lecture 12

- lighting
- materials: diffuse, specular, ambient
- shading: Flat vs. Gouraud vs Phong
Light Sources

- sunlight (parallel)
- spotlight
- light bulb
- ambient
Sunny day model: "point source at infinity"

Light reaching a unit area patch is proportional to $\vec{n} \cdot \vec{I}$
Why are the poles of the earth cold and the equator hot?
Spotlight model

other examples: lamp, ceiling light, window, .....
Spotlight model

Properties of a spotlight:

- 3D position
- peak direction
- spread (light bulb vs. laser beam)
- falloff in strength with distance
The illumination at $\hat{x}$ depends on direction of the spotlight and on the spread of the spotlight's beam (conceptually, the cone width).
\cos \chi = \hat{l}_{\text{peak}} \cdot \frac{\hat{x} - \hat{x}_e}{\| \hat{x} - \hat{x}_e \|}
The illumination depends on the spread of the spotlight's beam. We can model this effect as:

\[
\text{illumination at } \tilde{x} \sim (\cos \delta)^n
\]

very large \( n \)

\( \equiv \text{ laser beam} \)
$\mathbf{\vec{x}} = (x_e, y_e, z_e)$

\[
\mathbf{\vec{x}} = (x, y, z)
\]

\[
\mathbf{r} = \sqrt{(x_e - x)^2 + (y_e - y)^2 + (z_e - z)^2}
\]

How should the illumination at $x$ depend on distance?
Model the illumination from spotlight to be proportional to:

\[
\frac{1}{ar^2 + br + c}.
\]
Putting it all together... let $l(x)$ be the effective light source vector at $x$ that is due to the spotlight.

\[
\hat{\ell}(\vec{x}) = (\cos \sigma)^{\text{spread}} \frac{1}{(ar^2 + br + c)} \cdot \frac{\vec{x}_e - \vec{x}}{||\vec{x}_e - \vec{x}||}
\]
Special case: (non-directional) point light

\[ \mathbf{l}(\mathbf{x}) = \frac{1}{(ar^2 + br + c)} \cdot \frac{\mathbf{x}_l - \mathbf{x}}{||\mathbf{x}_l - \mathbf{x}||} \]

e.g. candle flame, light bulb
Ambient light

Same illumination everywhere in space.
lecture 12

- lighting

- materials: diffuse, specular, ambient

  Lighting + material allows us to calculate RGB.

  i.e. $\text{RGB}(x) = \ ?$

- shading: Flat vs. Gouraud vs Phong
Material (Reflectance)

diffuse  glossy  mirror
I will discuss them in order: diffuse, mirror, glossy.
Recall: light reaching a unit area patch is proportional to $\hat{n} \cdot \hat{l}$

$$I_{\text{diffuse}}(x) = I_{\text{light}} \cdot l_{\text{diffuse}}(x) \cdot \max(\hat{n} \cdot \hat{l}, 0)$$
Mirror  (extreme opposite of diffuse)

High school physics:  angle of incidence = angle of reflection.

In the next few slides, all vectors are unit length.
\[ \hat{\ell} + \vec{r} = 2(\hat{n} \cdot \hat{\ell}) \hat{n} \]
gives us an expression for \( \vec{r} \).
Glossy ("specular", shiny)

Bright regions are called "highlights".
Glossy ("specular", shiny)

Highlights occur at points near points where $r = V$. 
Phong model ("specular")

\[ I_{\text{specular}}(\hat{x}) = I_{\text{light}} k_{\text{specular}} (\hat{\vec{r}} \cdot \hat{\vec{v}})^e \]
Note the conceptual similarity to the spotlight model.

spotlight - spread of emitted beam

glossy highlight - spread of reflected beam
Blinn-Phong model ("specular")
-> used in OpenGL

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

\( \vec{H} \) is called the "half way vector"

Exercise: what is the computational advantage of Blinn-Phong over Phong?
OpenGL 1.0 (somewhat arbitrarily....)

\[ I_{\text{diffuse}}(x) = I_{\text{light}} \cdot k_{\text{diffuse}}(x) \cdot \max(\hat{n} \cdot \hat{l}, 0) \]

\[ I_{\text{specular}}(x) = I_{\text{light}} \cdot k_{\text{specular}}(x) \cdot (\hat{H} \cdot \hat{n})^e \]

\[ I_{\text{ambient}}(x) = I_{\text{light}} \cdot k_{\text{ambient}}(x) \]

\[ I(x) = I_{\text{diffuse}}(x) + I_{\text{specular}}(x) + I_{\text{ambient}}(x) \]

\[ I(x) \text{ is a triplet: RGB} \]
OpenGL lights

`glLightf( light, parameterName, parameter )`

`light` : a number (you can have up to 8 lights)

`parameterName`:

- `GL_AMBIENT`
- `GL_DIFFUSE`  
  color of the light
- `GL_SPECULAR`
- `GL_POSITION`
- `GL_SPOT_DIRECTION`
- `GL_SPOT_EXPONENT`
- `GL_SPOT_CUTOFF*`
- `GL_CONSTANT_ATTENUATION`
- `GL_LINEAR_ATTENUATION`
- `GL_QUADRATIC_ATTENUATION`

* cutoff in [0, 90] or 180 (uniform)
glEnable(GL_LIGHTING)
glEnable(GL_LIGHT0)

diffuseLight = (1, 1, .5, 1) // yellowish light
specularLight = (1, 1, .5, 1) // 
ambientLight = (1, 1, .5, 1) // 

// The above are RGBA values
// (A = alpha, we will cover it later in the course)

// OpenGL allows you to use different colored light source for
// ambient vs. diffuse vs. specular.
// However, IMHO, this makes no sense physically!

position = (-1.5, 1.0, -4.0, 1)

glLightfv(GL_LIGHT0, GL_AMBIENT, ambientLight)
glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuseLight)
glLightfv(GL_LIGHT0, GL_SPECULAR, specularLight)
glLightfv(GL_LIGHT0, GL_POSITION, position)
OpenGL Materials

```c
glMaterialfv( face, parameterName, parameters )
```

- **face**: GL_FRONT, GL_BACK, or GL_FRONT_AND_BACK
- **parameterName**: GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR, GL_SHININESS, ....

A few lectures from now, we will discuss how OpenGL does mirror surfaces ("environment mapping")
ambientMaterial = ( 0, 0.5, 0.5, 1 ) // middle cyan
diffuseMaterial = ( 1, 0, 1, 1 ) // magenta
specularMaterial = ( 1, 0, 0, 1 ) // red
shininess = (100.0,) // not a typo
              // rather, Python "tuple" notation

glMaterial(GL_FRONT, GL_AMBIENT, ambientMaterial )
glMaterial(GL_FRONT, GL_DIFFUSE, diffuseMaterial )
glMaterial(GL_FRONT, GL_SPECULAR, specularMaterial )
glMaterial(GL_FRONT, GL_SHININESS, shininess )

Exercise: which of the above values are in the formula below?

\[ I_{\text{specular}}(x) = I_{\text{light}} \cdot k_{\text{specular}}(x) \left( \frac{\vec{H}}{n} \right)^e \]
Material Modelling beyond OpenGL 1.0

The above examples are for the "fixed function pipeline" only.

With modern OpenGL (GLSL), you can code up whatever reflectance model and lighting model you wish.

This can be part of a vertex shader or fragment shader.
To fully characterize the reflection properties of a material at a point, you need 4 parameters.

In a real scene, the outgoing light in each outgoing direction is the sum that is due to all incoming directions.
"Measuring and modelling Anisotropic Reflection"
[Ward, 1992]
e.g. "brushed" metal

Recent models use subsurface scattering (especially for modelling wax, skin).
lecture 12

- lighting

- materials: diffuse, specular (Blinn-Phong), ambient

- shading: flat vs Gouraud vs Phong shading
It is natural to associate a surface normal with each *polygon*.

OpenGL allows us to explicitly define a surface normal at each *vertex*.
glBegin(GL_QUAD)
glNormal3f( __,____, ___)
glVertex3f( __, ____, ___)
glNormal3f( __,____, ___)
glVertex3f( __, ____, ___)
glNormal3f( __,____, ___)
glVertex3f( __, ____, ___)
glNormal3f( __,____, ___)
glVertex3f( __, ____, ___)
glEnd()
How to choose the RGB values, at each pixel?

Recall lecture 6: Filling a Polygon

for y = ymin to ymax {
    compute intersection of polygon edges with row y
    fill in pixels between adjacent pairs of edges
}
Linear Interpolation (LERP)

Compute the RGB at the vertices using a shading model (first half of today's lecture).

How do we interpolate?
\[ I(x', y) = I(x_0, y_0) + \left( I(x_2, y_2) - I(x_0, y_0) \right) \cdot \left( \frac{y - y_0}{y_2 - y_0} \right) \]
Similarly...

\( (x_i, y_i) \)

\( (x', y) \)

\( (x, y) \)

\( (x_0, y_0) \)

\( (x_2, y_2) \)

\[ I(x, y) = I(x'', y) + (I(x^1, y) - I(x'', y)) \left( \frac{x - x''}{x^1 - x''} \right) \]
A vertex belongs to more than one polygon. So, in principle, we could use different surface normals when the vertex is being used to fill in different polygons.

Each vertex may have different normals in different polygons.

Each vertex may have same normal in all polygons, and this normal is different than the normal of the polygons themselves. (See next slide.)
For smooth shading, we can define vertex normals to be an average of the normals of the faces that the vertex belongs to.

\[
\hat{n} = \frac{1}{\sum_i \|E_{ni}\|} \sum_i E_{ni}
\]
Flat vs. Smooth (Gouraud) Shading

http://www.felixgers.de/teaching/jogl/lightAlgo.html

(MODIFIED Feb. 22: color of one vertex is used for the entire polygon)

https://www.opengl.org/sdk/docs/man2/xhtml/glShadeModel.xml
Phong shading

Linearly interpolating the (RGB) intensities from the vertices is not quite right, especially for shiny surfaces.

Phong observed it would be better to *linearly interpolate the normals* and then compute the RGB values at each pixel (using the Phong model).
If the highlight occurs in the middle of a polygon, Gouraud shading will miss it but Phong shading will find it.
Shading in OpenGL 1.0

OpenGL 1.0 does not do Phong shading.

It does flat shading and smooth shading.

```c
void glShadeModel( GLenum mode )
{
  if( mode == GL_FLAT )
    { 
      // one color used for each polygon
      
    }
  else if( mode == GL_SMOOTH )
    { 
      // Linear interpolation from vertex colors
      // Smooth shading includes Gouraud shading
    }
}
```
Announcements

Next Tuesday: review Exercises

Next Thursday: midterm

Last name A-P (Trottier 0100),
Last name Q-Z (Rutherford Physics 114)

A2: posted soon (latest next week)
due date (to be determined)