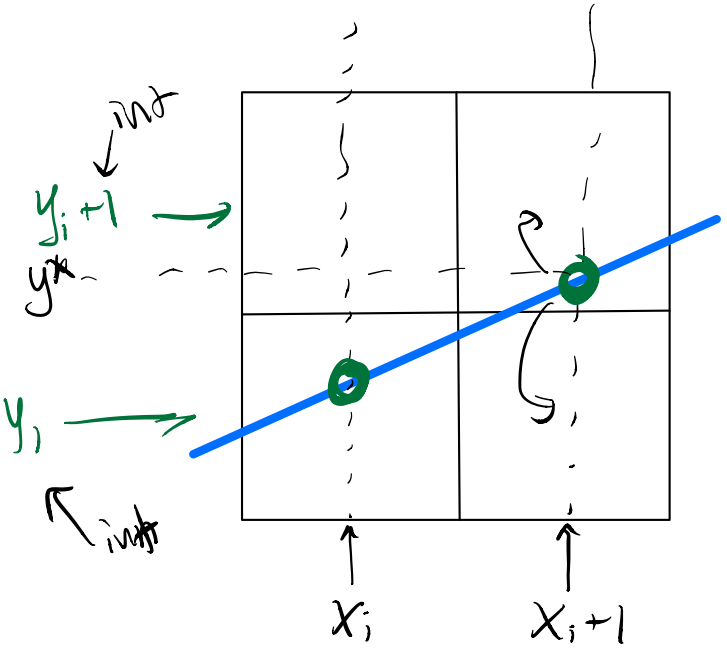
Midpoint Algorithm



Intuition: 1 pixel per col

pick pixel the line spends most time in

Equivalently: the pixel the line is in at integer x

Algorithm: $(y = mx+b)$

```
// compute m, b
for x = xmin : xmax
    y = m * x + b
    draw(x, round(y))
```

Efficiency?

3 flops

Faster Midpoint Algorithm

Original:

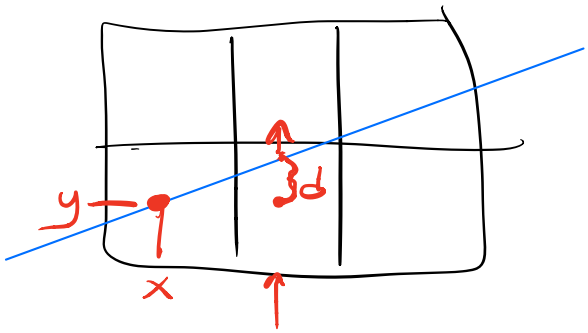
```
// compute m, b
for x = xmin : xmax:
    y ← b + m * x
    draw(x, round(y))
```

Faster:

```
// compute m, b
y = m * xmin + b
for x = xmin : xmax+1
    draw(x, round(y))
    y += m
```

Fasterer Midpoint Algorithm?!

```
//compute m, b
y = m * xmin + b
for x = xmin : xmax
    draw (x, round (y))
    d = m (x+1) + b  - y
    if( d > 0.5)
        y += 1
        d -= 1
```



← distance from line to current y

Strategy: incrementalize d as we did y above.

Flops : 1x >, 1x +, and 0 or 1x -

↑ shift by 0.5, This becomes a sign bit check!