Line Drawing

Rasterization

- First job: enumerate pixels covered by a primitive
  - simple, aliased definition: pixels whose centers fall inside
- Second job: interpolate values across primitive
  - e.g. colors computed at vertices
  - e.g. normals at vertices
**Rasterizing lines**

- Define line as a rectangle
- Specify by two endpoints
- Ideal image: black inside, white outside

**Point sampling**

- Approximate rectangle by drawing all pixels whose centers fall within the line
- Problem: sometimes turns on adjacent pixels
Point sampling in action

Bresenham algorithm in action
**Lane drawing algorithms**

- line equation: 
  \[ y = b + m \cdot x \]
- evaluate line equation per column

```plaintext
for x = ceil(x0) to floor(x1)
  y = b + m \cdot x
output(x, round(y))
```

\[ y = 1.91 + 0.37 \cdot x \]

---

**Optimizing line drawing**

- Multiplying and rounding is slow
- At each pixel the only options are E and NE
- \[ d = m(x + 1) + b - y \]
- \( d > 0.5 \) decides between E and NE
Optimizing line drawing

- \( d = m(x + 1) + b - y \)
- Only need to update \( d \) for integer steps in \( x \) and \( y \)

Bresenham line algorithm

```python
x = ceil(x0)
y = round(m*x + b)
d = m*(x + 1) + b - y
while x < floor(x1)
    if d > 0.8
        y += 1
        d -= 1
    x += 1
dx += m
output(x, y)
```
Your turn

- Code the Bresenham line drawing algorithm
- Make this image (or something similar)