# Computer Graphics

Bing-Yu Chen
National Taiwan University

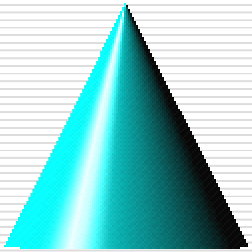
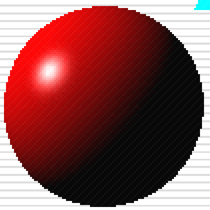
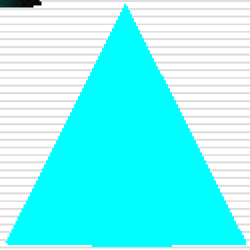
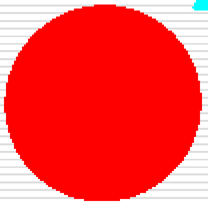
# Introduction to OpenGL

- General OpenGL Introduction
- An Example OpenGL Program
- Drawing with OpenGL
- Transformations
- Animation and Depth Buffering
- Lighting
- Evaluation and NURBS
- Texture Mapping
- Advanced OpenGL Topics
- Imaging

**modified from**
**Dave Shreiner, Ed Angel, and Vicki Shreiner.**
**An Interactive Introduction to OpenGL Programming.**
***ACM SIGGRAPH 2001 Conference Course Notes #54.***
**& *ACM SIGGRAPH 2004 Conference Course Notes #29.***

# Lighting Principles

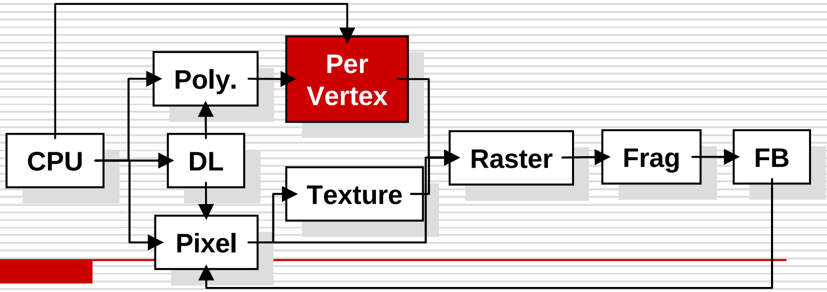
- Lighting simulates how objects reflect light
  - material composition of object
  - light’s color and position
  - global lighting parameters
    - ambient light
    - two sided lighting
  - available in both color index and RGBA mode

# How OpenGL Simulates Lights

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

# Surface Normals



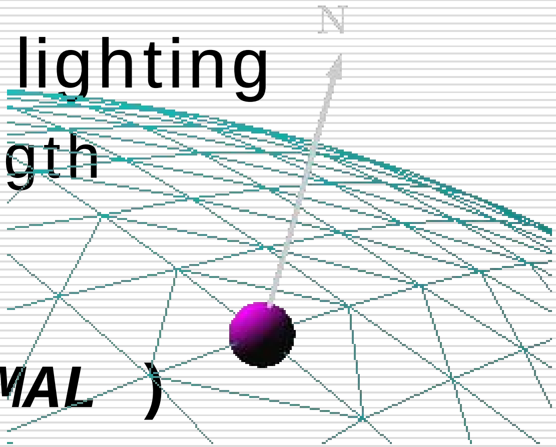

- Normals define how a surface reflects light

**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 )`**

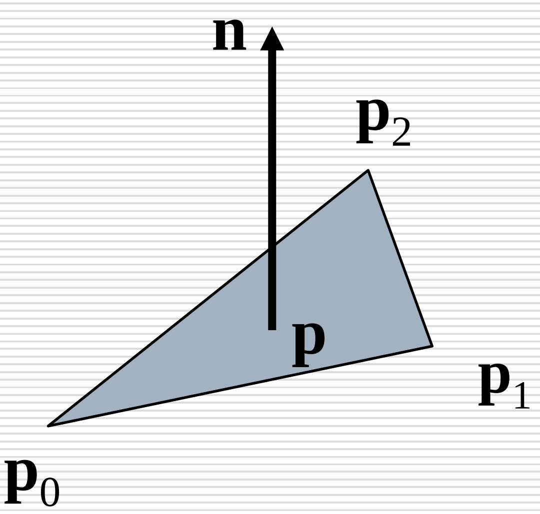


# Normal for Triangle

plane $\mathbf{n} \cdot (\mathbf{p} - \mathbf{p}_0) = 0$

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

normalize $\mathbf{n} \leftarrow \mathbf{n} / |\mathbf{n}|$




Note that right-hand rule determines outward face

# 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.)

- 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

$$w = 0 \quad \text{Infinite Light directed along} \begin{pmatrix} x & y & z \end{pmatrix}$$

$$w \neq 0 \quad \text{Local Light positioned at} \begin{pmatrix} x/w & y/w & z/w \end{pmatrix}$$

# Turning on the Lights

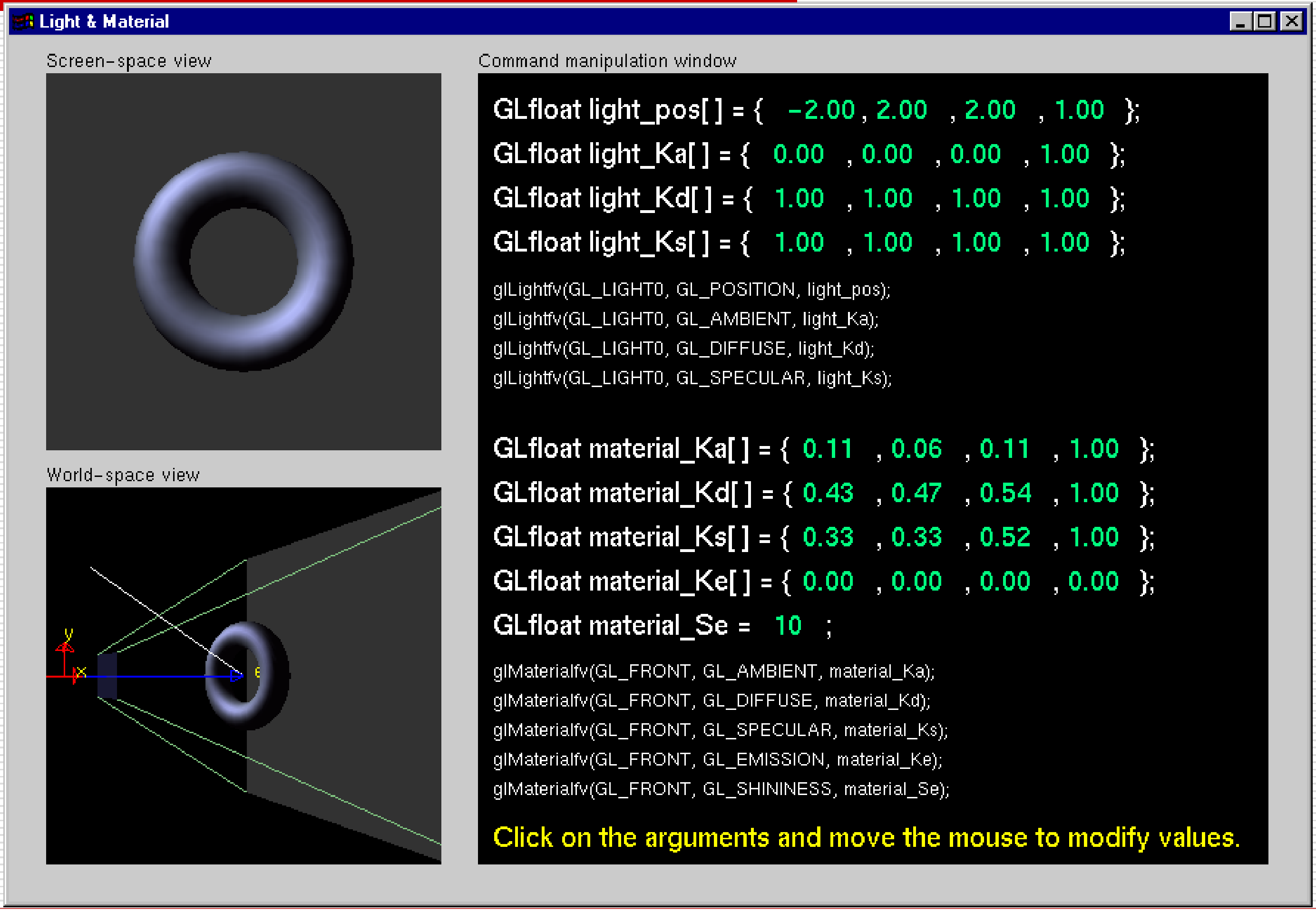
- Flip each light's switch

```
glEnable( GL_LIGHTn );
```

- Turn on the power

```
glEnable( GL_LIGHTING );
```

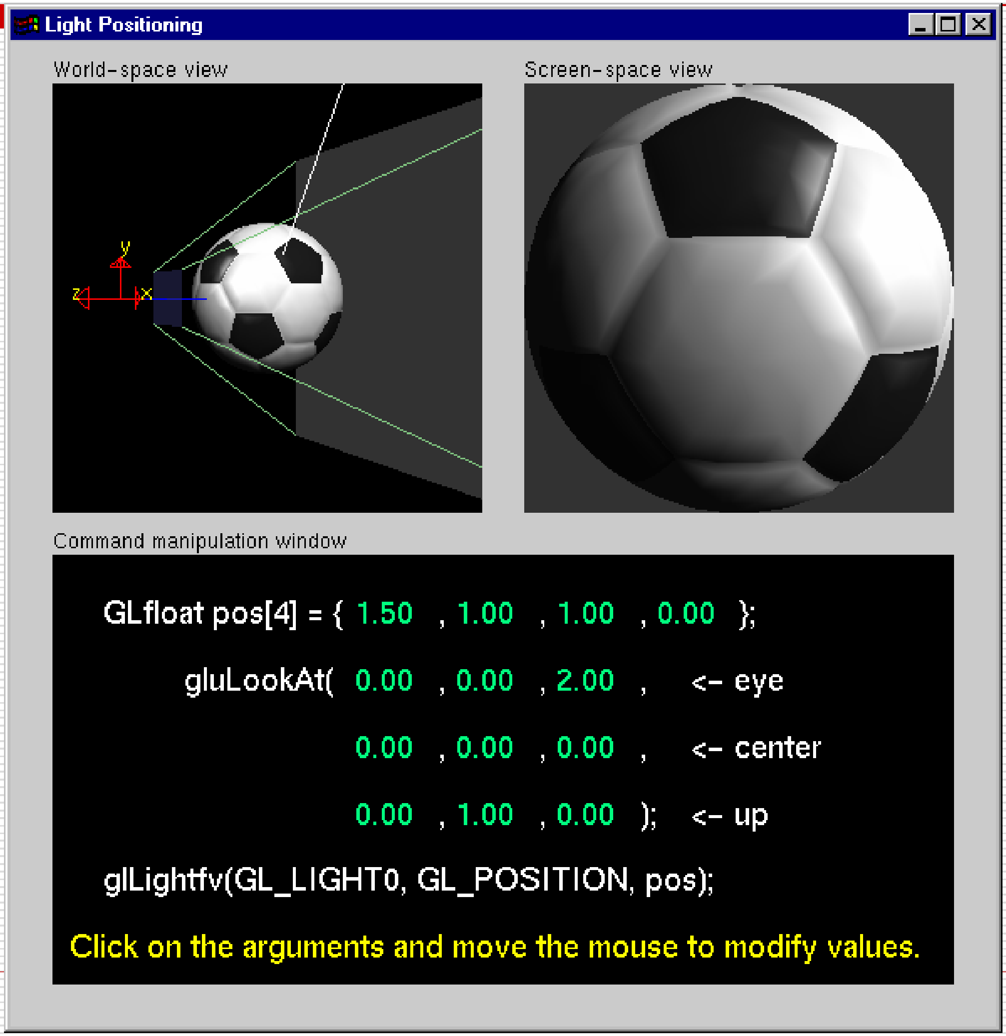
# Light Material Tutorial



# 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

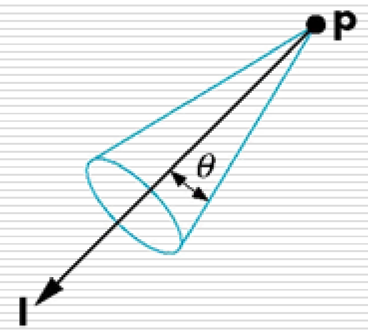


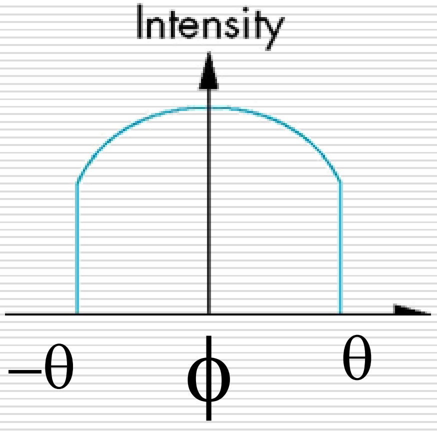
# Advanced Lighting Features

- Spotlights
  - localize lighting affects
    - ***GL_SPOT_DIRECTION***
    - ***GL_SPOT_CUTOFF***
    - ***GL_SPOT_EXPONENT***

# Spotlights

- Use **glLightv** to set
  - Direction **GL_SPOT_DIRECTION**
  - Cutoff **GL_SPOT_CUTOFF**
  - Attenuation **GL_SPOT_EXPONENT**
    - Proportional to $\cos^{\alpha}\phi$



# Advanced Lighting Features

- Light attenuation
  - decrease light intensity with distance
    - ***GL_CONSTANT_ATTENUATION***
    - ***GL_LINEAR_ATTENUATION***
    - ***GL_QUADRATIC_ATTENUATION***

$$f_i = \frac{1}{k_c + k_l d + k_q d^2}$$

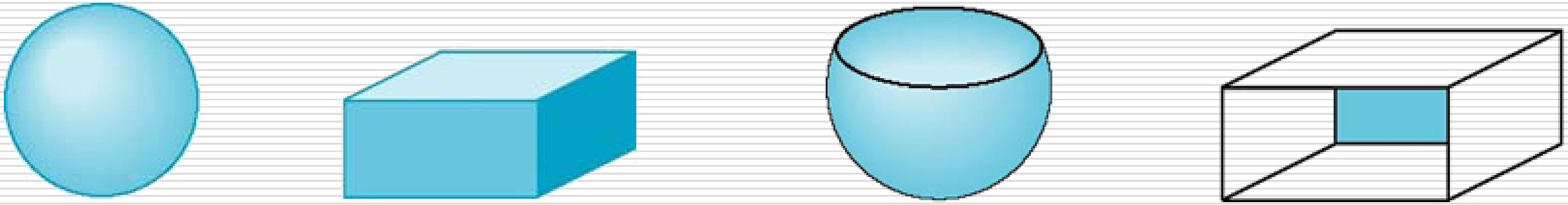
# Light Model Properties

**glLightModelfv(** ***property*****,** ***value*** **);**

- Enabling two sided lighting
  - ***GL_LIGHT_MODEL_TWO_SIDE***
- Global ambient color
  - ***GL_LIGHT_MODEL_AMBIENT***
- Local viewer mode
  - ***GL_LIGHT_MODEL_LOCAL_VIEWER***
- Separate specular color
  - ***GL_LIGHT_MODEL_COLOR_CONTROL***

# Front and Back Faces

- The default is shade only front faces which works correct for convex objects
- If we set two sided lighting, OpenGL will shaded both sides of a surface
- Each side can have its own properties which are set by using **`GL_FRONT`**, **`GL_BACK`**, or **`GL_FRONT_AND_BACK`** in **`glMaterialf`**



back faces not visible back faces visible

# Efficiency

- Because material properties are part of the state, if we change materials for many surfaces, we can affect performance
- We can make the code cleaner by defining a material structure and setting all materials during initialization

```
typedef struct materialStruct {
   GLfloat ambient[4];
   GLfloat diffuse[4];
   GLfloat specular[4];
   GLfloat shineness;
} MaterialStruct;
```

- We can then select a material by a pointer

# 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

# Steps in OpenGL shading

1. Enable shading and select model
2. Specify normals
3. Specify material properties
4. Specify lights

# Transparency

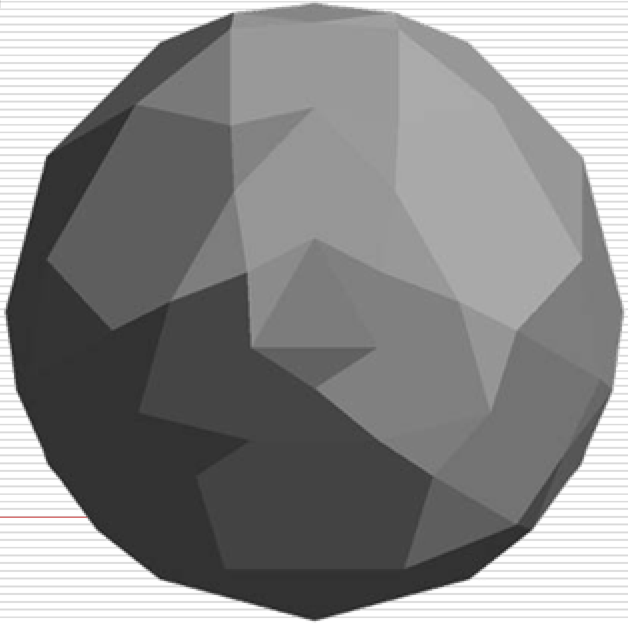
- Material properties are specified as RGBA values
- The A value can be used to make the surface translucent
- The default is that all surfaces are opaque regardless of A

# Polygonal Shading

- Shading calculations are done for each vertex
  - Vertex colors become vertex shades
- By default, vertex colors are interpolated across the polygon
  - **`glShadeModel(GL_SMOOTH);`**
- If we use **`glShadeModel(GL_FLAT);`** the color at the first vertex will determine the color of the whole polygon

# Polygon Normals

- Polygons have a single normal
  - Shades at the vertices as computed by the Phong model can be almost same
  - Identical for a distant viewer (default) or if there is no specular component
- Consider model of sphere
- Want different normals at each vertex even though this concept is not quite correct mathematically



# Smooth Shading

- We can set a new normal at each vertex
- Easy for sphere model
  - If centered at origin $\mathbf{n} = \mathbf{p}$
- Now smooth shading works
- Note *silhouette edge*