Binghamton University

EngiNet™

State University of New York

Thomas J. Watson
School of Engineering and Applied Science

EngiNet™

WARNING
All rights reserved. No Part of this video lecture series may be reproduced in any form or by any electronic or mechanical means, including the use of information storage and retrieval systems, without written approval from the copyright owner.

©2001 The Research Foundation of the State University of New York

CS 460/560
Computer Graphics
Professor Richard Eckert
Lecture # 23
May 2, 2001
Lecture 23
- 3D Graphics using OpenGL
  - Illumination and Shading
- Hidden Surface Removal Revisited
  - Depth Sort
- Shadows

Illumination & Reflection in OpenGL
- Uses the Phong Illumination/Reflection Model

Final Phong Illumination/Reflection Model Result (Single Light Source)
- Three color intensity equations:
  \[ l(r,g,b) = \text{Ambient} + \text{Point Diffuse} + \text{Point Specular} \]
  \[ l(r,g,b) = kd(r,g,b)Ia \]
  + \[ ip^*kd(r,g,b)(N.L) \]
  + \[ ip^*ks(R,V) \]

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
  \[ \text{GLfloat ambLight0}[] = \{0.3f, 0.3f, 0.3f, 1.0f\}; \ \
  \text{GLfloat diffLight0}[] = \{0.5f, 0.5f, 0.5f, 1.0f\}; \]
  \[ \text{GLfloat specLight0}[] = \{0.0f, 0.0f, 0.0f, 1.0f\}; \]
  \[ \text{GLfloat posnLight0}[] = \{1.0f, 1.0f, 1.0f, 0.0f\}; \ \
  \text{GLfloat x,y,z,w} \]
- Pass Arrays to OpenGL
  \[ \text{glLightfv(GL_LIGHT0, GL_AMBIENT, ambLight0)}; \]
  \[ \text{glLightfv(GL_LIGHT0, GL_DIFFUSE, diffLight0)}; \]
  \[ \text{glLightfv(GL_LIGHT0, GL_SPECULAR, specLight0)}; \]
  \[ \text{glLightfv(GL_LIGHT0, GL_POSITION, posnLight0)}; \]

Enabling a Light Source
- Turning on the light source
  \[ \text{glEnable(GL_LIGHTING)}; \]
  \[ \text{glEnable(GL_LIGHT0)}; \]
Material Reflection Properties

- Set up Material Arrays
  - ambient/diffuse reflection coefficients
    \[ \text{GLfloat mat\_ambient}[0] = [0.0f, 0.7f, 0.0f, 1.0f]; \]
  - specular reflection coefficient
    \[ \text{GLfloat mat\_spec}[0] = [1.0f, 1.0f, 1.0f, 1.0f]; \]
- Pass Material Arrays to OpenGL
  `glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, mat\_ambient);
  glMaterialfv(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)

`glShadeModel(GL_SMOOTH);
glEnable(GL_DEPTH_TEST);
gClearColor(1.0f, 1.0f, 1.0f, 1.0f);
gColor3f(GL_RED, 0.7f, 0.1f, 0.3f);
gMaterialfv(GL_FRONT, GL_AMBIENT, [0.0f, 0.7f, 0.0f, 1.0f]);
gMaterialfv(GL_FRONT, GL_SPECULAR, [1.0f, 1.0f, 1.0f, 1.0f]);
MatrixMode(GL_MODELVIEW);
gLoadIdentity();
gMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, mat\_ambient);
gMat\_spec[0] = [1.0f, 1.0f, 1.0f, 1.0f];
gMaterialf(GL_FRONT, GL_SHININESS, 20.0f);

Code from CalcNormal function

`glBegin(GL_POLYGON);
  glNormal3f(GLfloat[n][0], GLfloat[n][1], GLfloat[n][2]);
  glMat\_ambient[0] = [1.0f, 1.0f, 1.0f, 1.0f];
  glMaterialf(GL_FRONT, GL_SHININESS, 20.0f);
  glEnable(GL_LIGHT0);
  glEnable(GL_LIGHTING);
  glLightf(GL_LIGHT0, GL_AMBIENT, ambLight0);
  glLightf(GL_LIGHT0, GL_DIFFUSE, diffLight0);
  glLightf(GL_LIGHT0, GL_SPECULAR, specLight0);
  glEnable(GL_LIGHTING);
  DrawCube();
  glEnd();`

Code from DrawCube function

`glTranslate(0.0f, 0.0f, -3.0f); // position cube
  glRotatef(20.0f, 1.0f, 0.0f, 0.0f);
  glRotatef(20.0f, 0.0f, 1.0f, 0.0f);
  glMat\_ambient[0] = [1.0f, 1.0f, 1.0f, 1.0f];
  glMaterialf(GL_FRONT, GL_SHININESS, 20.0f);
  glLightf(GL_LIGHT0, GL_AMBIENT, ambLight0);
  glLightf(GL_LIGHT0, GL_DIFFUSE, diffLight0);
  glLightf(GL_LIGHT0, GL_SPECULAR, specLight0);
  glEnable(GL_LIGHTING);
  DrawCube();
  glEnd();`
Back to Hidden Surface Removal

- The Depth Sort Technique

Depth Sort Hidden Surface Removal

- Basic Idea
  - Order Polygons according to how far away from observer
  - Then “paint” them into picture, farthest first, closest last
    - Far surfaces overwritten by near surfaces
  - The way artists sometimes paint scenes

Algorithm

1. Remove back faces (preprocessing step)
2. Decompose remaining polygons into triangles
   - Depth determination easiest for triangles
3. Sort triangles into depth order
4. Apply Painters Algorithm

The Depth Sort (step 3)

For each triangle
  - Create a linked list with pointers to all triangles in front of the triangle
  - And a counter of all triangles in back of the triangle
  - Gives an array of linked lists

An Example

<table>
<thead>
<tr>
<th>Triangle counter</th>
<th>list</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6 3 2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>2 3 5</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>4 2</td>
</tr>
<tr>
<td>6</td>
<td>1 3</td>
</tr>
</tbody>
</table>

Painter’s Algorithm (step 4)

Repeat
  - Go through array & draw any triangle whose counter = 0  // it’s the farthest
  - Decrement counter of all triangles in list of drawn triangle  // since there’s 1 less in back
  - Mark counter of drawn triangle as finished
Until all triangles have been drawn
**Painter's Algorithm in Action (A)**

<table>
<thead>
<tr>
<th>Triangle</th>
<th>List</th>
<th>Coverage</th>
<th>Coverage after</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2 3 4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Painter's Algorithm in Action (B)**

<table>
<thead>
<tr>
<th>Triangle</th>
<th>List</th>
<th>Coverage</th>
<th>Coverage after</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3 4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Depth Sort (Step 3) Details**

Initialize all triangle counters to 0
for i = 1 to n-1
  for j = i+1 to n
    if triangles i & j "overlap"
      if triangle j is "in front of" triangle i
        add triangle j to i's list
        ctr[i]++
      else
        add triangle i to j's list
        ctr[j]++

**Determine if Triangles “overlap”**

1. Mini-max test (inexpensive)
   - look at containing rectangles
   - if rectangles don't overlap, triangles don't

2. So check for intersections of pairs of edges
   - we're working with x-y projections
   - if projected edges intersect, triangles overlap
   - 9 tests in all
   - Can use parametric equations of edges to look for intersections

**But if rectangles overlap, triangles may not**

---

---
2. Test for Intersection Points between Edges

OVERLAP-- Test for Intersection Points between edges

Can Use Parametric Equations

\[ x = x_1 + (x_2 - x_1)t \]
\[ y = y_1 + (y_2 - y_1)t \]

\[ x = x_3 + (x_4 - x_3)s \]
\[ y = y_3 + (y_4 - y_3)s \]

Equate & solve for \( s \) & \( t \)

if \( 0 < s < 1 \) & \( 0 < t < 1 \)

the edges intersect

3. But if there are no intersections, triangles still could overlap

- One triangle’s projection could contain the other’s
- So test for containment

3. Test for Containment

Which Side of a Line is a Point on

\[ f(x, y) = (x-x_1)(y-y_1) - (y-y_1)(x-x_1) \]

on line

\[ f(x, y) = 0 \Rightarrow (x, y) \text{ on line} \]

\[ f(x, y) < 0 \Rightarrow (x, y) \text{ on one side of line} \]

\[ f(x, y) > 0 \Rightarrow (x, y) \text{ on other side of line} \]

Determine which Triangle “in front”  

1. Mini-max test

- if all \( z_i \)'s of \( i \)'s vertices < all \( z_j \)'s of \( j \)'s, then \( i \) is in front of \( j \)

\[ z_{\text{min}} < z_{\text{max}} \]

or

\[ z_{\text{min}} > z_{\text{max}} \]
2. If test 1 inconclusive, find a point with same x,y on overlapping triangle
projections & compute z for each
- Triangle with smaller z is in front

```
2.5
2

(x,y,z)

(x,y,z)

z^2

Here z_i < z_j, so
Triangle i is in front of j

x,y plane
```

Algorithm won’t work for some cases
- Inter-penetrating triangles
- Cyclic overlapping triangles
- So decompose triangles into smaller ones

```
Two triangles
interpenetrating each other.

Three cyclic
overlapping triangles.
```

Performance of Depth Sort
- Depends on number of polygons
- More polygons means slower
- And it’s worse than linear

```
Performance of Depth Sort
```

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

```
Shadows
```

Sharp Shadows from Point Sources

```
Sharp Shadows from Point Sources
```

Soft Shadows from Extended Sources

```
Soft Shadows from Extended Sources
```

```
```
```
```
```
Shadows from Point Sources

- Look at shadows from point sources
- If a point is in shadow, set Phong $lp$ 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

1. Take 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]=infinity 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