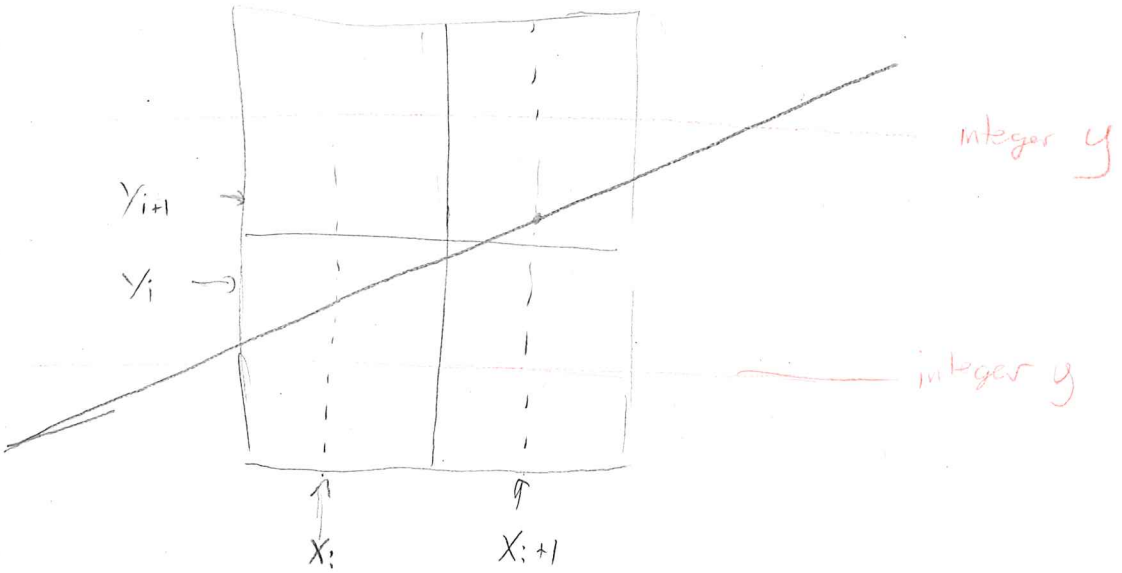
Midpoint Algorithm



Intuition: Pick 1 pixel per column
Pick the one the line spends the most time in.

Equivalently: The one the line is in at an integer value of x

Algorithm:

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

Exercise: write this line

Aside: given $(x_1, y_1), (x_2, y_2)$
what are m and b?

$$m = \frac{\text{rise}}{\text{run}} = \frac{y_2 - y_1}{x_2 - x_1}$$

$$b = y - mx$$
$$= y_1 - mx_1$$

Efficiency:

1 round, 1 fp mult, 1 fp add per x

Faster midpoint algorithm?

Take stuff out of the inner loop!

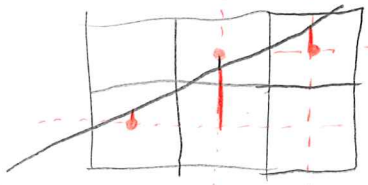
```
m =
b =
y = m xmin + b   (if xmin != x1)

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

Exercise: Speed it up!

Even faster?

Keep track of distance from integer y to line; if it's > 0.5, increment y and decrement d

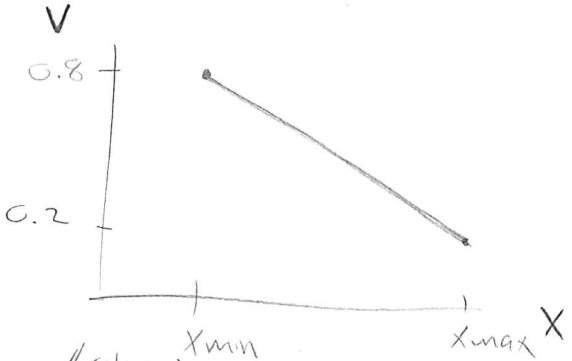


Adds and > only, no mult and no rounding

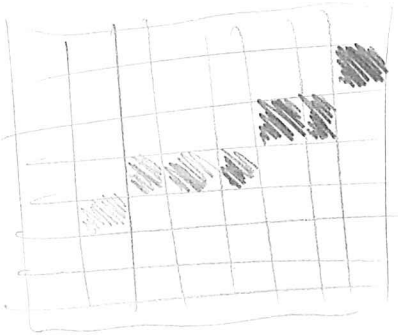
Interpolating Values

Given $P_1 = (x_1, y_1)$, $P_2 = (x_2, y_2)$ and V_1, V_2, values of same property at the endpoints, interpolate values at each pixel.

e.g., color



$V_0 = 0.8$ $V_1 = 0.2$



```
//calc mb
//calc vm, vb
y = M Xmin + b
V = Vm Xmin + Vb
for X = Xmin : Xmax
    draw(x, round(y), V)
    y += m
    V += Vm
```

One more aside: parametric view

$$V(x) = V_1 + \underbrace{\frac{x - x_1}{x_2 - x_1}}_{t}(V_2 - V_1)$$

$$= V_1 + t(V_2 - V_1)$$

$$= V_1 + tV_2 - tV_1$$

$$V(t) = (1-t)V_1 + tV_2$$