

Note Title

01/03/2006



Note Title

01/03/2006



Note Title

01/03/2006



## Illumination Models

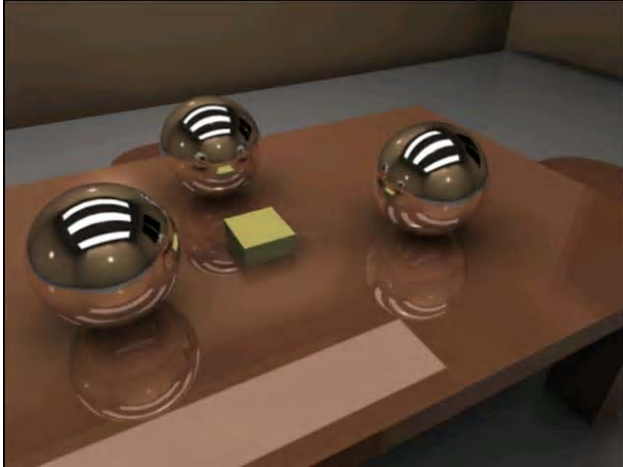


[electricimage.com]

77 K polygons  
24 area lights  
solution render time : around 7200 sec



# Images...



[electricimage.com]

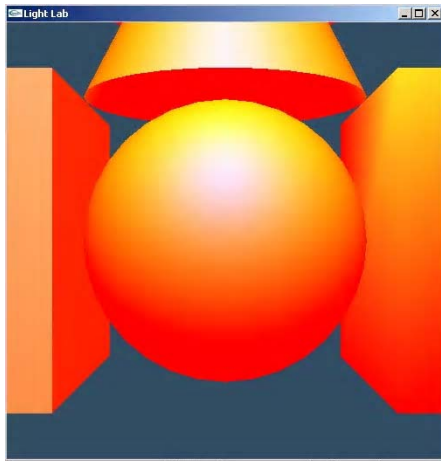


# Local Illumination

## Example



# Demo



# Local Illumination in the projective rendering pipeline

## Local Illumination

- only models light arriving directly from light source
- interreflections and shadows
  - added through tricks, multiple rendering passes

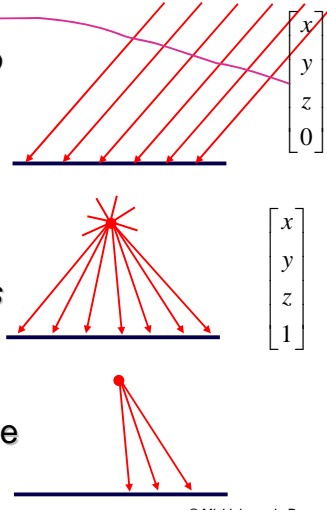
## Types of Models

- Simple, non-physical reflection models (Phong, Blinn)
- physically-based reflection models
  - BRDFs: *Bidirectional Reflection Distribution Functions*

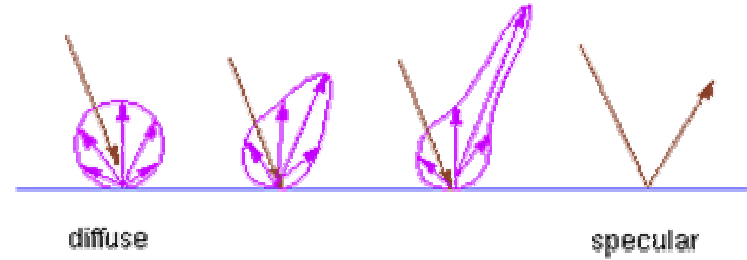
# Light Sources

## Types of light sources

- `glLightfv(GL_LIGHT0, GL_POSITION, light[])`
- Directional/parallel lights
  - E.g. sun
  - Homogeneous vector
- (Homogeneous) point lights
  - Same intensity in all directions
- Spot lights
  - Limited set of directions:
    - ▶ Point+direction+cutoff angle



# Local Illumination



# Commonly used model (simple, non-physical)

## Combine diffuse, specular, ambient

- E.g. OpenGL / graphics hardware:

$$I_{out}(\mathbf{x}) = k_a \cdot I_a + k_d \cdot (\mathbf{l} \cdot \mathbf{n}) I_{diff} + k_s \cdot (\mathbf{h} \cdot \mathbf{n})^n \cdot I_{spec}$$

ambient      diffuse      specular

*Blinn*

*$k_s (R \cdot V)^n I_{spec}$*   
*Phong*

# Materials

## Ambient Light

- Incoming light component that is identical everywhere in the scene
- No direction
- Hack for replacing true global illumination (light bouncing off from other objects)

$$I_{out}(\mathbf{x}) = k_a \cdot I_a$$

*$k_a \in [0, 1]$*   
 *$I_a \in [0, 1]$*

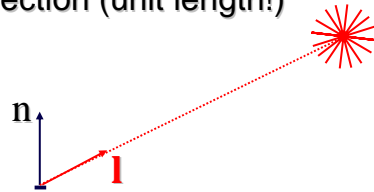
*light source*  
*material property*

# Diffuse component: Lambert's Law



## Johann Friedrich Lambert (1783):

- Power per unit area arriving at some object point  $x$  also depends on the angle of the surface to the light direction
- $dA$ : differential surface area surrounding  $x$
- $\mathbf{l}$ : light direction (unit length!)

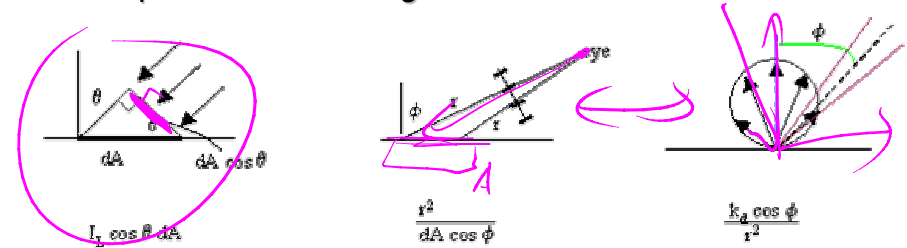


$$I'_{in}(\mathbf{x}) = \cos(\angle(\mathbf{n}, \mathbf{l})) \cdot I_{in}(\mathbf{x}) = (\mathbf{n} \cdot \mathbf{l}) \cdot I_{in}(\mathbf{x})$$

# Diffuse Component: a more detailed look



- independent of viewing direction



$I_D \propto$  incoming energy per unit surface area  
 $\times$  amount of surface area visible through pixel  
 $\times$  fraction of energy sent in the direction of the pixel

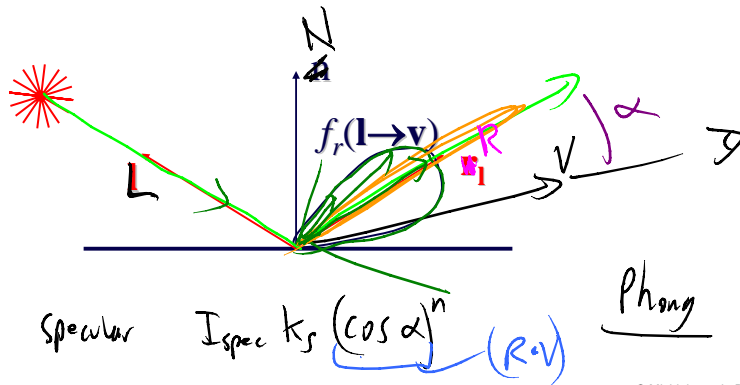
$$= I_L \cos \theta dA \times r^2 / (dA \cos \phi) \times k_D \cos \phi / r^2$$

# Materials



## Specular/Glossy

- Light is mostly reflected into the directions around the mirror direction  $\mathbf{r}_1$  of  $\mathbf{l}$



# Materials



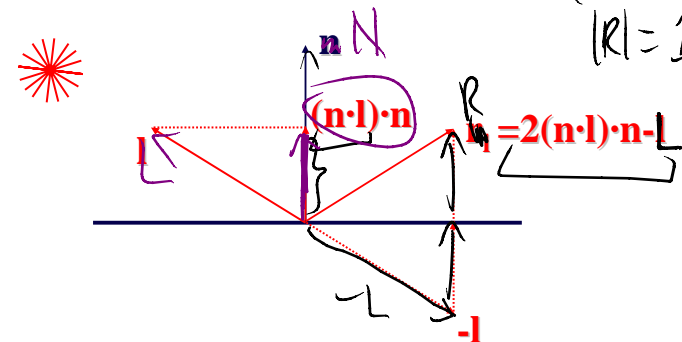
## Specular/Glossy

- Computing reflection direction  $\mathbf{r}_1$  of  $\mathbf{l}$
- $\mathbf{n}$  and  $\mathbf{l}$  are unit length!

$$|\mathbf{l}| = 1$$

$$|\mathbf{n}| = 1$$

$$|\mathbf{r}_1| = 1$$



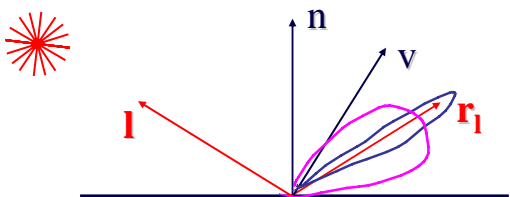
# Materials

## Phong Model (Phong Bui-Tuong, 1975)

- Use cosine power as heuristic

$$I_{spec}(\mathbf{x}) = k_s \cdot (\mathbf{v} \cdot \mathbf{n})^n \cdot I_{in}(\mathbf{x})$$

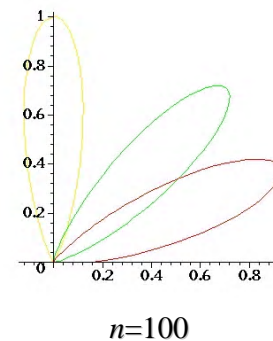
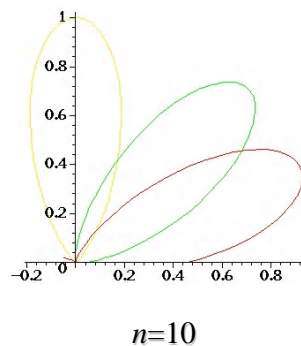
$n \approx 100$   
for highly specular surfaces  
 $n \approx 10$  for moderately specular



# Materials

## Phong model

- Polar plot



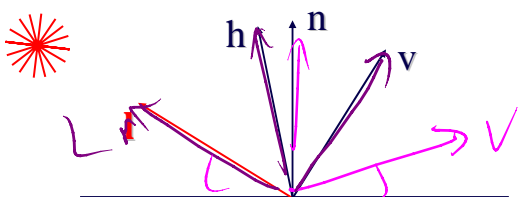
# Materials

## Blinn-Phong model (Jim Blinn, 1977)

- Variation with better physical interpretation
  - $\mathbf{h}$ : halfway vector;  $n$ : shininess

$$I_{out}(\mathbf{x}) = k_s \cdot (\mathbf{h} \cdot \mathbf{n})^n \cdot I_{in}(\mathbf{x}); \text{ with } \mathbf{h} = (\mathbf{l} + \mathbf{v}) / 2$$

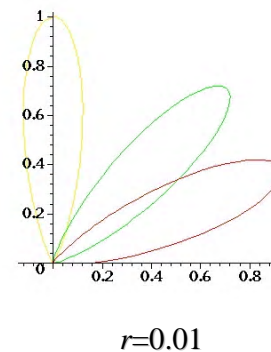
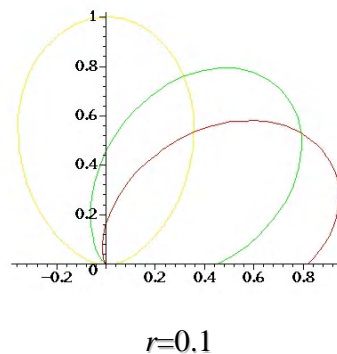
$(L+V)/2$



# Materials

## Blinn-Phong Model

- Polar plot



## Commonly used model (simple, non-physical)



### Combine diffuse, specular, ambient

- E.g. OpenGL / graphics hardware:

$$I_{out}(\mathbf{x}) = \underbrace{k_a}_{\text{ambient}} \cdot I_a + \underbrace{k_d}_{\text{diffuse}} \cdot (\mathbf{l} \cdot \mathbf{n}) \cdot I_{diff} + \underbrace{k_s}_{\text{specular}} \cdot (\mathbf{h} \cdot \mathbf{n})^n \cdot I_{spec}$$

## Lighting in OpenGL



- Light source: amount of RGB light emitted
  - value represents percentage of full intensity, e.g., (1.0,0.5,0.5)
  - every light source emits ambient, diffuse, and specular light
- Materials: amount of RGB light reflected
  - value represents percentage reflected e.g., (0.0,1.0,0.5)

## Lighting in OpenGL



```
glLightfv(GL_LIGHT0, GL_AMBIENT, amb_light_rgba );
glLightfv(GL_LIGHT0, GL_DIFFUSE, dif_light_rgba );
glLightfv(GL_LIGHT0, GL_SPECULAR, spec_light_rgba );
glLightfv(GL_LIGHT0, GL_POSITION, position);
glEnable(GL_LIGHT0);

glMaterialfv( GL_FRONT, GL_AMBIENT, ambient_rgba );
glMaterialfv( GL_FRONT, GL_DIFFUSE, diffuse_rgba );
glMaterialfv( GL_FRONT, GL_SPECULAR, specular_rgba );
glMaterialfv( GL_FRONT, GL_SHININESS, n );
```

```
void setProperties(void)
{
    // Default values for material and light properties.
    GLfloat mat_ambient[] = { 0.2,0.2,0.2,1};
    GLfloat mat_diffuse[] = { 0.6, 0.6, 0.6, 1};
    GLfloat mat_specular[] = { 0.5, 0.5 ,0.5, 1.0 };

    GLfloat lightAmbient[] = { 1,1,1,1};
    GLfloat lightDiffuse[] = { 1,1,1,1};
    GLfloat lightSpecular[] = { 1,1,1,1};
    GLfloat lightZero[] = {0.0,0.0,1};

    glShadeModel (GL_SMOOTH);

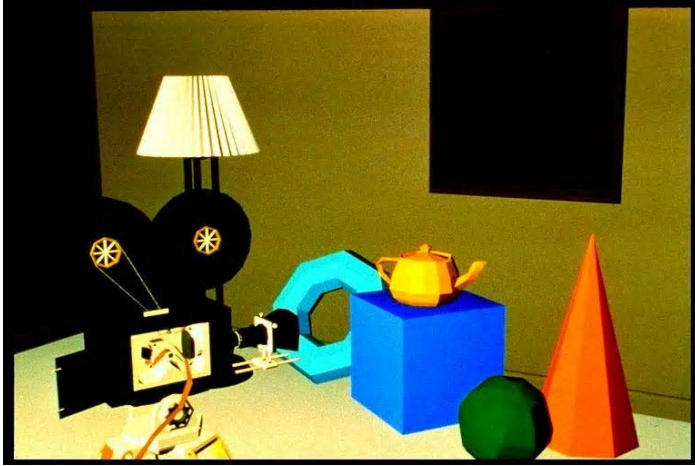
    // set material properties
    glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient); // material
    glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
    glMaterialfv(GL_FRONT, GL_SHININESS, &Shininess);

    // set light properties
    glLightModelfv( GL_LIGHT_MODEL_AMBIENT, lightZero);
    glLightfv(GL_LIGHT0, GL_AMBIENT, lightAmbient);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, lightDiffuse);
    glLightfv(GL_LIGHT0, GL_SPECULAR, lightSpecular);

    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_NORMALIZE);
}
```

## Flat Shading

### Example:



## Gouraud Shading

### Example:



## Materials

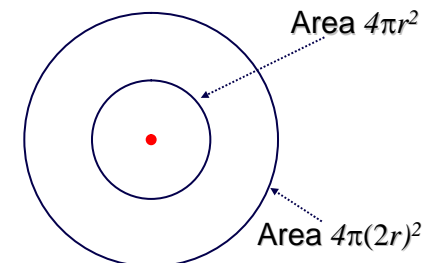
### Summary

- Very simple reflection models
- Fast (dot products & exponentiation)
- No physical justification
- Not very good for modeling real world

## Light Sources

### Quadratic falloff

- Brightness of objects depends on power per unit area that hits the object
- The power per unit area for a point or spot light decreases quadratically with distance



# Light Sources

## Non-quadratic falloff:

- Many systems allow for other falloffs
- Allows for faking of the effect of area light sources
- OpenGL / graphics hardware:
  - $I_o$ : intensity of light source
  - $\mathbf{x}$ : object point
  - $r$ : distance of light from  $\mathbf{x}$

$$I_{in}(\mathbf{x}) = \frac{1}{ar^2 + br + c} \cdot I_0$$

# Materials

## Bi-directional Reflectance Distribution Function (BRDF):

- Describes fraction of light reflected for all combinations of incoming (light) and outgoing (viewing) directions
- Color channels (R, G, B) are treated separately
  - Actually: wavelengths (see later in course)

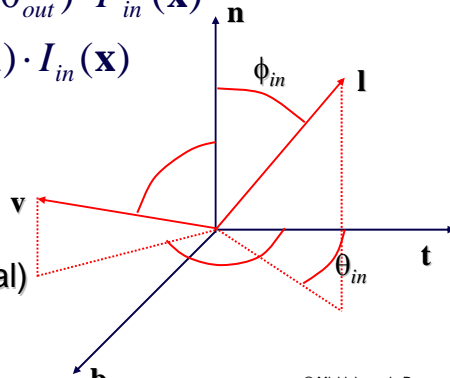
# Materials

## Bi-directional Reflectance Distribution Function (BRDF):

$$I_{out}(\mathbf{x}) = f_r(\phi_{in}, \theta_{in}, \phi_{out}, \theta_{out}) \cdot I'_{in}(\mathbf{x})$$

$$= f_r(\mathbf{l} \rightarrow \mathbf{v}) \cdot (\mathbf{n} \cdot \mathbf{l}) \cdot I_{in}(\mathbf{x})$$

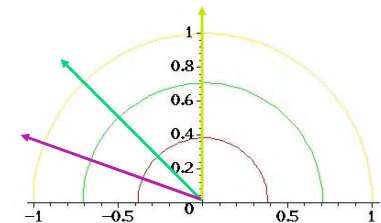
- $f_r(\mathbf{l} \rightarrow \mathbf{v})$  is called BRDF
- $(\mathbf{t}, \mathbf{n}, \mathbf{b})$  is local coordinate frame (normal, tangent, binormal)



# Materials

## Polar plot of BRDF

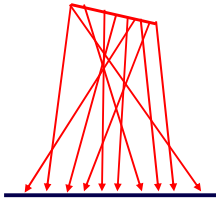
- Fix incoming light direction  $\mathbf{l}$
- Plot  $f_r(\mathbf{l} \rightarrow \mathbf{v}) \cdot \mathbf{v}$  for all viewing directions  $\mathbf{v}$
- Works for 2D and 3D plots
- Example: 2D polar plot for diffuse BRDF



## Light Sources

### **Area lights:**

- light sources with a finite area
- more realistic model of many light sources
- Not available with projective rendering pipeline, (i.e., not available with OpenGL)



## Gouraud Shading

### **Mach Bands:**

- Eye enhances discontinuity in first derivative
- Very disturbing, especially for highlights

