Midpoint Algorithm

Intuition: 1 pixel per col
pick pixel that line spends most time in
Equivalently: the pixel the line is in at integer $x$

Algorithm: $(y = mx + b)$
\[
\begin{align*}
\text{// compute } m, b \\
\text{for } x = x_{\text{min}} : x_{\text{max}} \\
y = m \times x + b \\
\text{draw}(x, \text{round}(y))
\end{align*}
\]

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: |x|, |x|+, and 0 or 1 -
Shift by 0.5, this becomes a sign bit check!