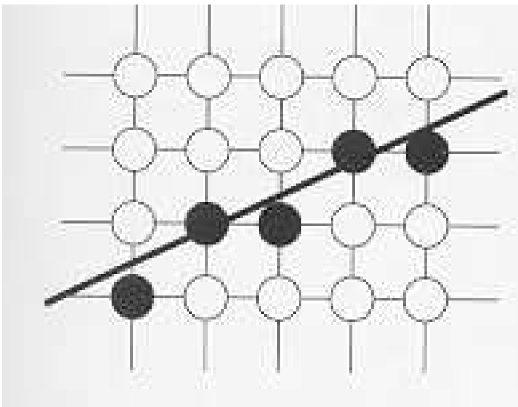
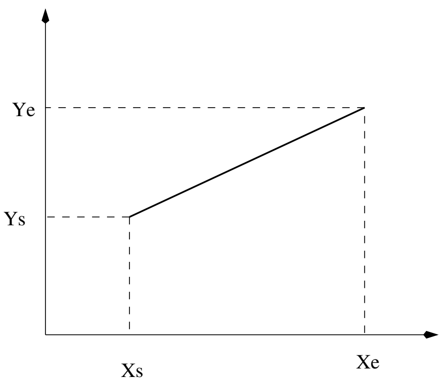
Scan Converting Lines

- Compute the coordinates of the pixels that lie on or near the ideal line



- **Desirable properties**
  - Straight lines should appear as straight lines
  - Lines should pass from the endpoints (if possible)
  - Uniform brightness (independent of slope)
  - Fast line drawing

- **Slope-intercept equation**



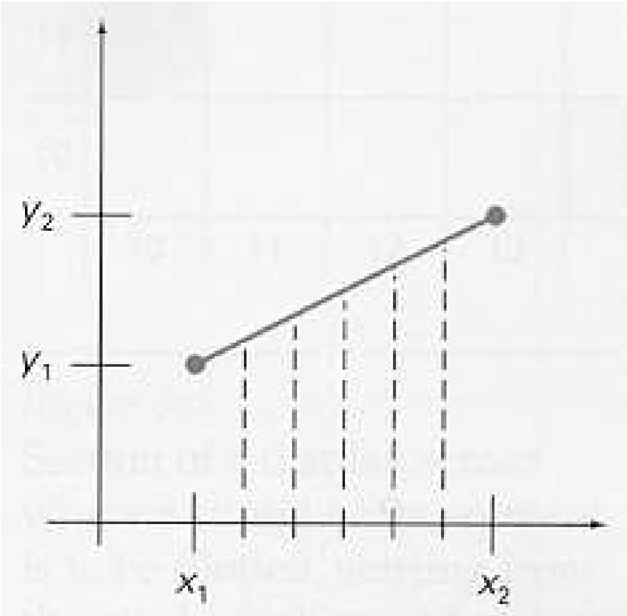
$Y = mX + B$

$m = (Ye-Ys)/(Xe-Xs)$ **slope**

$B = Ys - mXs$ **slope intercept**

- **Basic incremental algorithm**

- Idea: sample the line at unit intervals in one coordinate and determine corresponding integer values nearest the line path of the other coordinate



- Case 1: $|m| \le 1$, start from left endpoint

step1: set $x_1 = x_s$, $\Delta x = 1$

step2: calculate $y_i = mx_i + B$

step3: round y_i: $\hat{y}_i = \text{round}(y_i)$

step4: draw $(x_i, \hat{y}_i)$

step5: $x_i = x_i^{old} + \Delta x$

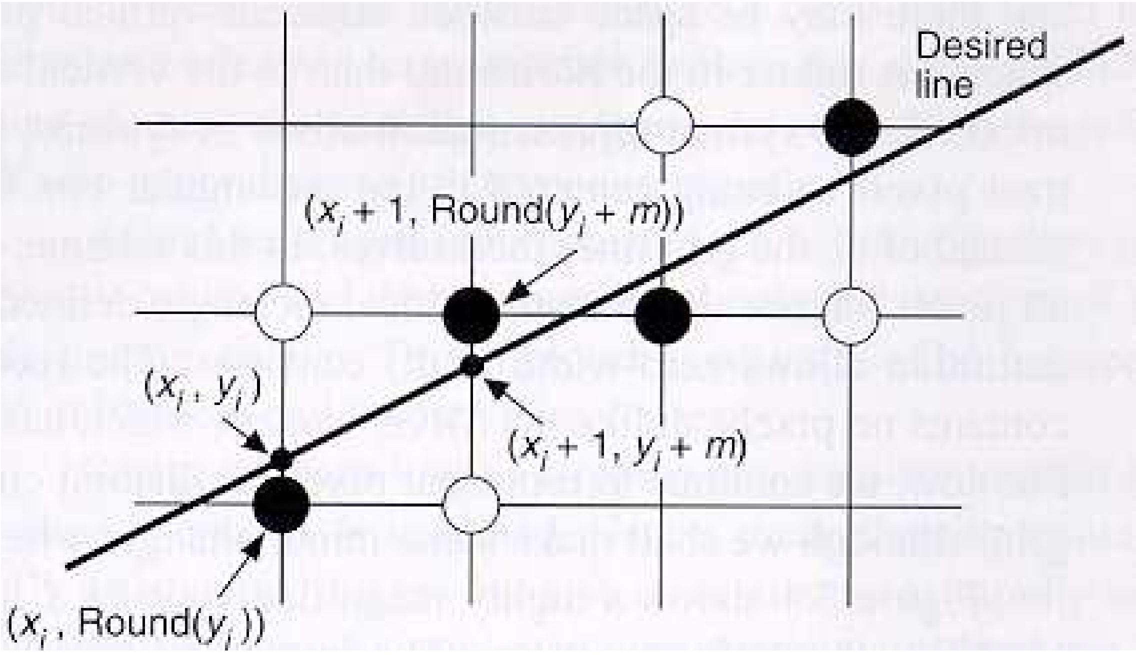
step6: if $x_i \le x_e$, goto step2

- Computational cost: a floating point multiplication, addition, rounding

- Simplification:

$$y_{i+1} = mx_{i+1} + B = m(x_i + \Delta x) + B = mx_i + m\Delta x + B = y_i + m\Delta x$$

$$\text{or } y_{i+1} = y_i + m \; (x_{i+1} = x_i + 1, \Delta x = 1)$$



- Case 2: $|m| > 1$, start from the left endpoint

  * A unit of change in x, changes y by m ($|m| > 1$)

  * Solution: reverse the roles of x and y

$$x_{i+1} = x_i + 1/m, \ (y_{i+1} = y_i + 1, \Delta y{=}1)$$

- Case 3: $|m| \le 1$, start from right endpoint

$$y_{i+1} = y_i - m \ (x_{i+1} = x_i - 1, \Delta x{=} -1)$$

- Case 4: $|m| > 1$, start from right endpoint

$$x_{i+1} = x_i - 1/m, \ (y_{i+1} = y_i - 1, \Delta y{=} -1)$$

- Drawbacks of the simple incremental algorithm

  * Floating point addition

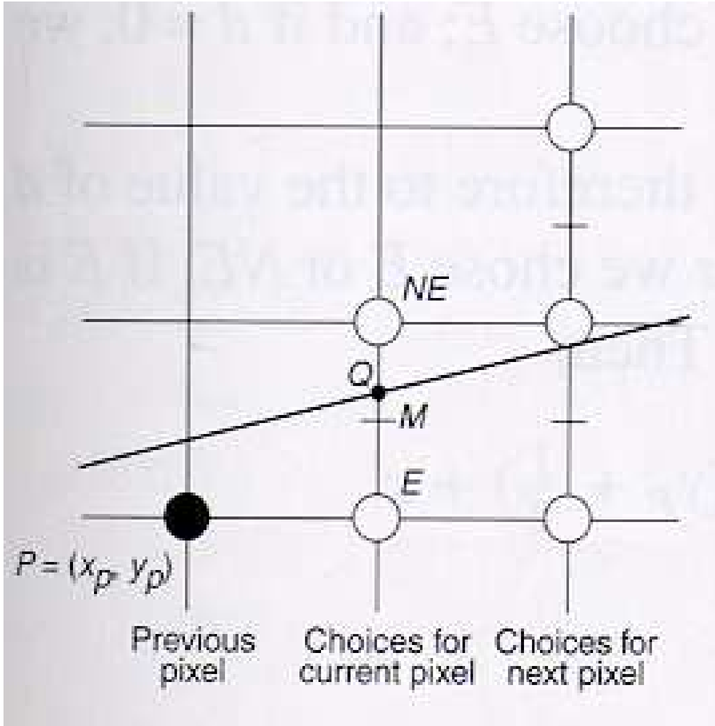
  * Round operation

  * Accumulation of roundoff errors

- **Midpoint algorithm**

  - It is a generalization of Bresenham's algorithm

  - Idea (assume $0 \leq m \leq 1$):



  * Assume that P has been selected and we must choose between E and NE

  * Define Q as the intersection of the line with $x = x_p + 1$

  * If Q lies above M, then pixel NE is closer to the line

  * If Q lies below M, then pixel E is closer to the line

  - Objective: calculate on which side of the line M lies !!

Background on line representations

* Implicit representation of lines

$$F(x, y) = ax + by + c = 0$$

* Slope-intercept representation of lines

$$y = mx + B = \frac{y_1 - y_0}{x_1 - x_0} x + B = \frac{dy}{dx} x + B$$

* How are the above representations related ?

$$xdy - ydx + Bdx = 0 \; (a = dy, \; b = -dx, \; c = Bdx)$$

* Sign of $F(x, y)$

$F(x, y) = 0$, for points on the line
$F(x, y) > 0$, for points below the line
$F(x, y) < 0$, for points above the line

- Check the sign of $d = F(x_p + 1, y_p + 1/2)$ to find on which side M lies

if $d > 0$, choose pixel NE

if $d < 0$, choose pixel E

if $d = 0$, pick either E or NE (we pick E)

- **How d should be computed ? Incrementally !!**

$$d = a(x_p + 1) + b(y_p + 1/2) + c$$

* If E is chosen, then we have:

$d_{new} = F(M_1) = F(x_p + 2, y_p + 1/2) = a(x_p + 2) + b(y_p + 1/2) + c$

Also, $d_{old} = a(x_p + 1) + b(y_p + 1/2) + c$

Take the difference: $\Delta_E = d_{new} - d_{old} = a$ or $d_{new} = d_{old} + a$

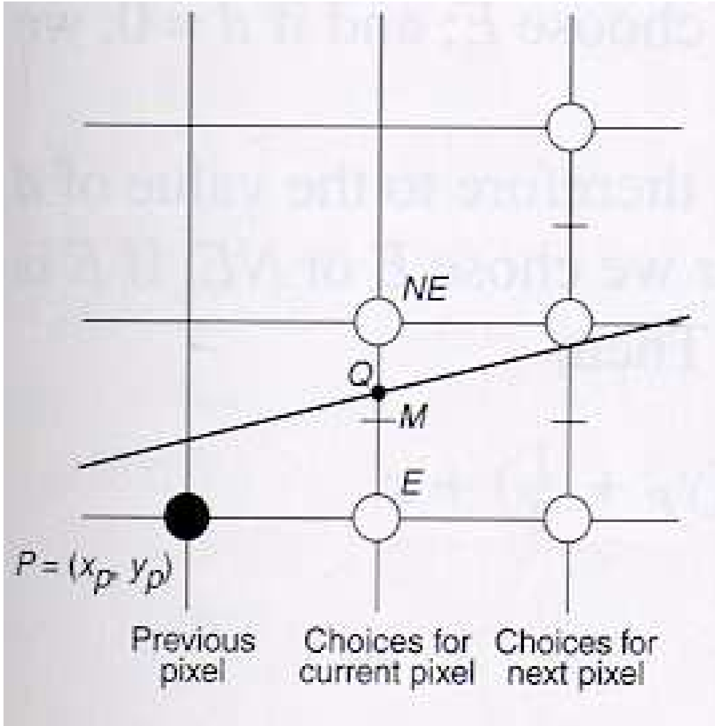
* If NE is chosen, then we have:

$d_{new} = F(M_2) = F(x_p + 2, y_p + 3/2) = a(x_p + 2) + b(y_p + 3/2) + c$

Also, $d_{old} = a(x_p + 1) + b(y_p + 1/2) + c$

Take the difference: $\Delta_{NE} = d_{new} - d_{old} = a + b$ or $d_{new} = d_{old} + a + b$

- We do not have to compute $F(M_1)$ and $F(M_2)$ directly !!



- **Summary of the midpoint algorithm**

  - Choose between E and NE, based on the sign of d

  - Update d by adding either Δ_E or Δ_{NE}

- **How should we compute d_{start} ?**

  - The first midpoint will be at $(x_0 + 1, y_0 + 1/2)$

    $F(x_0 + 1, y_0 + 1/2) = a(x_0 + 1) + b(y_0 + 1/2) + c =$

    $ax_0 + by_0 + c + a + b/2 = F(x_0, y_0) + a + b/2$

  - What is the value of $F(x_0, y_0)$?

    $F(x_0, y_0) = 0$ since (x_0, y_0) lines on the line !!

- Thus, $d_{start} = a + b/2 = dy - dx/2$

- Can we get rid of the division in *dstart* ? Yes !!

$$F'(x, y) = 2F(x, y)$$

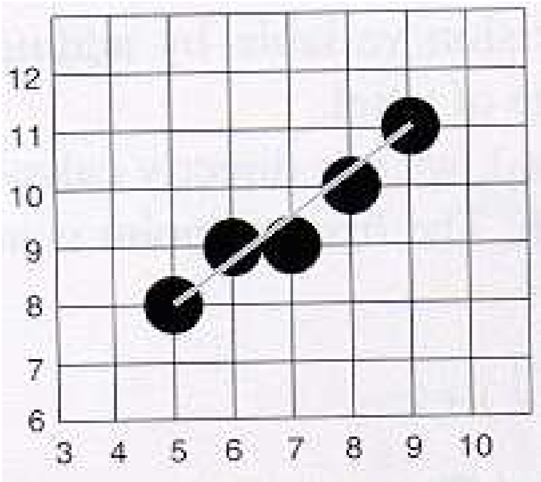
$$d' = 2d$$

$$d'_{start} = 2dy - dx$$

$$\Delta d' = 2\Delta d$$

- This is an algorithm based on integer arithmetic only !!

- **An example**



$(x_0, y_0) = (5, 8), (x_1, y_1) = (9, 11)$

$dy = y_1 - y_0 = 11 - 8 = 3,\ dx = x_1 - x_0 = 9 - 5 = 4$

$d_{start} = a + b/2 = dy - dx/2 = 3 - 4/2 = 1 > 0$, choose NE

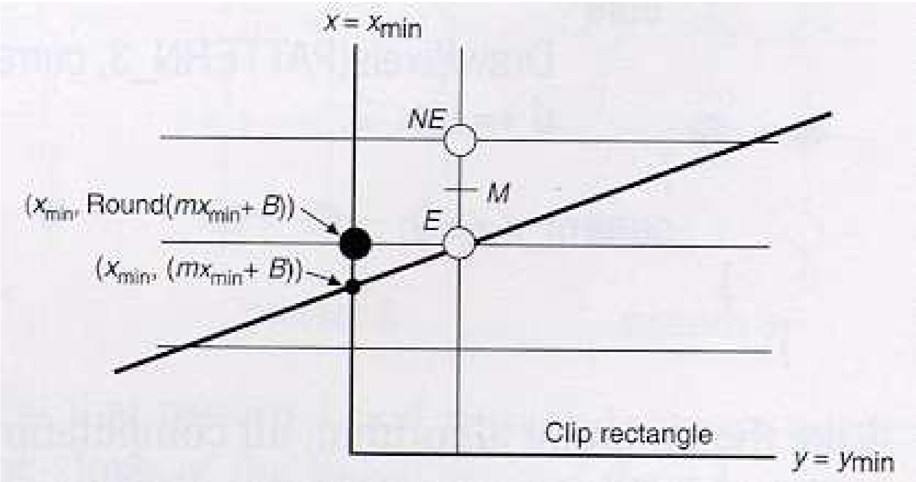
$d_{new} = d_{old} + a + b = 1 + 3 - 4 = 0$, choose E

$d_{new} = d_{old} + a = 0 + 3 = 3 > 0$, choose NE

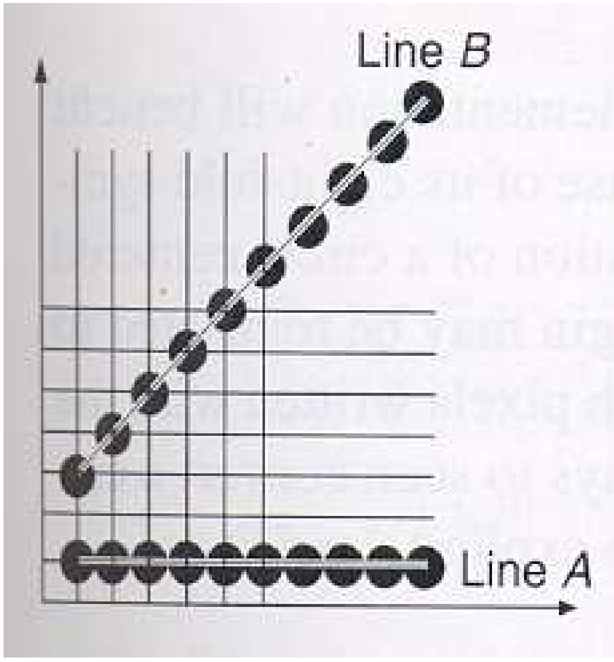
$d_{new} = d_{old} + a + b = 3 + 3 - 4 = 2 > 0$, choose NE

- **Additional issues**

  - Clipped lines



  - Intensity versus slope



* Distance of horizontal pixels: 1

* Distance of diagonal pixels: $\sqrt{2}$

* Length of line B: $\sqrt{2}$(length of line A)

* Intensity should be a function of slope (*anti-aliasing*)