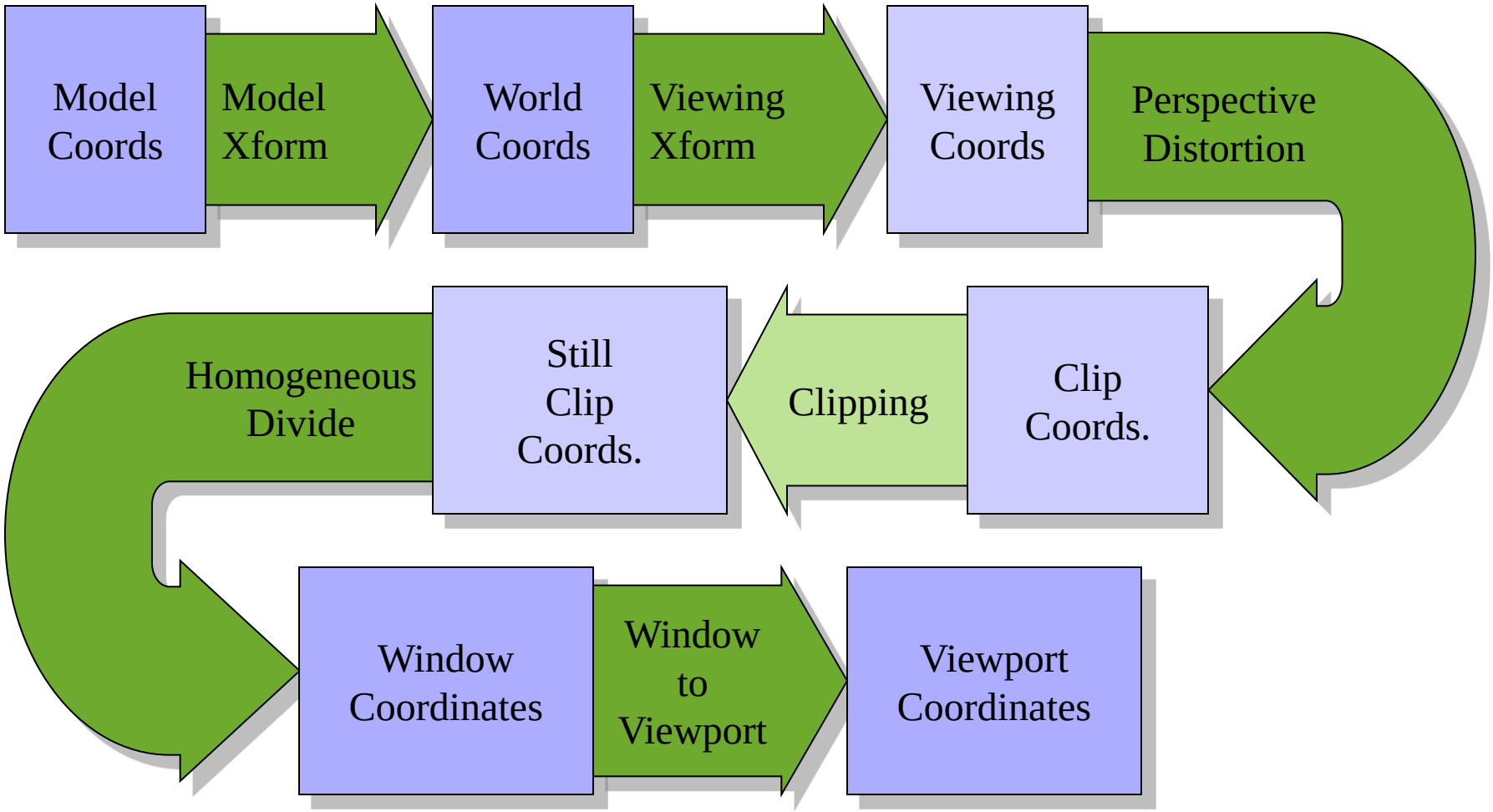
Lighting

CS418 Computer Graphics

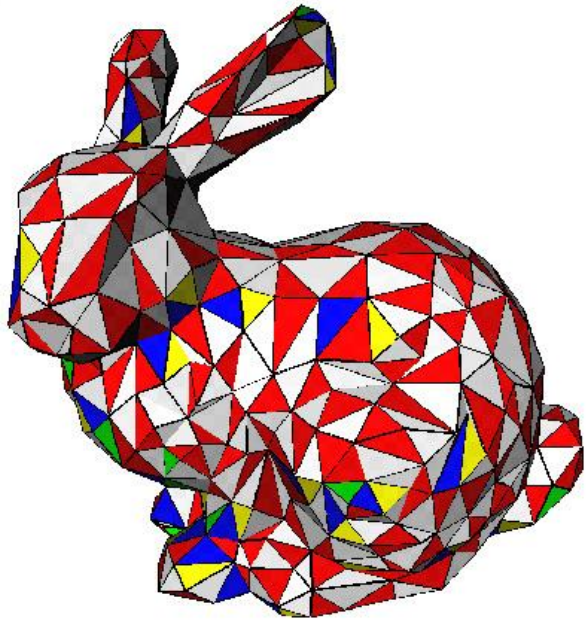
John C. Hart

Graphics Pipeline

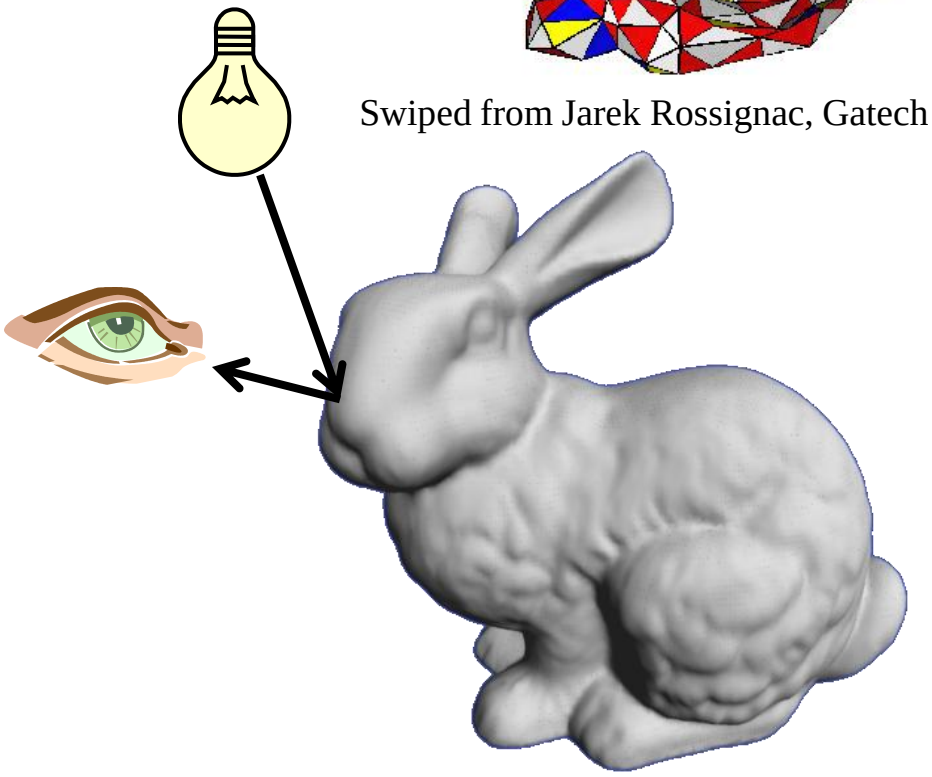


Lighting

- Makes viewed, projected, filled polygon meshes look more realistic by approximating how light would bounces off their surface



Swiped from Jarek Rossignac, Gatech



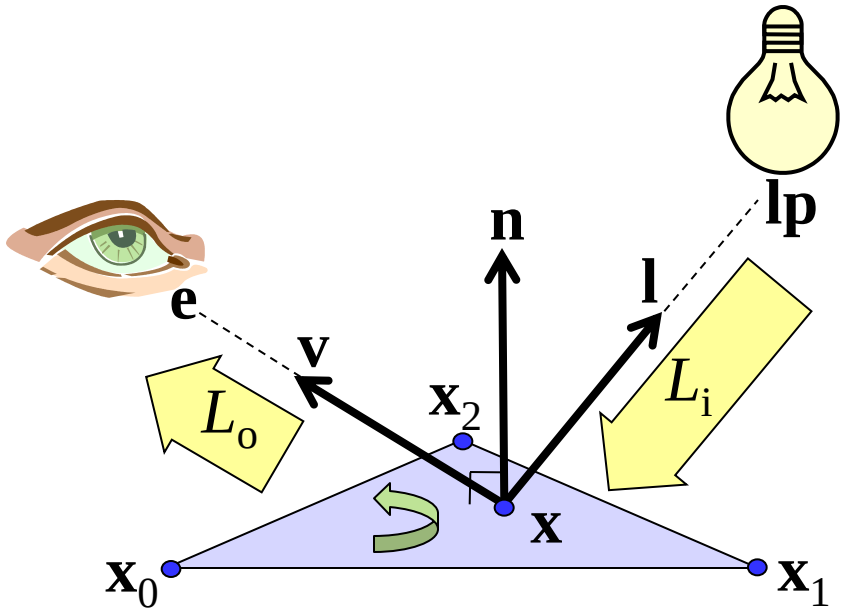
Swiped from Greg Turk, Gatech

Local Coordinates

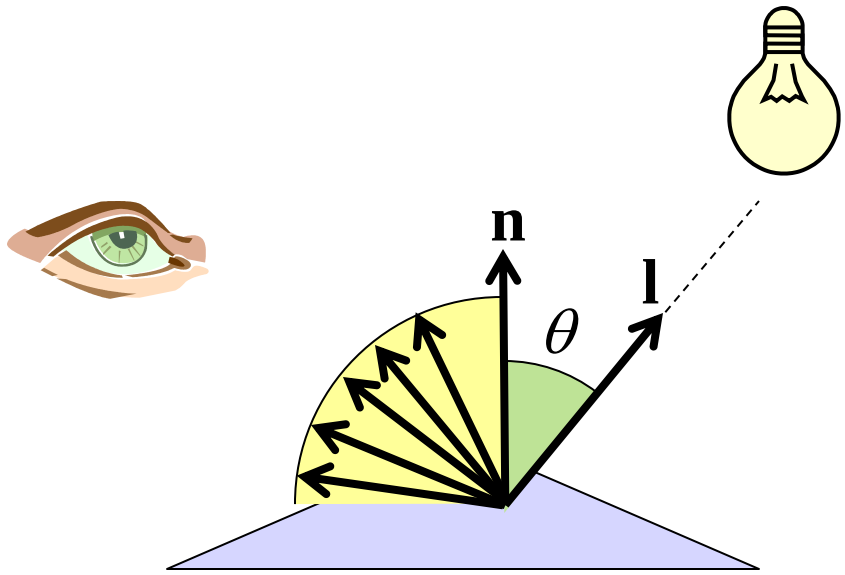
v – view vector: $\mathbf{v} = (\mathbf{e} - \mathbf{x})/\|\mathbf{e} - \mathbf{x}\|$

l – light vector: $\mathbf{l} = (\mathbf{lp} - \mathbf{x})/\|\mathbf{lp} - \mathbf{x}\|$

n – normal vector: $\mathbf{n} = (\mathbf{x}_1\text{-}\mathbf{x}_0)\times(\mathbf{x}_2\text{-}\mathbf{x}_0)/\|(\mathbf{x}_1\text{-}\mathbf{x}_0)\times(\mathbf{x}_2\text{-}\mathbf{x}_0)\|$

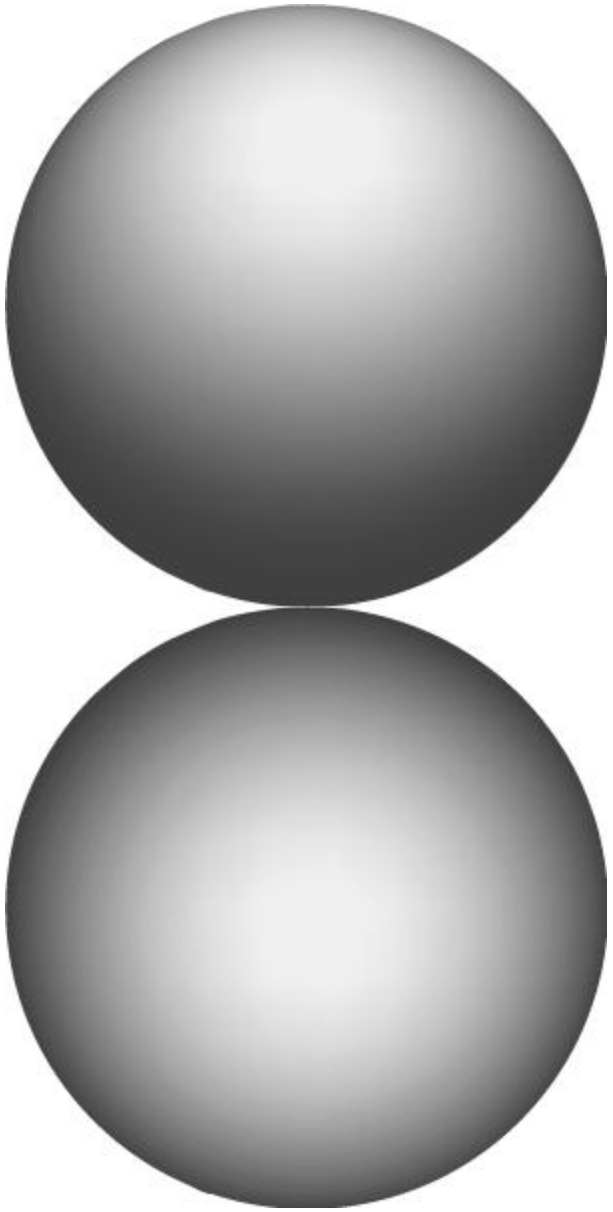


Lambertian Reflection

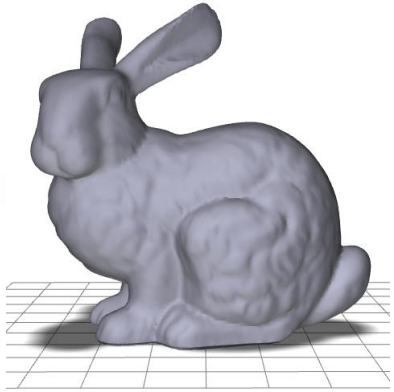


$$L_o = L_i\, k_d\, \mathbf{c}_d \cos\theta$$
$$= L_i\, k_d\, \mathbf{c}_d\, \mathbf{n}\cdot\mathbf{l}$$

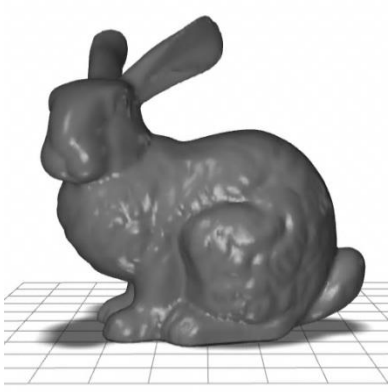
- $\mathbf{c}_d = (R_d, G_d, B_d)$: color surface diffusely reflects
- k_d = % of light reflected (rest is absorbed)
- view independent



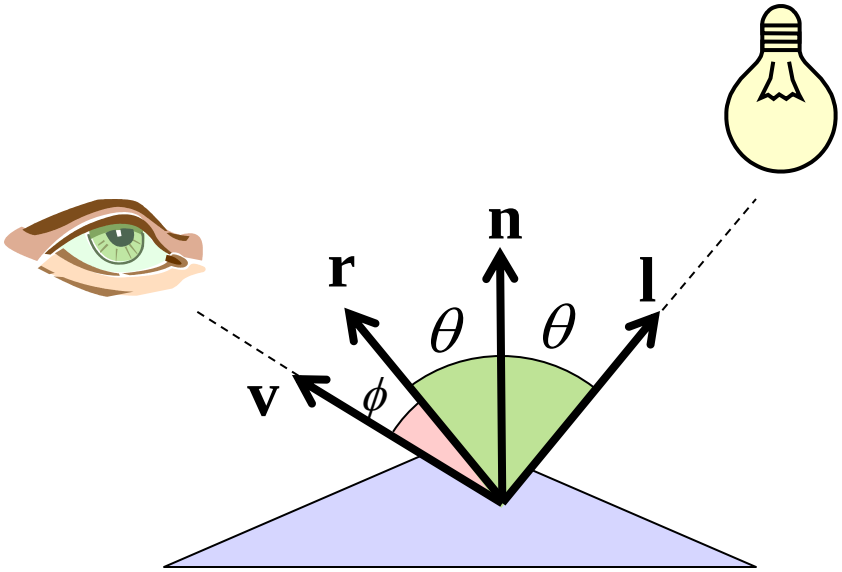
Specular Reflection



diffuse

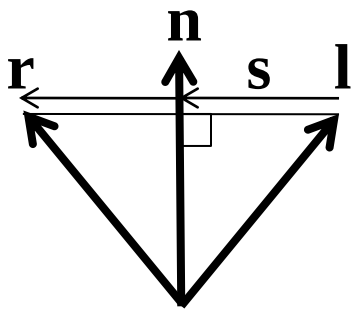


diffuse + specular



$$L_o = L_i\, k_s\, \mathbf{c}_s \cos^n \phi$$
$$= L_i\, k_s\, \mathbf{c}_s\, (\mathbf{v}\cdot\mathbf{r})^n$$

- $\mathbf{c}_s = (R_s, G_s, B_s)$: gleem reflection color
- k_s = % of light reflected (rest is absorbed)
- view dependent



$$\mathbf{s} = (\mathbf{n}\cdot\mathbf{l})\mathbf{n} - \mathbf{l}$$
$$\mathbf{r} = \mathbf{l} + 2\mathbf{s}$$
$$= \mathbf{l} + 2(\mathbf{n}\cdot\mathbf{l})\mathbf{n} - 2\mathbf{l}$$
$$= 2(\mathbf{n}\cdot\mathbf{l})\mathbf{n} - \mathbf{l}$$

The Phong Lighting Model

- Monochromatic

$$L_o = k_a L_a + L_i (k_d \, \mathbf{n}\cdot\mathbf{l} + k_s (\mathbf{v}\cdot\mathbf{r})^n)$$

- Tristimulus (RGB) color model

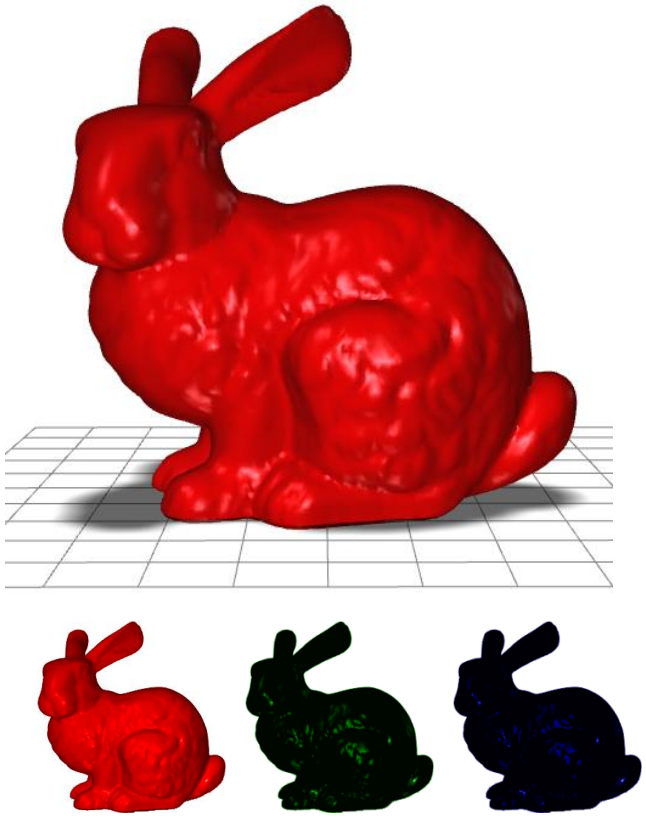
$$L_{o(R)} = k_{a(R)} L_{a(R)} + L_{i(R)} (k_{d(R)} \, \mathbf{n}\cdot\mathbf{l} + k_{s(R)} (\mathbf{v}\cdot\mathbf{r})^n)$$

$$L_{o(G)} = k_{a(G)} L_{a(G)} + L_{i(G)} (k_{d(G)} \, \mathbf{n}\cdot\mathbf{l} + k_{s(G)} (\mathbf{v}\cdot\mathbf{r})^n)$$

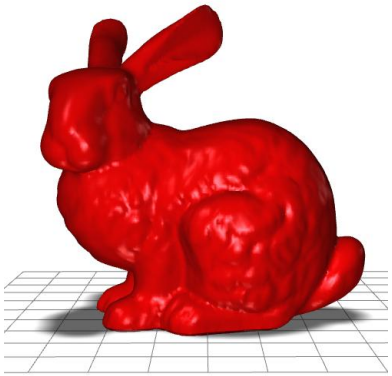
$$L_{o(B)} = k_{a(B)} L_{a(B)} + L_{i(B)} (k_{d(B)} \, \mathbf{n}\cdot\mathbf{l} + k_{s(B)} (\mathbf{v}\cdot\mathbf{r})^n)$$

- Multiple light sources

$$L_o = k_a L_a + L_{i(1)} (k_d \, \mathbf{n}\cdot\mathbf{l}_{(1)} + k_s (\mathbf{v}\cdot\mathbf{r}_{(1)})^n) + L_{i(2)} (k_d \, \mathbf{n}\cdot\mathbf{l}_{(2)} + k_s (\mathbf{v}\cdot\mathbf{r}_{(2)})^n) + \ldots$$



OpenGL Lighting



- Phong: $L_o = k_a L_a + L_i (k_d \mathbf{n}\cdot\mathbf{l} + k_s (\mathbf{v}\cdot\mathbf{r})^n)$
- OpenGL: $L_o = k_a L_{\#a} + L_{\#d} k_d \mathbf{n}\cdot\mathbf{l} + L_{\#s} k_s (\mathbf{v}\cdot\mathbf{r})^n$

```
GLfloat lpos[] = {1.0,10.0,-1.0,1.0};
GLfloat La[] = {1.0,1.0,1.0,1.0};
GLfloat Lid[] = {1.0,1.0,1.0,1.0};
GLfloat Lis[] = {1.0,1.0,1.0,1.0};

glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);

glLightfv(GL_LIGHT0, GL_POSITION, lpos);
glLightfv(GL_LIGHT0, GL_AMBIENT, La);
glLightfv(GL_LIGHT0, GL_DIFFUSE, Lid);
glLightfv(GL_LIGHT0, GL_SPECULAR, Lis);
```

```
GLfloat ka[] = {1.0,0.0,0.0,1.0};
GLfloat kd[] = {1.0,0.0,0.0,1.0};
GLfloat ks[] = {1.0,1.0,1.0,1.0};

#define BOTHSIDES GL_FRONT_AND_BACK

glMaterialfv(BOTHSIDES, GL_AMBIENT, ka);
glMaterialfv(BOTHSIDES, GL_DIFFUSE, kd);
glMaterialfv(BOTHSIDES, GL_SPECULAR, ks);
glMaterialf(BOTHSIDES,GL_SHININESS,50.0);
```

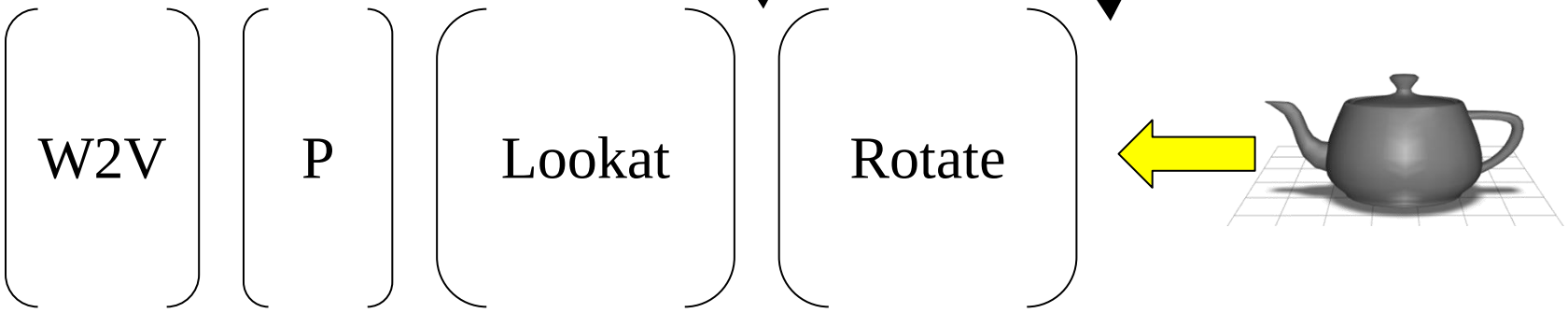
```
glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, GL_TRUE);
glLightModeli(GL_LIGHT_MODEL_TWO_SIDED, GL_TRUE);
```

Lighting Coordinates

- Lights specified in model (world) coordinates
- OpenGL computes lighting in viewing (eye) coordinates
- Light position specification treated as a vertex position
- Light position transformed by current ModelView matrix

Rotating object:
gluLookat(…,0,0,0,…);
glLightfv(…,…,lpos);
glRotatef(angle,0,1,0);
glutSolidTeapot(1.0);

Rotating view:
gluLookat(…,0,0,0,…);
glRotatef(angle,0,1,0);
glLightfv(...,...,lpos);
glutSolidTeapot(1.0);



Surface Normals

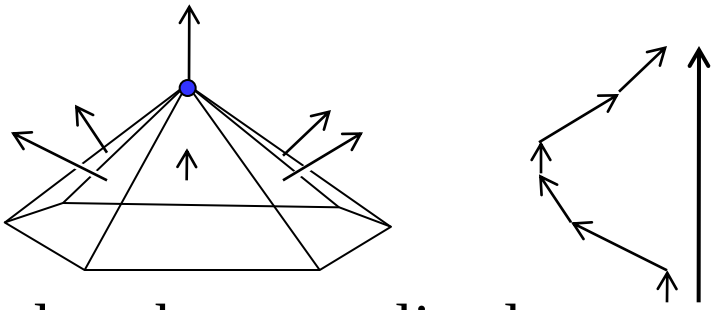
- Can be defined per face or per vertex
- Per face normal of a ccw face

$$\mathbf{n} = (\mathbf{x}_1 - \mathbf{x}_0)\times(\mathbf{x}_2 - \mathbf{x}_0)$$

  - Needs to be unitized before lighting
  - Automatically normalized by

```
glEnable(GL_NORMALIZE);
```

- Per vertex normal
  - Sum of normals of adjacent faces

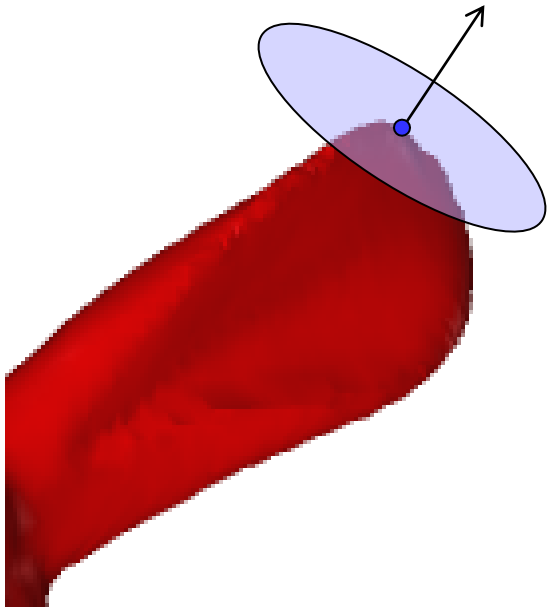


  - Needs to be normalized

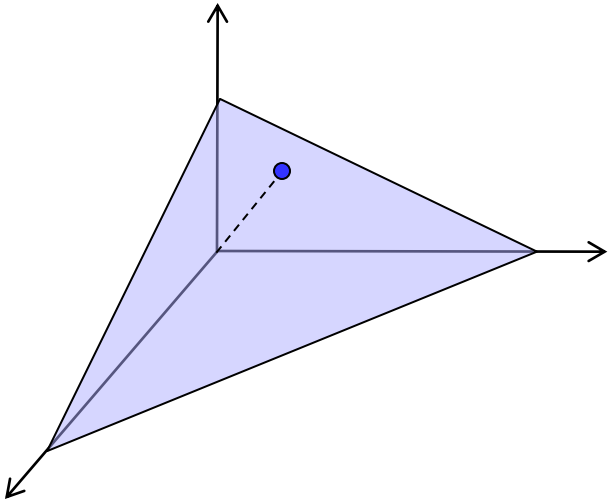
```
glNormal3f(nx,ny,nz)
glBegin(GL_POLYGON);
  glVertex3f(x0,y0,z0);
  glVertex3f(x1,y1,z1);
  glVertex3f(x2,y2,z2);
glEnd();
```

```
glBegin(GL_POLYGON);
  glNormal3f(nx0,ny0,nz0)
  glVertex3f(x0,y0,z0);

  glNormal3f(nx1,ny1,nz1)
  glVertex3f(x1,y1,z1);

  glNormal3f(nx2,ny2,nz2)
  glVertex3f(x2,y2,z2);
glEnd();
```

Transforming Normals



- First order neighborhood of a point on a surface described by a tangent plane
- Plane equation: $Ax + By + Cz + D = 0$
- Plane normal: (A,B,C)
- If $\|(A,B,C)\| = 1$
- Then D is distance from plane to origin



$$\begin{bmatrix} A & B & C & D \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = 0$$

Transforming Normals

$$\begin{bmatrix} A & B & C & D \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = 0$$

- Plane equation: $\mathbf{n}\,\mathbf{x} = 0$
  - $\mathbf{n}$ is a row vector
  - $\mathbf{x}$ is a column vector
- Let M be an affine transformation
- Transformed geometry $\mathbf{x}' = M\,\mathbf{x}$
- New normal $\mathbf{n}'$ such that $\mathbf{n}'\,\mathbf{x}' = 0$

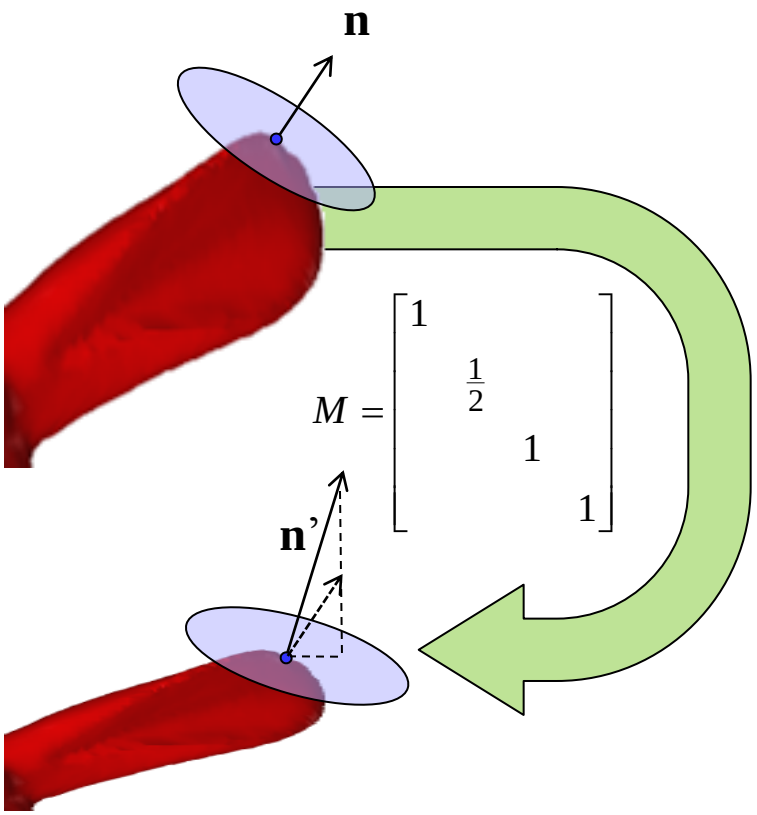
$$\mathbf{n}'\,M\,\mathbf{x} = 0$$

$$\mathbf{n}\,\mathbf{x} = 0$$

$$\mathbf{n}'\,M = \mathbf{n}$$

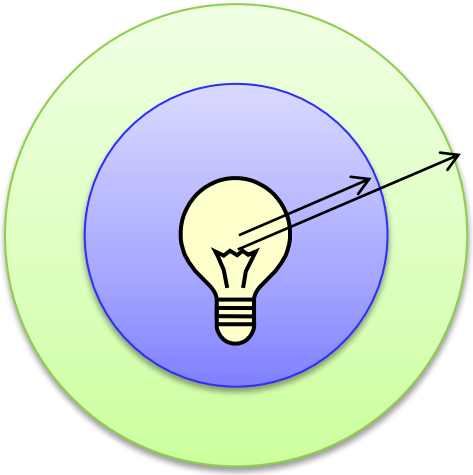
$$\mathbf{n}' = \mathbf{n}\,M^{-1}$$

- Needs to be normalized



Attenuation

Sphere Area = $4/3\ \pi\ r^2$



- Less light from sources far away
- Should be inverse square fall-off ($1/d^2$)
- Looks better with inverse ($1/d$)
- Inverse more expensive to compute

```
glLightf(GL_LIGHTi, GL_CONSTANT_ATTENUATION, a);
glLightf(GL_LIGHTi, GL_LINEAR_ATTENUATION, b);
glLightf(GL_LIGHTi, GL_QUADRATIC_ATTENUATION, c);

d = ||x – lpos[i]||;
att = 1.0/(a + b*d + c*d*d);
(light at x) = att*(light at source)
```