Scan Conversion 1

Scan Converting Lines

Line Drawing

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

Problem Statement

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

Finding the next pixel:

Special case:

- Horizontal Line:

  Draw pixel *P* and increment the *x* coordinate value by one to get the next pixel.

- Vertical Line:

  Draw pixel *P* and increment the *y* coordinate value by one to get the next pixel.

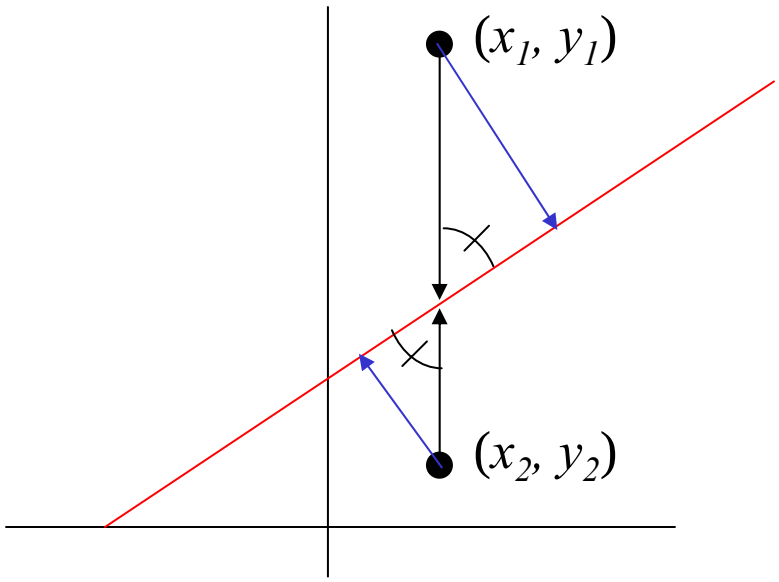
- Diagonal Line:

  Draw pixel *P* and increment both the *x* and the *y* coordinate by one to get the next pixel.

- What should we do in the general case?
  - Increment the x coordinate by 1 and choose the point closest to the line.
  - But how do we measure “closest”?

Vertical Distance

- Why can we use the vertical distance as a measure of which point is closer?
  - because the vertical distance is proportional to the actual distance
  - how do we show this?
  - with congruent triangles



- By similar triangles we can see that the true distances to the line (in blue) are directly proportional to the vertical distances to the line (in black) for each point
- Therefore, the point with the smaller vertical distance to the line is the closest to the line

Strategy 1 - Incremental Algorithm (1/2)

The Basic Algorithm

- Find the equation of the line that connects the two points P and Q
- Starting with the leftmost point P, increment x_i by 1 to calculate $y_i = mx_i + B$

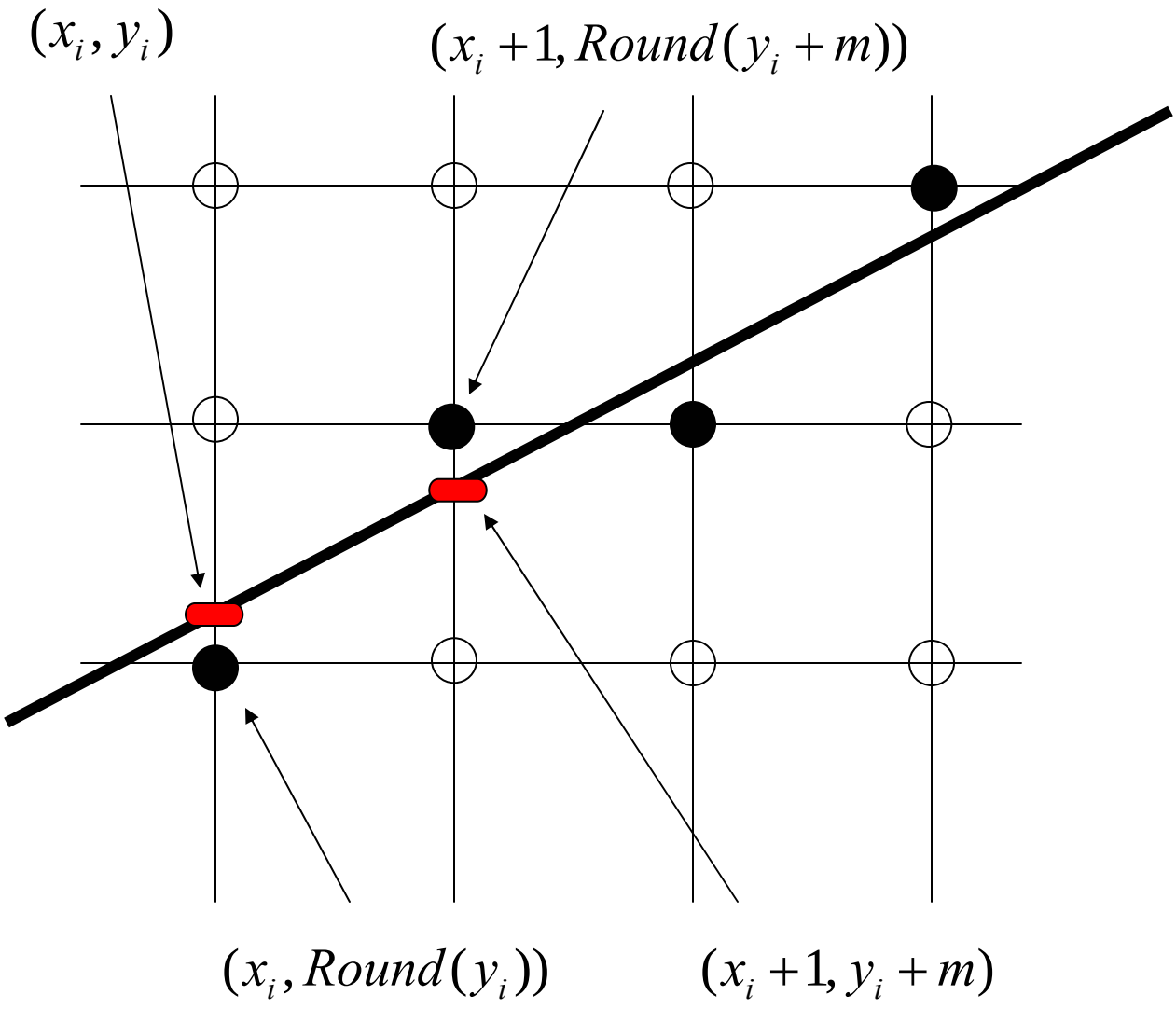
  where m = slope, $B = y$ intercept
- Intensify the pixel at $(x_i, \text{Round}(y_i))$ where

  $\text{Round}(y_i) = \text{Floor}(0.5 + y_i)$

The Incremental Algorithm:

- Each iteration requires a floating-point multiplication
  - therefore, modify the algorithm.
- $y_{i+1} = mx_{i+1} + B = m(x_i + \Delta x) + B = y_i + m\,\Delta x$
- If $\Delta x = 1$, then $y_{i+1} = y_i + m$
- At each step, we make incremental calculations based on the preceding step to find the next y value

Strategy 1 - Incremental Algorithm (2/2)



Example Code

```
// Incremental Line Algorithm
// Assumes –1 <= m <= 1, x0 < x1

void Line(int x0, int y0,
        int x1, int y1, int value) {
   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), value);
        y = y + m;
   }
}
```

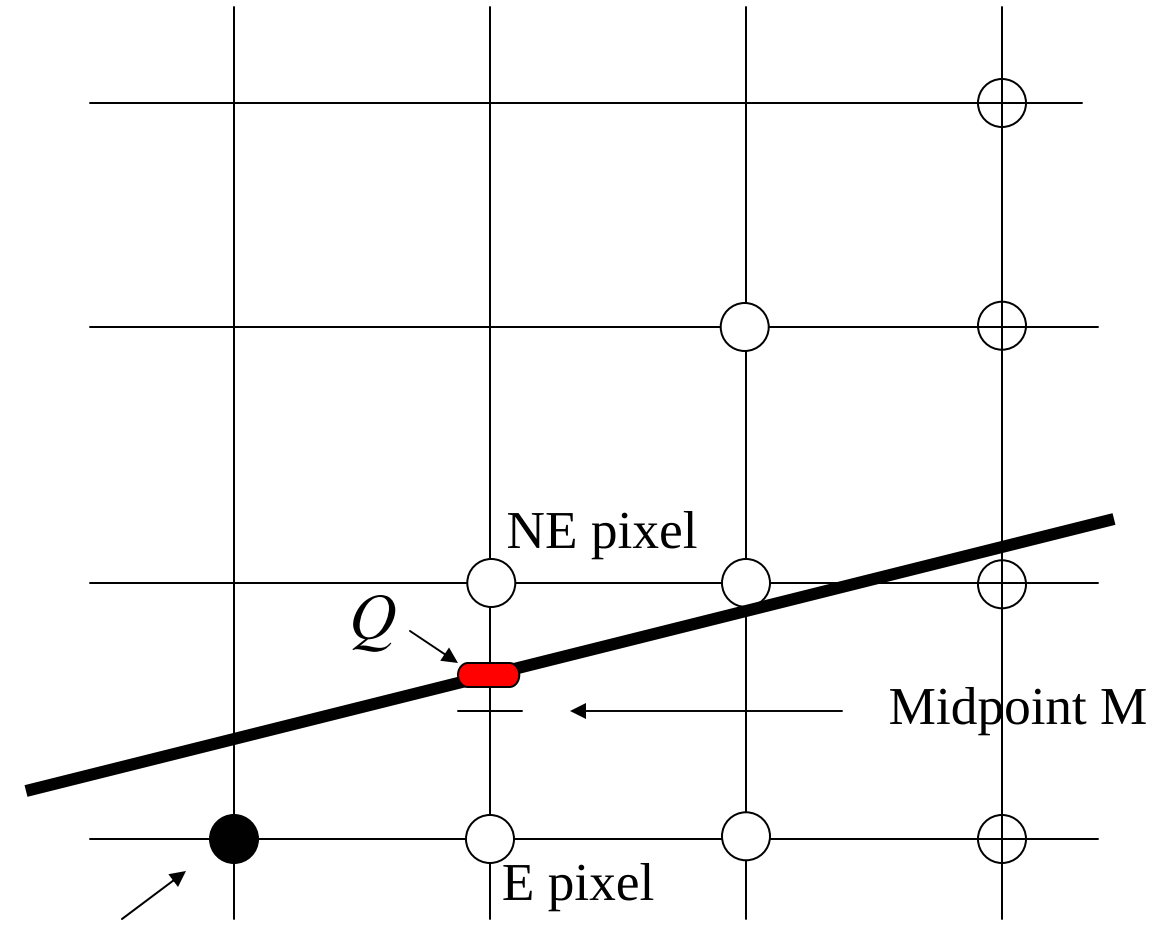
Problem with the Incremental Algorithm:

- Rounding integers takes time
- Variables *y* and *m* must be a real or fractional binary because the slope is a fraction
  - special case needed for vertical lines

Strategy 2 – Midpoint Line Algorithm (1/3)

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

Strategy 2 – Midpoint Line Algorithm (2/3)



Strategy 2 – Midpoint Line Algorithm (3/3)

- The line passes between E and NE
- The point that is closer to the intersection point *Q* must be chosen
- Observe on which side of the line the midpoint *M* lies:
  - E is closer to the line if the midpoint *M* lies above the line, i.e., the line crosses the bottom half
  - NE is closer to the line if the midpoint *M* lies below the line, i.e., the line crosses the top half
- The error, the vertical distance between the chosen pixel and the actual line, is always <= ½
- The algorithm chooses NE as the next pixel for the line shown
- Now, find a way to calculate on which side of the line the midpoint lies

The Line

Line equation as a function *f(x)*:

- $f(x) = m*x + B = dy/dx*x + B$

Line equation as an implicit function:

- $F(x, y) = a*x + b*y + c = 0$ for coefficients a, b, c, where $a, b \neq 0$

  from above, $y*dx = dy*x + B*dx$

  so $a = dy$, $b = -dx$, $c = B*dx$,

  $a > 0$ for $y_0 < y_1$

Properties (proof by case analysis):

- $F(x_m, y_m) = 0$ when any point M is on the line
- $F(x_m, y_m) < 0$ when any point M is above the line
- $F(x_m, y_m) > 0$ when any point M is below the line
- Our decision will be based on the value of the function at the midpoint M at $(x_p + 1, y_p + ½)$

Decision Variable

Decision Variable *d*:

- We only need the sign of $F(x_p + 1, y_p + ½)$ to see where the line lies, and then pick the nearest pixel
- $d = F(x_p + 1, y_p + ½)$

  - if $d > 0$ choose pixel NE
  - if $d < 0$ choose pixel E
  - if $d = 0$ choose either one consistently

How to update *d*:

- On the basis of picking E or NE, figure out the location of *M* for that pixel, and the corresponding value of *d* for the next grid line

If E was chosen:

M is incremented by one step in the *x* direction

$$d_{new} = F(x_p + 2, y_p + ½)$$
$$= a(x_p + 2) + b(y_p + ½) + c$$

$$d_{old} = a(x_p + 1) + b(y_p + ½) + c$$

- Subtract d_{old} from d_{new} to get the incremental difference ΔE

$$d_{new} = d_{old} + a$$
$$\Delta E = a = dy$$

- Derive the value of the decision variable at the next step incrementally without computing *F(M)* directly

$$d_{new} = d_{old} + \Delta E = d_{old} + dy$$

- ΔE can be thought of as the correction or update factor to take d_{old} to d_{new}
- It is referred to as the forward difference

If NE was chosen:

M is incremented by one step each in both the *x* and *y* directions

$$d_{new} = F(x_p + 2, y_p + 3/2)$$
$$= a(x_p + 2) + b(y_p + 3/2) + c$$

- Subtract d_{old} from d_{new} to get the incremental difference

$$d_{new} = d_{old} + a + b$$
$$\Delta NE = a + b = dy - dx$$

- Thus, incrementally,

$$d_{new} = d_{old} + \Delta NE = d_{old} + dy - dx$$

Example Code

```
void MidpointLine(int x0, int y0,
          int x1, int y1, int value) {
    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, value);

    while (x < x1) {
          if (d <= 0) {     //East Case
                    d = d + incrE;
          } else {          // Northeast Case
                    d = d + incrNE;
                    y++;
          }
          x++;
          writePixel(x, y, value);
    }               /* while */
}                   /* MidpointLine */
```