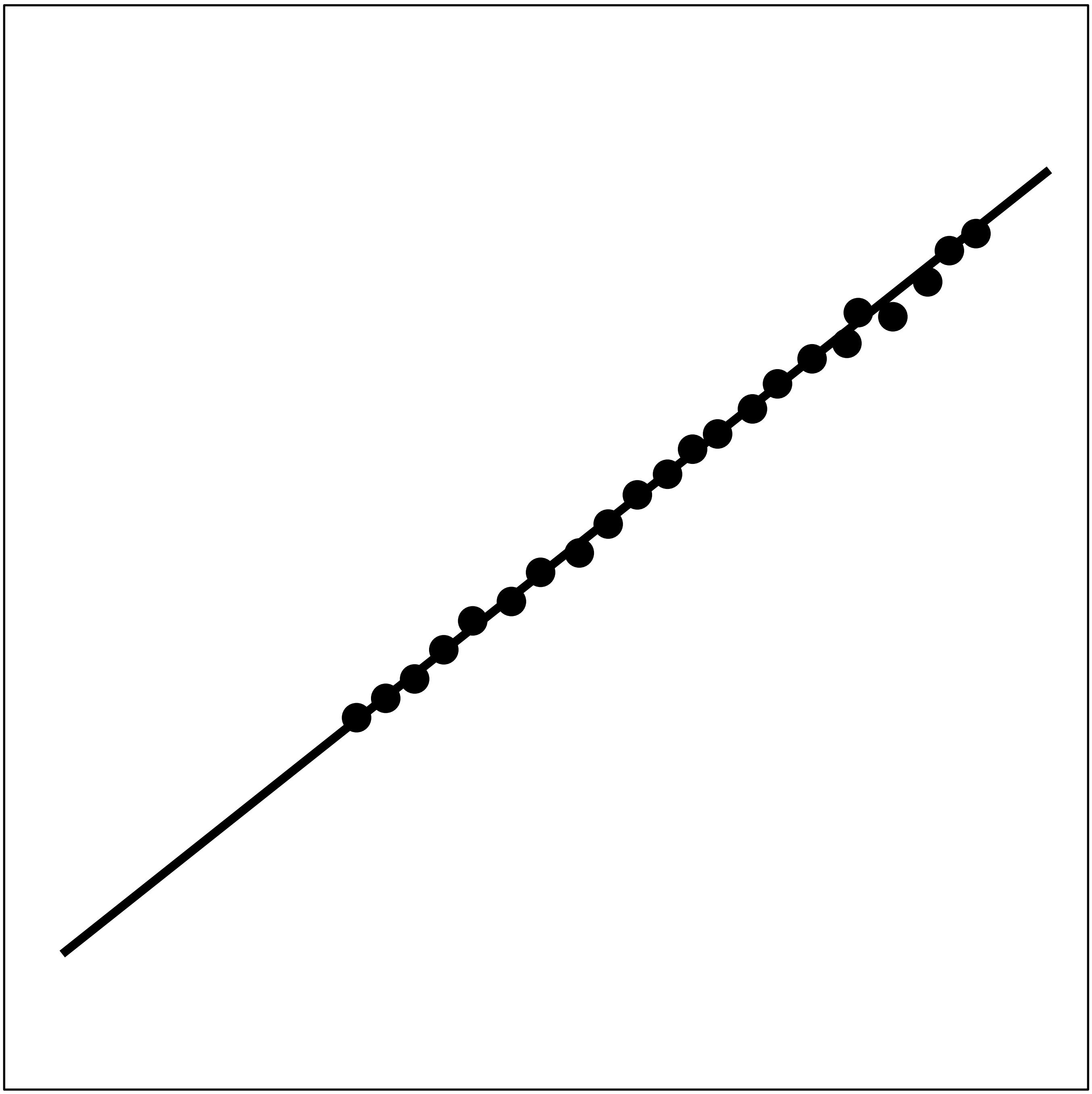
Line Drawing

- Why line drawing
  The line is the most fundamental drawing primitive with many uses
  - charts, engineering drawings, illustrations, 2D pencil-based animation, curve approximation
- Desired properties of line drawing algorithms
  - line should be straight
  - endpoint interpolation
  - uniform density for all lines
  - efficient
- Our ultimate goal —— efficient and correct line drawing algorithm
  draw-line(x_0, y_0, x_1, y_1)

Line Drawing



Algorithm Assumption

- Point samples on 2D integer lattice
- Bi-level display: on or off
- Line endpoints are integer coordinates
- All line slopes are: $|k| <= 1$
- Lines are ONE pixel thick

Simple Algorithm

- Draw-line (x_0, y_0, x_1, y_1)
  - let $\delta y = y_1 - y_0$, $\delta x = x_1 - x_0$
  - for $x = x_0$ to x_1
    $y = round(y_0 + (x - x_0)(\delta y / \delta x))$
    draw-point (x,y)
    end for
- Why does the above procedure work?
- Explicit definition of the line

$$y = \frac{\delta y}{\delta x}(x - x_0) + y_0$$

where $\delta y = y_1 - y_0$, and $\delta x = x_1 - x_0$

Line Equations

- **Parametric equation**

$$x(t) = x_0 + t(x_1 - x_0)$$

$$y(t) = y_0 + t(y_1 - y_0)$$

where $t \in [0, 1]$
how about when $t < 0$ **or** $t > 1$

- **Vector expression**

$$\mathbf{p}(t) = \mathbf{p}_0 + t(\mathbf{p}_1 - \mathbf{p}_0)$$

$$\mathbf{p}(t) = (1 - t)\mathbf{p}_0 + t\mathbf{p}_1$$

where $\mathbf{p} = [x, y]^\top$**, how about** $\mathbf{p}_0$ **and** $\mathbf{p}_1$

- **How do we improve the previous algorithm?**

- **Observations**

$$y_{curr} = y_0 + (x - x_0)\frac{\delta y}{\delta x}$$

$$y_{next} = y_0 + (x + 1 - x_0)\frac{\delta y}{\delta x}$$

$$y_{next} = y_{curr} + \frac{\delta y}{\delta x}$$

- **A more efficient algorithm**

  $x = x_0$; $y = y_0$
  draw-point (x,y)
  for x **from** $x_0 + 1$ **to** x_1
  $y = y + \frac{\delta y}{\delta x}$
  draw-point ($x, round(y)$**)**
  end for

Midpoint Algorithm

- Implicit equation (expression)

$$f(x, y) = (x - x_0)\delta y - (y - y_0)\delta x$$

If $f(x, y) = 0$, then (x, y) is on the line
If $f(x, y) > 0$, then (x, y) is below the line
If $f(x, y) < 0$, then (x, y) is above the line

- Midpoint algorithm is a recursive algorithm!

- Ideas!!!

- Consider $d = f(x_p + 1, y_p + 0.5)$

- There are three different cases

  - if $d < 0$, line is below midpoint, choose E
  - if $d > 0$, line is above midpoint, choose NE
  - if $d = 0$, line is passing midpoint, either E or NE

- For recursive algorithm, we MUST consider the subsequent step!

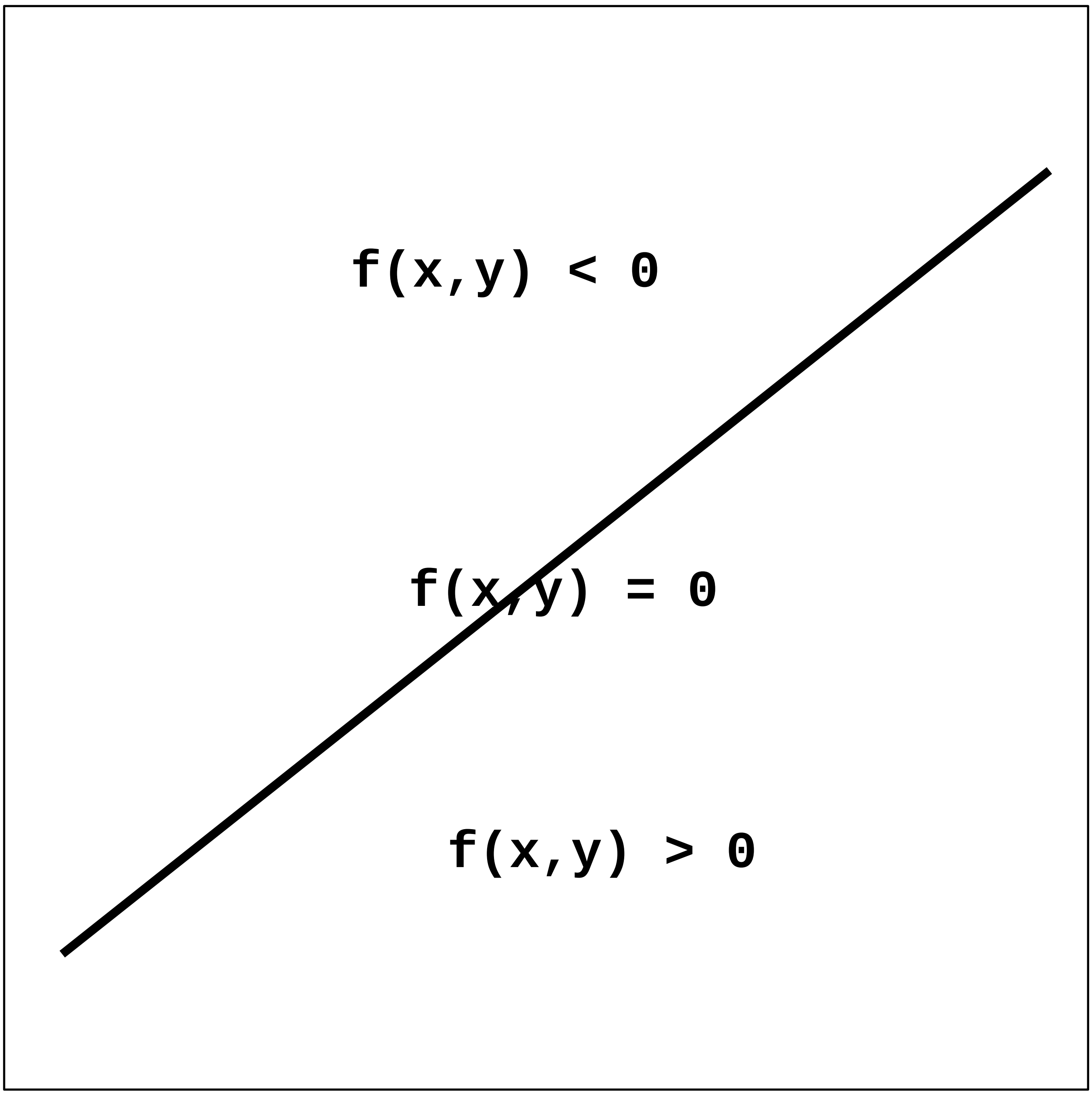
- **If E is chosen, the NEW E is** $(x_p + 2, y_p)$**,**
  the NEW NE is $(x_p + 2, y_p + 1)$**,**
  the NEW midpoint is $(x_p + 2, y + 0.5)$
  $d_{new} = f(x_p + 2, y + 0.5)$
  $d_{new} - d_{old} = \delta y$
  $d_{new} = d_{old} + \delta y$

- **If NE is chosen, the NEW E is** $(x_p + 2, y_p + 1)$**,**
  the NEW NE is $(x_p + 2, y_p + 2)$**,**
  the NEW midpoint is $(x_p + 2, y + 1.5)$
  $d_{new} = f(x_p + 2, y_p + 1.5)$
  $d_{new} - d_{old} = \delta y - \delta x$

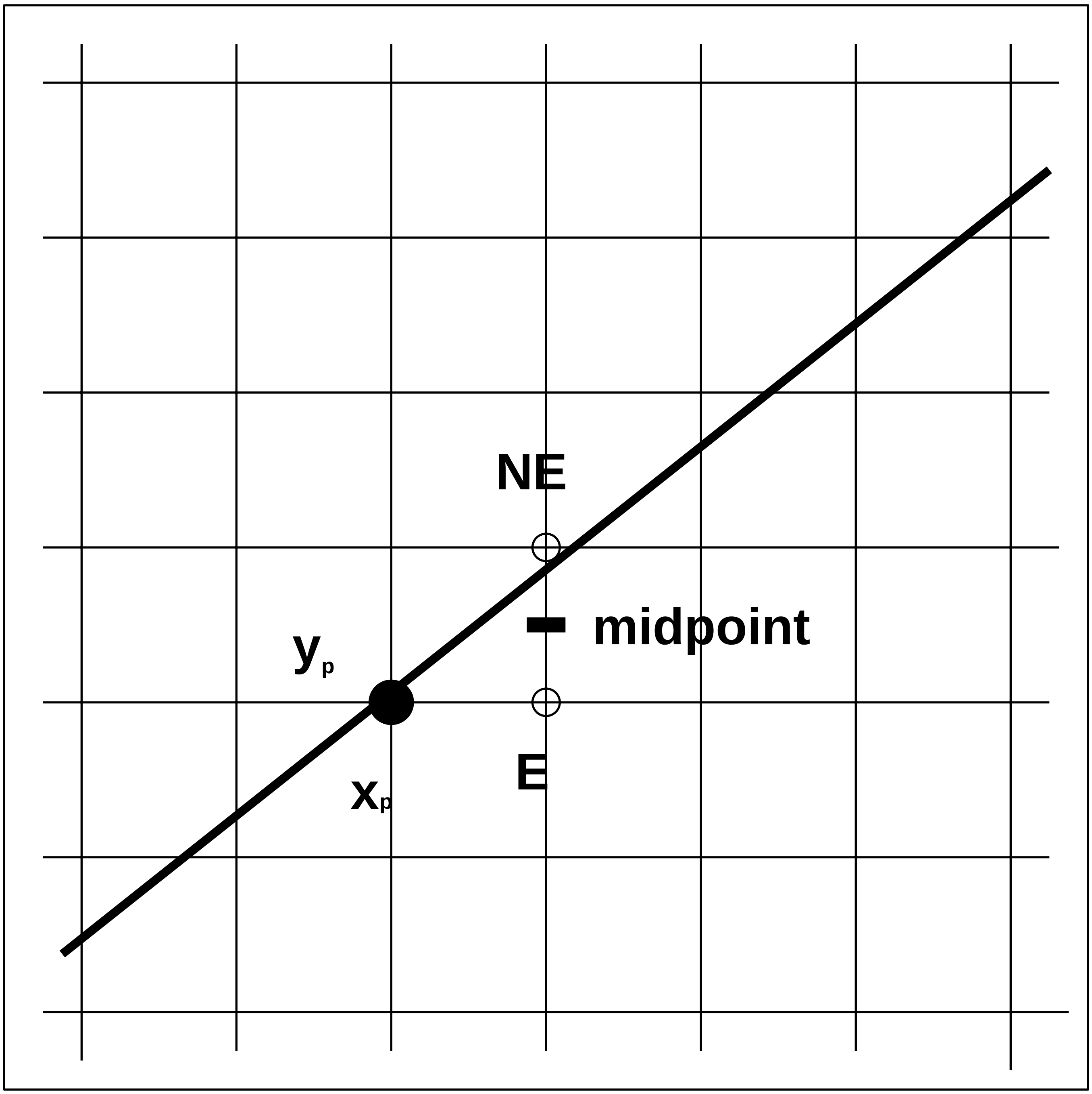
- **This process repeats recursively,**
  stepping along x **from** x_0 **to** x_1

- **How about initialization?**
  At the beginning, $x_p = x_0$**,** $y_p = y_0$
  $d = f(x_0 + 1, y_0 + 0.5) = \delta y + \frac{\delta x}{2}$

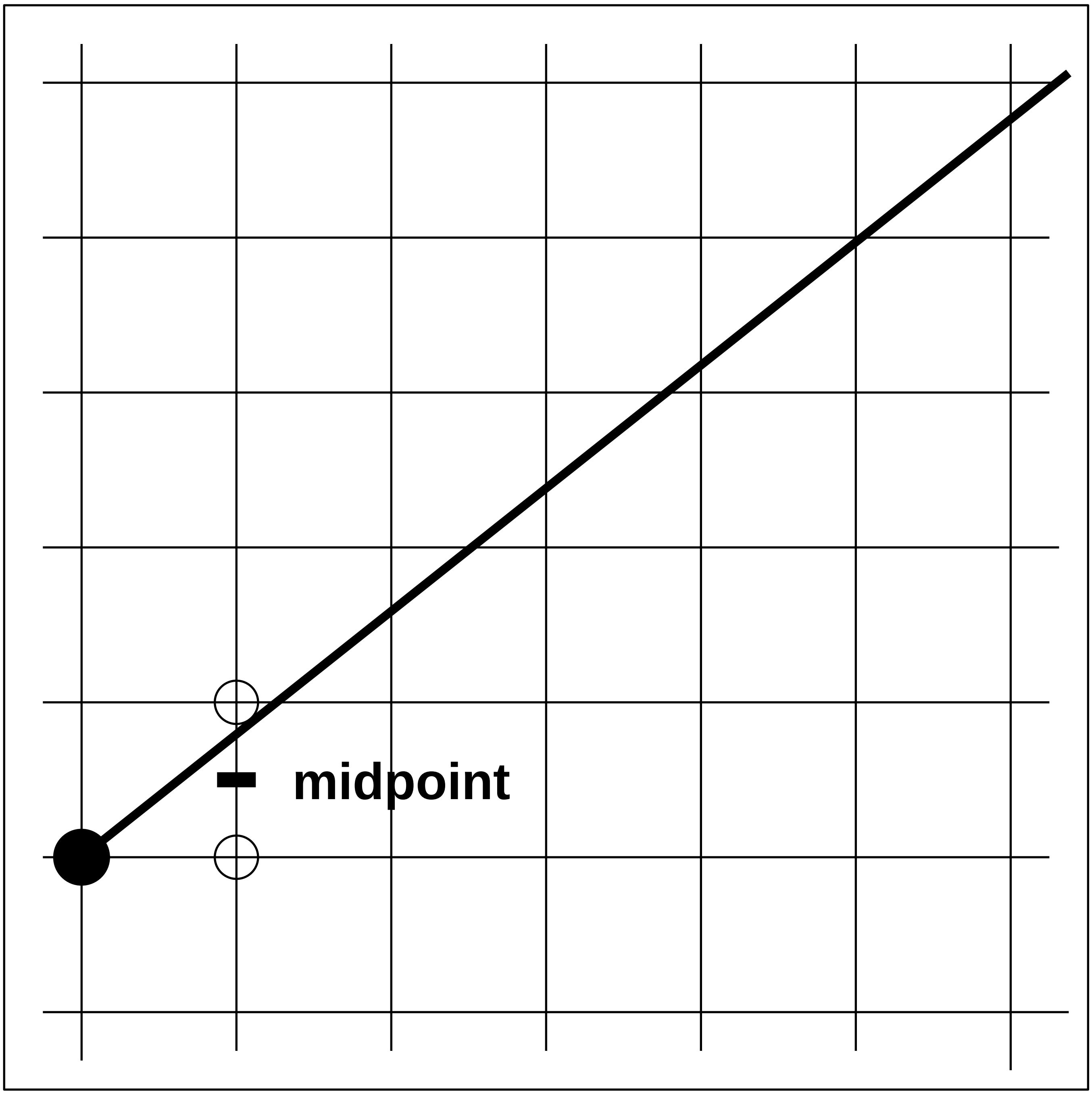
Implicit Equation



Midpoint Motivation



Midpoint Initialization



Midpoint Algorithm

- **draw-line** (x_0, y_0, x_1, y_1)
  int x_0, y_0, x_1, y_1
  $\{$ **int** $\delta x, \delta y, incE, incNE, x, y,$
  real d
  $\delta x = x_1 - x_0$ $\delta y = y_1 - y_0$ $d = \delta y - \frac{\delta x}{2}$
  $incE = \delta y, incNE = dy - dx$ $y = y_0$
  for x **from** x_0 **to** x_1
  draw-point (x,y)
  if $d > 0$**, then** $d = d + incNE$**,** $y = y + 1$
  else $d = d + incE$
  end for $\}$

- d **is not an integer, however, only the sign matters!**

- **We prefer an integer-only algorithm!!!**
  $f'(x, y) = 2f(x, y)$
  $d' = 2d$
  $d' = 2\delta y - \delta x$

Midpoint (integer-only) Algorithm

- **draw-line** (x_0, y_0, x_1, y_1)
  int x_0, y_0, x_1, y_1
  $\{$ **int** $\delta x, \delta y, incE, incNE, x, y, d$
  $\delta x = x_1 - x_0$
  $\delta y = y_1 - y_0$
  $d = 2\delta y - \delta x$
  $incE = 2\delta y, incNE = 2(\delta y - \delta x)$
  $y = y_0$
  for x **from** x_0 **to** x_1
  draw-point (x,y)
  if $d > 0$**, then** $d = d + incNE$**,** $y = y + 1$
  else $d = d + incE$
  end for $\}$

- **Assumptions**
  - **slopes:** $0 <= \frac{\delta y}{\delta x} <= 1$
  - **line endpoints are integer coordinates**

- **How about other cases**

– negative slope
– slope larger than 1

- If the slope is larger than 1, we use symmetry to switch x and y!

- If negative slope, we use x and $-y$

Generalizations

- Generalize to all cases for line drawing
- Algorithms for circle-drawing
- Algorithms for ellipses, conic section drawing
- Line drawing: P84-P92
- Circle-generating algorithms: P97-P102
- Ellipse-generating algorithms: P102-P107

Circle

- **Implicit function**

$$f(x, y) = (x - x_0)^2 + (y - y_0)^2 - r^2$$

- **If $f(x, y) = 0$, then (x, y) is on the circle**
- **If $f(x, y) > 0$, then (x, y) is outside the circle**
- **If $f(x, y) < 0$, then (x, y) is inside the circle**
- **Explicit definition**

$$y = y_0 + \sqrt{r^2 - (x - x_0)^2}$$

or

$$y = y_0 - \sqrt{r^2 - (x - x_0)^2}$$

where $-r <= (x - x_0) <= r$

- **Parametric definition**

$$x(\theta) = x_0 + r\cos(\theta); y(\theta) = y_0 + r\sin(\theta)$$

where $\theta \in [0, 2\pi]$

- **Equations for ellipses!**