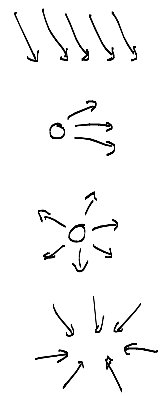
lecture 12

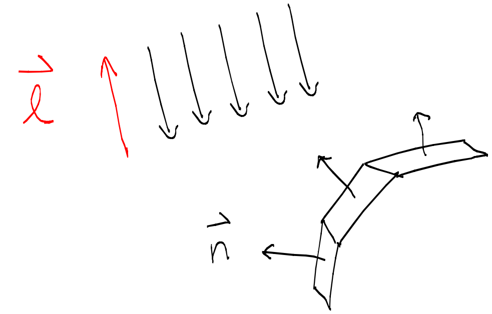
- lighting
- materials: diffuse, specular, ambient
- shading: Flat vs. Gouraud vs Phong

Light Sources

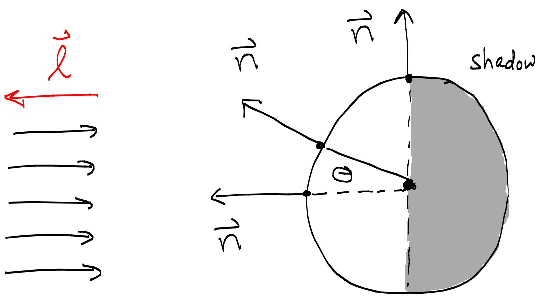


- sunlight (parallel)
- spotlight
- light bulb
- ambient

Sunny day model : "point source at infinity"



Light reaching a unit area patch is proportional to $\vec{n} \cdot \vec{\ell}$



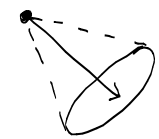
$$\vec{n} \cdot \vec{\ell} = \cos\theta$$

Why are the poles of the earth cold and the equator hot?

Spotlight model

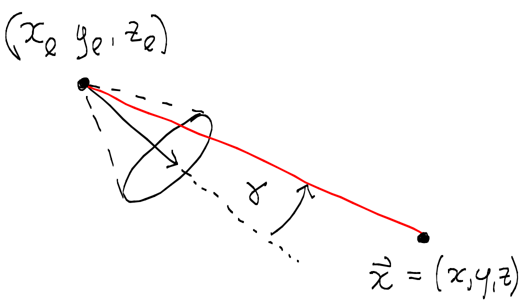
other examples: lamp, ceiling light, window, ...

Spotlight model

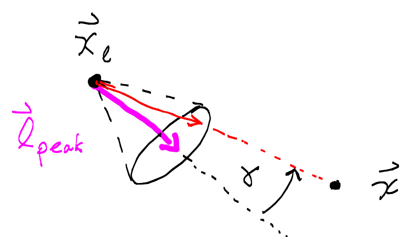


Properties of a spotlight:

- 3D position
- peak direction
- spread (light bulb vs. laser beam)
- falloff in strength with distance



The illumination at $\vec{x}$ depends on direction of the spotlight and on the spread of the spotlight's beam (conceptually, the cone width).

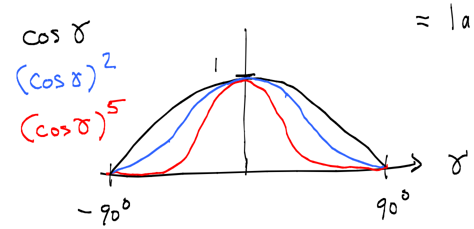


$$\cos\gamma = \vec{\ell}_{peak} \cdot \frac{\vec{x} - \vec{x}_\ell}{\|\vec{x} - \vec{x}_\ell\|}$$

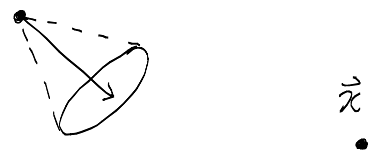
The illumination depends on the spread of the spotlight's beam. We can model this effect as:

illumination at $\vec{x} \sim (\cos\gamma)^n$

very large n
$\approx$ laser beam



$\vec{x}_\ell = (x_\ell, y_\ell, z_\ell)$

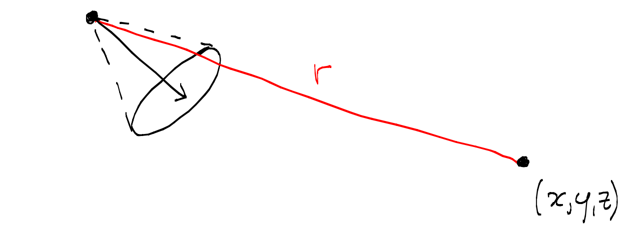


$\vec{x} = (x, y, z)$

$$r = \sqrt{(x_\ell - x)^2 + (y_\ell - y)^2 + (z_\ell - z)^2}$$

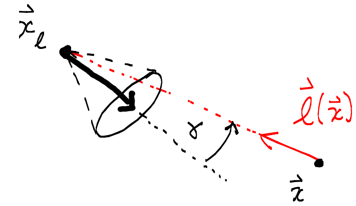
How should the illumination at $\vec{x}$ depend on distance ?

(x_ℓ, y_ℓ, z_ℓ)



Model the illumination from spotlight to be proportional to: $\dfrac{1}{ar^2 + br + c}$.

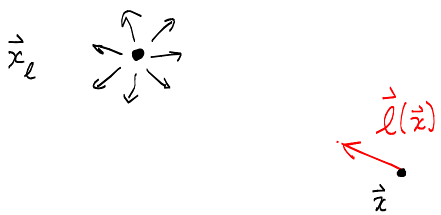
Putting it all together... let $\vec{l}(x)$ be the effective light source vector at x that is due to the spotlight.



$$\vec{\ell}(\vec{x}) = (\cos\gamma)^{spread} * \frac{1}{(ar^2+br+c)} * \frac{\vec{x}_\ell - \vec{x}}{\|\vec{x}_\ell - \vec{x}\|}$$

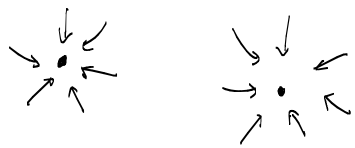
Special case: (non-directional) point light

e.g. candle flame, light bulb



$$\vec{\ell}(\vec{x}) = \frac{1}{(ar^2+br+c)} * \frac{\vec{x}_\ell - \vec{x}}{\|\vec{x}_\ell - \vec{x}\|}$$

Ambient light



Same illumination everywhere in space.

lecture 12

- lighting
- materials: diffuse, specular, ambient

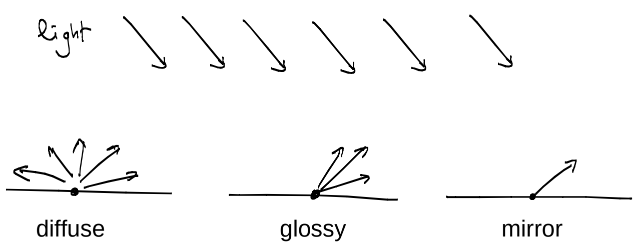
  Lighting + material allows us to calculate RGB.

  i.e. RGB(x) = ?
- shading: Flat vs. Gouraud vs Phong

Material (Reflectance)

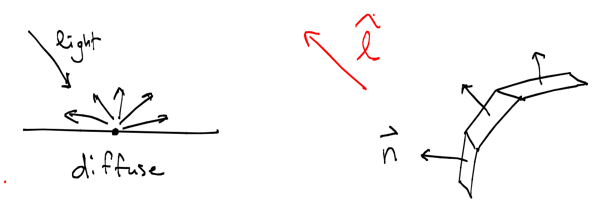
diffuse glossy mirror

Material (Reflectance)



I will discuss them in order: diffuse, mirror, glossy.

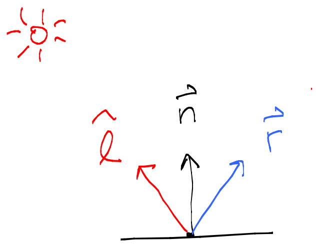
Diffuse / Matte / Lambertian



<u>Recall:</u> light reaching a unit area patch is proportional to $\vec{n} \cdot \hat{\ell}$

$$I_{diffuse}(x) = I_{light}\, k_{diffuse}(x) \max(\vec{n} \cdot \hat{\ell}, 0)$$

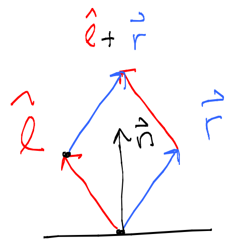
Mirror (extreme opposite of diffuse)



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

In the next few slides, all vectors are unit length.

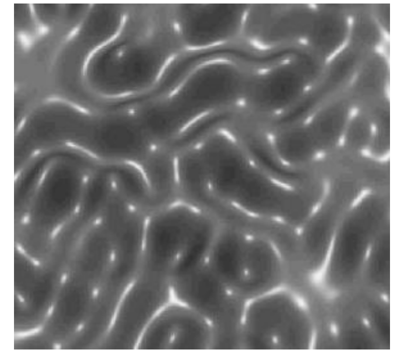
Mirror

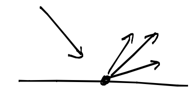


$$\hat{l} + \vec{r} = 2(\vec{n}\cdot\hat{l})\,\vec{n}$$

gives us an expression for $\vec{r}$.

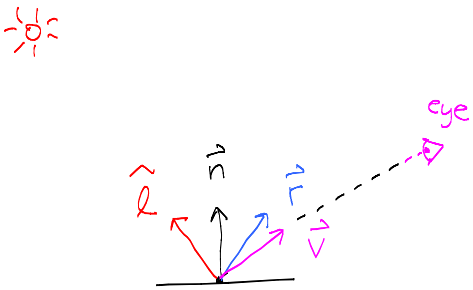
Glossy ("specular", shiny)





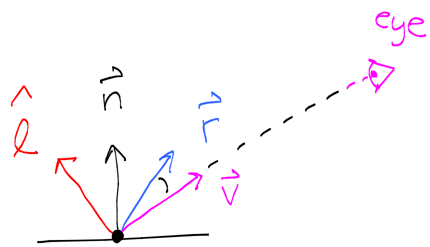
Bright regions are called "highlights".

Glossy ("specular", shiny)



Highlights occur at points near points where $\vec{r} = \vec{v}$.

Phong model ("specular")



$$I_{specular}(\vec{x}) = I_{light}\; k_{specular}\,(\vec{r}\cdot\vec{v})^{e}$$

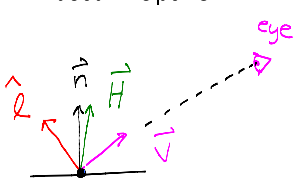
Note the conceptual similarity to the spotlight model.

spotlight - spread of emitted beam

glossy highlight - spread of reflected beam

Blinn-Phong model ("specular")

-> used in OpenGL



$$\vec{H} = \frac{\hat{l} + \vec{v}}{\|\hat{l} + \vec{v}\|}$$

H is called the "half way vector"

$$I_{specular}(\vec{x}) = I_{light}\; k_{specular}\,(\vec{H}\cdot\vec{n})^{e}$$

Exercise: what is the computational advantage of Blinn-Phong over Phong?

OpenGL 1.0 (somewhat arbitrarily....)

$$I_{diffuse}(x) = I_{light}\; k_{diffuse}(x)\; \max(\vec{n}\cdot\hat{l},\, 0)$$

$$I_{specular}(\vec{x}) = I_{light}\; k_{specular}(x)\; (\vec{H}\cdot\vec{n})^{e}$$

$$I_{ambient}(\vec{x}) = I_{light}\; k_{ambient}(x)$$

$$\boxed{I(x) = I_{diffuse}(x) + I_{specular}(x) + I_{ambient}(x)}$$

$I(x)$ is a triplet : RGB

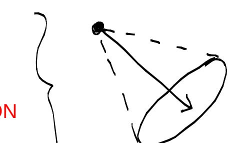
OpenGL lights

glLightf(light, parameterName, parameter **)**

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

parameterName:

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



$ar^2 + br + c$

* 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

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 ?

$$I_{specular}(\vec{x}) = I_{light}\; k_{specular}(x)\; (\vec{H} \cdot \vec{n})^{e}$$

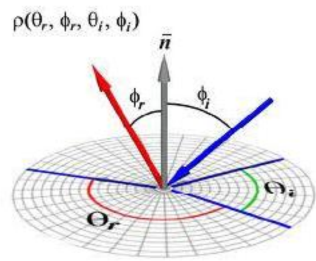
Material Modelling beyond OpenGL 1.0

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.

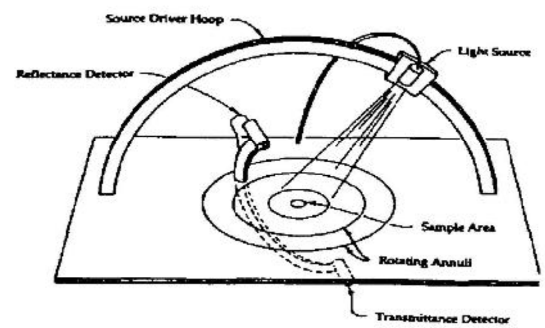
BRDF (Bidirectional Reflection Distribution Function)



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.

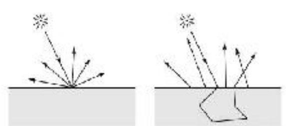
"Measuring and modelling Anisotropic Reflection" [Ward, 1992]



e.g. "brushed" metal

http://www.vrayc4d.com/manual/scenes/anisotropic-brushed-metal-186

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



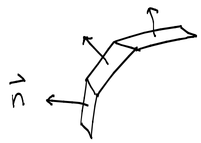
surface   sub-surface

https://graphics.stanford.edu/papers/bssrdf/bssrdf.pdf

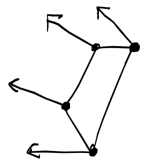
lecture 12

- lighting
- materials: diffuse, specular (Blinn-Phong), ambient
- shading: flat vs Gouraud vs Phong shading

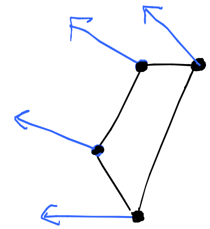
It is natural to associate a surface normal with each *polygon*.



OpenGL allows us to explicitly define a surface normal at each *vertex*.

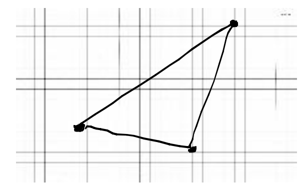


```
glBegin(GL_QUAD)
glNormal3f( __,___, __)
glVertex3f( __, ___, ___)
glNormal3f( __,___, __)
glVertex3f( __, ___, ___)
glNormal3f( __,___, __)
glVertex3f( __, ___, ___)
glNormal3f( __,___, __)
glVertex3f( __, ___, ___)
glEnd()
```



How to choose the RGB values at each pixel ?

Recall lecture 6: Filling a Polygon

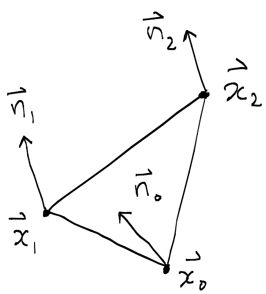


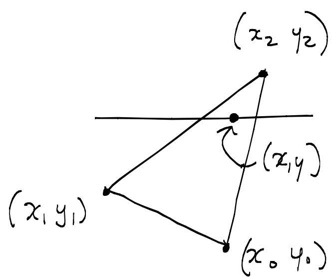
```
for y = ymin to ymax {
    compute intersection of polygon edges with row y
    fill in pixels between adjacent pairs of edges
}
```

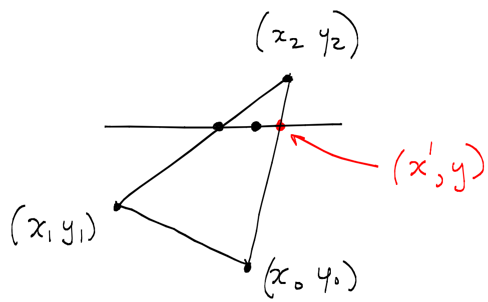
Linear Interpolation (LERP)

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

How do we interpolate ?

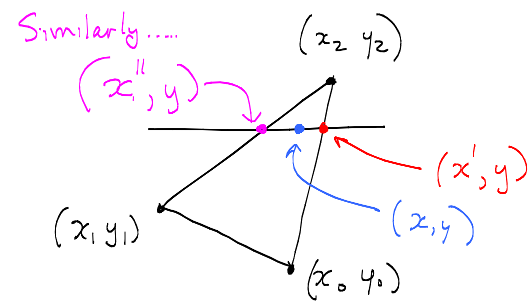






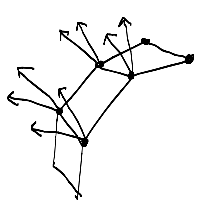
$$I(x',y) = I(x_0\ y_0) + \left(I(x_2,y_2) - I(x_0,y_0)\right)\left(\frac{y-y_0}{y_2-y_0}\right)$$

Similarly.....

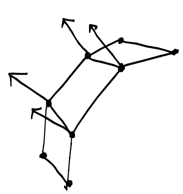


$$I(x,y) = I(x'',y) + \left(I(x',y) - I(x'',y)\right)\left(\frac{x-x''}{x'-x''}\right)$$

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.

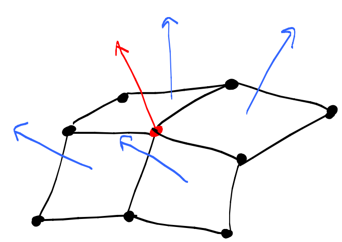


Each vertex may have different normals in different polygons.



Each vertex may have same normal in all polygons, and this normal is different than the normal of the polygons themselves. (See next slide.)

Smooth (Gouraud) Shading

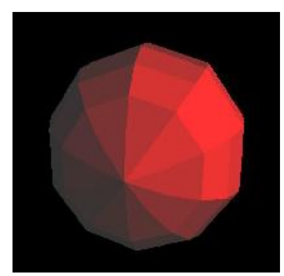


For smooth shading, we can define vertex normals to be an average of the normals of the faces that the vertex belongs to.

$$\vec{n} = \sum_i n_i \Big/ \left\| \sum_i n_i \right\|$$

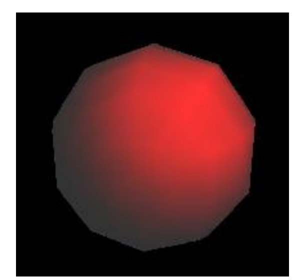
Flat vs. Smooth (Gouraud) Shading

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



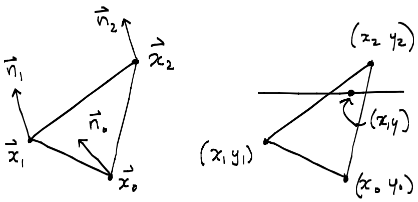
Flat

(MODIFIED Feb. 22:
color of one vertex is used for the entire polygon)
https://www.opengl.org/sdk/docs/man2/xhtml/glShadeModel.xml



Smooth / Gouraud
(vertex normals are averages of neighboring face normals.)

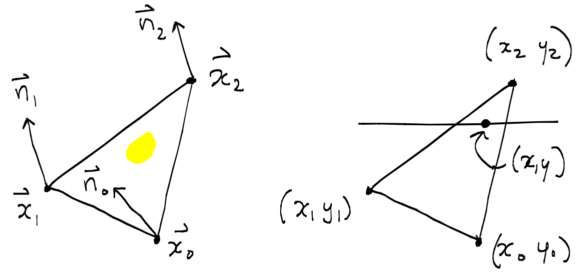
Phong shading



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).

Phong shading



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 <u>not</u> do Phong shading.

It does flat shading and smooth shading.

```
glShadeModel( GL_FLAT )
    //  one color used for each polygon

glShadeModel( GL_SMOOTH )   // default
      //  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)