 0)\) for homogeneous coordinates
Local Shading Models

- *Local shading models* provide a way to determine the intensity and color of a point on a surface
  - The models are local because they **don’t consider other objects**
  - We use them because they are fast and simple to compute
  - They do not require knowledge of the entire scene, only the current piece of surface.

- For the moment, assume:
  - We are applying these computations at a particular point on a surface
  - We have a normal vector for that point
Local Shading Models

- What they capture:
  - Direct illumination from light sources
  - Diffuse and Specular reflections
  - (Very) Approximate effects of global lighting

- What they don’t do:
  - Shadows
  - Mirrors
  - Refraction
  - Lots of other stuff ...
“Standard” Lighting Model

- Consists of three terms linearly combined:
  - **Diffuse** component for the amount of incoming light from a point source reflected equally in all directions
  - **Specular** component for the amount of light from a point source reflected in a mirror-like fashion
  - **Ambient** term to approximate light arriving via other surfaces
Diffuse Illumination

\[ k_d I_i (L \cdot N) \]

- Incoming light, \( I_i \), from direction \( L \), is reflected equally in all directions
  - No dependence on viewing direction
- Amount of light reflected depends on:
  - Angle of surface with respect to light source
    - Actually, determines how much light is collected by the surface, to then be reflected
  - Diffuse reflectance coefficient of the surface, \( k_d \)
- Don’t want to illuminate back side. Use \( k_d I_i \max(L \cdot N, 0) \)
Diffuse Example

Where is the light?

Which point is brightest (how is the normal at the brightest point related to the light)?

Diffuse Lighting
Illustrating Shading Models

- Show the polar graph of the amount of light leaving for a given incoming direction:

  Diffuse?

- Show the intensity of each point on a surface for a given light position or direction:

  Diffuse?
Specular Reflection
(Phong Reflectance Model)

\[ k_s I_i (R \cdot V)^p \]

- Incoming light is reflected primarily in the mirror direction, \( R \)
  - Perceived intensity depends on the relationship between the viewing direction, \( V \), and the mirror direction
  - Bright spot is called a *specularity*

- Intensity controlled by:
  - The specular reflectance coefficient, \( k_s \)
  - The *Phong Exponent*, \( p \), controls the apparent size of the specularity
    - Higher \( p \), smaller highlight
Specular Example
Illustrating Shading Models

- Show the polar graph of the amount of light leaving for a given incoming direction:
  
  Specular?

- Show the intensity of each point on a surface for a given light position or direction
  
  Specular?
Alternative Specular Reflection Model

\[ H = \frac{(L + V)}{\|L + V\|} \]

\[ k_s I_i (H \cdot N)^p \]

- Compute based on normal vector and “halfway” vector, \( H \)
Putting It Together

\[ I = k_a I_a + I_i \left( k_d (\mathbf{L} \cdot \mathbf{N}) + k_s (\mathbf{H} \cdot \mathbf{N})^p \right) \]

- Global ambient intensity, \( I_a \):
  - Gross approximation to light bouncing around of all other surfaces
  - Modulated by ambient reflectance \( k_a \)
- Just sum all the terms
- If there are multiple lights, sum contributions from each light
- Several variations, and approximations …
Color

\[ I_r = k_{a,r}I_{a,r} + I_{i,r}(k_{d,r}(L \cdot N) + k_{s,r}(H \cdot N)^n) \]

- Do everything for three colors, r, g and b
- Note that some terms (the expensive ones) are constant
- Using only three colors is an approximation, but few graphics practitioners realize it
  - \( k \) terms depend on wavelength, should compute for continuous spectrum
Approximations for Speed

- The viewer direction, $V$, and the light direction, $L$, depend on the surface position being considered, $x$.
- Distant light approximation:
  - Assume $L$ is constant for all $x$.
  - Good approximation if light is distant, such as sun.
- Distant viewer approximation:
  - Assume $V$ is constant for all $x$.
  - Rarely good, but only affects specularities.
Distant Light Approximation

- Distant light approximation:
  - Assume $L$ is constant for all $x$
  - Good approximation if light is distant, such as sun
  - Generally called a directional light source

- What aspects of surface appearance are affected by this approximation?
  - Diffuse?
  - Specular?
Distant Viewer Approximation

- Specularities require the viewing direction:
  - $V(x) = ||c-x||$
  - Slightly expensive to compute

- Distant viewer approximation uses a global $V$
  - Independent of which point is being lit
  - Use the view plane normal vector
  - Error depends on the nature of the scene

- Is the diffuse component affected?
Describing Surfaces

- The various parameters in the lighting equation describe the appearance of a surface.
- \((k_d, r, k_d, g, k_d, b)\): The *diffuse color*, which most closely maps to what you would consider the “color” of a surface.
  - Also called *diffuse reflectance coefficients*.
- \((k_s, r, k_s, g, k_s, b)\): The *specular color*, which controls the color of specularities.
  - Some systems do not let you specify this color separately.
- \((k_a, r, k_a, g, k_a, b)\): The *ambient color*, which controls how the surface looks when not directly lit.
  - Normally the same as the diffuse color.
OpenGL Commands (1)

- **glMaterial**(face, parameter, value)
  - Changes one of the coefficients for the front or back side of a face (or both sides)

- **glLight**(light, property, value)
  - Changes one of the properties of a light (intensities, positions, directions, etc)
  - There are 8 lights: GL_LIGHT0, GL_LIGHT1, ...

- **glLightModel**(property, value)
  - Changes one of the global light model properties (global ambient light, for instance)

- **glEnable**(GL_LIGHT0) enables GL_LIGHT0
  - You must enable lights before they contribute to the image
  - You can enable and disable lights at any time
OpenGL Commands (2)

- `glEnable(GL_LIGHTING)` turns on lighting
  - You must enable lighting explicitly - it is off by default
- Don’t use specular intensity if you don’t have to
  - It’s expensive - turn it off by giving 0,0,0 as specular color of the lights
- Don’t forget normals
  - If you use scaling transformations, must enable GL_NORMALIZE to keep normal vectors of unit length
- Many other things to control appearance
Light Sources

- Two aspects of light sources are important for a local shading model:
  - Where is the light coming from (the $L$ vector)?
  - How much light is coming (the $I$ values)?

- Various light source types give different answers to the above questions:
  - *Point light source*: Light from a specific point
  - *Directional*: Light from a specific direction
  - *Spotlight*: Light from a specific point with intensity that depends on the direction
  - *Area light*: Light from a continuum of points (later in the course)
Point and Directional Sources

- **Point light:** \( L(x) = \frac{p_{\text{light}} - x}{\| p_{\text{light}} - x \|} \)
  - The \( L \) vector depends on where the surface point is located
  - Must be normalized - slightly expensive
  - To specify an OpenGL light at 1,1,1:
    
    ```c
    GLfloat light_position[] = { 1.0, 1.0, 1.0, 1.0 }; 
    glLightfv(GL_LIGHT0, GL_POSITION, light_position);
    ```

- **Directional light:** \( L(x) = L_{\text{light}} \)
  - The \( L \) vector does not change over points in the world
  - OpenGL light traveling in direction 1,1,1 (\( L \) is in opposite direction):
    
    ```c
    GLfloat light_position[] = { 1.0, 1.0, 1.0, 0.0 }; 
    glLightfv(GL_LIGHT0, GL_POSITION, light_position);
    ```
Spotlights

- Point source, but intensity depends on $L$:
  - Requires a position: the location of the source
    
    \[
    \text{glLightfv(GL_LIGHT0, GL_POSITION, light_posn);} \]
  
  - Requires a direction: the center axis of the light
    
    \[
    \text{glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, light_dir);} \]
  
  - Requires a cut-off: how broad the beam is
    
    \[
    \text{glLightfv(GL_LIGHT0, GL_SPOT_CUTOFF, 45.0);} \]
  
  - Requires and exponent: how the light tapers off at the edges of the cone
    
    \[
    \text{Intensity scaled by } (L \cdot D)^n \]
    
    \[
    \text{glLightfv(GL_LIGHT0, GL_SPOT_EXPONENT, 1.0);} \]
So far, we have discussed illuminating a single point

\[ I = k_a I_a + I_i \left( k_d (L \cdot N) + k_s (H \cdot N)^p \right) \]

We have assumed that we know:

- The point
- The surface normal
- The viewer location (or direction)
- The light location (or direction)

But commonly, normal vectors are only given at the vertices

It is also expensive to compute lighting for every point
Shading Interpolation

- Take information specified or computed at the vertices, and somehow propagate it across the polygon (triangle)

- Several options:
  - Flat shading
  - Gouraud interpolation
  - Phong interpolation
Flat shading

- Compute shading at a representative point and apply to whole polygon
  - OpenGL uses one of the vertices

- Advantages:
  - Fast - one shading computation per polygon

- Disadvantages:
  - Inaccurate
  - What are the artifacts?
Gouraud Shading

- Shade each *vertex* with its own location and normal
- *Linearly interpolate* the color across the face

**Advantages:**
- Fast: incremental calculations when rasterizing
- Much smoother - use same normal every time a vertex is used for a face

**Disadvantages:**
- What are the artifacts?
- Is it accurate?
Phong Interpolation

- Interpolate normals across faces
- Shade each pixel individually

Advantages:
- High quality, narrow specularities

Disadvantages:
- Expensive
- Still an approximation for most surfaces

Not to be confused with Phong’s specularity model
Shading and OpenGL

- OpenGL defines two particular shading models
  - Controls how colors are assigned to pixels
  - `glShadeModel(GL_SMOOTH)` interpolates between the colors at the vertices (the default, Gouraud shading)
  - `glShadeModel(GL_FLAT)` uses a constant color across the polygon

- Phong shading requires a significantly greater programming effort - beyond the scope of this class
  - Also requires fragment shaders on programmable graphics hardware
Next Time

- Texture mapping