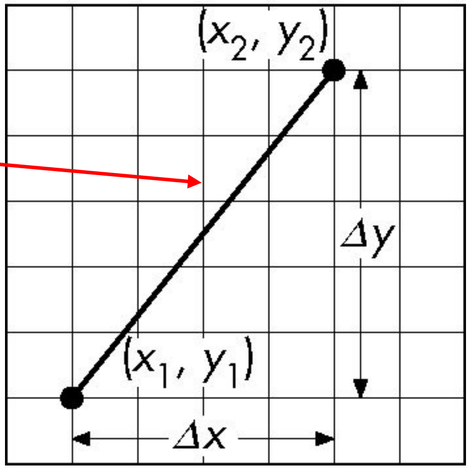
Scan Conversion of Line Segments

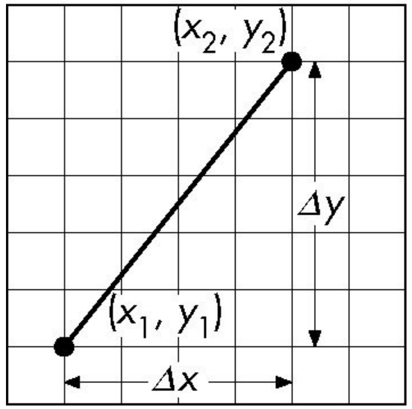
- Start with line segment in window coordinates with integer values for endpoints
- Assume implementation has a **`write_pixel`** function

$$k = \frac{\Delta y}{\Delta x}$$

$y = kx + m$



DDA Algorithm



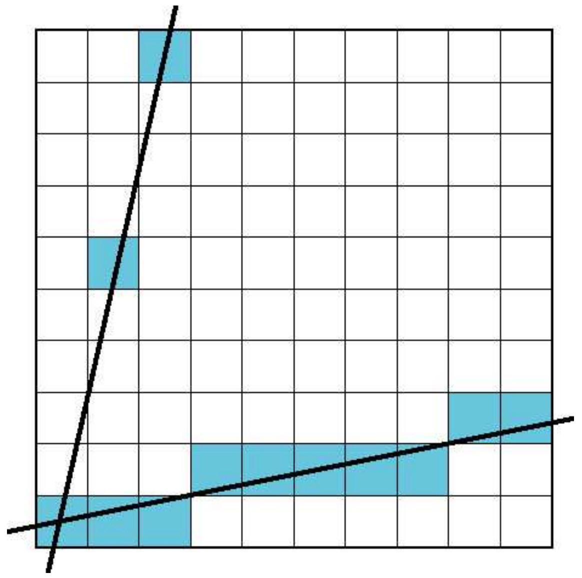
- Digital Differential Analyzer
  - DDA was a mechanical device for numerical solution of differential equations
  - Line $y=kx+m$ satisfies differential equation

    $dy/dx = k = \Delta y/\Delta x = y_2-y_1/x_2-x_1$
- Along scan line $\Delta x = 1$

```
y=y1;
For(x=x1; x<=x2,ix++) {
  write_pixel(x, round(y), line_color)
  y+=k;
}
```

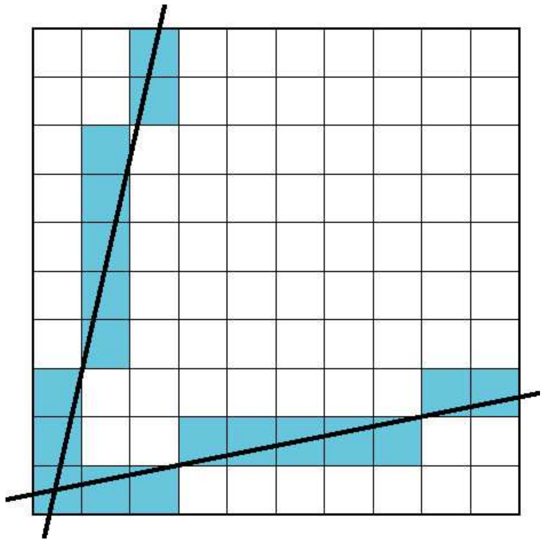
Problem

- DDA = for each x plot pixel at closest y
  - –Problems for steep lines



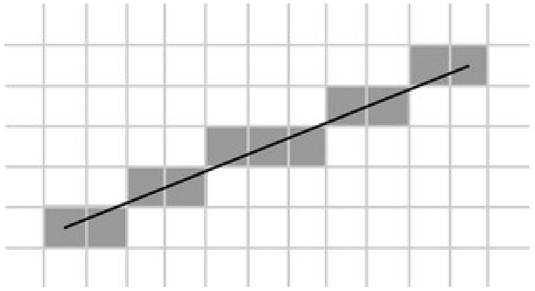
Using Symmetry

- Use for $1 \geq k \geq 0$
- For $k > 1$, swap role of x and y
  - For each y, plot closest x



- The problem with DDA is that it uses floats which was slow in the old days
- Bresenhams algorithm only uses integers

Bresenham's line drawing algorithm

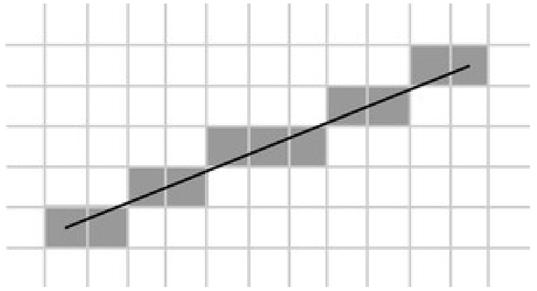


- The line is drawn between two points (x_0, y_0) and (x_1, y_1)
- Slope $k = \dfrac{(y_1 - y_0)}{(x_1 - x_0)}$ ($y = kx + m$)
- Each time we step 1 in x-direction, we should increment *y* with *k.* Otherwise the error in y increases with *k.*
- If the error surpasses 0.5, the line has become closer to the next *y*-value, so we add 1 to *y,* simultaneously decreasing the error by 1

```
function line(x0, x1, y0, y1)
    int deltax := abs(x1 - x0)
    int deltay := abs(y1 - y0)
    real error := 0
    real deltaerr := deltay / deltax
    int y := y0
    for x from x0 to x1
        plot(x,y)
        error := error + deltaerr
        if error ≥ 0.5
            y := y + 1
            error := error - 1.0
```

See also
http://en.wikipedia.org/wiki/Bresenham's_line_algorithm

Bresenham’s line drawing algorithm



- Now, convert algorithm to only using integer computations
- Trick: multiply the fractional number, *deltaerr*, by *deltax*
  - enables us to express *deltaerr* as an integer.
  - The comparison *if error>=0.5* is multiplied on both sides by *2*deltax*

Old float version:

```
function line(x0, x1, y0, y1)
    int deltax := abs(x1 - x0)
    int deltay := abs(y1 - y0)
    real error := 0
    real deltaerr := deltay / deltax
    int y := y0
    for x from x0 to x1
        plot(x,y)
        error := error + deltaerr
        if error ≥ 0.5
            y := y + 1
            error := error - 1.0
```

New integer version:

```
function line(x0, x1, y0, y1)
    int deltax := abs(x1 - x0)
    int deltay := abs(y1 - y0)
    real error := 0
    real deltaerr := deltay          ←  Multiply by deltax
    int y := y0
    for x from x0 to x1
        plot(x,y)
        error := error + deltaerr    ←  error implicitly mult by deltax
        if 2*error ≥ deltax          ←  Multiply by 2 deltax
            y := y + 1
            error := error - deltax  ←  Multiply by deltax
```

Complete Bresenham's line drawing algorithm

```
function line(x0, x1, y0, y1)
    *boolean* steep := abs(y1 - y0) > abs(x1 - x0)
    if steep then
        swap(x0, y0)
        swap(x1, y1)
    if x0 > x1 then
        swap(x0, x1)
        swap(y0, y1)
    *int* deltax := x1 - x0
    *int* deltay := abs(y1 - y0)
    *int* error := 0
    *int* ystep
    *int* y := y0
    if y0 < y1 then ystep := 1 else ystep := -1
    for x from x0 to x1
        if steep then plot(y,x) else plot(x,y)
        error := error + deltay
        if 2×error ≥ deltax
            y := y + ystep
            error := error - deltax
```

Swap loop axis

Swap start and end points

The first case is allowing us to draw lines that still slope downwards, but head in the opposite direction. I.e., swapping the initial points if x0 > x1.

To draw lines that go up, we check if y0 >= y1; if so, we step y by -1 instead of 1.

To be able to draw lines with a slope less than one, we take advantage of the fact that a steep line can be reflected across the line y=x to obtain a line with a small slope. The effect is to switch the x and y variables.