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 \(x = x_{\text{min}} : x_{\text{max}}\)
\(y = m \times x + b\)
draw((x, round(y)))

Efficiency? 3 Flops
Faster Midpoint Algorithm

Original:

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

Faster:

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

Fasterer Midpoint Algorithm?!

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

Strategy: incrementalize `d` as we did `y` above.

Flops: 1x>, 1x+, and 0 or 1x-
\[ d > 0.0 \]

**Interpolating Values**

**Given:** \( p_1, p_2, v_1, v_2 \)

- Endpoints
- Values of same property at \( p_1, p_2 \) (e.g.: color, any vertex data)

**Interpolate a \( V \) for each pixel**

**Example:** \( V_1 = 0.8 \)

\[ \begin{align*}
0.8 & \quad \rightarrow \quad V_2 = 0.2 \\
0.2 & \quad \rightarrow \quad V_3 = 0.0 \\
\end{align*} \]

```
// calc. m, b
y = m * x_m + b

// calc v_m, v_b
V = v_m * x_m + v_b

for x = x_min : x_max
    draw (x, round(y), V)
    y += m
    V += v_m
```
- Setup RP repo by tomorrow night
  (GitHub classroom, push proposal + feedback)

- A3 due W night
  - Point lights have $r^2$ falloff
  - Ambient light $l_n$ is a constant added to all pixel colors

- A2 grading feedback (except extensions)
  is on Github in the “Feedback” pull request

- Today's lab is completed in your AO repo
  - push finished changes by F night.
  - 10 pts in written HW category