Announcements

Assignment 2 due on Friday
Written Assignment 2 out later today.

Midterm next Thursday—or we could move it to 10/24 or 10/31?

Shading

Light Sources
Diffuse & Specular Reflection
Phong Illumination Model
Transmission with Refraction
Texture Mapping

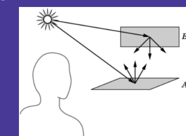
Watt, Chapter 6.2 and 6.3

COMPUTER GRAPHICS

15-462

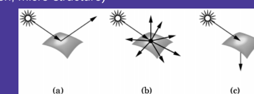
9/23/02

Illumination



Light Sources emit light
- EM spectrum
- Position and direction

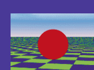
Surfaces reflect light
- Reflectance
- Geometry (position, orientation, micro-structure)
- Absorption
- Transmission



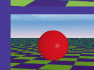
Illumination determined by the interactions between light sources and surfaces

Types of Light Sources

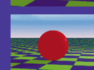
- Ambient: equal light in all directions
  - a hack to model inter-reflections



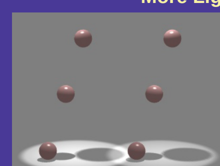
- Directional: light rays oriented in same direction
  - good for distance light sources (sunlight)

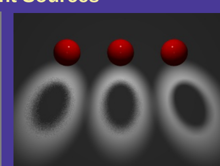


- Point: light rays diverge from a single point
  - approximation to a light bulb (but harsher)



More Light Sources

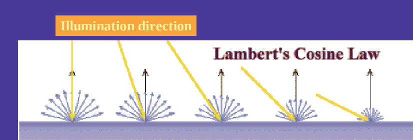




- Spotlight: point source with directional fall-off
  - intensity is maximal along some direction D, falls off away from D
  - specified by color, point, direction, fall-off parameters
- Area Source: Luminous 2D surface
  - radiates light from all points on its surface
  - generates soft shadows

Diffuse Reflection

- Simplest kind of reflector (also known as *Lambertian Reflection*)
- Models a matte surface -- rough at the microscopic level
- Ideal diffuse reflector
  - incoming light is scattered equally in all directions
  - viewed brightness does not depend on viewing direction
  - brightness *does* depend on direction of illumination



Lambert's Law

$$I_{diffuse} = k_d I_{light} \cos\theta$$
$$= k_d I_{light} (N \bullet L)$$

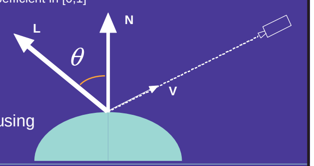
I_{light} : Light Source Intensity

k_d : Surface reflectance coefficient in [0,1]

θ : Light/Normal angle

$$\cos\theta = \frac{N \bullet L}{|N||L|}$$

See Watt if this is confusing



Examples of Diffuse Illumination

Same sphere lit diffusely from different lighting angles

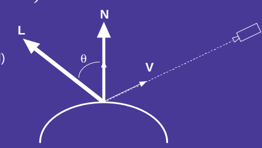
Ambient + Diffuse Reflection

CG started using the Lambertian model and then added more terms as extra effects were required

$$I_{d+a} = k_a I_a + k_d I_{light} (N \bullet L)$$

I_a : Ambient light intensity (global)

k_a : Ambient reflectance (local)



This is diffuse illumination plus a simple ambient light term

a trick to account for a background light level caused by multiple reflections from all objects in the scene (less harsh appearance)

Further Simple Illumination Effects

- Light attenuation:
  - light intensity falls off with the square of the distance from the source - so we add an extra term for this

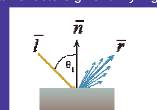
$$I_{d+a} = k_a I_a + f_{att} k_d I_{light} (N \bullet L) \quad \textbf{where} \quad f_{att} = \frac{1}{d^2}$$

with d the light source to surface distance - more complicated formulae are possible (see Foley) and work better

- Colored lights and surfaces:
  - just have three separate equations for RGB
- Atmospheric attenuation:
  - use viewer-to-surface distance to give extra effects
  - the distance is used to blend the object's radiant color with a "far" color (e.g., a nice hazy gray)

Specular Reflection

- Shiny surfaces change appearance when viewpoint is varied
  - specularities (highlights) are view-dependent
  - caused by surfaces that are microscopically smooth
- For shiny surfaces part of the incident light reflects coherently
  - an incoming ray is reflected in a single direction (or narrow beam)
  - direction is defined by the incoming direction and the surface normal
- A mirror is a perfect specular reflector
  - approximate specular reflectors give fuzzy highlights



Phong Illumination

- One function that approximates specular falloff is called the *Phong Illumination* model

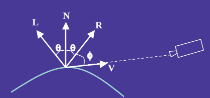
$$I_{specular} = k_s I_{light} (\cos\phi)^{n_{shiny}}$$

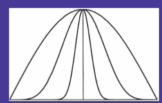
- No real physical basis, yet widespread use in computer graphics

ϕ : Angle between reflected light ray **R** and viewer **V**

k_s : Specular reflectance

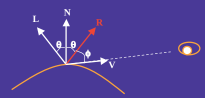
n_{shiny} : Rate of specular falloff

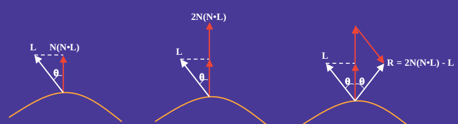




Greater n_{shiny}, more focused beam

Computing the Reflected Ray





Project L onto N | Double length of vector | Subtract L

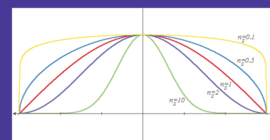
$$|X| = N \bullet L$$

Phong Illumination Curves

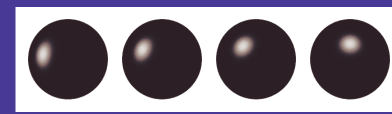
- The specular exponents are often much larger than 1; values of 100 are not uncommon.

$$I_{specular} = k_s I_{light} (\cos\phi)^{n_{shiny}}$$

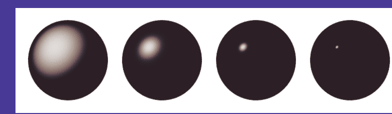
ϕ : angle between line of sight and perfect reflection
k_s : Specular reflectance
n_{shiny} : Rate of specular falloff



Phong Illumination



Moving the light source



Changing n_{shiny}

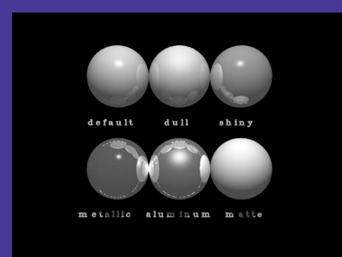
Putting It All Together

- Combining ambient, diffuse, and specular illumination

$$I = k_a I_a + f_{att} I_{light} \left[k_d \cos\theta + k_s (\cos\phi)^{n_{shiny}} \right]$$

- For multiple light sources
  - Repeat the diffuse and specular calculations for each light source
  - Add the components from all light sources
  - The ambient term contributes only once
- The different reflectance coefficients can differ.
  - Simple “metal”: k_a and k_d share material color, k_s is white
  - Simple plastic: k_s also includes material color

Some Examples



OpenGL Materials

```
GLfloat white8[] = {.8, .8, .8, 1.}, white2 = {.2,.2,.2,1.},black={0.,0.,0.};
GLfloat mat_shininess[] = {50.};          /* Phong exponent */

glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, black);
glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, white8);
glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, white2);
glMaterialfv(GL_FRONT_AND_BACK, GL_SHININESS, mat_shininess);
```

OpenGL Lighting

```
GLfloat white[] = {1., 1., 1., 1.};
GLfloat light0_position[] = {1., 1., 5., 0.}; /* directional light (w=0) */

glLightfv(GL_LIGHT0, GL_POSITION, light0_position);
glLightfv(GL_LIGHT0, GL_DIFFUSE, white);
glLightfv(GL_LIGHT0, GL_SPECULAR, white);
glEnable(GL_LIGHT0);

glEnable(GL_NORMALIZE);     /* normalize normal vectors */
glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE);     /* two-sided lighting*/

glEnable(GL_LIGHTING);
```

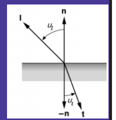

Transmission with Refraction

- Refraction:
  - the bending of light due to its different velocities through different materials
- Refractive index:
  - light travels at speed *c/n* in a material of refractive index *n*
  - *c* is the speed of light in a vacuum
  - varies with wavelength hence rainbows and prisms

MATERIAL	INDEX OF REFRACTION
Air/Vacuum	1
Water	1.33
Glass	about 1.5
Diamond	2.4

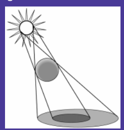
Snell's Law

- Light bends by the physics *principle of least time*, a consequence of *Huygens' Principle*
  - light travels from point A to point B by the fastest path
  - when passing from a material of index n_1 to one of index n_2 *Snell's law* gives the angle of refraction: $n_1 \sin \theta_1 = n_2 \sin \theta_2$ where θ_1 and θ_2 are the angles from perpendicular
- When traveling into a denser material (larger *n*), light bends to be more perpendicular (eg air to water) and vice versa
  - light travels further in the faster material
  - if the indices are the same the light doesn't bend
- When traveling into a less dense material total internal reflection occurs if $\theta_1 > \sin^{-1}(n_2/n_1)$



Shadows

- Shadows occur where objects are hidden from a light source
  - omit any intensity contribution from hidden light sources
- Working out what it hidden is simply a visibility problem
  - can the light source see the object?
  - use the z-buffer shadow algorithm:
    - » run the algorithm from the light source's viewpoint
    - » save the z-buffer as the shadow buffer
    - » run the real z-buffer algorithm, transforming each point into the light source's coordinates and comparing the z value against the shadow buffer



Shading

Given an equation to calculate surface radiance, we still must apply it to the real model
- Usually performed during scan conversion
- There are efficient methods for doing this quickly (which we will discuss in more detail later in the semester


Flat shaded
Gouraud: Normal at vertex is average of normals for adjacent faces
Phong: interpolate normals instead of intensities

Uniformly shaded surfaces are still unrealistic

Real objects have surface features, or texture

One option: use a huge number of polygons with appropriate surface coloring and reflectance characteristics

Texture mapping gets you further
- Assign radiance based on an image

Even better: use *Procedural shaders* to specify any function you want to define radiance
- The possibilities are endless…
- Generate radiance on the fly, during shading
- Key ingredient of high-end rendering systems
  - » Pixar's Renderman (used for "Toy Story", "Bug's Life", etc.)

Break for video…