Chapter 8

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

Idea: Use Explicit Line Formula

Explicit - one coordinate as function of the others

\[ y = f(x) \]
\[ z = f(x, y) \]

Line

\[ y = mx + b \]
\[ y = \frac{(y_2 - y_1)}{(x_2 - x_1)}(x - x_1) + y_1 \]

Typically separate into 8 cases (why?)
Basic Line Drawing

Assume $x_1 < x_2$ & line slope absolute value is $\leq 1$

\[
\text{Line} \ (x_1, y_1, x_2, y_2) \\
\text{begin} \\
\text{float} \ dx, dy, x, y, \text{slope} \ ; \\
dx \leftarrow x_2 - x_1; \\
dy \leftarrow y_2 - y_1; \\
slope \leftarrow \frac{dy}{dx}; \\
y \leftarrow y_1; \\
\text{for} \ x \ \text{from} \ x_1 \ \text{to} \ x_2 \ \text{do} \\
\text{begin} \\
\quad \text{PlotPixel} \ (x, \text{Round} \ (y)); \\
\quad y \leftarrow y + \text{slope}; \\
\text{end} \ ; \\
\text{end} ;
\]

Questions:
Can this algorithm use integer arithmetic?

Midpoint (Bresenham) Algorithm

- **Key Observation 1:**
  - At each step have ONLY 2 choices
    - East/North-East
Midpoint (Bresenham) Algorithm

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)
\[
\tau = \{ (x, y) | ax + by + c = xdy - ydx + c = 0 \} \\
(d(x, y) = 2(xdy - ydx + c)
\]

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

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

\[
\begin{align*}
\text{Line} & \quad (x_1, y_1, x_2, y_2) \\
\text{begin} & \\
\text{int} & \quad x, y, dx, dy, d \\
& x \leftarrow x_1 \\
& y \leftarrow y_1 \\
& dx \leftarrow x_2 - x_1 \\
& dy \leftarrow y_2 - y_1 \\
\text{PlotPixel} & \quad (x, y) \\
\text{while} & \quad (x < x_2) \text{ do} \\
& \quad d = (2x + 2)dy - (2y + 1)dx + 2c; // 2((x + 1)dy - (y + .5)dx + c) \\
& \quad \text{if} \quad (d < 0) \text{ then} \\
& \quad \text{begin} \\
& \quad \quad x \leftarrow x + 1 \\
& \quad \quad \text{end} \\
& \quad \text{else} \text{ begin} \\
& \quad \quad x \leftarrow x + 1 \\
& \quad \quad y \leftarrow y + 1 \\
& \quad \quad \text{end} \\
& \quad \text{PlotPixel} \quad (x, y) \\
& \text{end} \\
\end{align*}
\]
Bresenham (Midpoint) Algorithm

- Insanely efficient version (less computations inside the loop)
  - compute d incrementally
- At \((x_i, y_i)\)
  \[d_{\text{start}} = d(x_i + 1, y_i + \frac{1}{2}) = 2dy - dx\]

Increment in \(d\) (after each step)
- If move east
  \[\Delta_e = d(x + 2, y + \frac{1}{2}) - d(x + 1, y + \frac{1}{2}) = 2((x + 2)dy - (y + \frac{1}{2})(dx + c)) - 2((x + 1)dy - (y + \frac{1}{2})(dx + c)) = 2dy\]
- If move northeast
  \[\Delta_{ne} = d(x + 1, y + \frac{3}{2}) - d(x + 1, y + \frac{1}{2}) = 2((x + 2)dy - (y + \frac{3}{2})(dx + c)) - 2((x + 1)dy - (y + \frac{1}{2})(dx + c)) = 2(dy - dx)\]
Bresenham Examples

- Intensity depends on angle

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

Comparison: float/integer

Assume $x_1 < x_2$ & line slope is $\leq 1$

**Line** $(x_1, y_1, x_2, y_2)$

```
begin
  float dx, dy, x, y, slope;
  dx := x_2 - x_1;
  dy := y_2 - y_1;
  slope := dy / dx;
  y := y_1;
  for $x$ from $x_1$ to $x_2$ do
    begin
      PlotPixel ($x$, Round ($y$));
      y := y + slope;
    end;
end;
```

**Line** $(x_1, y_1, x_2, y_2)$

```
begin
  int x, y, dx, dy, d, $\Delta_x$, $\Delta_y$;
  x := x_1;
  y := y_1;
  dx := x_2 - x_1;
  dy := y_2 - y_1;
  d := 2 * dy - dx;
  $\Delta_x$ := 2 * dy;
  $\Delta_y$ := 2 * (dy - dx);
  PlotPixel ($x$, $y$);
  while ($x < x_2$) do
    if ($d < 0$) then
      begin
        d := d + $\Delta_x$;
      end;
    else
      begin
        d := d + $\Delta_y$;
        y := y + 1;
      end;
    x := x + 1;
    PlotPixel ($x$, $y$);
end;
```
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