Illumination and Reflection in OpenGL

Shadows

Illumination & Reflection in OpenGL

OpenGL Uses the Phong Illumination/Reflection Model

Final Phong Illumination/Reflection Model Result (Single White Light Source)

Three color intensity equations:
\[ I(r,g,b) = Ambient + Point Diffuse + Point Specular \]
\[ I(r,g,b) = kd(r,g,b)I_a + Ipkd(r,g,b)(N \cdot L) + Ipks(R \cdot V)n \]

OpenGL generalizes this to include colored light sources

Illumination & Reflection in OpenGL

Define Light Sources
Define Material Properties
Define Normal Vectors
Specify Shading Model
Enable Depth Testing (Z-Buffer)

Defining a Light Source

Set up Arrays of lighting values

- Intensities:
  - GLfloat ambLight0[] = {0.3f, 0.3f, 0.3f, 1.0f}; // R,G,B,A
  - GLfloat diffLight0[] = {0.5f, 0.5f, 0.5f, 1.0f};
  - GLfloat specLight0[] = {0.0f, 0.0f, 0.0f, 1.0f};

- Position:
  - GLfloat posnLight0[] = {1.0f, 1.0f, 1.0f, 0.0f}; // x,y,z,w

Pass Arrays to OpenGL

glLightfv(GL_LIGHT0, GL_AMBIENT, ambLight0);
glLightfv(GL_LIGHT0, GL_DIFFUSE, diffLight0);
glLightfv(GL_LIGHT0, GL_SPECULAR, specLight0);
glLightfv(GL_LIGHT0, GL_POSITION, posnLight0);
### Enabling a Light Source
- Turn on Lighting
  ```
glEnable(GL_LIGHTING);
  ```
- Turn on a Light Source
  ```
glEnable(GL_LIGHT0);
  ```

### Material Reflection Properties
- Set up Material Arrays
  - ambient/diffuse reflection coefficients
    ```
    GLfloat mat_ambdiff[] = {0.0f, 0.7f, 0.0f, 1.0f};
    ```
  - specular reflection coefficient
    ```
    GLfloat mat_spec[] = {1.0f, 1.0f, 1.0f, 1.0f};
    ```
- Pass Material Arrays to OpenGL
  ```
glMaterialf(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, mat_ambdiff);
glMaterialf(GL_FRONT, GL_SPECULAR, mat_spec);
glMaterialf(GL_FRONT, GL_SHININESS, 20.0f);
  ```

### Defining Normals
- Must compute normals for all polygons
- OpenGL has no function to do that
  - So write your own
- Assume the result is:
  ```
  double n[3];
  ```
- Use this when you define the polygon
  ```
  glBegin(GL_POLYGON)
  glNormal3f((GLfloat)n[0], (GLfloat)n[1], (GLfloat)n[2]); // glVertex3f() calls here for polygon vertices
  glEnd();
  ```

### Specify a Shading Model & Enable Depth Testing
- glShadeModel(GL_FLAT); // use GL_SMOOTH
  ```
  // for Gouraud shading
  ```
- glEnable(GL_DEPTH_TEST);
- glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  ```
  // clear frame buffer and z-buffer
  ```

### Some sample code - view class::OnDraw()
```java
  glShadeModel(GL_SMOOTH);
glEnable(GL_DEPTH_TEST);
glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
glMatrixMode(GL_MODELVIEW); glLoadIdentity();
glLightfv(GL_LIGHT0, GL_AMBIENT, ambLight0);         // the ambient...
glLightfv(GL_LIGHT0, GL_DIFFUSE, diffLight0);            // and diffuse...
glLightfv(GL_LIGHT0, GL_SPECULAR, specLight0);     // and specular...
glLightfv(GL_LIGHT0, GL_POSITION, posnLight0);       // position...
glEnable(GL_LIGHTING);glEnable(GL_LIGHT0);
DrawCube();
gFlush();
  ```

### Code from DrawCube() function
```java
  glTranslatef(0.0f, 0.0f, -3.0f);    // position cube
  glRotatef(20.0f, 1.0f, 0.0f, 0.0f);
glEnable(GL_DEPTH_TEST);
  ```
  // for Gouraud shading
  ```
glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  ```
```java
  glShadeModel(GL_SMOOTH);
glEnable(GL_DEPTH_TEST);
glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
glMatrixMode(GL_MODELVIEW); glLoadIdentity();
glLightfv(GL_LIGHT0, GL_AMBIENT, ambLight0);         // the ambient...
glLightfv(GL_LIGHT0, GL_DIFFUSE, diffLight0);            // and diffuse...
glLightfv(GL_LIGHT0, GL_SPECULAR, specLight0);     // and specular...
glLightfv(GL_LIGHT0, GL_POSITION, posnLight0);       // position...
glEnable(GL_LIGHTING);glEnable(GL_LIGHT0);
DrawCube();
gFlush();
  ```
Code from CalcNormal (double *p1, double *p2, double *p3, double *n)

// Form two vectors from the points.
double a[3], b[3];
// Calculate the cross product of the two vectors.
// Normalize the new vector.
double length = sqrt(r[0]*r[0]+r[1]*r[1]+r[2]*r[2]);
r[0] = r[0] / length;

Shadows

- Very important to our perception of depth
- Shadow position/orientation give information as to how objects relate to each other in space

Sharp Shadows from Point Sources

<table>
<thead>
<tr>
<th>Point source of light</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRIGHT</td>
<td>This region receives no light...completely dark.</td>
</tr>
</tbody>
</table>

Soft Shadows from Extended Sources

Outside end lines: light received from B

A

B

Object

Outside black line: light received from A

Light received from both A and B

Light from A, but not B

Completely Dark

Light from B, but not A

Light received from both A and B

Undefined: central area that receives no light (complete shadow)

Fombrum: areas in partial shadow (receive light from part of source)

Shadows from Point Sources

- Look at shadows from point sources
- If a point is in shadow, set Phong I_p to 0
  - Source gets no light from point source
  - So no reflection from point source
  - Still must include ambient term
- Lots of algorithms
- One of simplest: Shadow Z-Buffer
Shadow Z-Buffer Algorithm
A two-stage process
1. Take Light Source as viewpoint & compute depths
   • Store results in shadow Z-buffer $Z'[x'[y']$
   • Each $Z'[x'[y']$ will contain distance of closest
     surface to light source
2. Normal Z-Buffer rendering
   • But if $(x,y)$ is closest (visible), transform to light
     space coordinates $(x',y',z')$
   • If $z' > Z'[x'[y']$ point is in shadow
     - Some object is closer to light & will block it
     - So only include ambient term in computation

Shadow Z-Buffer
Set up shadow Z-buffer $Z'[x'[y']$ using coordinate
system whose origin is at light source
$Z-buf[x][y]=\infty$ for all $x,y$
for each polygon
for each pixel $x,y$
calculate $z$
if $z < Z-buf[x][y]$
   transform $x,y,z$ to light coord space $x',y',z'$
if $z' > Z'[x'[y']$
   reduce intensity (include only ambient)
$Z-buf[x][y]=z; fb[x][y]=intensity$