Scan Converting Lines

Line Drawing
- Draw a line on a raster screen between two points
- What's wrong with statement of problem?
  - doesn't say anything about which points are allowed as endpoints
  - doesn't give a clear meaning of “draw”
  - doesn't say what constitutes a “line” in raster world
  - doesn't say how to measure success of proposed algorithms

Problem Statement
- Given two points \( P \) and \( Q \) in XY plane, both with integer coordinates, determine which pixels on raster screen should be on in order to make picture of a unit-width line segment starting at \( P \) and ending at \( Q \)

Finding next pixel:

Special case:
- Horizontal Line:
  Draw pixel \( P \) and increment \( x \) coordinate value by 1 to get next pixel.
- Vertical Line:
  Draw pixel \( P \) and increment \( y \) coordinate value by 1 to get next pixel.
- Diagonal Line:
  Draw pixel \( P \) and increment both \( x \) and \( y \) coordinate by 1 to get next pixel.
- What should we do in general case?
  - Increment \( x \) coordinate by 1 and choose point closest to line.
  - But how do we measure “closest”?

Vertical Distance
- Why can we use vertical distance as measure of which point is closer?
  - because vertical distance is proportional to actual distance
  - how do we show this?
  - with similar triangles
- By similar triangles we can see that true distances to line (in blue) are directly proportional to vertical distances to line (in black) for each point.
- Therefore, point with smaller vertical distance to line is closest to line
Strategy 1 - Incremental Algorithm (1/2)

Basic Algorithm
- Find equation of line that connects two points P and Q
- Starting with leftmost point P, increment x_i by 1 to calculate y_i = m*x_i + B
  where m = slope, B = y intercept
- Draw pixel at (x_i, Round(y_i)) where Round(y_i) = Floor(0.5 + y_i)

Incremental Algorithm:
- Each iteration requires a floating-point multiplication
  - Modify algorithm to use deltas
    - (y_{i+1} - y_i) = m*(x_{i+1} - x_i) + B - B
    - y_{i+1} = y_i + m*(x_{i+1} - x_i)
  - If Δx = 1, then y_{i+1} = y_i + m
- At each step, we make incremental calculations based on preceding step to find next y value

Strategy 1 - Incremental Algorithm (2/2)

Example Code

```c
// Incremental Line Algorithm
// Assume x0 < x1
void Line(int x0, int y0, int x1, int y1) {
    int x, y;
    float dy = y1 - y0;
    float dx = x1 - x0;
    float m = dy / dx;
    y = y0;
    for (x = x0; x < x1; x++) {
        WritePixel(x, Round(y));
        y = y + m;
    }
}
```

Problem with Incremental Algorithm:

```c
void Line(int x0, int y0, int x1, int y1) {
    int x, y;
    float dy = y1 - y0;
    float dx = x1 - x0;
    float m = dy / dx;
    y = y0;
    for (x = x0; x < x1; x++) {
        WritePixel(x, Round(y));
        y = y + m;
    }
}
```

Since slope is fractional, need special case for vertical lines
Strategy 2 – Midpoint Line Algorithm (1/3)

- Assume that line’s slope is shallow and positive (0 < slope < 1); other slopes can be handled by suitable reflections about principle axes.
- Call lower left endpoint \((x_0, y_0)\) and upper right endpoint \((x_1, y_1)\).
- Assume that we have just selected pixel \(P\) at \((x_p, y_p)\).
- Next, we must choose between pixel to right (E pixel), or one right and one up (NE pixel).
- Let \(Q\) be intersection point of line being scan-converted and vertical line \(x = x_p + 1\).

Strategy 2 – Midpoint Line Algorithm (2/3)

- Previous pixel
- Choices for current pixel
- Choices for next pixel
- \(P = (x_p, y_p)\)
- \(Q\)
- \(M\)
- \(E\) pixel
- \(NE\) pixel

Strategy 2 – Midpoint Line Algorithm (3/3)

- Line passes between E and NE.
- Point that is closer to intersection point \(Q\) must be chosen.
- Observe on which side of line midpoint \(M\) lies:
  - E is closer to line if midpoint \(M\) lies above line, i.e., line crosses bottom half.
  - NE is closer to line if midpoint \(M\) lies below line, i.e., line crosses top half.
- Error (vertical distance between chosen pixel and actual line) is always \(\leq \frac{1}{2}\).
- Algorithm chooses NE as next pixel for line shown.
- Now, need to find a way to calculate on which side of line midpoint lies.

Line equation as function \(f(x)\):
\[
y = mx + B
\]
\[
y = \frac{dy}{dx} x + B
\]

Line equation as implicit function:
\[
f(x, y) = ax + by + c = 0
\]
for coefficients \(a, b, c\), where \(a, b \neq 0\).

From above,
\[
y \cdot dx = dy \cdot x + B \cdot dx
\]
\[
dy \cdot x - y \cdot dx + B \cdot dx = 0
\]
\[
\therefore a = dy, b = -dx, c = B \cdot dx
\]

Properties (proof by case analysis):
- \(f(x_p, y_p) = 0\) when any point \(M\) is on line.
- \(f(x_p, y_p) < 0\) when any point \(M\) is above line.
- \(f(x_p, y_p) > 0\) when any point \(M\) is below line.
- Our decision will be based on value of function at midpoint \(M\) at \((x_p + 1, y_p + \frac{1}{2})\).
Decision Variable

- We only need sign of \( f(x_p + 1, y_p + \frac{1}{2}) \) to see where line lies, and then pick nearest pixel
- \( d = f(x_p + 1, y_p + \frac{1}{2}) \)
  - if \( d > 0 \) choose pixel NE
  - if \( d < 0 \) choose pixel E
  - if \( d = 0 \) choose either one consistently

How do we incrementally update \( d \)?

- On basis of picking E or NE, figure out location of \( M \) for that pixel, and corresponding value \( d \) for next grid line
- We can derive \( d \) for the next pixel based on our current decision

If E was chosen:

Increment \( M \) by one in \( x \) direction

\[
d_{\text{new}} = f(x_p + 2, y_p + \frac{3}{2}) = a(x_p + 2) + b(y_p + 3/2) + c
\]

- \( d_{\text{new}} - d_{\text{old}} \) is the incremental difference \( \Delta E \)
  \[
d_{\text{new}} = d_{\text{old}} + a \\
\Delta E = a = dy \quad \text{(2 slides back)}
\]
- We can compute value of decision variable at next step incrementally without computing \( F(M) \) directly
  \[
d_{\text{new}} = d_{\text{old}} + \Delta E = d_{\text{old}} + dy
\]
- \( \Delta E \) can be thought of as correction or update factor to take \( d_{\text{old}} \) to \( d_{\text{new}} \)
- It is referred to as forward difference

If NE was chosen:

Increment \( M \) by one in both \( x \) and \( y \) directions

\[
d_{\text{new}} = F(x_p + 2, y_p + 3/2) = a(x_p + 2) + b(y_p + 3/2) + c
\]

- \( \Delta NE = d_{\text{new}} - d_{\text{old}} \)
  \[
d_{\text{new}} = d_{\text{old}} + a + b \\
\Delta NE = a + b = dy - dx
\]
- Thus, incrementally,
  \[
d_{\text{new}} = d_{\text{old}} + \Delta NE = d_{\text{old}} + dy - dx
\]

Summary

- At each step, algorithm chooses between 2 pixels based on sign of decision variable calculated in previous iteration.
- It then updates decision variable by adding either \( \Delta E \) or \( \Delta NE \) to old value depending on choice of pixel. Simple additions only!
- First pixel is first endpoint \((x_0, y_0)\), so we can directly calculate initial value of \( d \) for choosing between E and NE.
**Summary (2/2)**

- First midpoint for first \( d = d_{\text{start}} \) is at \((x_0 + 1, y_0 + 1/2)\)
- \( f(x_0 + 1, y_0 + 1/2) = a(x_0 + 1) + b(y_0 + 1/2) + c \)
- \( = a*x_0 + b*y_0 + c + a + b/2 \)
- \( = f(x_0, y_0) + a + b/2 \)
- But \((x_0, y_0)\) is point on line and \( f(x_0, y_0) = 0 \)
- Therefore, \( d_{\text{start}} = a + b/2 = dy - dx/2 \)
  - use \( d_{\text{start}} \) to choose second pixel, etc.
- To eliminate fraction in \( d_{\text{start}} \):
  - redefine \( f \) by multiplying it by 2; \( f(x, y) = 2(ax + by + c) \)
  - this multiplies each constant and decision variable by 2, but does not change sign
- Bresenham’s line algorithm is same but doesn’t generalize as nicely to circles and ellipses

**Example Code**

```c
void MidpointLine(int x0, int y0,
                   int x1, int y1) {
  int dx = x1 - x0;
  int dy = y1 - y0;
  int d = 2 * dy - dx;
  int incrE = 2 * dy;
  int incrNE = 2 * (dy - dx);
  int x = x0;
  int y = y0;
  writePixel(x, y);
  while (x < x1) {
    if (d <= 0) {
      // East Case
      d = d + incrE;
    } else {
      // Northeast Case
      d = d + incrNE;
      y++;
    }
    x++;
    writePixel(x, y);
  } /* while */
} /* MidpointLine */
```

Andries van Dam   October 1, 2009   Scan Conversion 1 18/18