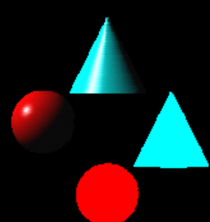
Lighting

Objectives

- Learn to shade objects so their images appear three-dimensional
- Introduce the types of light-material interactions
- Build a simple reflection model---the Phong model--- that can be used with real time graphics hardware

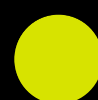
Lighting Principles

- Lighting simulates how objects reflect light
  - material composition of object
  - light's color and position
  - global lighting parameters
    - ambient light
    - two sided lighting
  - available in both color index and RGBA mode

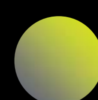


Why we need shading

- Suppose we build a model of a sphere using many polygons and color it with `glColor`. We get something like



- But we want



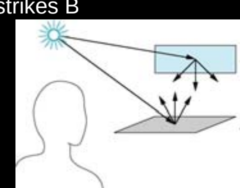
Shading

- Why does the image of a real sphere look like

- Light-material interactions cause each point to have a different color or shade
- Need to consider
  - Light sources
  - Material properties
  - Location of viewer
  - Surface orientation

Scattering

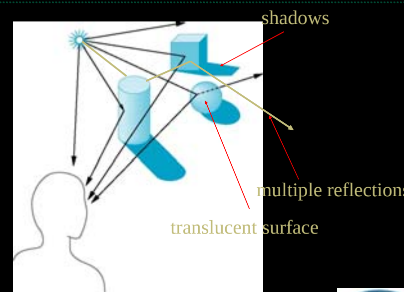
- Light strikes A
  - Some scattered
  - Some absorbed
- Some of scattered light strikes B
  - Some scattered
  - Some absorbed
- Some of this scattered light strikes A and so on



Rendering Equation

- The infinite scattering and absorption of light can be described by the *rendering equation*
  - Cannot be solved analytically in general
  - Ray tracing is a special case for perfectly reflecting surfaces
- Rendering equation is global and includes
  - Shadows
  - Multiple scattering from object to object

Global Effects



Local vs Global Rendering

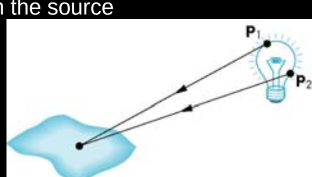
- Correct shading requires a global calculation involving all objects and light sources
  - Incompatible with pipeline model which shades each polygon independently (local rendering)
- However, in computer graphics, especially real time graphics, we are happy if things "look right"
  - Exist many techniques for approximating global effects

Light-Material Interaction

- Light that strikes an object is partially absorbed and partially scattered (reflected)
- The amount reflected determines the color and brightness of the object
  - A surface appears red under white light because the red component of the light is reflected and the rest is absorbed
- The reflected light is scattered in a manner that depends on the smoothness and orientation of the surface

Light Sources

General light sources are difficult to work with because we must integrate light coming from all points on the source

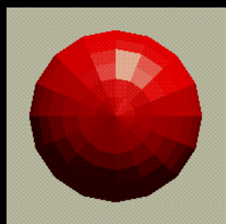


Simple Light Sources

- Point source
  - Model with position and color
  - Distant source = infinite distance away (parallel)
- Spotlight
  - Restrict light from ideal point source
- Ambient light
  - Same amount of light everywhere in scene
  - Can model contribution of many sources and reflecting surfaces

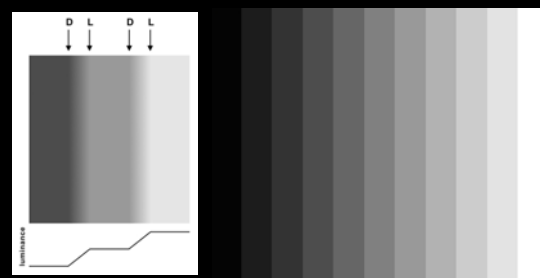
Shading Schemes

Flat Shading: same shade to entire polygon



CS5600

Mach Band Illusion



CS5600

Surface Types

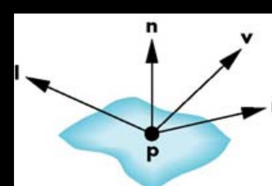
- The smoother a surface, the more reflected light is concentrated in the direction a perfect mirror would reflected the light
- A very rough surface scatters light in all directions

smooth surface

rough surface

Phong Model

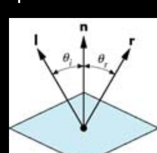
- A simple model that can be computed rapidly
- Has three components
  - Diffuse
  - Specular
  - Ambient
- Uses four vectors
  - To source
  - To viewer
  - Normal
  - Perfect reflector



Ideal Reflector

- Normal is determined by local orientation
- Angle of incidence = angle of relection
- The three vectors must be coplanar

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

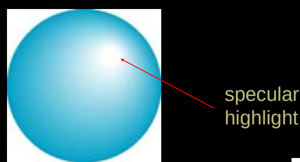


Lambertian Surface

- Perfectly diffuse reflector
- Light scattered equally in all directions
- Amount of light reflected is proportional to the vertical component of incoming light
  - reflected light ~cos θ_i
  - cos θ_i = $\mathbf{l} \cdot \mathbf{n}$ if vectors normalized
  - There are also three coefficients, k_r, k_b, k_g that show how much of each color component is reflected

Specular Surfaces

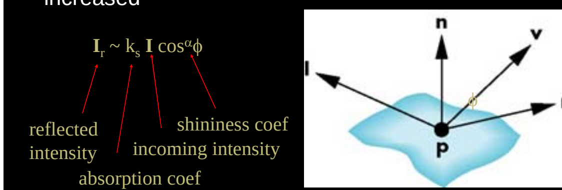
- Most surfaces are neither ideal diffusers nor perfectly specular (ideal reflectors)
- Smooth surfaces show specular highlights due to incoming light being reflected in directions concentrated close to the direction of a perfect reflection



Angel: Interactive Computer Graphics 5E © Addison-Wesley 2009

Modeling Specular Relections

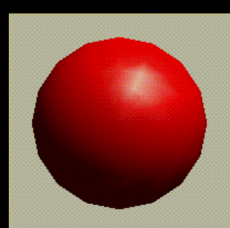
- Phong proposed using a term that dropped off as the angle between the viewer and the ideal reflection increased



Angel: Interactive Computer Graphics 5E © Addison-Wesley 2009

Shading Schemes

Gouraud Shading: smoothly blended intensity across each polygon



CS5600

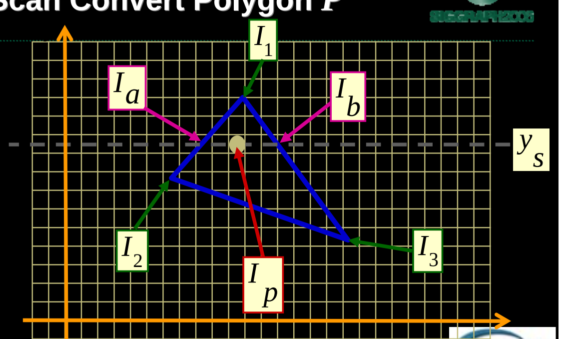
Shading Schemes

Phong Shading: interpolated *normals* to compute intensity at each point

Bui Toung Phong Thesis

CS5600

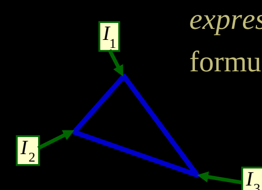
Scan Convert Polygon *P*



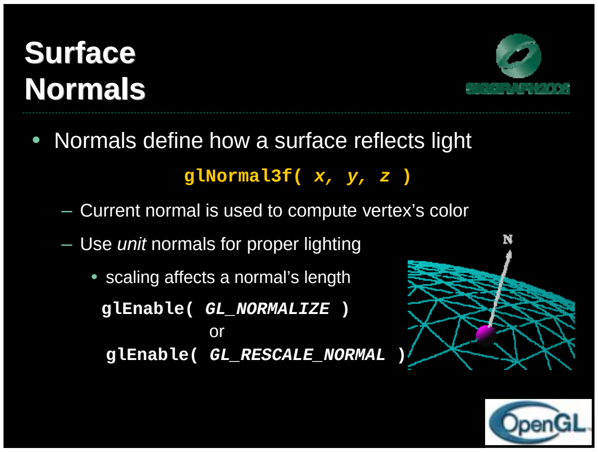
CS5600

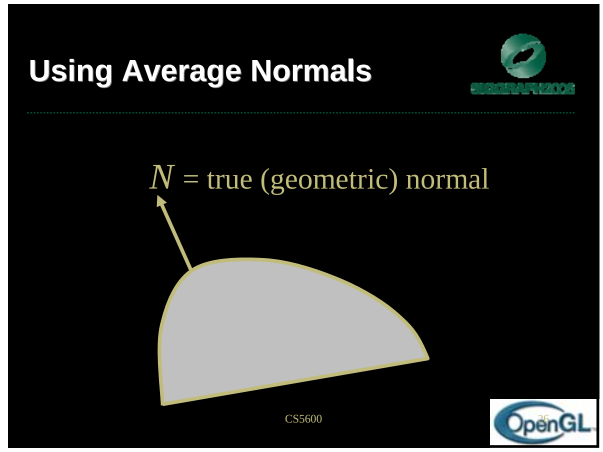
Intensity Interpolation

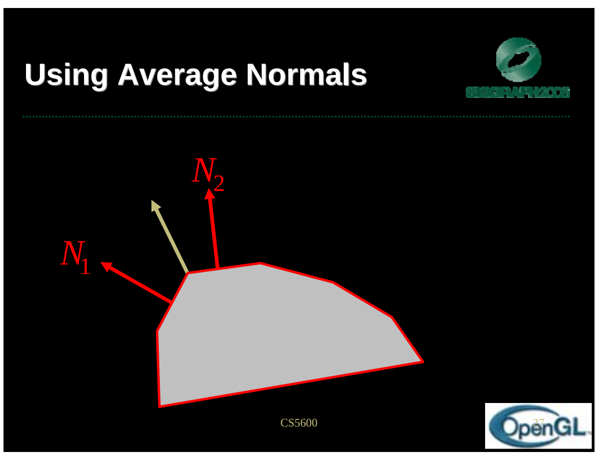
I_1, I_2, I_3: Compute by direction evaluation of *illumination expression*, whichever formula is being used

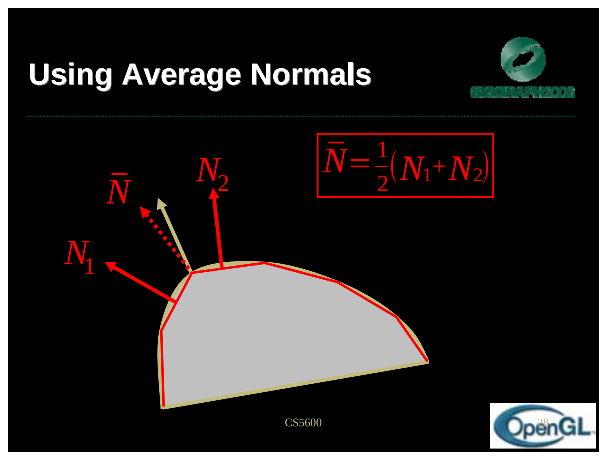


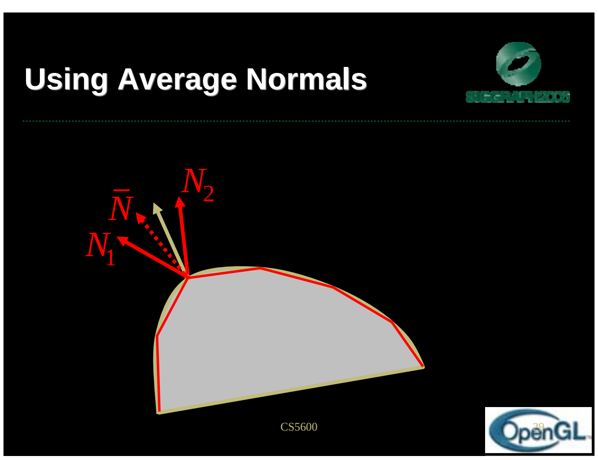
CS5600

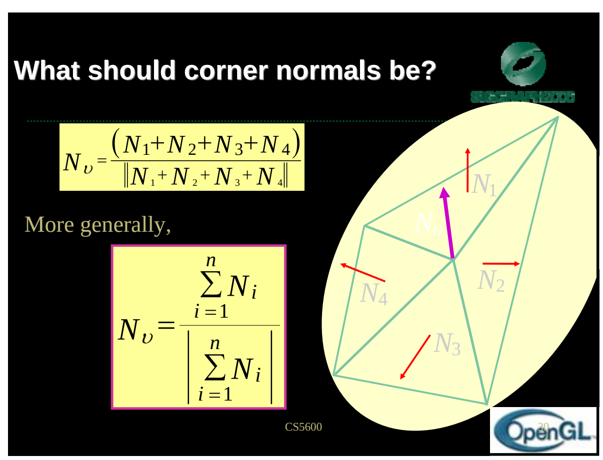

Surface Normals
Normals define how a surface reflects light
glNormal3f(x, y, z)
Current normal is used to compute vertex's color
Use unit normals for proper lighting
scaling affects a normal's length
glEnable(GL_NORMALIZE)
or
glEnable(GL_RESCALE_NORMAL)
N
OpenGL

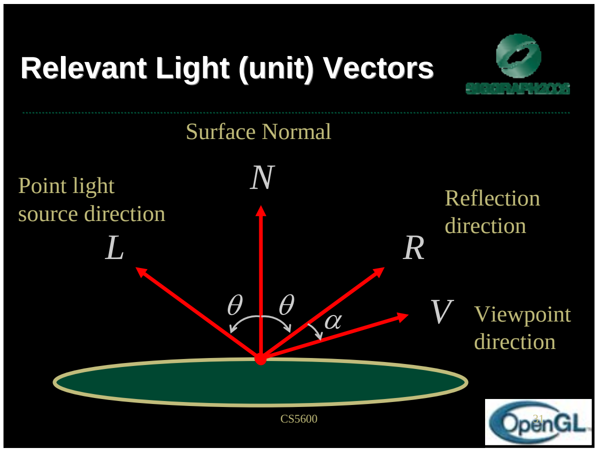

Using Average Normals
N = true (geometric) normal
CS5600
OpenGL

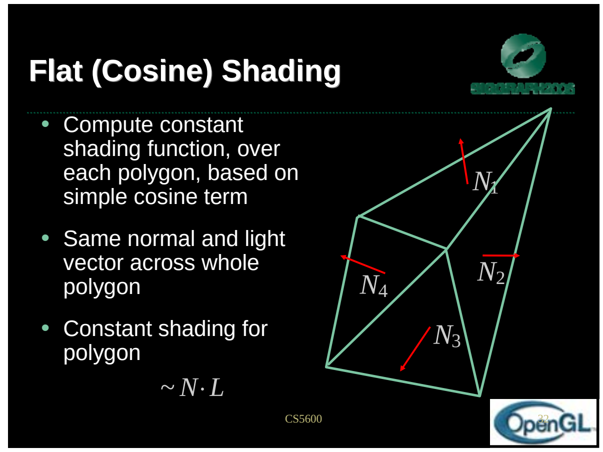

Using Average Normals
N_2
N_1
CS5600
OpenGL

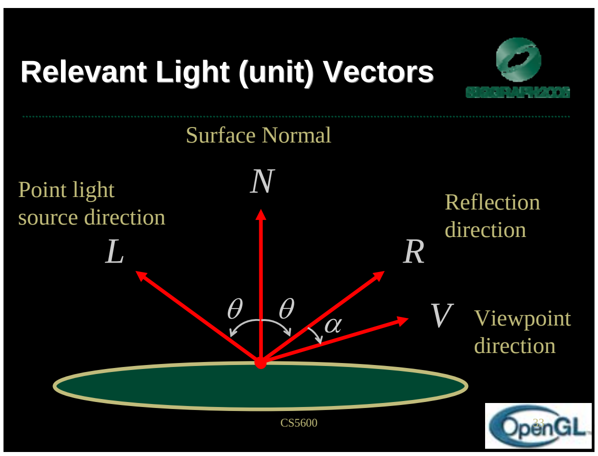

Using Average Normals
$\bar{N} = \frac{1}{2}(N_1 + N_2)$
$\bar{N}$
N_2
N_1
CS5600
OpenGL

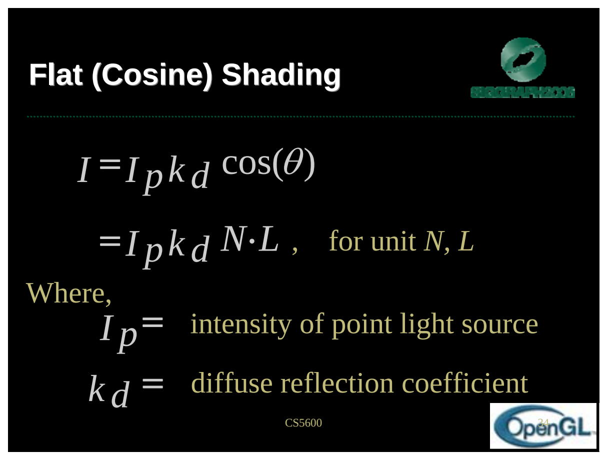

Using Average Normals
$\bar{N}$
N_2
N_1
CS5600
OpenGL

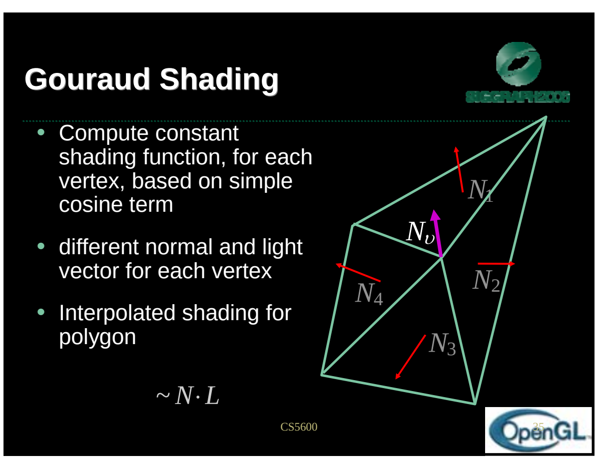

What should corner normals be?
$N_\upsilon = \frac{(N_1+N_2+N_3+N_4)}{\|N_1+N_2+N_3+N_4\|}$
More generally,
$N_\upsilon = \frac{\sum_{i=1}^{n} N_i}{\left|\sum_{i=1}^{n} N_i\right|}$
N_1
N_2
N_3
N_4
CS5600
OpenGL

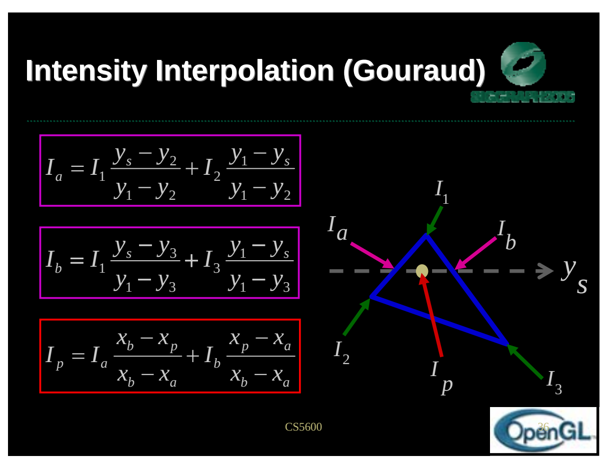

Relevant Light (unit) Vectors
Surface Normal
Point light source direction
N
Reflection direction
L
R
θ θ α
V
Viewpoint direction
CS5600


Flat (Cosine) Shading
Compute constant shading function, over each polygon, based on simple cosine term
Same normal and light vector across whole polygon
Constant shading for polygon
$\sim N \cdot L$
N_1
N_2
N_3
N_4
CS5600


Relevant Light (unit) Vectors
Surface Normal
Point light source direction
N
Reflection direction
L
R
θ θ α
V
Viewpoint direction
CS5600


Flat (Cosine) Shading
$I = I_p k_d \cos(\theta)$
$= I_p k_d N \cdot L$, for unit N, L
Where,
$I_p =$ intensity of point light source
$k_d =$ diffuse reflection coefficient
CS5600

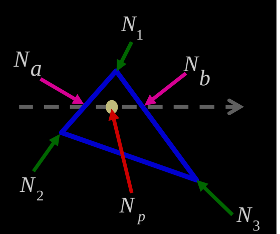

Gouraud Shading
Compute constant shading function, for each vertex, based on simple cosine term
different normal and light vector for each vertex
Interpolated shading for polygon
$\sim N \cdot L$
N_1
N_v
N_2
N_3
N_4
CS5600


Intensity Interpolation (Gouraud)
$I_a = I_1 \frac{y_s - y_2}{y_1 - y_2} + I_2 \frac{y_1 - y_s}{y_1 - y_2}$
$I_b = I_1 \frac{y_s - y_3}{y_1 - y_3} + I_3 \frac{y_1 - y_s}{y_1 - y_3}$
$I_p = I_a \frac{x_b - x_p}{x_b - x_a} + I_b \frac{x_p - x_a}{x_b - x_a}$
I_1
I_a
I_b
y_s
I_2
I_p
I_3
CS5600

Normal Interpolation (Phong)

$$N_a = N_1 \frac{y_s - y_2}{y_1 - y_2} + N_2 \frac{y_1 - y_s}{y_1 - y_2}$$

$$N_b = N_1 \frac{y_s - y_3}{y_1 - y_3} + N_3 \frac{y_1 - y_s}{y_1 - y_3}$$



CS5600

Normal Interpolation (Phong)

$$\tilde{N}_p = \frac{N_a}{\|N_a\|}\left[\frac{x_b - x_p}{x_b - x_a}\right] + \frac{N_b}{\|N_b\|}\left[\frac{x_p - x_a}{x_b - x_a}\right]$$

$$N_p = \frac{\tilde{N}_p}{\|\tilde{N}_p\|}$$

Normalizing makes this a unit vector

CS5600

Phong Illumination Formula (1/2)

$$I_\lambda = I_{a\lambda} k_{a\lambda} + f_{att} I_{l\lambda}\left[k_{d\lambda}(N \cdot L) + k_{s\lambda}(R \cdot V)^n\right]$$

$\cos^n(\alpha)$

CS5600

Illumination Formula (2/2)

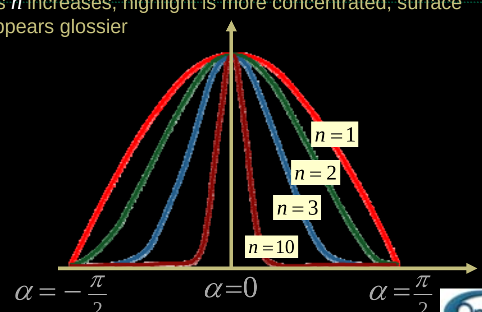
Where,

- a denotes *ambient* term
- d denotes *diffuse* term
- s denotes *specular* term
- k denotes *coefficient*
- I denotes *intensity*

CS5600

Effect of Exponent Parameter

As n increases, highlight is more concentrated, surface appears glossier



How OpenGL Simulates Lights

- Phong lighting model
  - Computed at vertices
- Lighting contributors
  - Surface material properties
  - Light properties
  - Lighting model properties

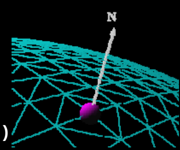
Surface Normals

- Normals define how a surface reflects light

```
glNormal3f( x, y, z )
```

  - Current normal is used to compute vertex's color
  - Use *unit* normals for proper lighting
    - scaling affects a normal's length

```
glEnable( GL_NORMALIZE )
or
glEnable( GL_RESCALE_NORMAL )
```



Material Properties

- Define the surface properties of a primitive

```
glMaterialfv( face, property, value );
```

GL_DIFFUSE	Base color
GL_SPECULAR	Highlight Color
GL_AMBIENT	Low-light Color
GL_EMISSION	Glow Color
GL_SHININESS	Surface Smoothness

  - separate materials for front and back

Light Properties

```
glLightfv( light, property, value );
```

- ***light*** specifies which light
  - multiple lights, starting with **GL_LIGHT0**

```
glGetIntegerv( GL_MAX_LIGHTS, &n );
```

- ***properties***
  - colors
  - position and type
  - attenuation

Light Sources (cont'd.)

- Light color properties
  - **GL_AMBIENT**
  - **GL_DIFFUSE**
  - **GL_SPECULAR**

Types of Lights

- OpenGL supports two types of Lights
  - Local (Point) light sources
  - Infinite (Directional) light sources
- Type of light controlled by w coordinate

$w = 0$ ***Infinite Light directed along*** $(x \quad y \quad z)$

$w \neq 0$ ***Local Light positioned at*** $(x/w \quad y/w \quad z/w)$

Turning on the Lights

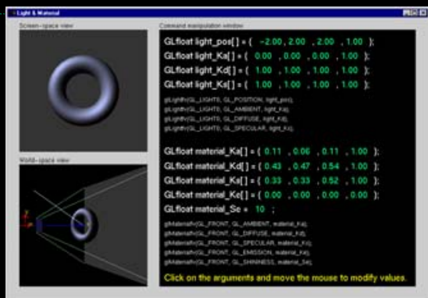
- Flip each light's switch

```
glEnable( GL_LIGHTn );
```

- Turn on the power

```
glEnable( GL_LIGHTING );
```

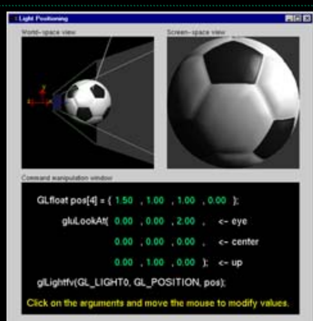
Light Material Tutorial



Controlling a Light's Position

- Modelview matrix affects a light's position
  - Different effects based on <u>when</u> position is specified
    - eye coordinates
    - world coordinates
    - model coordinates
  - Push and pop matrices to uniquely control a light's position

Light Position Tutorial



Tips for Better Lighting

- Recall lighting computed only at vertices
  - model tessellation heavily affects lighting results
    - better results but more geometry to process
- Use a single infinite light for fastest lighting
  - minimal computation per vertex

Lighting Models

From \ To	Somewhere	Everywhere
Somewhere	*Specular*	*Lambertian*
Everywhere	*Cautics*	*Ambient*

CS5600