

Computer Graphics (Spring 2008)

COMS 4160, Lecture 14: OpenGL 3

<http://www.cs.columbia.edu/~cs4160>

Methodology for Lecture

- Lecture deals with lighting (teapot shaded as in HW1)
- Some Nate Robbins tutor demos in lecture
- Briefly explain OpenGL color, lighting, shading
- Demo [4160-opengl.opengl3.opengl3-orig.exe](#)
- Lecture corresponds chapter 5 (and some of 4)
 - But of course, better off doing rather than reading

Importance of Lighting

- Important to bring out 3D appearance (compare teapot now to in previous demo)
- Important for correct shading under lights
- The way shading is done also important



`glShadeModel(GL_FLAT)`



`glShadeModel(GL_SMOOTH)`

Outline

- *Basic ideas and preliminaries*
- Types of materials and shading
 - Ambient, Diffuse, Emissive, Specular
- Source code
- Moving light sources

Brief primer on Color

- Red, Green, Blue primary colors
 - Can be thought of as vertices of a color cube
 - R+G = Yellow, B+G = Cyan, B+R = Magenta, R+G+B = White
 - Each color channel (R,G,B) treated separately
- RGBA 32 bit mode (8 bits per channel) often used
 - A is for alpha for transparency if you need it
- Colors normalized to 0 to 1 range in OpenGL
 - Often represented as 0 to 255 in terms of pixel intensities
- Also, color index mode (not so important)

Shading Models

- So far, lighting disabled: color explicit at each vertex
- This lecture, enable lighting
 - Calculate color at each vertex (based on shading model, lights and material properties of objects)
 - Rasterize and interpolate vertex colors at pixels
- Flat shading: single color per polygon (one vertex)
- Smooth shading: interpolate colors at vertices
- Wireframe: `glPolygonMode(GL_FRONT, GL_LINE)`
 - Also, polygon offsets to superimpose wireframe
 - Hidden line elimination? (polygons in black...)

Demo and Color Plates

- See OpenGL color plates 1-8
- Demo: [4160-opengl/opengl3/opengl3-orig.exe](#)
- Question: Why is blue highlight jerky even with smooth shading, while red highlight is smooth?

Lighting

- Rest of this lecture considers lighting on vertices
- In real world, complex lighting, materials interact
- We study this more formally in next unit
- OpenGL is a hack that efficiently captures some qualitative lighting effects. But not physical
- Modern programmable shaders allow arbitrary lighting and shading models (not covered in class)

Types of Light Sources

- Point
 - Position, Color [separate diffuse/specular]
 - Attenuation (quadratic model) $atten = \frac{1}{k_c + k_d + k_s d^2}$
- Directional (w=0, infinitely far away, no attenuation)
- Spotlights
 - Spot exponent
 - Spot cutoff
- All parameters: page 195 (should have already read HW1)

Material Properties

- Need normals (to calculate how much diffuse, specular, find reflected direction and so on)
- Four terms: Ambient, Diffuse, Specular, Emissive

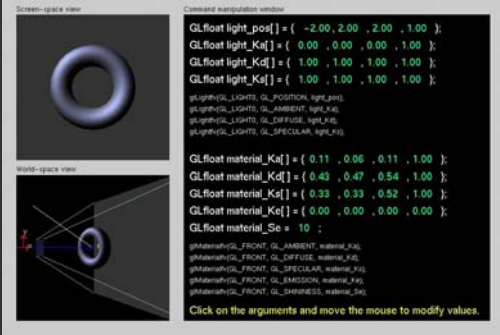
Specifying Normals

- Normals are specified through `glNormal`
- Normals are associated with vertices
- Specifying a normal sets the *current* normal
 - Remains unchanged until user alters it
 - Usual sequence: `glNormal, glVertex, glNormal, glVertex, glNormal, glVertex...`
- Usually, we want unit normals for shading
 - `glEnable(GL_NORMALIZE)`
 - This is slow – either normalize them yourself or don't use `glScale`
- Evaluators will generate normals for curved surfaces
 - Such as splines. GLUT does it automatically for teapot, cylinder,...

Outline

- Basic ideas and preliminaries
- *Types of materials and shading*
 - *Ambient, Diffuse, Emissive, Specular*
- Source code
- Moving light sources

LightMaterial Demo



Screen-space view

World-space view

```

Command assignment window
GLfloat light_pos[] = { -2.00, 2.00, 2.00, 1.00 };
GLfloat light_Ka[] = { 0.00, 0.00, 0.00, 1.00 };
GLfloat light_Kd[] = { 1.00, 1.00, 1.00, 1.00 };
GLfloat light_Ks[] = { 1.00, 1.00, 1.00, 1.00 };

glLight(GL_LIGHT0, GL_POSITION, light_pos);
glLight(GL_LIGHT0, GL_AMBIENT, light_Ka);
glLight(GL_LIGHT0, GL_DIFFUSE, light_Kd);
glLight(GL_LIGHT0, GL_SPECULAR, light_Ks);

GLfloat material_Ka[] = { 0.11, 0.06, 0.11, 1.00 };
GLfloat material_Kd[] = { 0.43, 0.47, 0.54, 1.00 };
GLfloat material_Ks[] = { 0.33, 0.33, 0.52, 1.00 };
GLfloat material_Ke[] = { 0.00, 0.00, 0.00, 0.00 };
GLfloat material_Se = 10 ;

glMaterial(GL_FRONT, GL_AMBIENT, material_Ka);
glMaterial(GL_FRONT, GL_DIFFUSE, material_Kd);
glMaterial(GL_FRONT, GL_SPECULAR, material_Ks);
glMaterial(GL_FRONT, GL_EMISSION, material_Ke);
glMaterial(GL_FRONT, GL_SHININESS, material_Se);

Click on the arguments and move the mouse to modify values.
    
```

Emissive Term

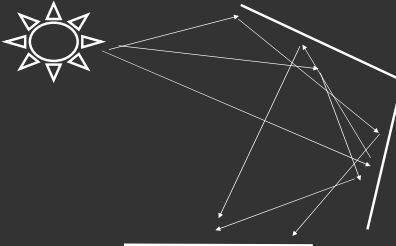


$$I = Emission_{material}$$

- Only relevant for light sources when looking directly at them
- Gotcha: must create geometry to actually see light
- Emission does not in itself affect other lighting calculations

Ambient Term

- Hack to simulate multiple bounces, scattering of light
- Assume light equally from all directions



Ambient Term

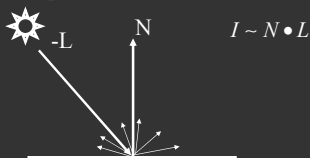
- Associated with each light and overall light
- E.g. skylight, with light from everywhere

$$I = ambient_{global} * ambient_{material} + \sum_{i=0}^n ambient_{light_i} * ambient_{material} * atten_i$$

Most effects per light involve linearly combining effects of light sources

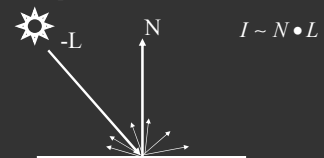
Diffuse Term

- Rough matte (technically Lambertian) surfaces
- Light reflects equally in all directions



Diffuse Term

- Rough matte (technically Lambertian) surfaces
- Light reflects equally in all directions

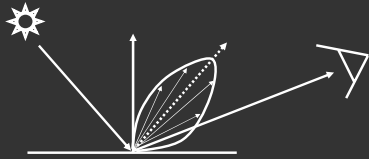


$$I = \sum_{i=0}^n diffuse_{light_i} * diffuse_{material} * atten_i * [\max(L \cdot N, 0)]$$

- Why is diffuse of light diff from ambient, specular?

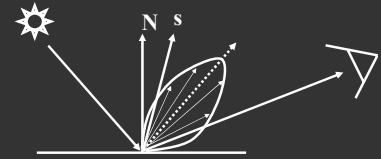
Specular Term

- Glossy objects, specular reflections
- Light reflects close to mirror direction



Specular Term

- Glossy objects, specular reflections
- Light reflects close to mirror direction
- Consider half-angle between light and viewer



$$I = \sum_{i=0}^n \text{specular}_{light_i} * \text{specular}_{material} * \text{atten}_i * [\max(N \cdot s, 0)]^{\text{shininess}}$$

Demo

- What happens when we make surface less shiny?
- What happens to jerkiness of highlights?

Outline

- Basic ideas and preliminaries
- Types of materials and shading
 - Ambient, Diffuse, Emissive, Specular
- *Source code*
- Moving light sources

Source Code (in display)

```
/* New for Demo 3; add lighting effects */
/* See hw1 and the red book (chapter 5) for details */
{
  GLfloat one[] = {1, 1, 1, 1};
  //   GLfloat small[] = {0.2, 0.2, 0.2, 1};
  GLfloat medium[] = {0.5, 0.5, 0.5, 1};
  GLfloat small[] = {0.2, 0.2, 0.2, 1};
  GLfloat high[] = {100};
  GLfloat light_specular[] = {1, 0.5, 0, 1};
  GLfloat light_specular1[] = {0, 0.5, 1, 1};
  GLfloat light_position[] = {0.5, 0, 0, 1};
  GLfloat light_position1[] = {0, -0.5, 0, 1};

  /* Set Material properties for the teapot */
  glMaterialfv(GL_FRONT, GL_AMBIENT, one);
  glMaterialfv(GL_FRONT, GL_SPECULAR, one);
  glMaterialfv(GL_FRONT, GL_DIFFUSE, medium);
  glMaterialfv(GL_FRONT, GL_SHININESS, high);
}
```

Source Code (contd)

```
/* Set up point lights, Light 0 and Light 1 */
/* Note that the other parameters are default values */

glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
glLightfv(GL_LIGHT0, GL_DIFFUSE, small);
glLightfv(GL_LIGHT0, GL_POSITION, light_position);

glLightfv(GL_LIGHT1, GL_SPECULAR, light_specular1);
glLightfv(GL_LIGHT1, GL_DIFFUSE, medium);
glLightfv(GL_LIGHT1, GL_POSITION, light_position1);

/* Enable and Disable everything around the teapot */
/* Generally, we would also need to define normals etc. */
/* But glut already does this for us */

glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glEnable(GL_LIGHT1);
if (smooth) glShadeModel(GL_SMOOTH); else glShadeModel(GL_FLAT);
}
```

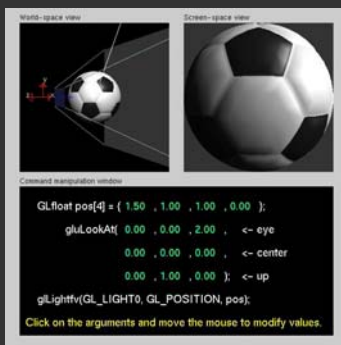
Outline

- Basic ideas and preliminaries
- Types of materials and shading
 - Ambient, Diffuse, Emissive, Specular
- Source code
- *Moving light sources*

Moving a Light Source

- Lights transform like other geometry
- Only modelview matrix (not projection). The only real application where the distinction is important
- See types of light motion pages 202-
 - Stationary light: set the transforms to identity before specifying it
 - Moving light: Push Matrix, move light, Pop Matrix
 - Moving light source with viewpoint (attached to camera). Can simply set light to 0 0 0 so origin wrt eye coords (make modelview matrix identity before doing this)

Lightposition demo



The demo interface consists of two side-by-side windows at the top. The left window, labeled 'World-space view', shows a 3D scene with a soccer ball and a light source (represented by a red dot) with a cone of light rays. The right window, labeled 'Screen-space view', shows a 2D projection of the soccer ball. Below these windows is a 'Command manipulation window' containing the following code:

```
GLfloat pos[4] = { 1.50 , 1.00 , 1.00 , 0.00 };
gluLookAt( 0.00 , 0.00 , 2.00 , <- eye
          0.00 , 0.00 , 0.00 , <- center
          0.00 , 1.00 , 0.00 ); <- up
glLightfv(GL_LIGHT0, GL_POSITION, pos);
```

Click on the arguments and move the mouse to modify values.