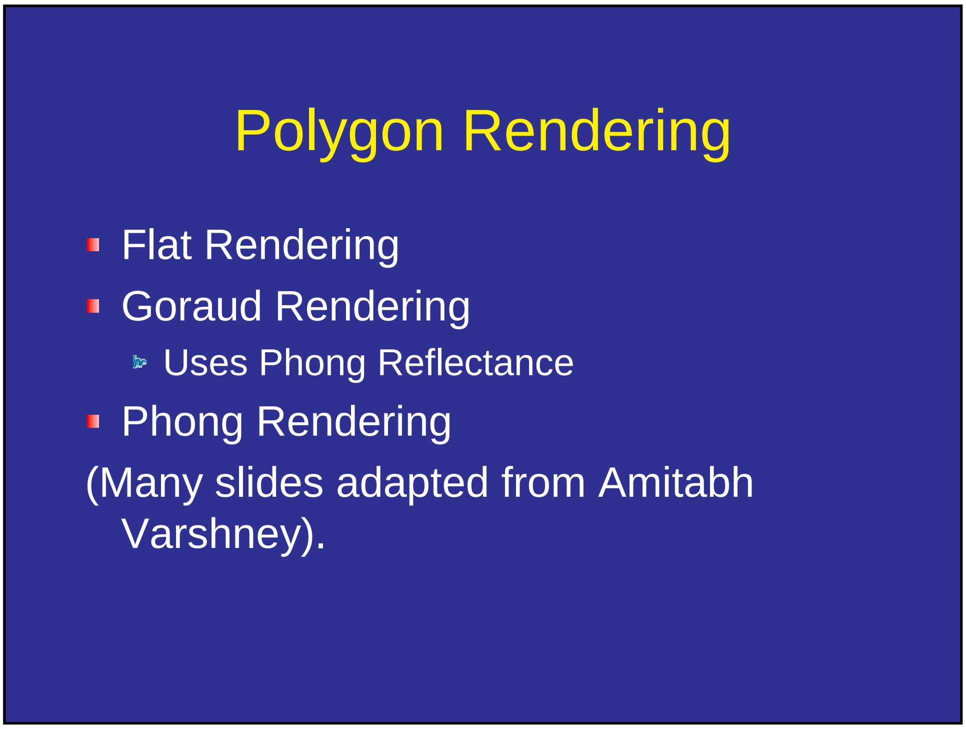
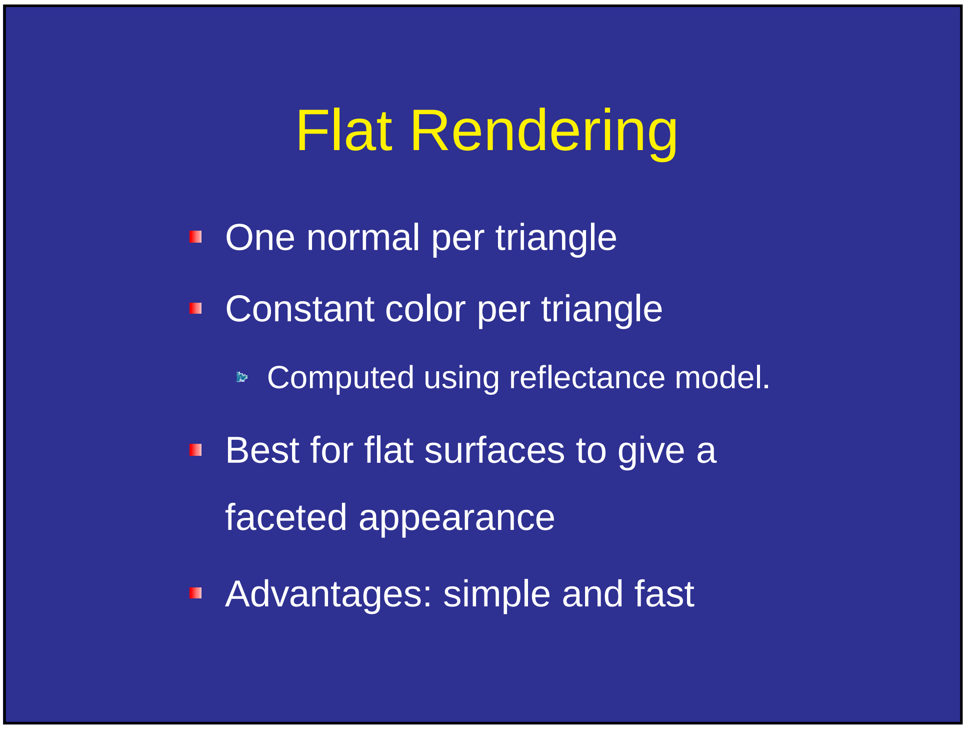


Polygon Rendering

- Flat Rendering
- Goraud Rendering
  - Uses Phong Reflectance
- Phong Rendering

(Many slides adapted from Amitabh Varshney).



Flat Rendering

- One normal per triangle
- Constant color per triangle
  - Computed using reflectance model.
- Best for flat surfaces to give a faceted appearance
- Advantages: simple and fast

Diffuse Illumination & Flat Rendering

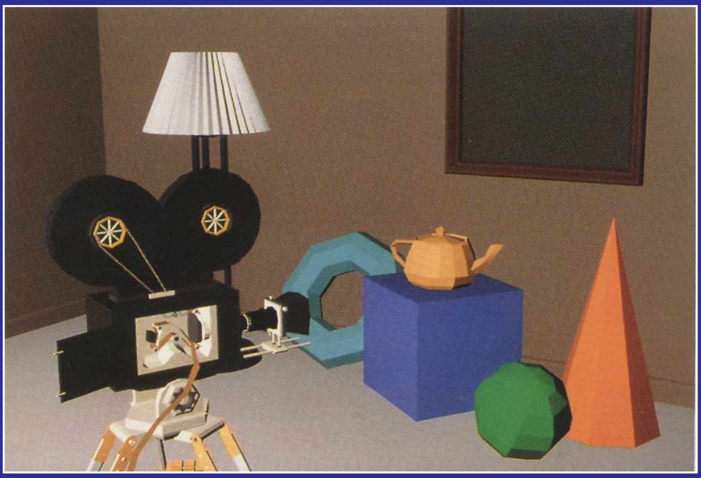


Image courtesy, Foley, van Dam, Feiner, Hughes

Gouraud Rendering

- One normal per vertex
- Compute color per vertex
- Interpolate color per pixel (one add per R, G, B channel)
- Tolerable results for curved surfaces

Diffuse & Gouraud Shading

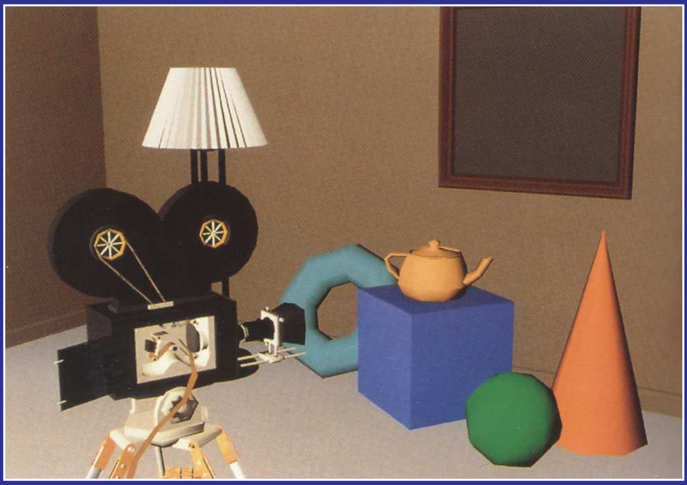


Image courtesy, Foley, van Dam, Feiner, Hughes

Specular & Gouraud Shading

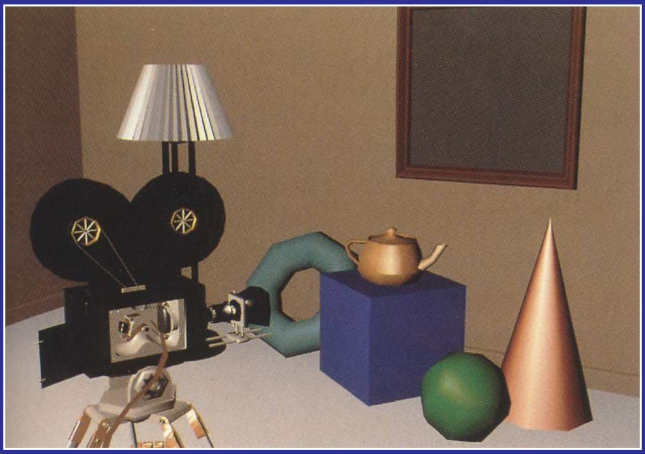
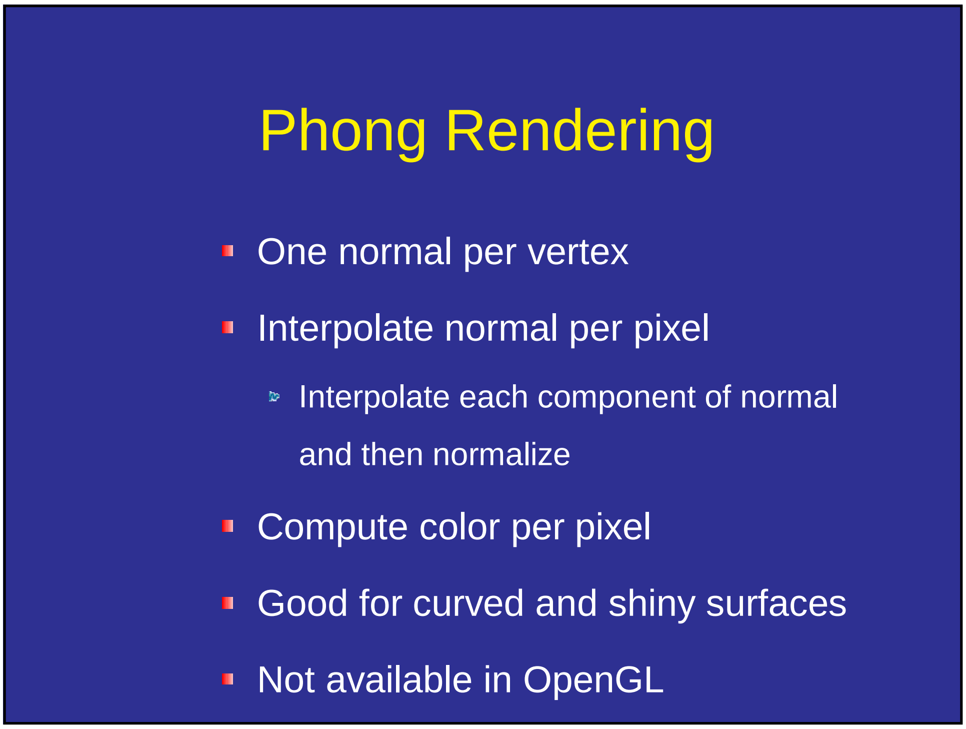
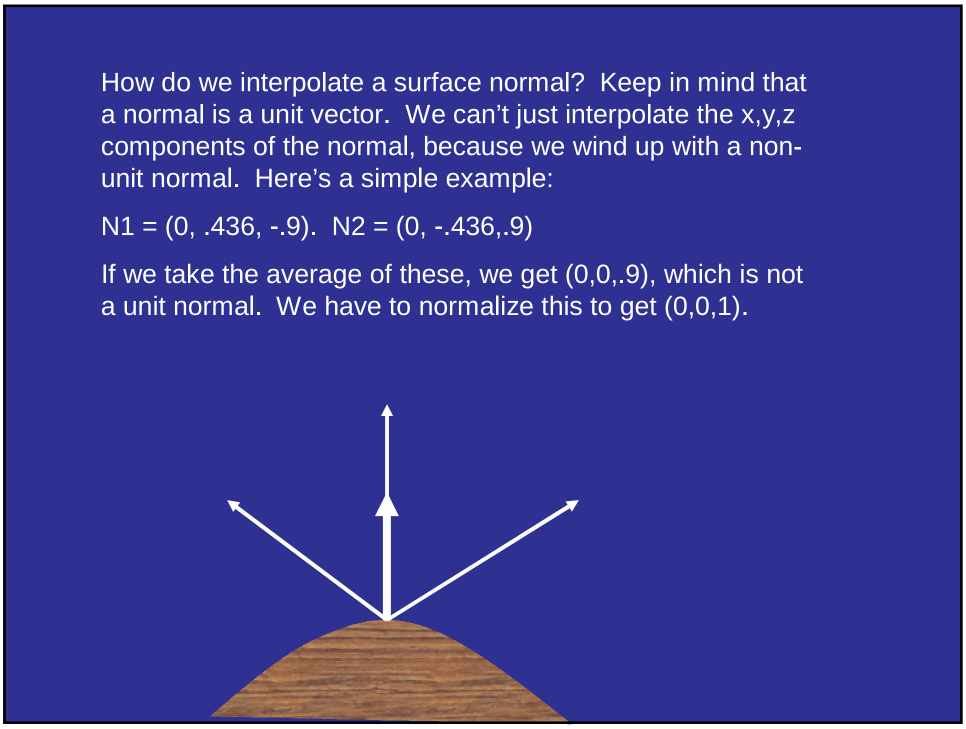


Image courtesy, Foley, van Dam, Feiner, Hughes



Phong Rendering

- One normal per vertex
- Interpolate normal per pixel
  - Interpolate each component of normal and then normalize
- Compute color per pixel
- Good for curved and shiny surfaces
- Not available in OpenGL



How do we interpolate a surface normal? Keep in mind that a normal is a unit vector. We can't just interpolate the x,y,z components of the normal, because we wind up with a non-unit normal. Here's a simple example:

N1 = (0, .436, -.9). N2 = (0, -.436,.9)

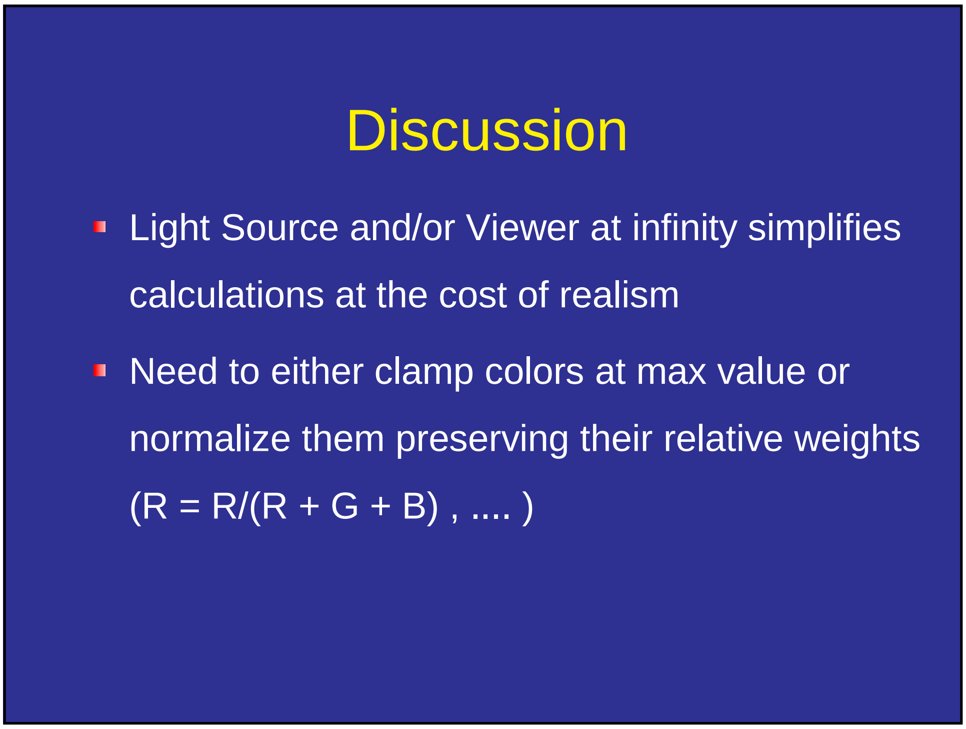
If we take the average of these, we get (0,0,.9), which is not a unit normal. We have to normalize this to get (0,0,1).

Specular & Phong Rendering

Image courtesy, Foley, van Dam, Feiner, Hughes

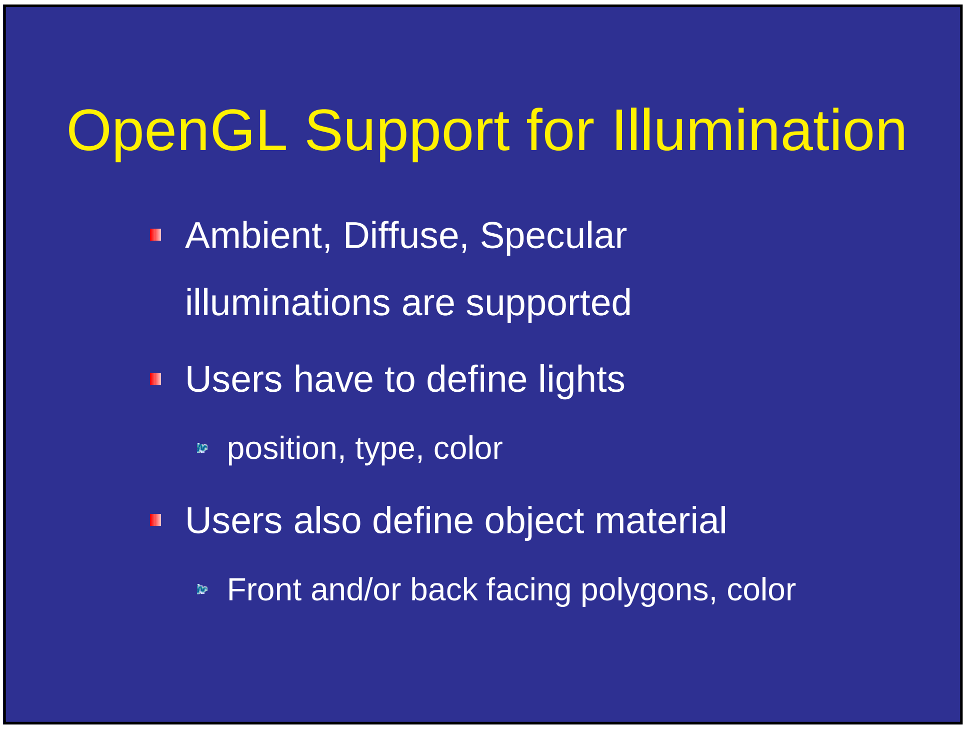
Gouraud vs. Phong

- Gouraud is faster
  - Interpolate 1 value instead of 3
  - Don’t need to normalize
  - Don’t need to render at each point.
- Phong much more accurate
  - Especially when lighting effects change rapidly with surface normal.
  - True for shiny objects
  - And for cast shadows.



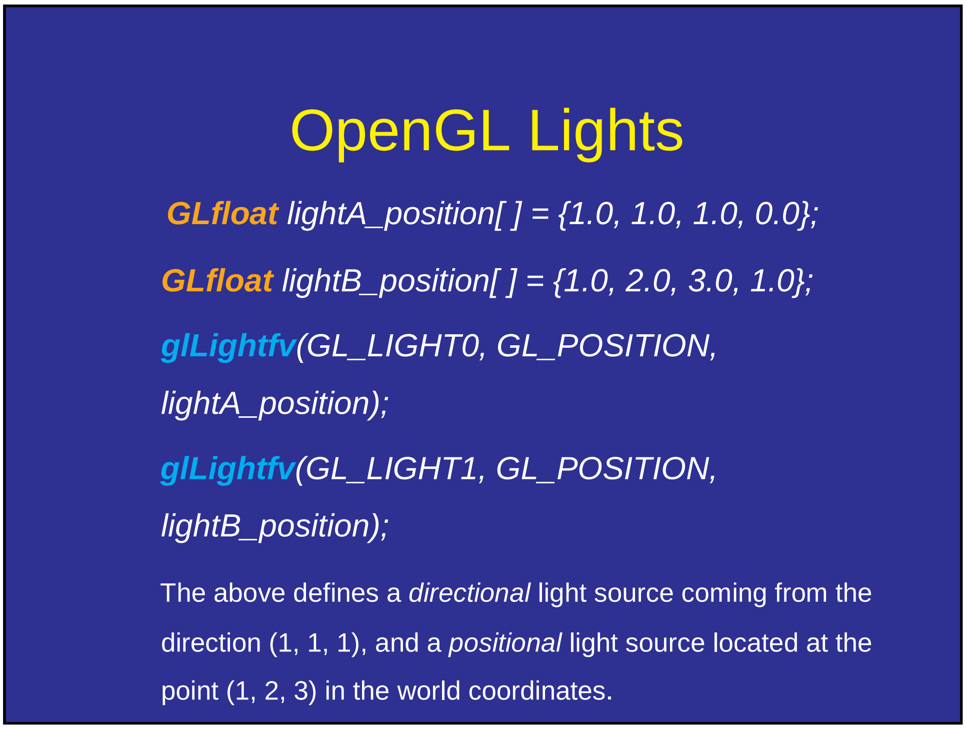
Discussion

- Light Source and/or Viewer at infinity simplifies calculations at the cost of realism
- Need to either clamp colors at max value or normalize them preserving their relative weights (R = R/(R + G + B) , ...)



OpenGL Support for Illumination

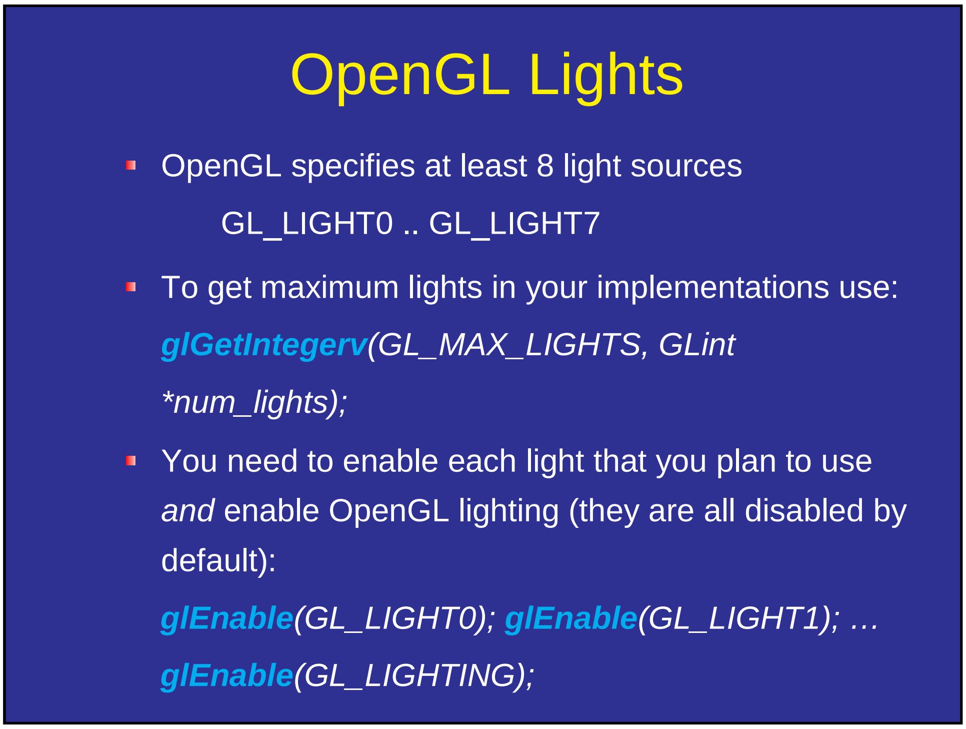
- Ambient, Diffuse, Specular illuminations are supported
- Users have to define lights
  - position, type, color
- Users also define object material
  - Front and/or back facing polygons, color



OpenGL Lights

```
GLfloat lightA_position[ ] = {1.0, 1.0, 1.0, 0.0};
GLfloat lightB_position[ ] = {1.0, 2.0, 3.0, 1.0};
glLightfv(GL_LIGHT0, GL_POSITION,
lightA_position);
glLightfv(GL_LIGHT1, GL_POSITION,
lightB_position);
```

The above defines a *directional* light source coming from the direction (1, 1, 1), and a *positional* light source located at the point (1, 2, 3) in the world coordinates.



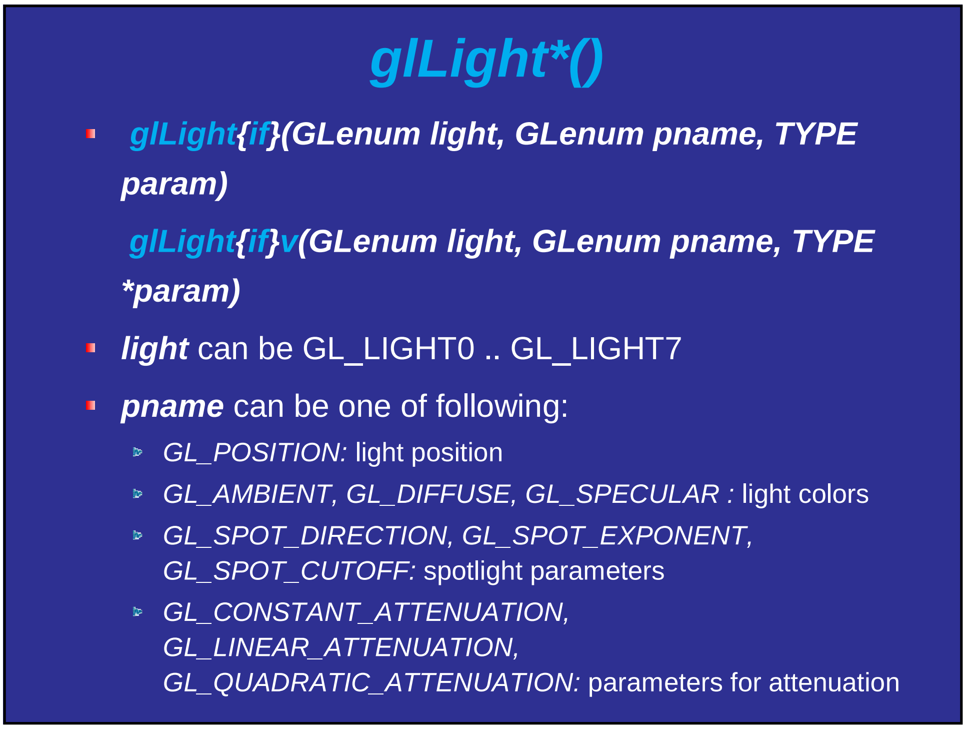
OpenGL Lights

- OpenGL specifies at least 8 light sources

  GL_LIGHT0 .. GL_LIGHT7
- To get maximum lights in your implementations use:

  ```
  glGetIntegerv(GL_MAX_LIGHTS, GLint
  *num_lights);
  ```
- You need to enable each light that you plan to use *and* enable OpenGL lighting (they are all disabled by default):

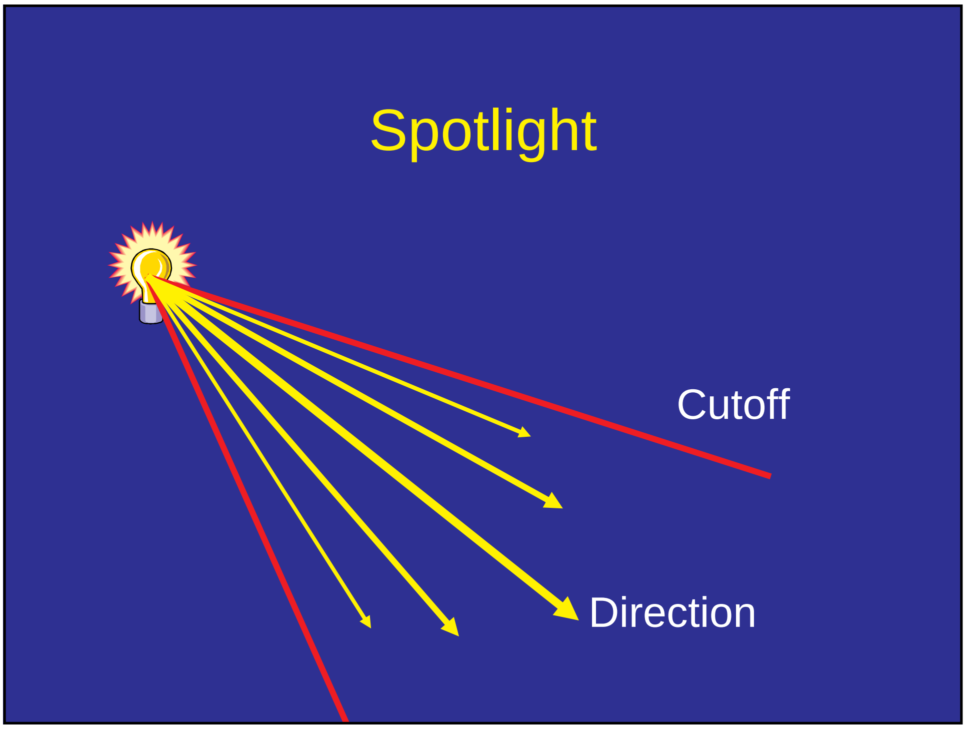
  ```
  glEnable(GL_LIGHT0); glEnable(GL_LIGHT1); …
  glEnable(GL_LIGHTING);
  ```



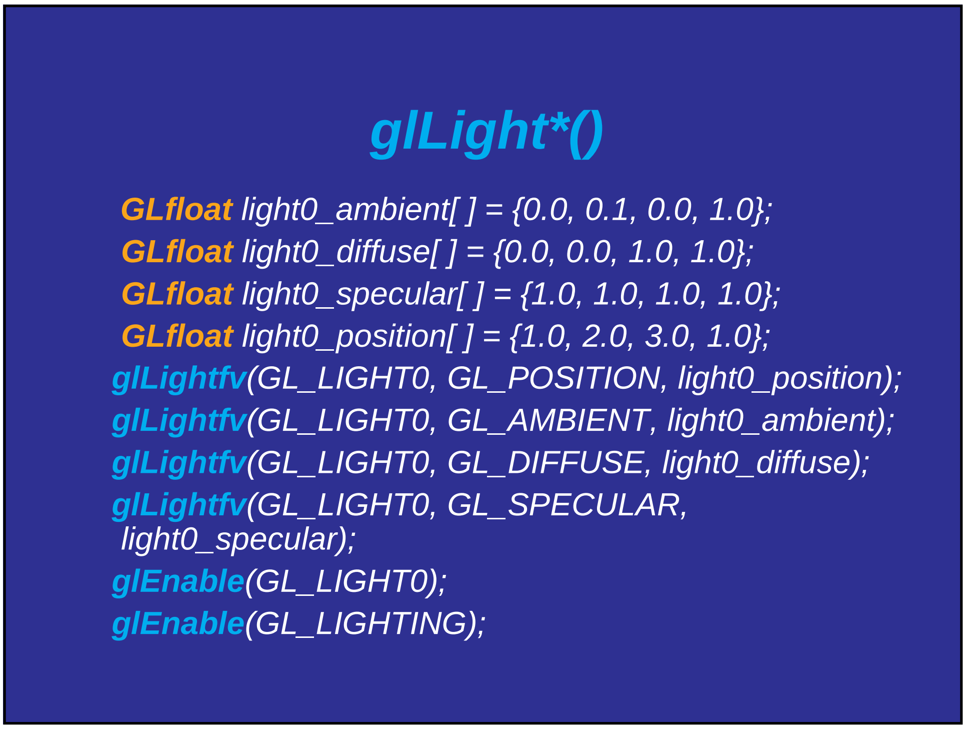
glLight()*

- ***glLight{if}(GLenum light, GLenum pname, TYPE param)***

  glLight{if}v(GLenum light, GLenum pname, TYPE *param)
- ***light*** can be GL_LIGHT0 .. GL_LIGHT7
- ***pname*** can be one of following:
  - *GL_POSITION:* light position
  - *GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR :* light colors
  - *GL_SPOT_DIRECTION, GL_SPOT_EXPONENT, GL_SPOT_CUTOFF:* spotlight parameters
  - *GL_CONSTANT_ATTENUATION, GL_LINEAR_ATTENUATION, GL_QUADRATIC_ATTENUATION:* parameters for attenuation

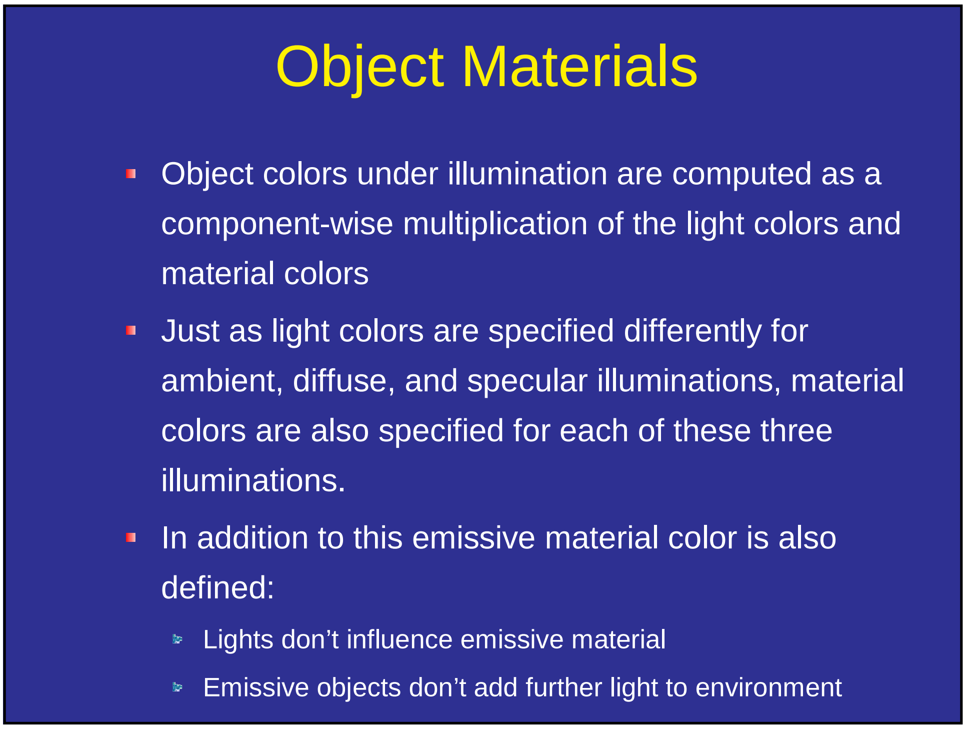


Spotlight



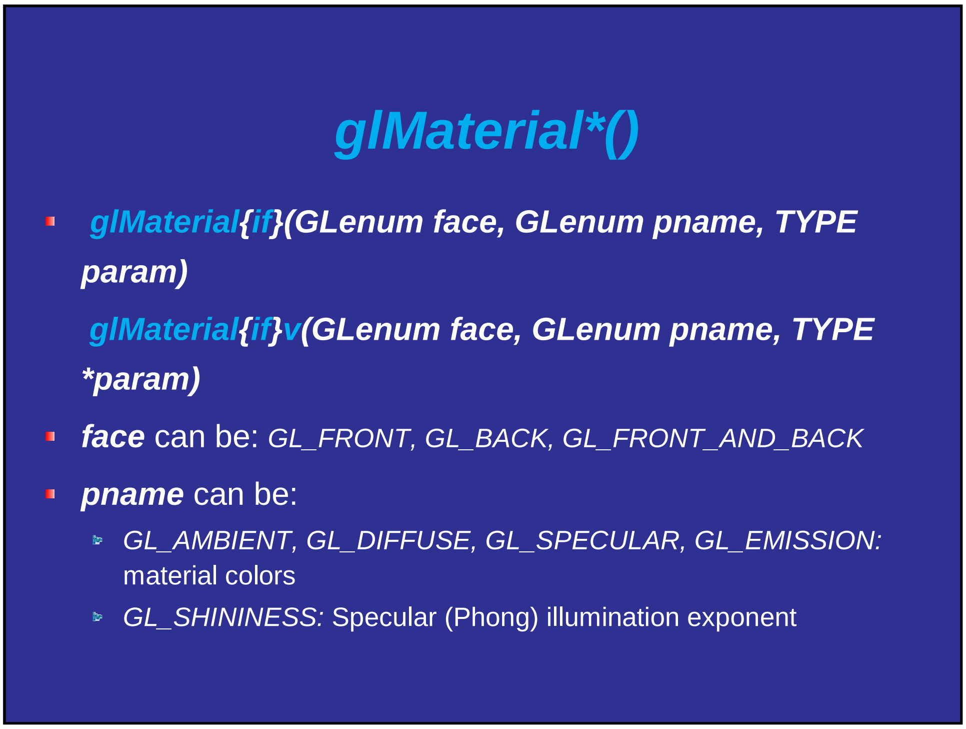
glLight*()

```
GLfloat light0_ambient[ ] = {0.0, 0.1, 0.0, 1.0};
GLfloat light0_diffuse[ ] = {0.0, 0.0, 1.0, 1.0};
GLfloat light0_specular[ ] = {1.0, 1.0, 1.0, 1.0};
GLfloat light0_position[ ] = {1.0, 2.0, 3.0, 1.0};
glLightfv(GL_LIGHT0, GL_POSITION, light0_position);
glLightfv(GL_LIGHT0, GL_AMBIENT, light0_ambient);
glLightfv(GL_LIGHT0, GL_DIFFUSE, light0_diffuse);
glLightfv(GL_LIGHT0, GL_SPECULAR,
 light0_specular);
glEnable(GL_LIGHT0);
glEnable(GL_LIGHTING);
```



Object Materials

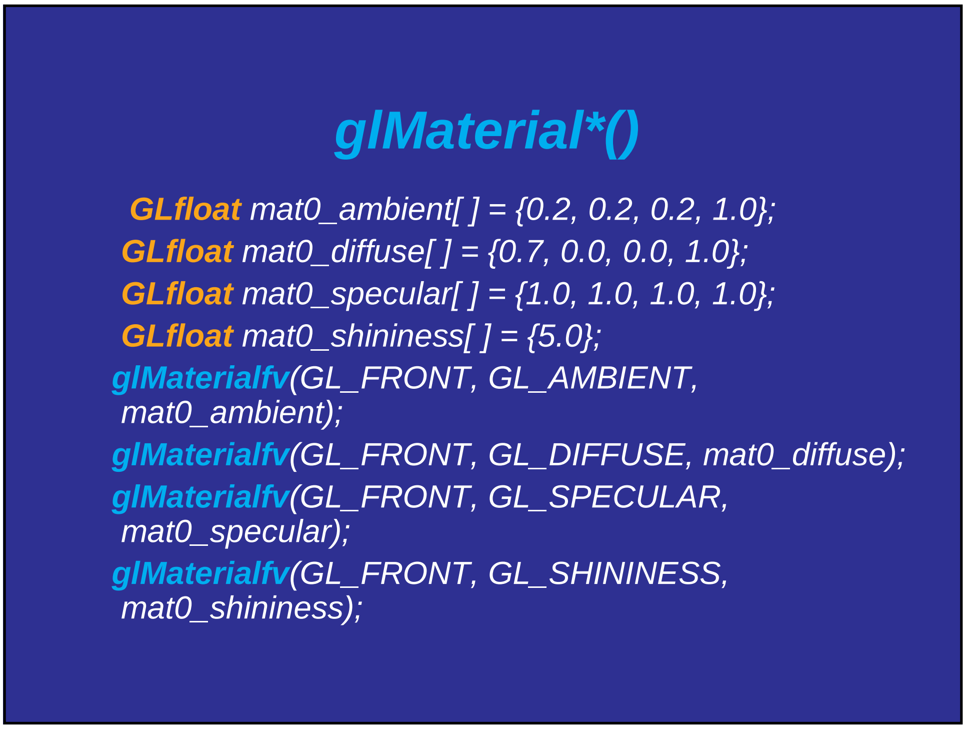
- Object colors under illumination are computed as a component-wise multiplication of the light colors and material colors
- Just as light colors are specified differently for ambient, diffuse, and specular illuminations, material colors are also specified for each of these three illuminations.
- In addition to this emissive material color is also defined:
  - Lights don’t influence emissive material
  - Emissive objects don’t add further light to environment



glMaterial*()

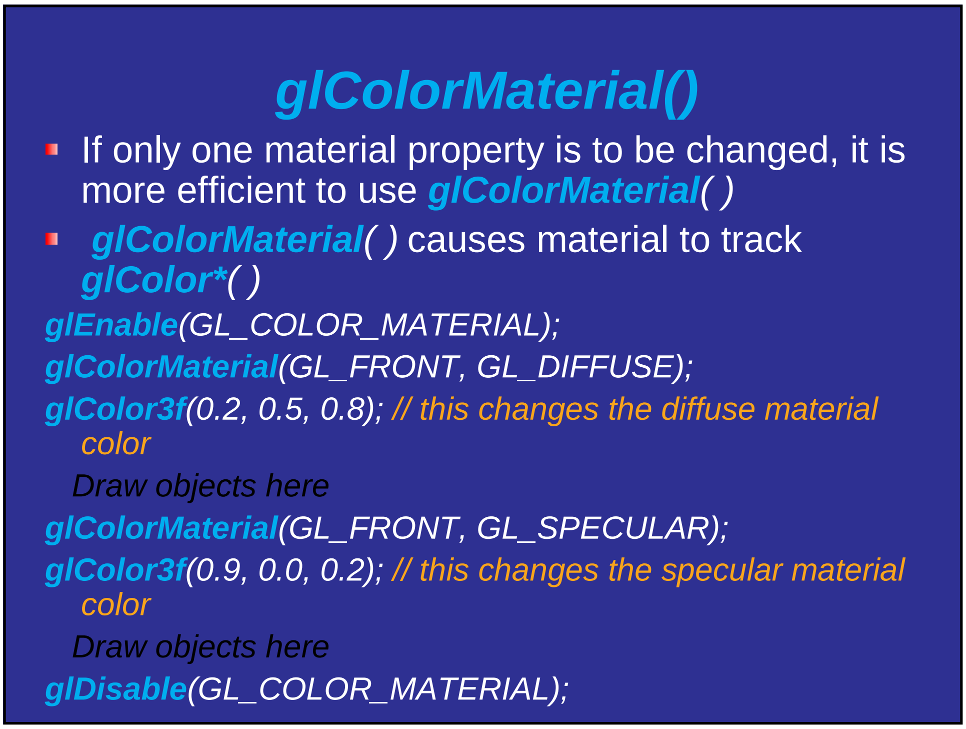
- glMaterial{if}(GLenum face, GLenum pname, TYPE param)

  glMaterial{if}v(GLenum face, GLenum pname, TYPE *param)
- **face** can be: *GL_FRONT, GL_BACK, GL_FRONT_AND_BACK*
- **pname** can be:
  - *GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR, GL_EMISSION:* material colors
  - *GL_SHININESS:* Specular (Phong) illumination exponent



glMaterial*()

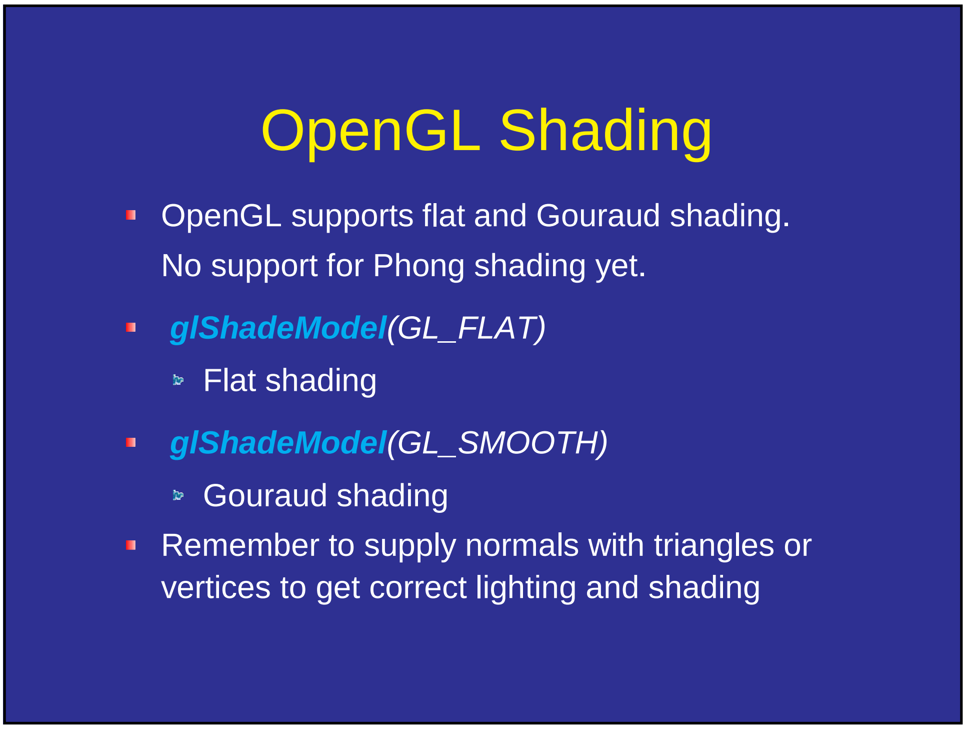
```
 GLfloat mat0_ambient[ ] = {0.2, 0.2, 0.2, 1.0};
GLfloat mat0_diffuse[ ] = {0.7, 0.0, 0.0, 1.0};
GLfloat mat0_specular[ ] = {1.0, 1.0, 1.0, 1.0};
GLfloat mat0_shininess[ ] = {5.0};
glMaterialfv(GL_FRONT, GL_AMBIENT,
 mat0_ambient);
glMaterialfv(GL_FRONT, GL_DIFFUSE, mat0_diffuse);
glMaterialfv(GL_FRONT, GL_SPECULAR,
 mat0_specular);
glMaterialfv(GL_FRONT, GL_SHININESS,
 mat0_shininess);
```



glColorMaterial()

- If only one material property is to be changed, it is more efficient to use *glColorMaterial( )*
- *glColorMaterial( )* causes material to track *glColor*( )*

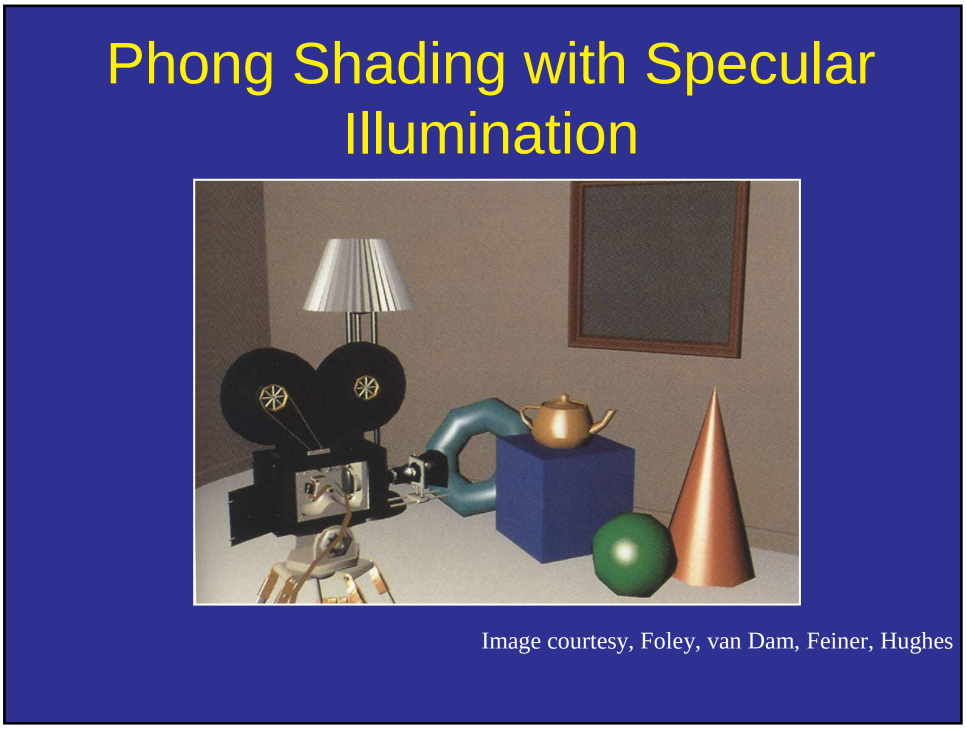
```
glEnable(GL_COLOR_MATERIAL);
glColorMaterial(GL_FRONT, GL_DIFFUSE);
glColor3f(0.2, 0.5, 0.8); // this changes the diffuse material
   color
  Draw objects here
glColorMaterial(GL_FRONT, GL_SPECULAR);
glColor3f(0.9, 0.0, 0.2); // this changes the specular material
   color
  Draw objects here
glDisable(GL_COLOR_MATERIAL);
```



OpenGL Shading

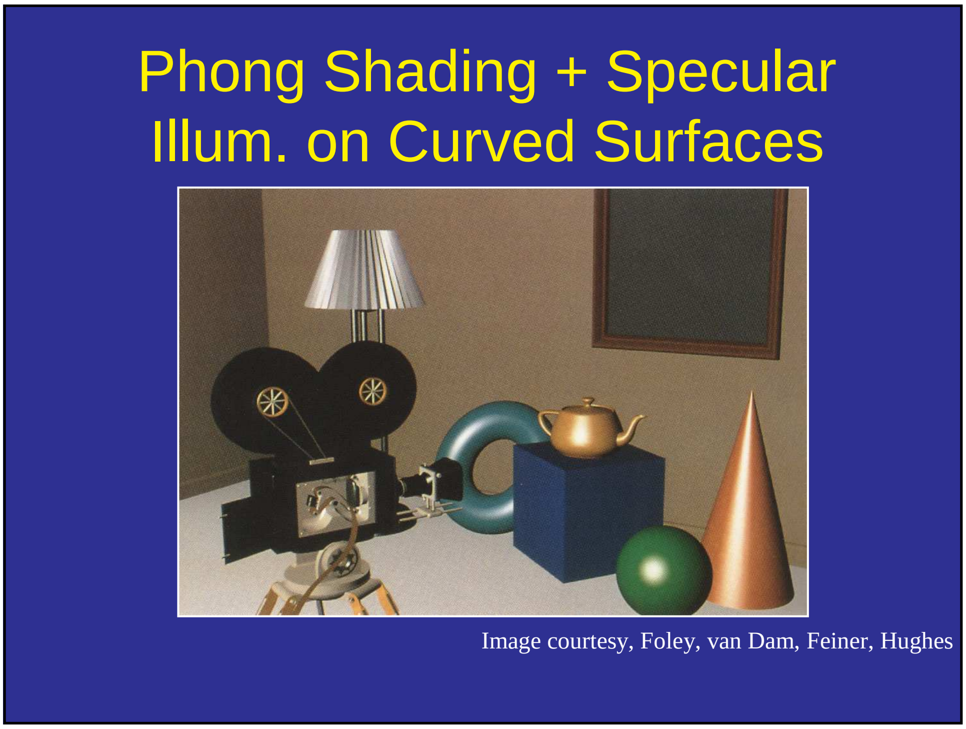
- OpenGL supports flat and Gouraud shading.

  No support for Phong shading yet.
- *glShadeModel(GL_FLAT)*
  - Flat shading
- *glShadeModel(GL_SMOOTH)*
  - Gouraud shading
- Remember to supply normals with triangles or vertices to get correct lighting and shading



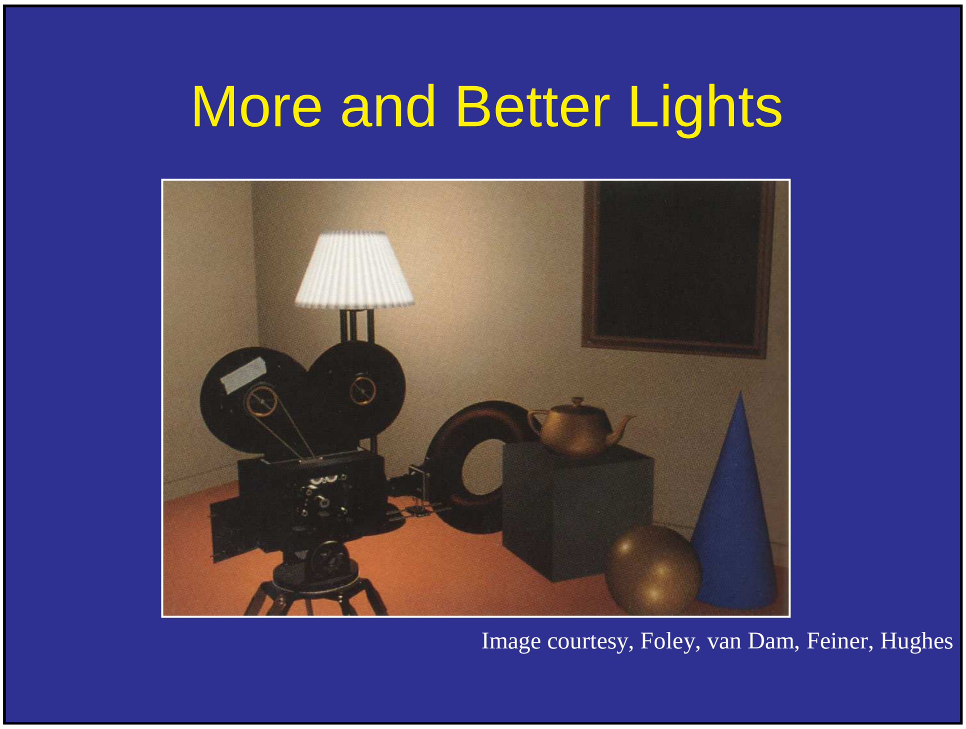
Phong Shading with Specular Illumination

Image courtesy, Foley, van Dam, Feiner, Hughes



Phong Shading + Specular Illum. on Curved Surfaces

Image courtesy, Foley, van Dam, Feiner, Hughes



More and Better Lights

Image courtesy, Foley, van Dam, Feiner, Hughes

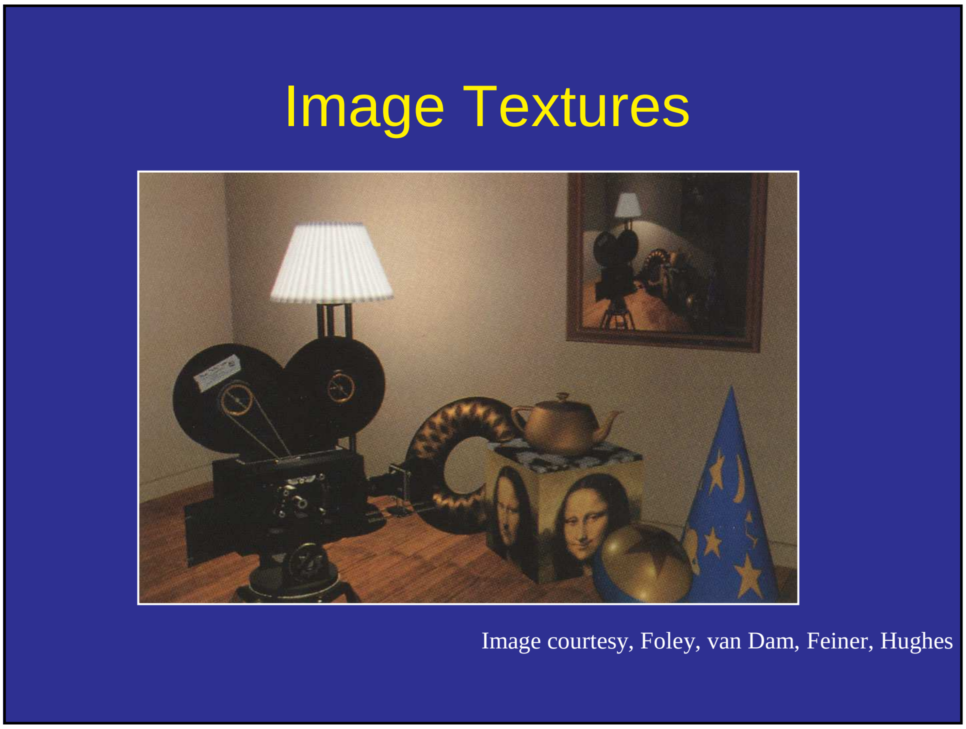
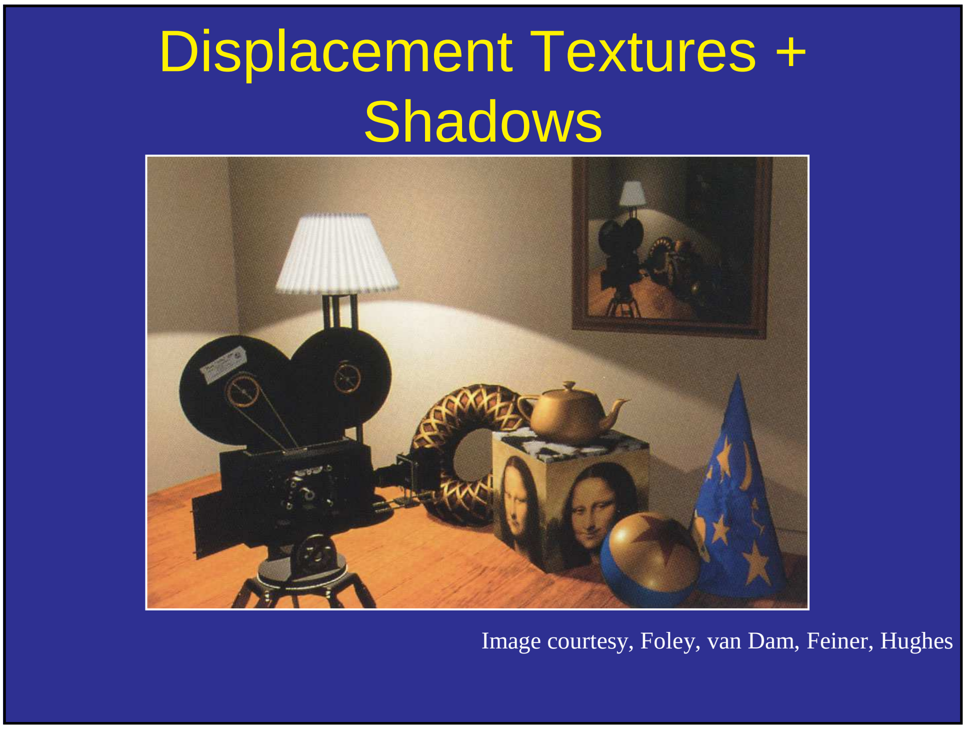


Image Textures

Image courtesy, Foley, van Dam, Feiner, Hughes



Displacement Textures + Shadows

Image courtesy, Foley, van Dam, Feiner, Hughes