Chapter 8

Scan Conversion - Drawing on Raster Display (part 1 - Lines)

Scan Conversion - Lines

Rendering Pipeline

- Discard geometry outside viewport window

Scan Conversion - Rasterization

- Convert continuous rendering primitives into discrete fragments/pixels:
  - Lines
  - Bresenham (Midpoint)
  - Triangles
  - Flood fill
  - Implicit formulation
  - Scanline
  - Interpolation

Scan Conversion - Lines

Idea: Use Explicit Line Formula

- Explicit - one coordinate as function of the others:
  - $y = f(x)$
  - $z = f(x, y)$

- Line:
  - $y = mx + b$
  - $y = (y_2 - y_1)(x - x_1) + y_1$

- Typically separate into 8 cases (why?)
Computer Graphics

Basic Line Drawing

Assume \(a \leq x \leq b\) & line slope absolute value is \(\leq 1\).

Line \((x_1, y_1), (x_2, y_2)\):

Begin:

\[
\begin{align*}
& a \leftarrow x_1, b \leftarrow x_2, \text{ slope } \leftarrow \frac{y_2 - y_1}{x_2 - x_1}, \\
& \text{ if } x_1 \lt x_2 \text{ then swap } a, b, \text{ slope, } x_1, x_2, y_1, y_2, \text{ end if,} \\
& \text{ for } x 	ext{ from } a \text{ to } b \\
& \text{ PlotPixel } (x + 1, y_1 + \text{ slope }) \text{ end for,} \\
& \text{ end.}
\end{align*}
\]

Midpoint (Bresenham) Algorithm

**Key Observation 1:**
- At each step have ONLY 2 choices
  - East/North-East

**Key Observation 2:**
- Can decide based on whether midpoint is above/below line
  - How?
    - Evaluate implicit line equation at \((x+1, y+1/2)\)

Bresenham Algorithm

**Implicit formulation = distance (up to scale)**

\[
\tau = \{(x, y) \mid ax + by + c = axy - ydx + c = 0 \}
\]

\[
d(x, y) = 2(ax - yd) + c
\]

- Given point \(P = (x, y)\) \(d(x, y)\) is signed distance of \(P\)
  to \(\tau\) (up to scale)
- \(d\) is zero for \(P \in \tau\)

Midpoint (Bresenham) Algorithm

1. Starting point satisfies \(d(x_1, y_1) = 0\)
2. Each step moves right (east) or upper right (northeast)
3. Sign of \(d(x + 1; y + \frac{1}{2})\) indicates if to move east or northeast

Bresenham (Midpoint) Algorithm

- Evaluate implicit line equation at \((x+1, y+1/2)\)
- \(d\) is zero for \(P \in \tau\)
Bresenham (Midpoint) Algorithm

- Insanely efficient version (less computations inside the loop)
  - compute d incrementally
- At $(x_i, y_i)$
  \[ d_{ext} = d_{ext} + 1; \]
- Increment in d (after each step)
  - If move east
    \[ d_{ext} = d_{ext} + 1 \]
  - If move northeast
    \[ d_{ext} = d_{ext} + 1 \]

**Intensity depends on angle**

Comment: extends to higher order curves - e.g. circles

Scan Conversion of Lines

Discussion

- Integer: Bresenham
  - Good for hardware implementations (integer!)
- Floating Point
  - May be faster for software (depends on system!)
  - May be faster for software (depends on system!)
  - Floating point ops higher parallelized (pipelined)
    - E.g. RISC CPUs from MIPS, SUN
    - No 'if' statements in inner loop
    - More efficient use of processor pipelining

Comparison: float/integer

Assume $x_i < x_f$ & line slope is $< 1$

Scan Conversion of Lines

Discussion

- Integer: Bresenham
  - Good for hardware implementations (integer!)
- Floating Point
  - May be faster for software (depends on system!)
  - Floating point ops higher parallelized (pipelined)
    - E.g. RISC CPUs from MIPS, SUN
    - No 'if' statements in inner loop
    - More efficient use of processor pipelining