Lecture Overview:

- Administrative
- Motivation
- Z-buffering (Depth-buffer shading)
- Scan-line approaches
- Relative performance comparison of shading approaches
- Basic lighting concepts
- Shading shortcuts
- Relations to OpenGL
MAE 473-573: Lecture #28 - Shading, Lighting

• Administrative

  • Project 1 demos - November 1 - November 7!

  • Dr. Chugh OpenGL/Glut compilation (handout)

  • Attendance sheet policy

  • After class today - can’t meet!
• Motivation - again!

“The Quest for Visual Realism”

We’re transitioning from 2D to 3D graphics. Once the 3D object is projected onto the 2D display plane, we must extract information to enhance the visual quality of the screen image.

Transition from wireframe to “filled” solid (hidden lines removed)

Eventually: Shaded (colored) images, texturing, etc.
Z-buffering (depth buffer shading) - Catmull, 1974

(DEF: Scan conversion: on raster displays, primitives (lines/polygons) are specified by their end vertices. The frame buffer scan converts these polygons into their component (discrete) pixels.)

• Simple to implement, either in software or in hardware
• Uses a frame buffer (F) (color), and a z-buffer (Z) (depth)
• Z - initialized to zero (back clipping plane)
• F - initialized to background color

• Largest Z? (common sense…..) - Z value of the FRONT c.p.
- Z-buffering (depth buffer shading) (cont.)

- Polygons - scan converted to F in an arbitrary order

- Logic: during scan conversion, if the polygon point \((x, y)\) is no farther from the viewer than is the point whose F and Z are currently in the buffers, then the new points F and Z replace the old values!
Figure 13.9 The z-buffer. A pixel’s shade is shown by its color, its z value is shown as a number. (a) Adding a polygon of constant z to the empty z-buffer. (b) Adding another polygon that intersects the first.
• Z-buffering (depth buffer shading) (cont.)

Pseudo-code segment:

```c
Program 13.1

void zBuffer()
{
    int pz;
    /* Polygon's z at pixel coords (x, y) */
    for (y = 0; y < YMAX; y++) {
        for (x = 0; x < XMAX; x++) {
            WritePixel (x, y, BACKGROUND_VALUE);
            WriteZ (x, y, 0);
        }
    }
    for (each polygon) {
        for (each pixel in polygon's projection) {
            pz = polygon's z-value at pixel coords (x, y);
            if (pz >= ReadZ (x, y)) { /* New point is not farther */
                WriteZ (x, y, pz);
                WritePixel (x, y, polygon's color at pixel coords (x, y));
            }
        }
    }
}
```
• Basic lighting concepts (mathematics)

• Illumination models - expressed by an illumination equation expressed in variables associated with the point on the object being shaded...

• Lets start with the basic illumination model, and gradually add numerous factors to the math model...

A. Ambient light: \[ I = I_a k_a \]

where: **I** - resulting pixel intensity
\( I_a \) - intensity of the ambient light
\( k_a \) - ambient-reflection coefficient, \([0,1]\)
(a material property)
MAE 473-573: Lecture #28 - Shading, Lighting

• Basic lighting concepts (cont.)

B. Diffuse reflection (dull surfaces)

\[ I = I_p k_d \cos \theta = I_p k_d (\mathbf{N} \cdot \mathbf{L}) \]

where:

- \( I_p \) - intensity of the point light source
- \( k_d \) - diffuse-reflection coefficient, [0,1]
  (a material property)
- \( \theta \) - angle between light source and surface normal
  (must be between 0 and 90 for light source to have direct effect)
C. Specular reflection (shiny surfaces)

Here,

- \( L \) is the light source,
- \( N \) is the surface normal,
- \( R \) is the direction of the principal reflected ray,
- \( V \) is the viewing vector
• Basic lighting concepts (cont.)

\[ I = W(?) \cos^n a \ I_p \]  \hspace{1cm} (The *Phong* illumination model)

where:

- \( I_p \) - intensity of the point light source
- \( W(?) \) - fraction of specularly reflected light - often set to \( k_s \), the specular-reflection coefficient, \([0,1]\)
- \( a \) - angle between view vector and reflected ray. Maximum specular reflectance occurs when \( a = 0 \); falls sharply as \( a \) increases
• Basic lighting concepts (cont.)

n - specular-reflection exponent, [0, 100+] (material property)

\( ? \) - angle between light source and surface normal (must be between 0 and 90 for light source to have any direct effect)

*Figure 14.9* Different values of \( \cos^n \alpha \) used in the Phong illumination model.
• Shading shortcuts (remember: performance vs. accuracy…)

• **Constant shading** - perform the shading calculation for a full polygon based on a single representative point (i.e. the center), OR: perform shading calculations at vertices; shade based on the average

• **Gouraud shading** - computes pixel brightness at each vertex; interpolates between vertices to get a brightness for each pixel
Shading shortcuts (cont.)

Phong shading - detailed surface normal vector information is obtained at each vertex - this info is then interpolated to the pixel level - along the polygon edges or the scan lines.

Subsequent brightness calculations of each pixel are performed using these interpolated vectors.
Scan-line algorithms:

- Wylie, Romney, Evans, Erdahl (1967)
- Bouknight (1970)
- Watkins (1970)

Image is generated by considering one horizontal line at a time, and plotting pixels along the line as appropriate.

Makes use of:

- edge coherence - visibility of an edge only changes as it interacts with another edge (like Appel’s!)
- scan-line coherence - edges visible on one line are likely visible on the next line
• Scan-line algorithms (cont.):

First, create an **edge** table (ET), where each entry has:

1. A polygon identifier for the edge
2. $Y_{\text{min}}$, and its corresponding $x$
3. $Y_{\text{max}}$, and its corresponding $x$
4. An increment $\Delta x$, which defines the distance between scan lines

Also, define a **polygon** table (PT), where each entry has:

1. An ID number
2. Coefficients for the polygonal plane equation
3. Shading/color info for the polygon
4. An “in-out” Boolean flag, initialized to “out”
Scan-line algorithms (cont.):

Active edge table (AET) - lists edges which intersect a given scan line, ordered in accordance with their “x” value.

Figure 13.11  Two polygons being processed by a scan-line algorithm.
Scan-line algorithms (cont.):

Here, our AET would appear as follows:

<table>
<thead>
<tr>
<th>Scan line</th>
<th>Entries</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>AB, AC</td>
</tr>
<tr>
<td>β</td>
<td>AC, AC, FD, FE</td>
</tr>
<tr>
<td>?</td>
<td>AB, DE, CB, FE</td>
</tr>
<tr>
<td>?+1</td>
<td>AB, DE, CB, FE</td>
</tr>
<tr>
<td>?+2</td>
<td>AB, CB, DE, FE</td>
</tr>
</tbody>
</table>

Logic:
Ex: a - hits AB, flag “on” (inside 1 polygon); hits AC, flag “off”
Ex: ? - hits AB, flag “on” (inside 1 polygon); hits DE, flag “on”, z-buffer to determine which is in front; hits BC, flag “off”, hits EF, flag “off”
Relative performance comparison of shading approaches 
(Sutherland, Sproull, Schumacker, 1974)

Argue that *sorting* is the key to an efficient visible-surface algorithm

- Painter’s - sorts on z, and then on x,y
- Scan-line - sort on y, then x, then z
- Warnock - parallel sort on x,y, then z
- Z-buffer - searches only in z
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>100</th>
<th>2500</th>
<th>60,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depth sort</td>
<td>1 *</td>
<td>10</td>
<td>507</td>
</tr>
<tr>
<td>z-buffer</td>
<td>54</td>
<td>54</td>
<td>54</td>
</tr>
<tr>
<td>Scan line</td>
<td>5</td>
<td>21</td>
<td>100</td>
</tr>
<tr>
<td>Warnock area subdivision</td>
<td>11</td>
<td>64</td>
<td>307</td>
</tr>
</tbody>
</table>

*Entries are normalized such that this case is unity.*
• Relations to OpenGL:

Shading - end of Chapter 4 ("colors") in OpenGL text

Use: `glShadeModel(GLenum mode)`

mode - single color or "flat" shading - GL_FLAT
multiple color or "smooth" shading - GL_SMOOTH

note: latter is in fact Gouraud shading…
note: as always, performance trade-off….

Lighting - Chapter 5 in OpenGL text (Pages 211-218)
“Color produced by lighting a vertex is computed as follows:"

- material emission at vertex +
- the global ambient light (scaled) +
- the ambient, diffuse, and specular contributions for the light sources

………The basic theory of which we just discussed!

• See text for implementation details