

CS559: Computer Graphics

Lecture 16: Shading and OpenGL

Li Zhang

Spring 2008

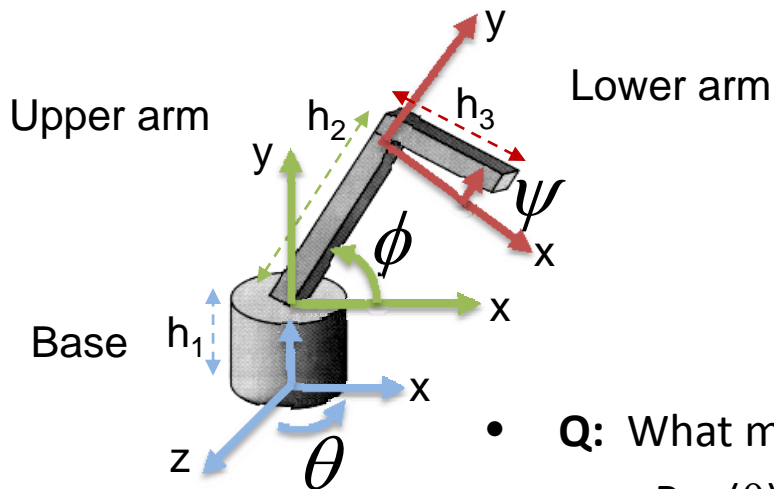
Today

- Finish shading
- How to do shading in OpenGL

- Reading
 - Shirley, Ch 13.3
 - Red book, Ch 4&5 (except color index mode)

3D Example: A robot arm

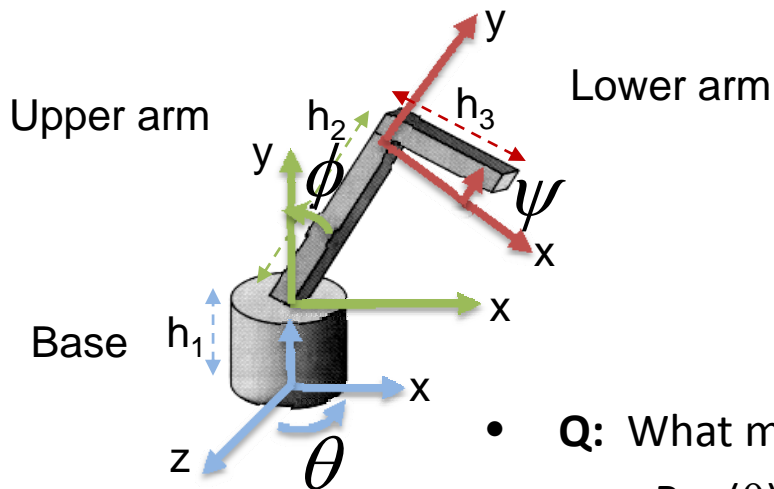
- Consider this robot arm with 3 degrees of freedom:
 - Base rotates about its vertical axis by θ
 - Upper arm rotates in its xy -plane by ϕ
 - Lower arm rotates in its xy -plane by ψ



- **Q:** What matrix do we use to transform the base to the world?
 - $R_y(\theta)$
- **Q:** What matrix for the upper arm to the base?
 - $T(0, h_1, 0)R_z(\phi)$
- **Q:** What matrix for the lower arm to the upper arm?
 - $T(0, h_2, 0)R_z(\psi)$

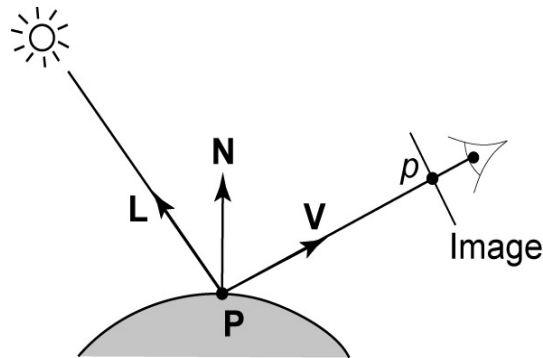
3D Example: A robot arm

- Consider this robot arm with 3 degrees of freedom:
 - Base rotates about its vertical axis by θ
 - Upper arm rotates in its xy -plane by ϕ
 - Lower arm rotates in its xy -plane by ψ



- **Q:** What matrix do we use to transform the base to the world?
 - $R_y(\theta)$
- **Q:** What matrix for the upper arm to the base?
 - $T(0, h_1, 0)R_z(\phi)$
- **Q:** What matrix for the lower arm to the upper arm?
 - $T(0, h_2, 0)R_z(\psi)$

Shading problem

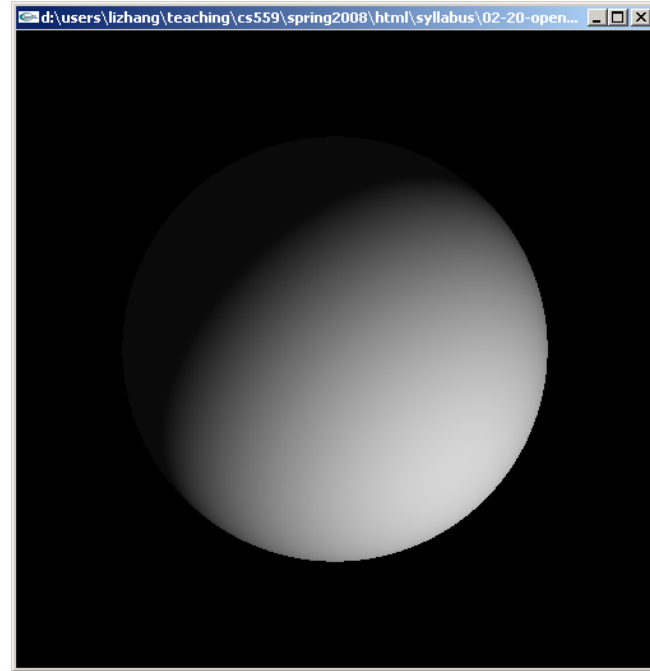
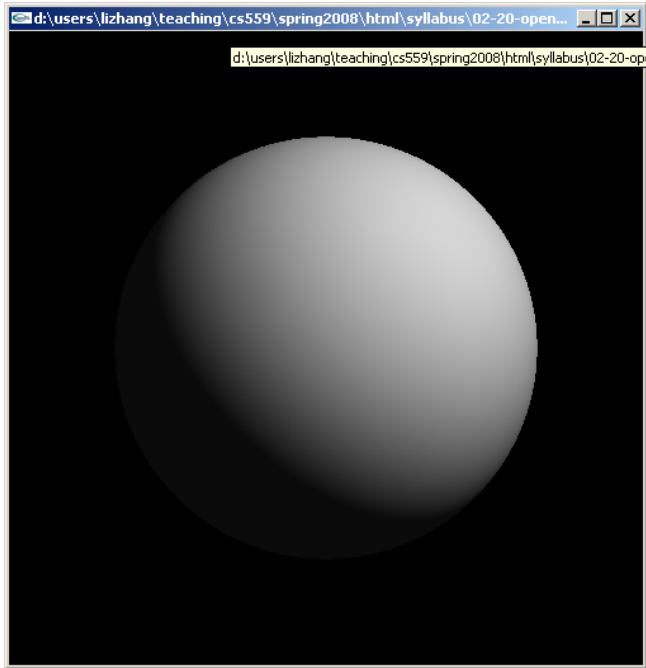


$$\|\mathbf{N}\| = \|\mathbf{L}\| = \|\mathbf{V}\| = 1$$

- Given:
 - a point \mathbf{P} on a surface visible through pixel p
 - The normal \mathbf{N} at \mathbf{P}
 - The lighting direction, \mathbf{L} , and intensity, L , at \mathbf{P}
 - The viewing direction, \mathbf{V} , at \mathbf{P}
 - The shading coefficients at \mathbf{P}
- Compute the color, I , of pixel p .

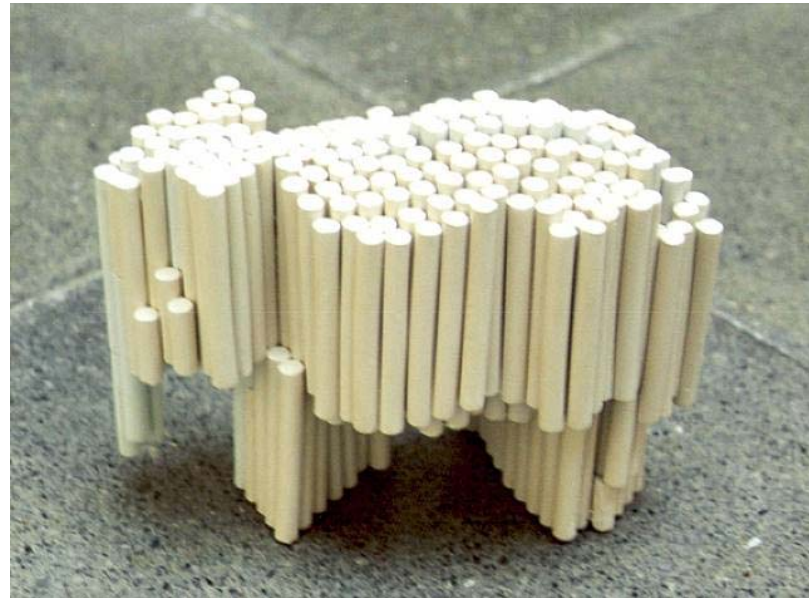
$$I = k_e + k_a L_a + k_d L_d \cdot \max(0, \mathbf{L} \cdot \mathbf{N})$$

Diffuse Shading



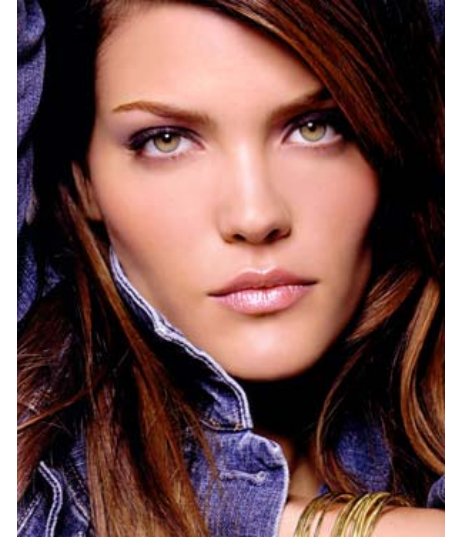
$$I = k_e + k_a L_a + k_d L_d \cdot \max(0, \mathbf{L} \cdot \mathbf{N})$$

Diffuse Shading



$$I = k_e + k_a L_a + k_d L_d \cdot \max(0, \mathbf{L} \cdot \mathbf{N})$$

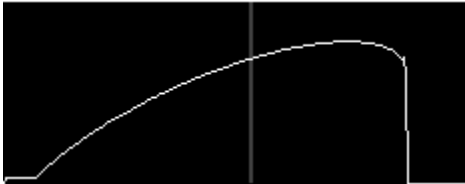
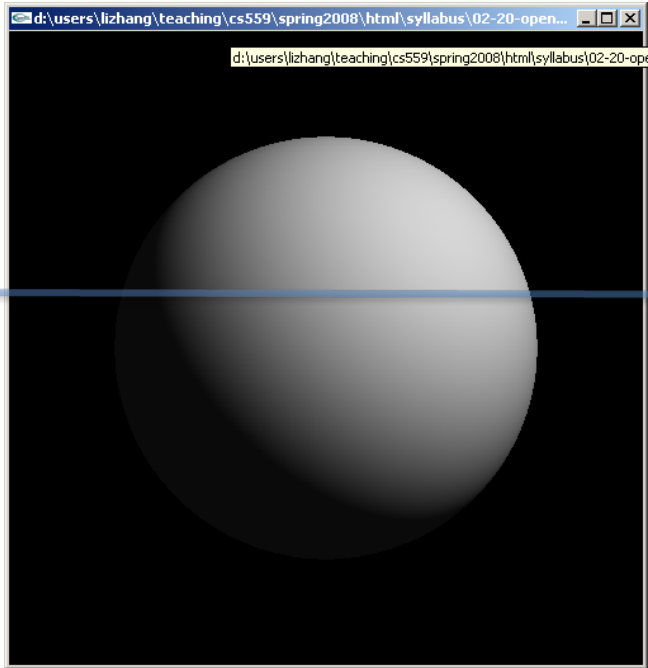
Specular reflection



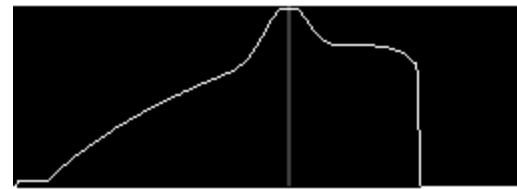
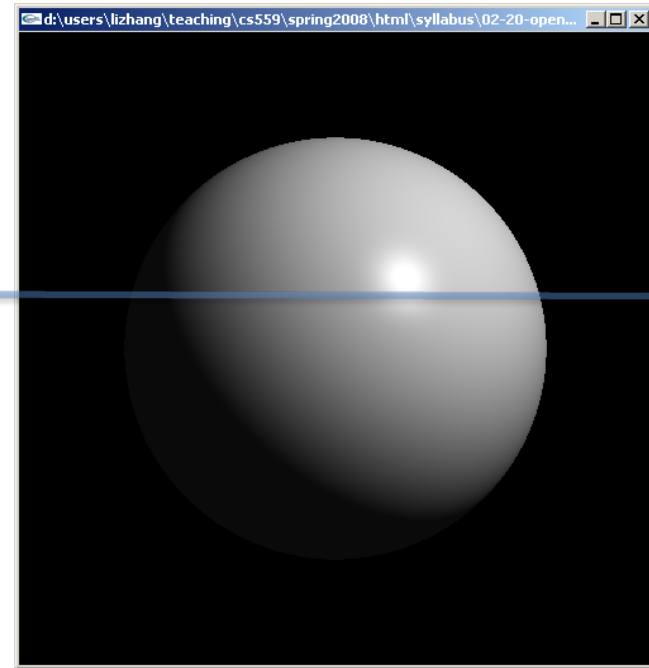
- **Specular reflection** accounts for the highlight that you see on some objects.
- It is particularly important for *smooth, shiny* surfaces, such as:
 - Metal, polished stone, plastics, apples, Skin

Specular Reflection

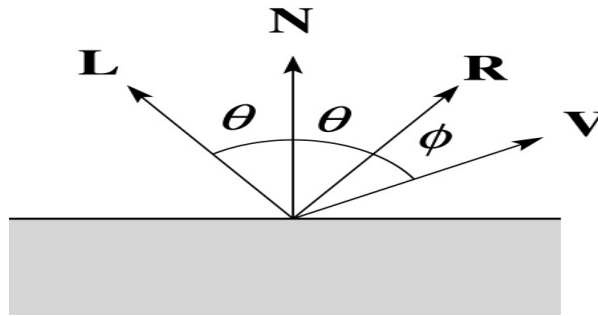
Diffuse



Specular



Specular reflection “derivation”

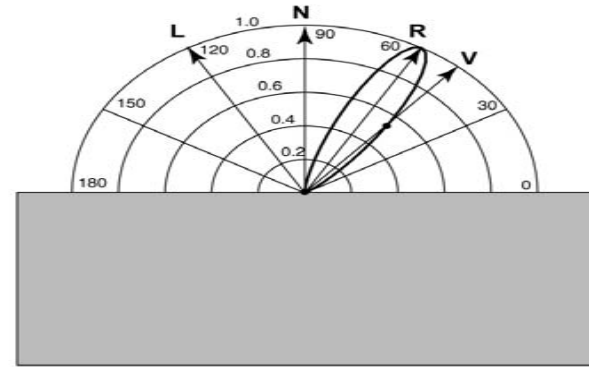
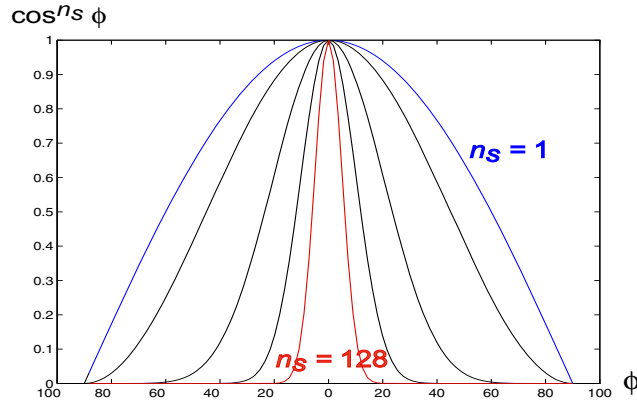


- For a perfect mirror reflector, light is reflected about **N**, so

$$I = \begin{cases} L & \text{if } \mathbf{V} = \mathbf{R} \\ 0 & \text{o t h e r w i s e} \end{cases}$$

- For a near-perfect reflector, you might expect the highlight to fall off quickly with increasing angle ϕ .
- Also known as:
 - “rough specular” reflection
 - “directional diffuse” reflection
 - “glossy” reflection

Derivation, cont.



- One way to get this effect is to take $(R \cdot V)$, raised to a power n_s .
- As n_s gets larger,
 - the dropoff becomes {more,less} gradual
 - gives a {larger,smaller} highlight
 - simulates a {more,less} mirror-like surface

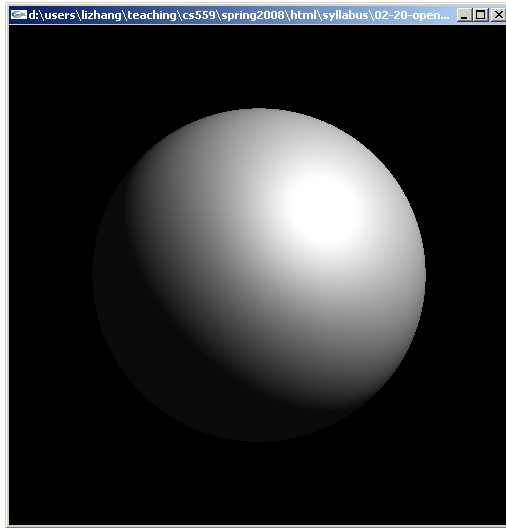
“Iteration three”

- The next update to the Phong shading model is then:

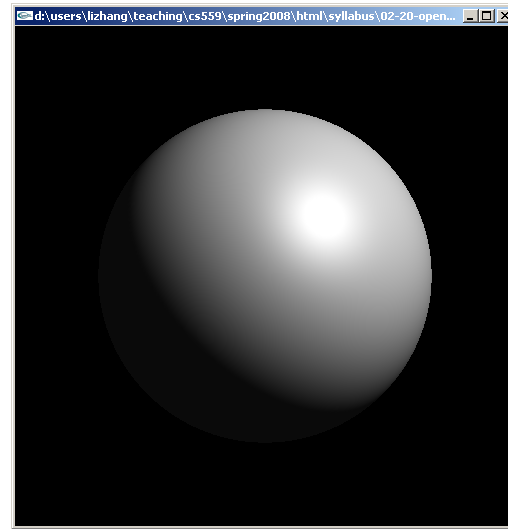
$$I = k_e + k_a L_a + k_d L_d \cdot \max(0, \mathbf{L} \cdot \mathbf{N}) + k_s L_s \cdot \max(0, \mathbf{V} \cdot \mathbf{R})^{n_s}$$

- where:
 - k_s is the **specular reflection coefficient**
 - n_s is the **specular exponent** or **shininess**
 - \mathbf{R} is the reflection of the light about the normal (unit vector)
 - \mathbf{V} is viewing direction (unit vector)

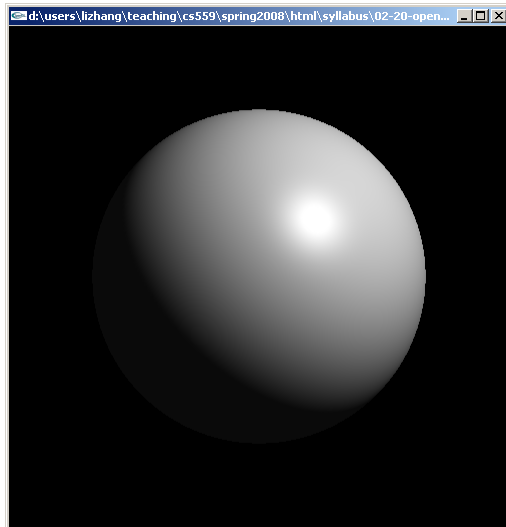
Shininess



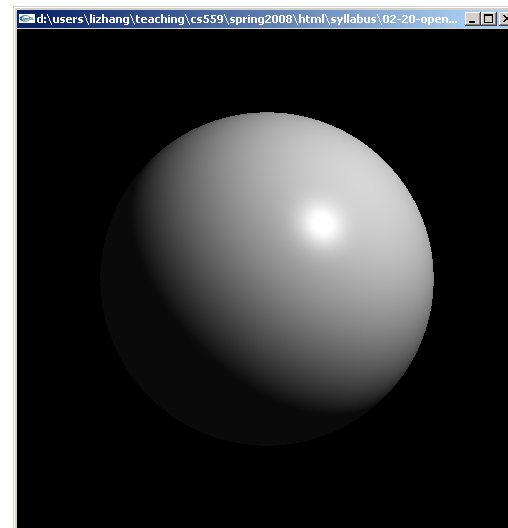
$N_s=10$



$N_s=20$



$N_s=50$



$N_s=100$

Specular vs Diffuse reflection

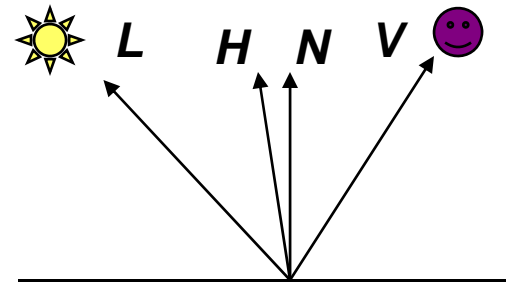
- What's the key difference Properties:
 - Specular reflection depends on the viewing direction \mathbf{V} .

Specular Reflection Improvement

$$I = k_e + k_a L_a + k_d L_d \cdot \max(0, \mathbf{L} \cdot \mathbf{N}) + k_s L_s \cdot \max(0, \mathbf{V} \cdot \mathbf{R})^{n_s}$$

$$\mathbf{H} = (\mathbf{L} + \mathbf{V}) / \|\mathbf{L} + \mathbf{V}\|$$

$$k_s L_s (\mathbf{H} \cdot \mathbf{N})^p$$



- Compute based on normal vector and “halfway” vector, \mathbf{H}
 - Always positive when the light and eye are above the tangent plane
 - Not quite the same result as the other formulation

Putting It Together

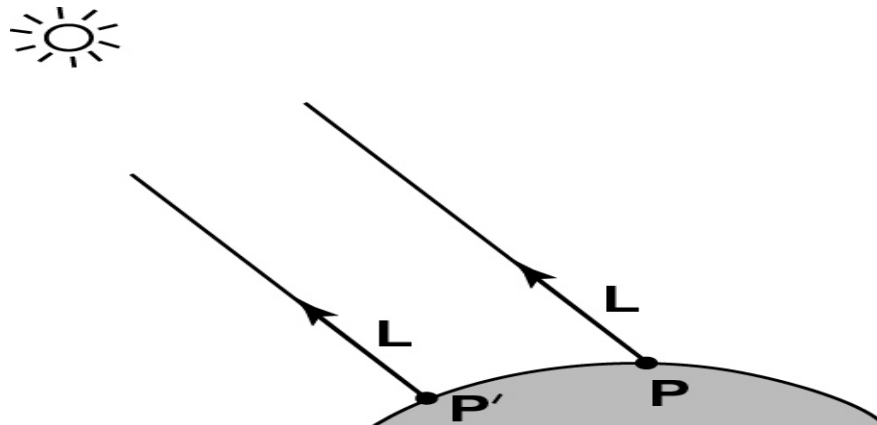
Phong Shading Model

$$I = k_e + k_a L_a + k_d L_d \cdot \max(0, \mathbf{L} \cdot \mathbf{N}) + k_s L_s \cdot (\mathbf{H} \cdot \mathbf{N})^{n_s}$$

```
GLfloat ke[] = { 0.1, 0.15, 0.05, 1.0 };
GLfloat ka[] = { 0.1, 0.15, 0.1, 1.0 };
GLfloat kd[] = { 0.3, 0.3, 0.2, 1.0 };
GLfloat ks[] = { 0.2, 0.2, 0.2, 1.0 };
GLfloat ns[] = { 50.0 };
glMaterialfv(GL_FRONT, GL_EMISSION, ke);
glMaterialfv(GL_FRONT, GL_AMBIENT, ka);
glMaterialfv(GL_FRONT, GL_DIFFUSE, kd);
glMaterialfv(GL_FRONT, GL_SPECULAR, ks);
glMaterialfv(GL_FRONT, GL_SHININESS, ns);
```

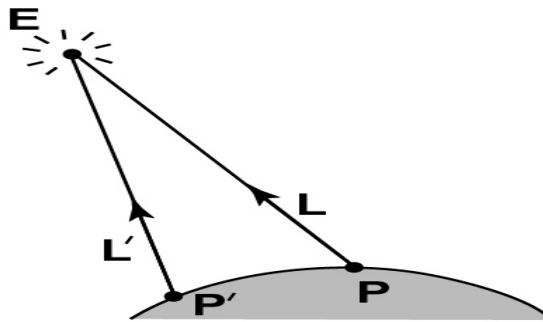

Lights

- OpenGL supports three different kinds of lights: ambient, directional, and point. Spot lights are also supported as a special form of point light.
- We've seen ambient light sources, which are not really geometric.
- **Directional light** sources have a single direction and intensity associated with them.



Point lights

- The direction of a **point light** source is determined by the vector from the light position to the surface point.



$$L = \frac{E - P}{\|E - P\|}$$

$$d = \|E - P\|$$

- Physics tells us the intensity must drop off inversely with the square of the distance:

$$f_{\text{atten}} = \frac{1}{d^2}$$

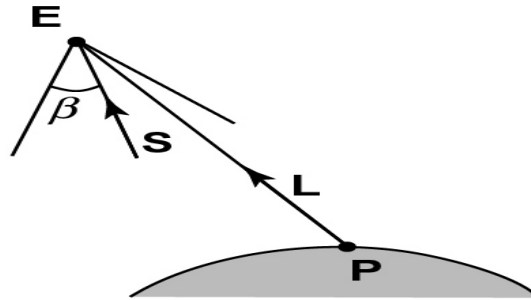
- Sometimes, this distance-squared dropoff is considered too “harsh.” A common alternative is:

$$f_{\text{atten}} = \frac{1}{a + b d + c d^2}$$

- with user-supplied constants for a , b , and c .

Spotlights

- OpenGL also allows one to apply a *directional attenuation* of a point light source, giving a **spotlight** effect



- The spotlight intensity factor is computed in OpenGL as:

$$f_{spot} = [\max\{\beta - \arccos(\mathbf{L} \cdot \mathbf{S}), 0\}]^e$$

- where
 - \mathbf{L} is the direction to the point light.
 - \mathbf{S} is the center direction of the spotlight.
 - β is the cutoff angle for the spotlight
 - e is the angular falloff coefficient

“Iteration four”

- Since light is additive, we can handle multiple lights by taking the sum over every light.
- Our equation is now:

$$I = k_e + k_a L_a + \sum_j \frac{f_{spot_j}}{a_j + b_j D + c_j D^2} \left[k_d L_{d_j} \cdot \max(0, \mathbf{L} \cdot \mathbf{N}) + k_s L_{s_j} \cdot (\mathbf{H} \cdot \mathbf{N})^{n_s} \right]$$

- This is the Phong illumination model in OpenGL.
- Which quantities are spatial vectors?
- Which are RGB triples?

Choosing the parameters

- Experiment with different parameter settings. To get you started, here are a few suggestions:
 - Try n_s in the range [0,100]
 - Try $k_a + k_d + k_s < 1$
 - Use a small k_a (~ 0.1)

	n_s	k_d	k_s
Metal	large	Small, color of metal	Large, color of metal
Plastic	medium	Medium, color of plastic	Medium, white
Planet	0	varying	0

Shading in OpenGL

- The OpenGL lighting model allows you to associate different lighting colors according to material properties they will influence.
- Thus, our original shading equation becomes:

$$I = k_e + k_a L_a + \sum_j \frac{f_{spot_j}}{a_j + b_j D + c_j D^2} \left[k_a L_{a_j} + k_d L_{d_j} \cdot \max(0, \mathbf{L} \cdot \mathbf{N}) + k_s L_{s_j} \cdot (\mathbf{H} \cdot \mathbf{N})^{n_s} \right]$$

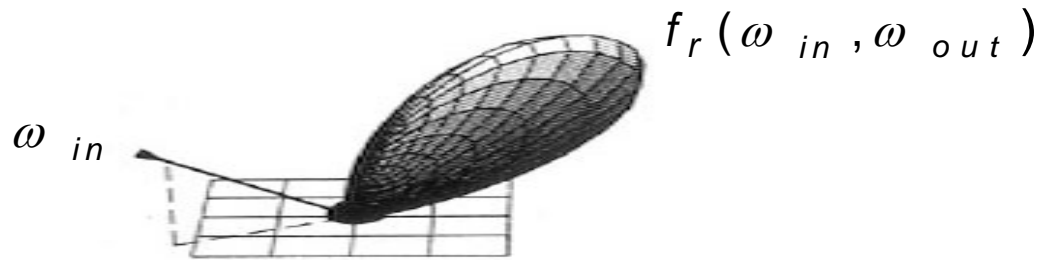
- where you can have a global ambient light with intensity L_a in addition to having an ambient light intensity L_{a_j} associated with each individual light.

BRDF

- The Phong illumination model is a function that maps light from incoming (light) directions ω_{in} to outgoing (viewing) directions ω_{out} :

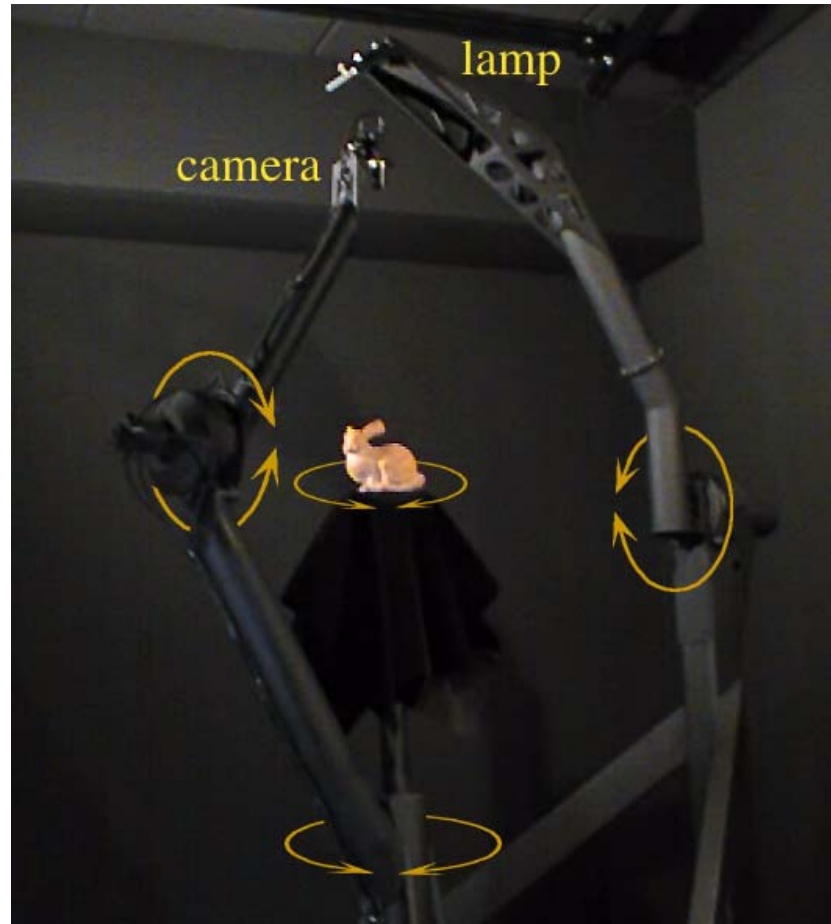
$$f_r(\omega_{in}, \omega_{out})$$

- Here's a plot with ω_{in} held constant:



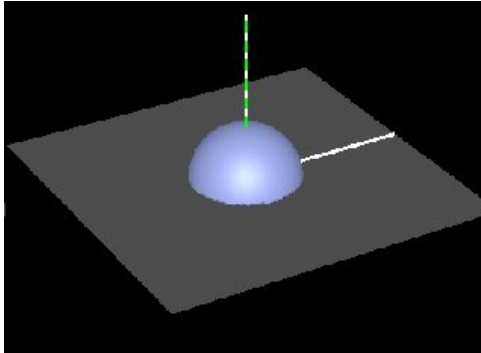
- This function is called the **Bi-directional Reflectance Distribution Function (BRDF)**.
- BRDF's can be quite sophisticated...

BRDF measurement

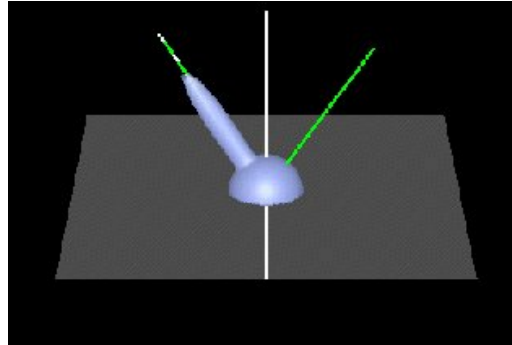


Stanford Graphics Lab

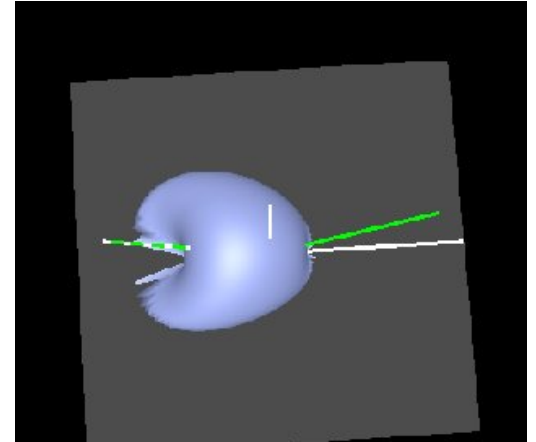
Brdf Viewer plots



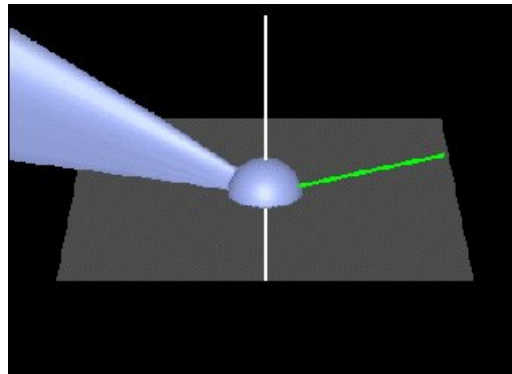
Diffuse



Torrance-Sparrow

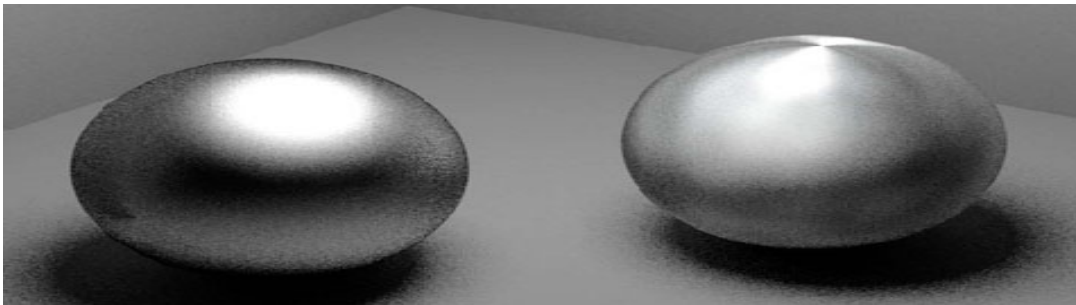
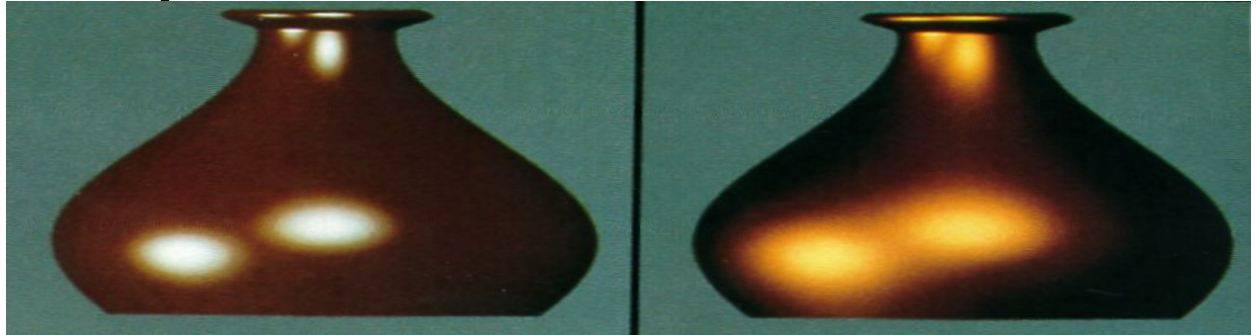


Anisotropic



More sophisticated BRDF's

Cook and
Torrance, 1982



Westin, Arvo, Torrance 1992

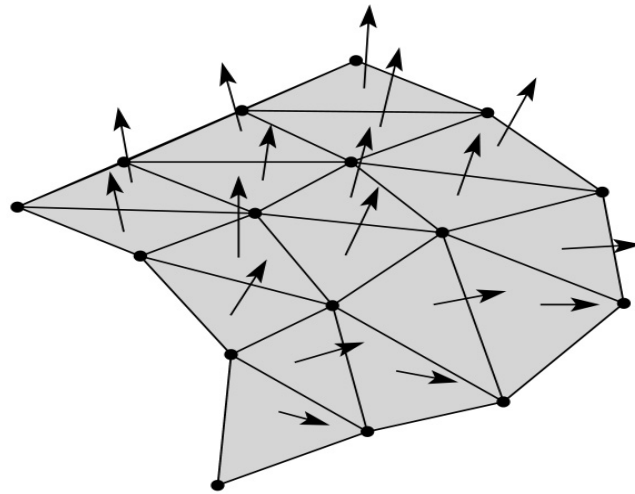


Surface Shading

- Now we know how to compute the color at a point on a surface using the Phong lighting model.
- Does graphics hardware do this calculation at every point? Typically not (although this is changing)...
- Smooth surfaces are often approximated by polygonal facets. So How do we compute the shading for such a surface?

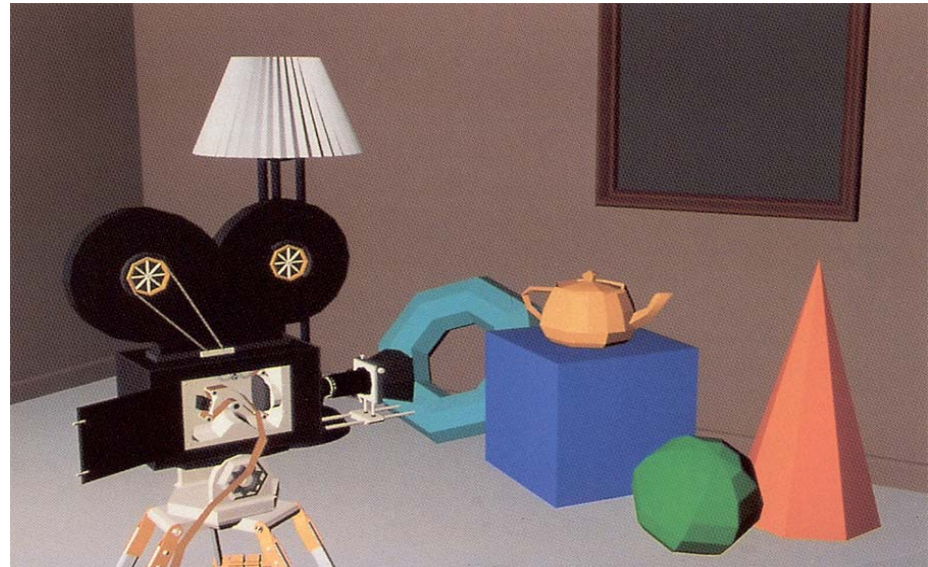
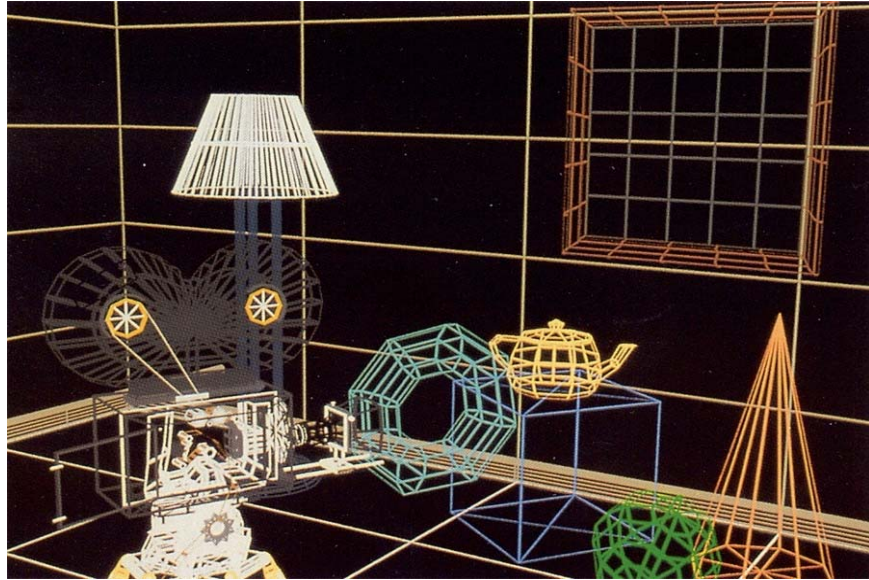
Faceted shading

- Assume each face has a constant normal:



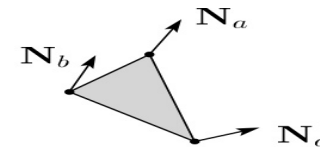
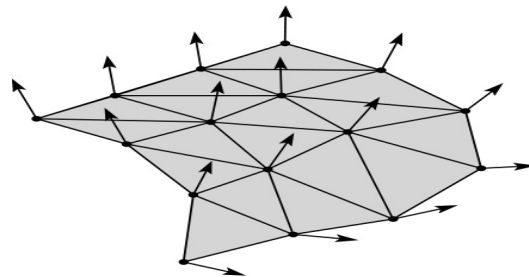
- If we have constant material properties over the surface, how will the color of each triangle vary?
- Result: faceted, not smooth, appearance.

Faceted shading (cont'd)

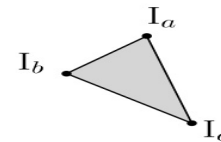


Gouraud interpolation

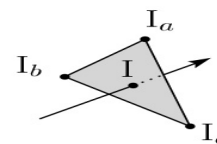
- To get a smoother result that is easily performed in hardware, we can do **Gouraud interpolation**.
- Here's how it works:
 1. Compute normals at the vertices.
 2. Shade only the vertices.
 3. Interpolate the resulting vertex colors.



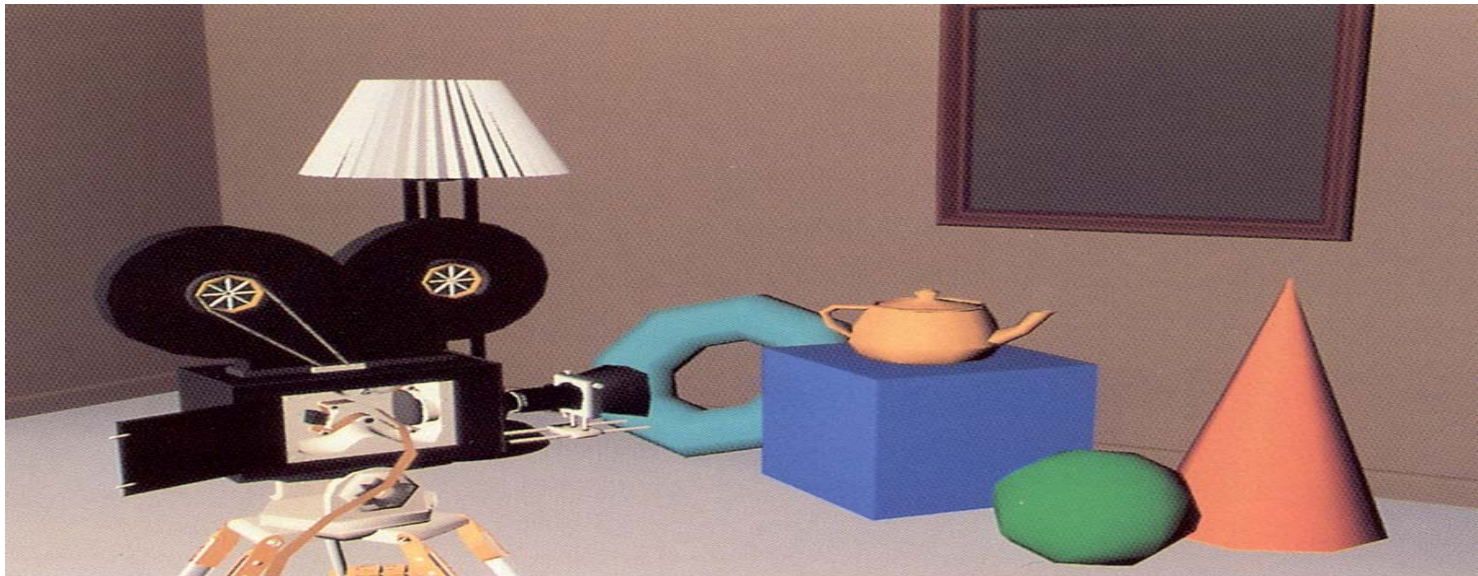
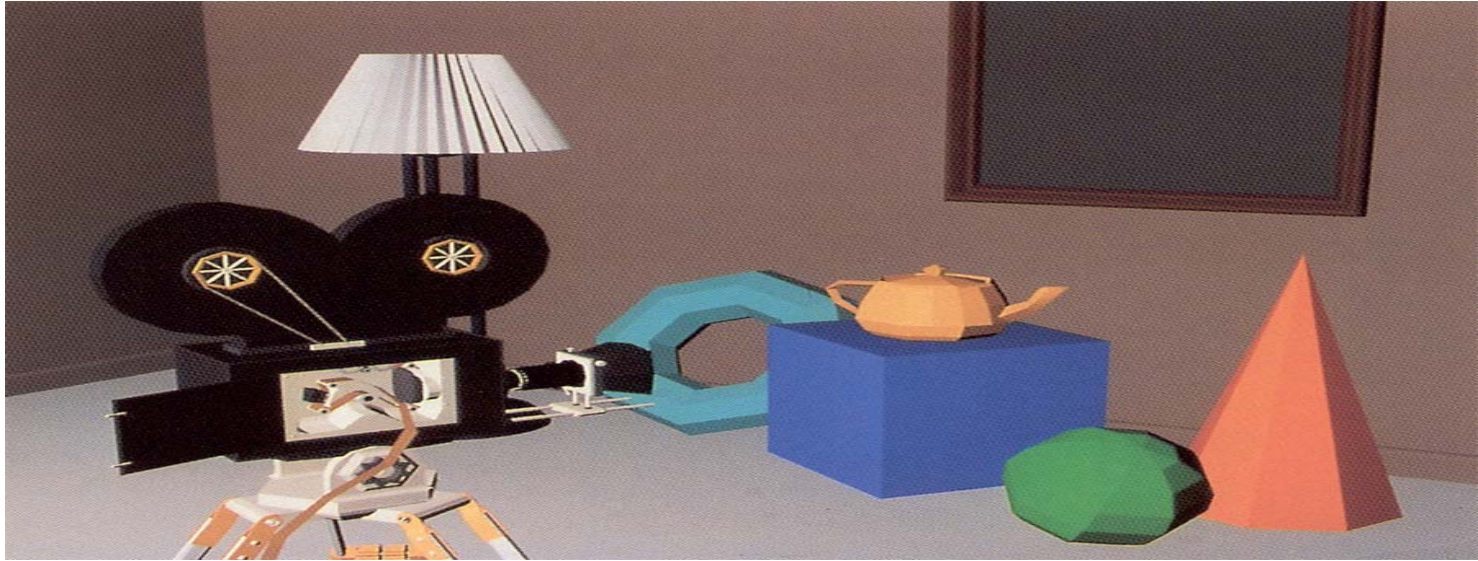
Shade



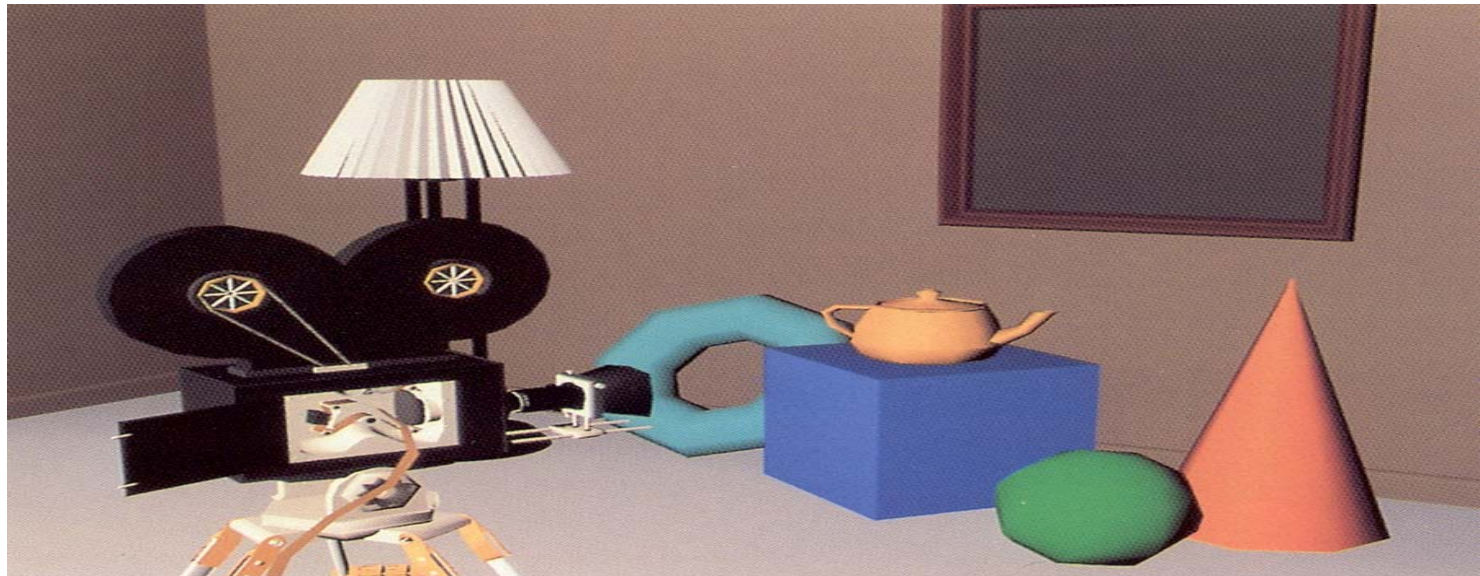
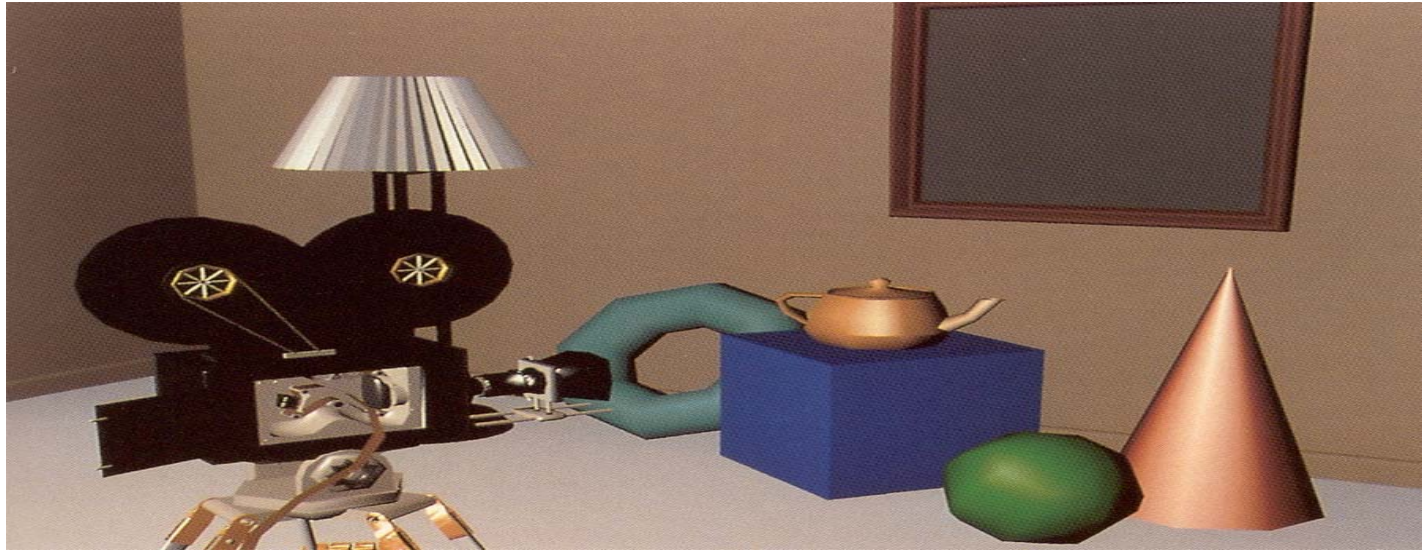
Interpolate



Faced shading vs. Gouraud interpolation

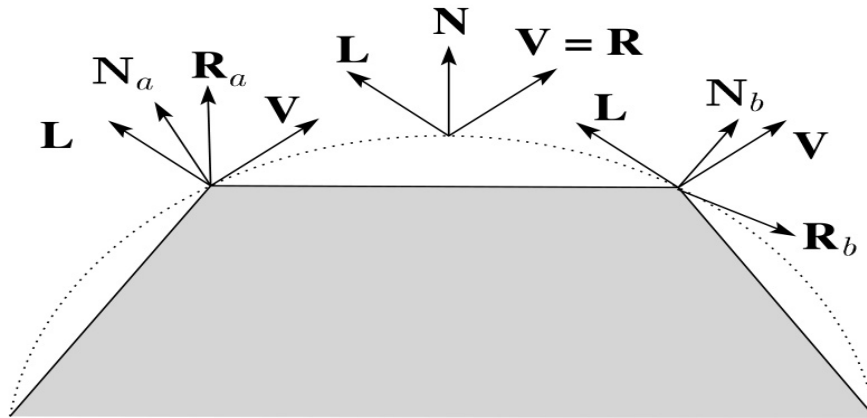


Gouraud interpolation



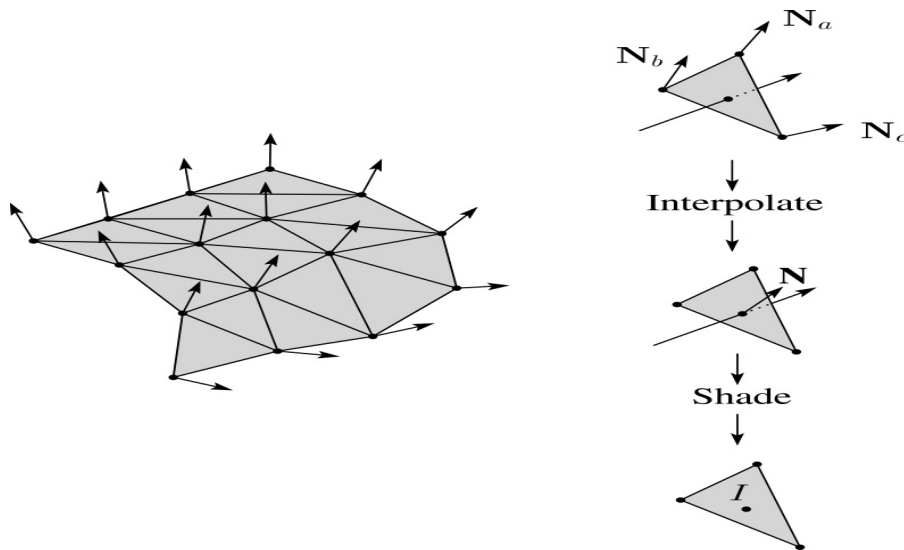
Gouraud interpolation artifacts

- Gouraud interpolation has significant limitations.
 - If the polygonal approximation is too coarse, we can miss specular highlights.

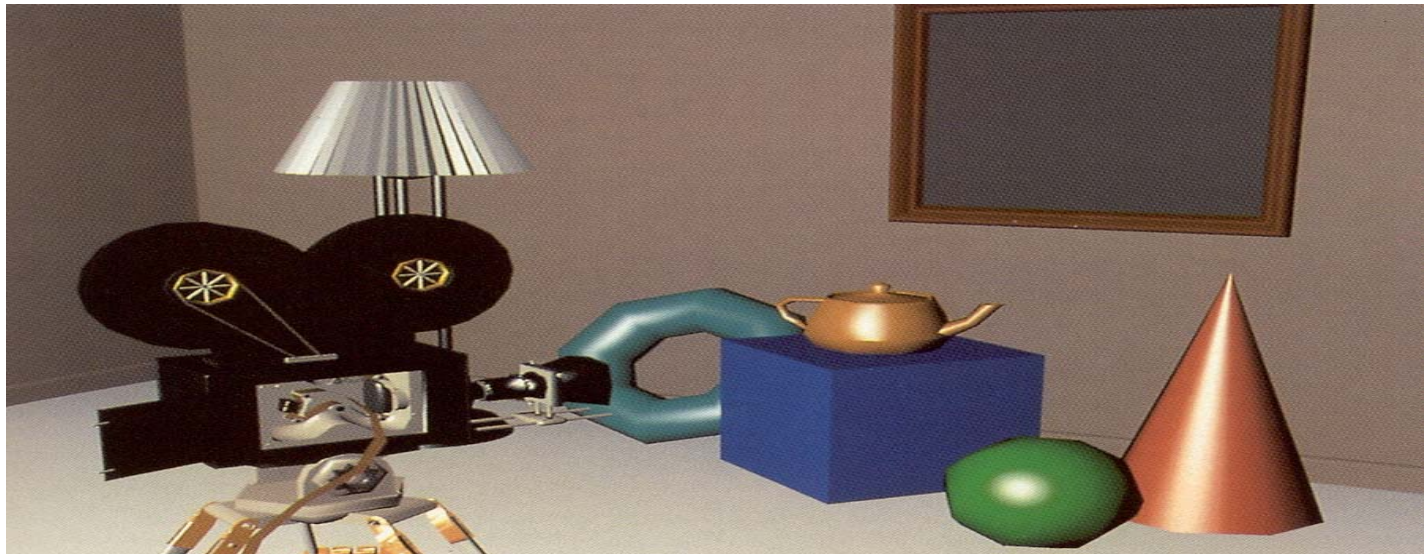
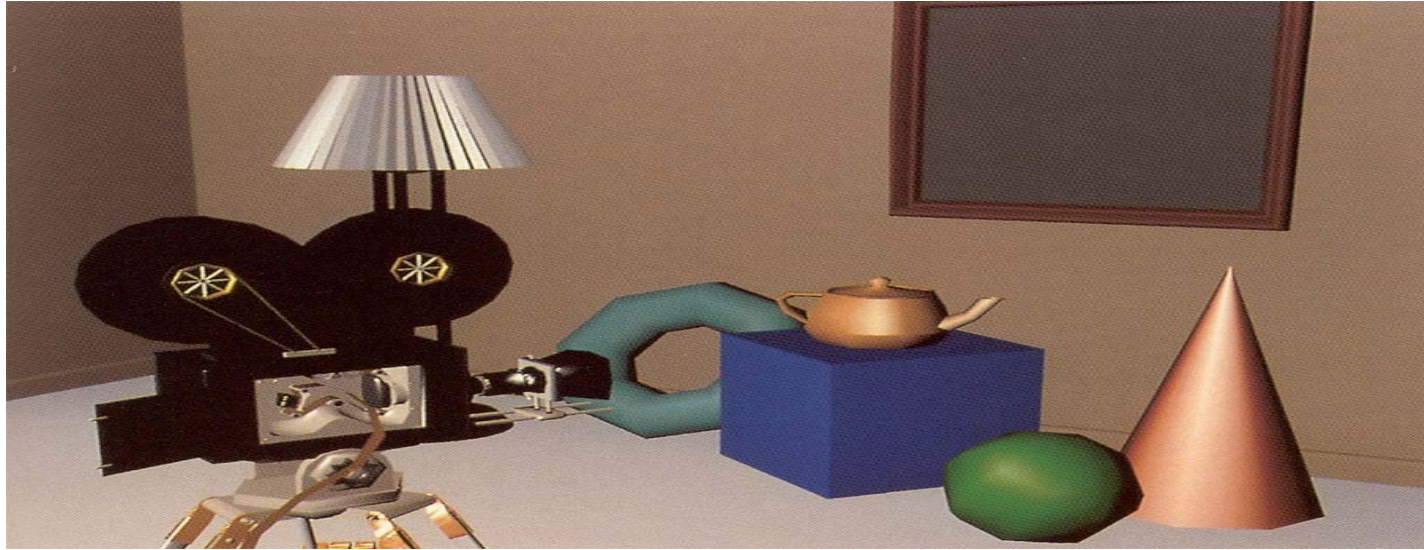


Phong interpolation

- To get an even smoother result with fewer artifacts, we can perform **Phong interpolation**.
- Here's how it works:
 1. Compute normals at the vertices.
 2. Interpolate normals and normalize.
 3. Shade using the interpolated normals.

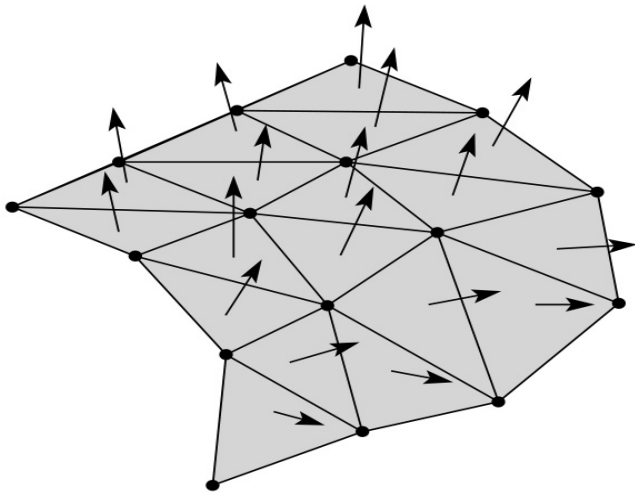


Gouraud vs. Phong interpolation



How to compute vertex normals

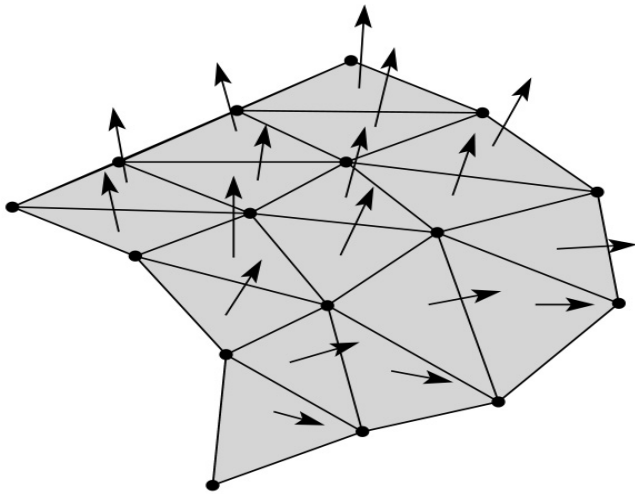
A weighted average of normals of neighboring triangles



$$\mathbf{n}_{vertex} = \frac{\sum_{triangle} area_{triangle} \mathbf{n}_{triangle}}{\sum_{triangle} area_{triangle}}$$

How to compute vertex normals

A weighted average of normals of neighboring triangles



$$\mathbf{n}_{vertex} = \frac{\sum_{triangle} area_{triangle} \mathbf{n}_{triangle}}{\left\| \sum_{triangle} area_{triangle} \mathbf{n}_{triangle} \right\|}$$

Define a light in OpenGL

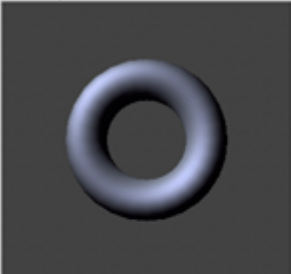
```
GLfloat ke[] = { 0.1, 0.15, 0.05, 1.0 };
GLfloat ka[] = { 0.1, 0.15, 0.1, 1.0 };
GLfloat kd[] = { 0.3, 0.3, 0.2, 1.0 };
GLfloat ks[] = { 0.2, 0.2, 0.2, 1.0 };
GLfloat ns[] = { 50.0 };
glMaterialfv(GL_FRONT, GL_EMISSION, ke);
glMaterialfv(GL_FRONT, GL_AMBIENT, ka);
glMaterialfv(GL_FRONT, GL_DIFFUSE, kd);
glMaterialfv(GL_FRONT, GL_SPECULAR, ks);
glMaterialfv(GL_FRONT, GL_SHININESS, ns);
```

$$I = k_e + k_a L_a + k_d L_d \cdot \max(0, \mathbf{L} \cdot \mathbf{N}) + k_s L_s \cdot (\mathbf{H} \cdot \mathbf{N})^{n_s}$$

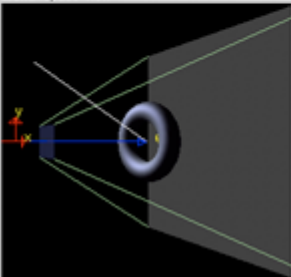
```
GLfloat light_ambient[] = { 0.0, 0.0, 0.0, 1.0 };
GLfloat light_diffuse[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat light_specular[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat light_position[] = { 1.0, 1.0, 1.0, 0.0 };
glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
```

Demo

Screen-space view



World-space view



Command manipulation window

```
GLfloat light_pos[] = { -2.00 , 2.00 , 2.00 , 1.00 };
GLfloat light_Ka[] = { 0.00 , 0.00 , 0.00 , 1.00 };
GLfloat light_Kd[] = { 1.00 , 1.00 , 1.00 , 1.00 };
GLfloat light_Ks[] = { 1.00 , 1.00 , 1.00 , 1.00 };

glLightfv(GL_LIGHT0, GL_POSITION, light_pos);
glLightfv(GL_LIGHT0, GL_AMBIENT, light_Ka);
glLightfv(GL_LIGHT0, GL_DIFFUSE, light_Kd);
glLightfv(GL_LIGHT0, GL_SPECULAR, light_Ks);

GLfloat material_Ka[] = { 0.11 , 0.06 , 0.11 , 1.00 };
GLfloat material_Kd[] = { 0.43 , 0.47 , 0.54 , 1.00 };
GLfloat material_Ks[] = { 0.33 , 0.33 , 0.52 , 1.00 };
GLfloat material_Ke[] = { 0.00 , 0.00 , 0.00 , 0.00 };
GLfloat material_Se = 10 ;

gMaterialfv(GL_FRONT, GL_AMBIENT, material_Ka);
gMaterialfv(GL_FRONT, GL_DIFFUSE, material_Kd);
gMaterialfv(GL_FRONT, GL_SPECULAR, material_Ks);
gMaterialfv(GL_FRONT, GL_EMISSION, material_Ke);
gMaterialfv(GL_FRONT, GL_SHININESS, material_Se);
```

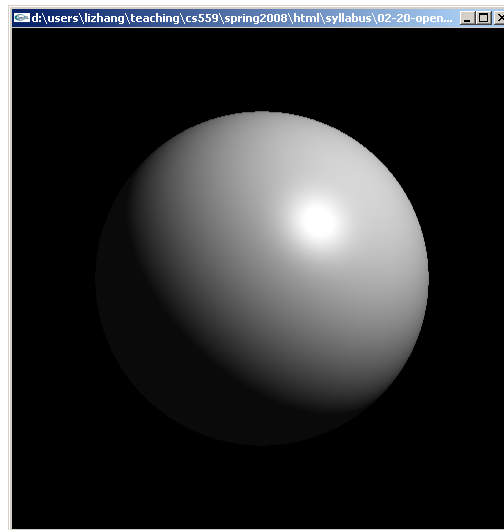
Click on the arguments and move the mouse to modify values.

Light.c

```
void init(void) {
    GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
    GLfloat mat_shininess[] = { 50.0 };
    GLfloat light_position[] = { 1.0, 1.0, 1.0, 0.0 };
    glClearColor (0.0, 0.0, 0.0, 0.0);
    glShadeModel (GL_SMOOTH);
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
    glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
    glLightfv(GL_LIGHT0, GL_POSITION, light_position);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_DEPTH_TEST);
}
```

```
void display(void) {
    glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glutSolidSphere (1.0, 20, 16);
    glFlush ();
}
```


Light.c



Complex lighting

- How to have multiple lights?
- How to change lighting positions?
- How to change color material?

Multiple lights

$$I = k_e + k_a L_a + \sum_j \frac{f_{spot_j}}{a_j + b_j D + c_j D^2} \left[k_a L_{a_j} + k_d L_{d_j} \cdot \max(0, \mathbf{L} \cdot \mathbf{N}) + k_s L_{s_j} \cdot (\mathbf{H} \cdot \mathbf{N})^{n_s} \right]$$

```
GLfloat light1_ambient[] = { 0.2, 0.2, 0.2, 1.0 };
GLfloat light1_diffuse[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat light1_specular[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat light1_position[] = { -2.0, 2.0, 1.0, 1.0 };
GLfloat spot_direction[] = { -1.0, -1.0, 0.0 };

glLightfv(GL_LIGHT1, GL_AMBIENT, light1_ambient);
glLightfv(GL_LIGHT1, GL_DIFFUSE, light1_diffuse);
glLightfv(GL_LIGHT1, GL_SPECULAR, light1_specular);

glLightfv(GL_LIGHT1, GL_POSITION, light1_position);
glLightf(GL_LIGHT1, GL_CONSTANT_ATTENUATION, 1.5);
glLightf(GL_LIGHT1, GL_LINEAR_ATTENUATION, 0.5);
glLightf(GL_LIGHT1, GL_QUADRATIC_ATTENUATION, 0.2);

glLightfv(GL_LIGHT1, GL_SPOT_DIRECTION, spot_direction);
glLightf(GL_LIGHT1, GL_SPOT_CUTOFF, 45.0);
glLightf(GL_LIGHT1, GL_SPOT_EXPONENT, 2.0);

glEnable(GL_LIGHT1);
```

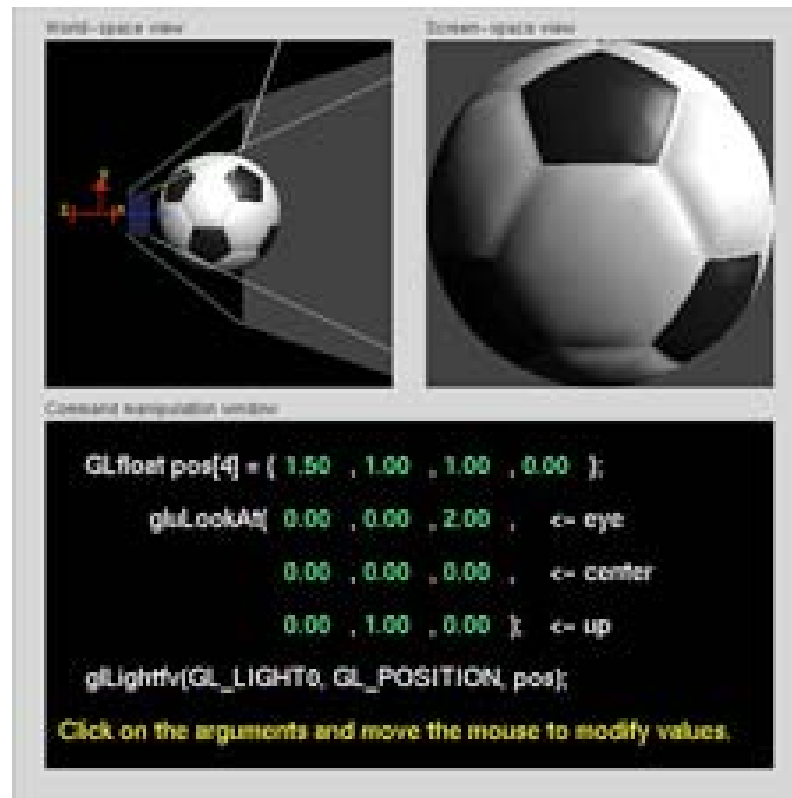
Moving light source



- Method 1:
 - Use `glLightfv(GL_LIGHT1, GL_POSITION, light1_position);`
- Method 2:
 - Use transformation

Moving a light source

- Use `glLightfv(GL_LIGHT1, GL_POSITION, light1_position);`



Moving light source

- Use transformation



```
static GLdouble spin;
void display(void) {
    GLfloat light_position[] = { 0.0, 0.0, 1.5, 1.0 };
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glPushMatrix();
        gluLookAt (0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
        glPushMatrix();
            glRotated(spin, 1.0, 0.0, 0.0);
            glLightfv(GL_LIGHT0, GL_POSITION, light_position);
        glPopMatrix();
        glutSolidTorus (0.275, 0.85, 8, 15);
    glPopMatrix();
    glFlush();
}
```

Demo

- Rotating a light source demo.

glColorMaterial

```
glEnable(GL_COLOR_MATERIAL);

glColorMaterial(GL_FRONT, GL_DIFFUSE);
/* now glColor* changes diffuse reflection */
glColor3f(0.2, 0.5, 0.8);
/* draw some objects here */

glColorMaterial(GL_FRONT, GL_SPECULAR);
/* glColor* no longer changes diffuse reflection */
/* now glColor* changes specular reflection */
glColor3f(0.9, 0.0, 0.2);

/* draw other objects here */
glDisable(GL_COLOR_MATERIAL);
```


glColorMaterial

- Demo

```
void mouse(int button, int state, int x, int y) {
    switch (button) {
        case GLUT_LEFT_BUTTON:
            if (state == GLUT_DOWN) {
                /* change red */
                diffuseMaterial[0] += 0.1;
                if (diffuseMaterial[0] > 1.0) diffuseMaterial[0] = 0.0;
                glColor4fv(diffuseMaterial);
                glutPostRedisplay();
            }
            break;
        case GLUT_MIDDLE_BUTTON:
            if (state == GLUT_DOWN) {
                /* change green */
                diffuseMaterial[1] += 0.1;
                if (diffuseMaterial[1] > 1.0) diffuseMaterial[1] = 0.0;
                glColor4fv(diffuseMaterial);
                glutPostRedisplay();
            }
            break;
        case GLUT_RIGHT_BUTTON:
            if (state == GLUT_DOWN) {
                /* change blue */
                diffuseMaterial[2] += 0.1;
                if (diffuseMaterial[2] > 1.0) diffuseMaterial[2] = 0.0;
                glColor4fv(diffuseMaterial);
                glutPostRedisplay();
            }
            break;
        default:
            break;
    }
}
```

Shading in OpenGL, cont'd

Notes:

You can have as many as `GL_MAX_LIGHTS` lights in a scene. This number is system-dependent.

For directional lights, you specify a light direction, not position, and the attenuation and spotlight terms are ignored.

The directions of directional lights and spotlights are specified in the world coordinate systems, not the object coordinate system.