OpenGL
Lighting and Texturing

Jacobo Bibliowicz
University of Toronto
March 16, 2008

Why use lighting and textures?

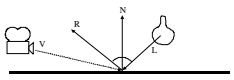
- Add realism.
- Does not increase geometric complexity!
- Models the real world.

Lighting in OpenGL

- OpenGL provides:
  - Phong lighting (materials).
  - Gouraud shading (color interpolation).
- Phong materials are composed of a diffuse (Lambertian) term and a specular term.

$$C \quad k_d \; N \quad L \quad k_s \; R \quad V^n$$

What goes where?



$$C \quad k_d \; N \quad L \quad k_s \; R \quad V^n$$

- Material properties: k_d, k_s, n
- Surface properties: N
  - With the light: L, R
  - With the camera: V

Specifying Materials in OpenGL

- One function for all material properties!

```
glMaterialf{v}(GLenum face, TYPE param);
```

- Example:

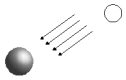
```
GLfloat diff[] = {1.0, 0.0, 0.0, 1.0};
Glfloat spec[] = {0.0, 1.0, 1.0, 1.0};

glMaterialfv(GL_FRONT, GL_DIFFUSE, diff);
glMaterialfv(GL_FRONT, GL_SPECULAR, spec);
glMaterialf(GL_FRONT, GL_SHININESS, 15.0);
```


OpenGL Lights

- Lights are placed in the scene only once.
- OpenGL supports a number of lighting effects:
  - Directional v. Positional lights.
  - Spot lights.
  - Attenuation.
  - Ambient lighting.

Directional v. Positional Lights

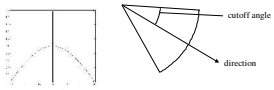
- A directional light is assumed to be infinitely far away (ex. sun)



- A positional light is not so far! (ex. lightbulb)

Spot lights

- Cone shaped lighting.
- Shape determined by
  - cutoff angle
  - direction (assume point light)
  - cosine exponent (shape of fall-off curve).



Attenuation, Ambient Lighting

- Attenuation
  - Reduction in light intensity due to distance from the light.
- Ambient lighting
  - Refers to background lighting caused by many lights.
  - Light scattered by environment
    - Unable to determine its direction.

Specifying OpenGL lights

- Once again, one function! :)
  glLight{if}{v}(<light>, <property>, <val>)

- Example:

```
GLfloat pos[] = {5.0, -3.0, 8.0, 1.0};
GLfloat pos2[] = {1.0, -1.0, 1.0, 0.0};
Glfloat diff[] = {1.0, 1.0, 1.0, 1.0};
Glfloat dir[] = {0.0, -3.0, 3.0};

glLightfv(GL_LIGHT0, GL_POSITION, pos2);
glLightfv(GL_LIGHT0, GL_DIFFUSE, diff);
```

w != 0 for positional light

w == 0 for directional light

Specifying OpenGL lights

```
glLightfv(GL_LIGHT1, GL_POSITION, pos);
glLightfv(GL_LIGHT1, GL_DIFFUSE, diff);
glLightfv(GL_LIGHT1, GL_SPOT_DIRECTION, dir);

glLightf(GL_LIGHT1, GL_SPOT_CUTOFF, 45.0);
glLightf(GL_LIGHT1, GL_SPOT_EXPONENT, 30.0);

...
```


Ambient lighting

- Both lights and materials support an ambient color:

```
glLightfv(GL_LIGHT0, GL_AMBIENT, amb);
glMaterialfv(GL_FRONT, GL_AMBIENT, amb);
```

- OpenGL also provides a GLOBAL ambient term.

```
glLightModelfv(GL_LIGHT_MODEL_AMBIENT, amb);
```

- Objects can glow: emissive materials

```
glMaterialfv(GL_FRONT, GL_EMISSION, em);
```

Mathematics of Lighting

- The final color of a vertex is computed as follows:

```
C = <GLOBAL ambient>*<mat ambient>
                    + <mat emission>;
for each light L
  C += <L attenuation>*<spotlight effects>
     *[ <L ambient>*<mat ambient>
      + <L diffuse>*<mat diffuse>*<diffuse term>
      + <L spec>*<mat spec>*<spec term>];
```


Putting it all together...

- Steps to perform lighting in OpenGL
  1. Create and position your lights (glLight()).
  2. Enable lighting and individual lights
     glEnable(GL_LIGHT0); glEnable(GL_LIGHT1);
     glEnable(GL_LIGHTING);
  3. For each rendered object
     1. Define material properties (glMaterial()).
     2. Provide normals for each rendered vertex!

Lighting Demo

OpenGL Texturing

- Supports traditional 2D textures, also 1D and 3D textures!
- Provides filtering and texture pyramid (mipmap) schemes.
- Multitexturing.
- Texture subimages.
- Texture compression.
- Using image in frame buffer as texture.

Specifying Textures—Step 1

- Suppose scene requires N textures.
- OpenGL needs a mechanism to know what texture you are talking about.
- Texture Objects
  - texture handles (identifiers)

Specifying Textures—Step 1

- Use of texture objects:

```
Gluint texObjs[3];

glGenTextures(3, texObjs);
glBindTexture(GL_TEXTURE_2D, texObjs[0]);
...
// specify texObjs[0] image
...
glBindTexture(GL_TEXTURE_2D, texObjs[1]);
...
// specify texObjs[1] image
...
// etc...
```

Specifying Textures—Step 1

- If you are unsure if an id refers to a texture, try this:

```
glIsTexture(texObj[i]);
```

- When you are done w/ your texture handles, call this:

```
glDeleteTextures(3, texObj);
```


Specifying Textures—Step 2

- Now that we have a texture object, we need to specify the texture image!
  - OpenGL is limited to images whose widths and heights are powers of 2.
  - Specify format of texture image:
    - Of original image (processor memory).
    - Of stored image (graphics board memory).
  - All caveats of glDrawPixels apply here as well (glPixelTransfer, etc...)
  - Textures support an optional pixel for a border.

Specifying Textures—Step 2

- Example:

```
glBindTexture(GL_TEXTURE_2D, texObj);
glTexImage2D(GL_TEXTURE_2D          <- target
             , 0                    <- pyramid level
             , GL_RGBA              <- format when stored inside graphics board
             , width, height, border   <- 0 or 1 only!
             , GL_RGBA
             , GL_UNSIGNED_BYTE
             , pixels
            );
```

- Note: width and height must be of the form 2^m + 2*border.

Specifying Textures—Step 3

- Image is in texture memory.
- Next task: define how texture affects color.
  - Options
    - Replace color completely.
    - Draw on top as a decal.
    - Multiply texture and color (or lighting) values.
    - Blending or adding.
- Specify how to filter textures, and whether to clamp or repeat.

Specifying Textures—Step 3

- Affecting color:

```
glTexEnv(GL_TEXTURE_ENV
         , GL_TEXTURE_ENV_MODE
         , GL_DECAL
        );
```

  - Options: GL_DECAL, GL_REPLACE, GL_MODULATE, GL_ADD, GL_BLEND...
- Filtering:

```
glTexParameterf(GL_TEXTURE_2D
                , GL_MAG_FILTER
                , GL_NEAREST <or GL_LINEAR>);
```


Specifying Textures—Step 3

- Wrapping behaviour:
  - Clamping

  - Repeating

Specifying Textures—Step 3

- Wrapping:

```
glTexParameterfv(GL_TEXTURE_2D
                 , GL_TEXTURE_WRAP_S
                 , GL_CLAMP
                 );
```

  - s,t,r are the texture coordinates.
  - Options are GL_CLAMP, GL_CLAMP_TO_EDGE, GL_CLAMP_TO_BORDER, GL_MIRRORED_REPEAT, and GL_REPEAT.

Specifying Textures—Step 4

- Now we are ready to render!
  - Make sure the correct texture is bound.
  - Draw, specifying texture coordinates:
    glTexCoord{1234}{sifd}{v}().
  - Coordinates range from 0 to 1.0
    - Values > 1.0 cause wrapping!
  - Example:

```
glBegin(GL_TRIANGLE_STRIP);
  glTexCoord2f(0.0, 1.0);
  glVertex3d(x, y, z);
  .....
```


Specifying Textures—Recap

1. Create texture objects
- Provide texture image pixels.

3. Define color modification, filtering, and wrapping behaviour.
- Render specifying texture coordinates.

5. Delete texture objects when you no longer need them.

Texturing Demo

Simple as Beans... hopefully? ;)

- More details in “The Red Book.”
- Any questions?