Implementation III
Objectives

• Survey Line Drawing Algorithms
  - DDA
  - Bresenham
Rasterization

• Rasterization (scan conversion)
  - Determine which pixels that are inside primitive specified by a set of vertices
  - Produces a set of fragments
  - Fragments have a location (pixel location) and other attributes such color and texture coordinates that are determined by interpolating values at vertices

• Pixel colors determined later using color, texture, and other vertex properties
Scan Conversion of Line Segments

- Start with line segment in window coordinates with integer values for endpoints
- Assume implementation has a `write_pixel` function

\[ m = \frac{\Delta y}{\Delta x} \]

\[ y = mx + h \]
**DDA Algorithm**

- **Digital Differential Analyzer**
  - DDA was a mechanical device for numerical solution of differential equations
  - Line $y = mx + h$ satisfies differential equation
    
    \[
    \frac{dy}{dx} = m = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1}
    \]

- Along scan line $\Delta x = 1$

  ```
  For(x=x1; x<=x2, ix++) {
      y+=m;
      write_pixel(x, round(y), line_color)
  }
  ```
Problem

• DDA = for each x plot pixel at closest y
  - Problems for steep lines
Using Symmetry

• Use for $1 \geq m \geq 0$

• For $m > 1$, swap role of $x$ and $y$
  - For each $y$, plot closest $x$
Bresenham’s Algorithm

• DDA requires one floating point addition per step
• We can eliminate all fp through Bresenham’s algorithm
• Consider only $1 \geq m \geq 0$
  - Other cases by symmetry
• Assume pixel centers are at half integers
• If we start at a pixel that has been written, there are only two candidates for the next pixel to be written into the frame buffer
Candidate Pixels

$1 \geq m \geq 0$

Note that line could have passed through any part of this pixel
Decision Variable

\[ d = \Delta x(b-a) \]

- \( d \) is an integer
- \( d > 0 \) use upper pixel
- \( d < 0 \) use lower pixel
Incremental Form

• More efficient if we look at $d_k$, the value of the decision variable at $x = k$

$$d_{k+1} = d_k + 2\Delta y, \quad \text{if } d_k < 0$$

$$d_{k+1} = d_k + 2(\Delta y - \Delta x), \quad \text{otherwise}$$

• For each $x$, we need do only an integer addition and a test
• Single instruction on graphics chips
Example

• Consider line from (20,10) to (30,18)
  \[ \Delta x = 10, \]
  \[ \Delta y = 8, \]
  \[ 2\Delta y = 16 \]
  \[ 2(\Delta y - \Delta x) = -4 \]
  \[ d = 2\Delta y - \Delta x = 6 \]
Polygon Scan Conversion

• Scan Conversion = Fill
• How to tell inside from outside
  - Convex easy
  - Nonsimple difficult
  - Odd even test
    • Count edge crossings
    - Winding number

odd-even fill
Winding Number

• Count clockwise encirclements of point

winding number = 1

winding number = 2

• Alternate definition of inside: inside if winding number \neq 0
Filling in the Frame Buffer

• Fill at end of pipeline
  - Convex Polygons only
  - Nonconvex polygons assumed to have been tessellated
  - Shades (colors) have been computed for vertices (Gouraud shading)
  - Combine with z-buffer algorithm
    • March across scan lines interpolating shades
    • Incremental work small
Using Interpolation

$C_1, C_2, C_3$ specified by `glColor` or by vertex shading
$C_4$ determined by interpolating between $C_1$ and $C_2$
$C_5$ determined by interpolating between $C_2$ and $C_3$
interpolate between $C_4$ and $C_5$ along span

Angel: Interactive Computer Graphics 5E © Addison-Wesley 2009
Flood Fill

• Fill can be done recursively if we know a seed point located inside (WHITE)
• Scan convert edges into buffer in edge/inside color (BLACK)

```c
flood_fill(int x, int y) {
    if(read_pixel(x,y)==WHITE) {
        write_pixel(x,y,BLACK);
        flood_fill(x-1, y);
        flood_fill(x+1, y);
        flood_fill(x, y+1);
        flood_fill(x, y-1);
    }
}
```
Scan Line Fill

- Can also fill by maintaining a data structure of all intersections of polygons with scan lines
  - Sort by scan line
  - Fill each span

vertex order generated by vertex list
desired order
Data Structure

Scanlines

Intersections

\[ j \rightarrow x_1 \rightarrow x_2 \]
\[ j + 1 \rightarrow x_3 \rightarrow x_4 \]
\[ j + 2 \rightarrow x_4 \rightarrow x_5 \rightarrow x_7 \rightarrow x_8 \]
Aliasing

- Ideal rasterized line should be 1 pixel wide

- Choosing best y for each x (or visa versa) produces aliased raster lines
Antialiasing by Area
Averaging

- Color multiple pixels for each x depending on coverage by ideal line

original

antialiased

magnified

Angel: Interactive Computer Graphics 5E © Addison-Wesley 2009
Polygon Aliasing

• Aliasing problems can be serious for polygons
  - Jaggedness of edges
  - Small polygons neglected
  - Need compositing so color of one polygon does not totally determine color of pixel

All three polygons should contribute to color