Foundations of Computer Graphics (Spring 2010)
CS 184, Lecture 11: OpenGL 3
http://inst.eecs.berkeley.edu/~cs184

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
- Also talk a little about programmable shaders
- Demo
- 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
- 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 (briefly covered)

### Types of Light Sources
- **Point**
  - Position, Color [separate diffuse/specular]
  - Attenuation (quadratic model) \( \text{atten} = \frac{1}{k_d + k_a d^2 + k_r d^4} \)
- **Directional** (\( w=0 \), infinitely far away, no attenuation)
- **Spotlights**
  - Spot exponent
  - Spot cutoff
- All parameters: page 215 (should have already read HW1; see there for page numbers in previous editions)

### 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
**Light Material Demo**

**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

\[ I = \text{ambient}_{global} \cdot \text{ambient}_{material} + \sum_{\text{ambient}_i} \text{ambient}_{material}_i \cdot \text{atten}_i \]

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

\[ I = \sum_{\text{diffuse}_i} \text{diffuse}_{material}_i \cdot \text{atten}_i \cdot [\max (L \cdot N, 0)] \]

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

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

\[
I = \sum_{i=0}^{n} \text{specular}_{i} \times \text{specular}_{i} \times \text{atten}_{i} \times \max (N \cdot x, 0)^{\text{shininess}}
\]

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)

```c
/* 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 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)

```c
/* 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 222-
  - 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

Pixel or Fragment Pipeline

Rasterization (scan conversion)  Texture Mapping  Z-buffering  Framebuffer

These fixed function stages can be replaced by a general per-fragment calculation using fragment shaders in modern programmable hardware

Shading Languages

- Vertex / Fragment shading described by small program
- Written in language similar to C but with restrictions
- Long history. Cook’s paper on Shade Trees, Renderman for offline rendering
- Stanford Real-Time Shading Language, work at SGI
- Cg from NVIDIA, HLSL
- GLSL directly compatible with OpenGL 2.0 (So, you can just read the OpenGL Red Book to get started)

Shader Setup

Cliff Lindsay web.cs.wpi.edu/~rich/courses/imgd4000-d09/lectures/gpu.pdf
Phong Shader: Vertex

This Shader Does
- Gives eye space location for \( v \)
- Transform Surface Normal
- Transform Vertex Location

```cpp
varying vec3 N;
varying vec3 v;

void main(void)
{
    v = vec3(gl_ModelViewMatrix * gl_Vertex);
    N = normalize(gl_NormalMatrix * gl_Normal);
    gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
    (Updates OpenGL Built-In Variable for Vertex Position)
}
```

Cliff Lindsay web.cs.wpi.edu/~rich/courses/imgd4000-d09/lectures/gpu.pdf

Phong Shader: Fragment

```cpp
void main (void) {
    if (we are in Eye Coordinates, so EyePos is (0,0,0))
        vec3 E = normalize(gl_Position, eye_pos);
    else
        vec3 E = normalize(eye_pos - gl_Position);

    // Calculate Ambient Term:
    vec4 lamba = gl_FrontLightProduct[0].ambient;

    // Calculate Diffuse Term:
    vec3 N = normalize(gl_NormalMatrix * gl_Normal);
    float diffuse = max(dot(E, N), 0.0);

    // Calculate Specular Term:
    vec3 R = normalize(gl_FrontLightProduct[0].specular);
    float specular = pow(max(dot(R, E), 0.0), gl_FrontMaterial.shininess);

    // Write Total Color:
    gl_FragColor = gl_FrontLightModelProduct.ambientColor + lamba + diffuse + specular;
}
```

Cliff Lindsay web.cs.wpi.edu/~rich/courses/imgd4000-d09/lectures/gpu.pdf