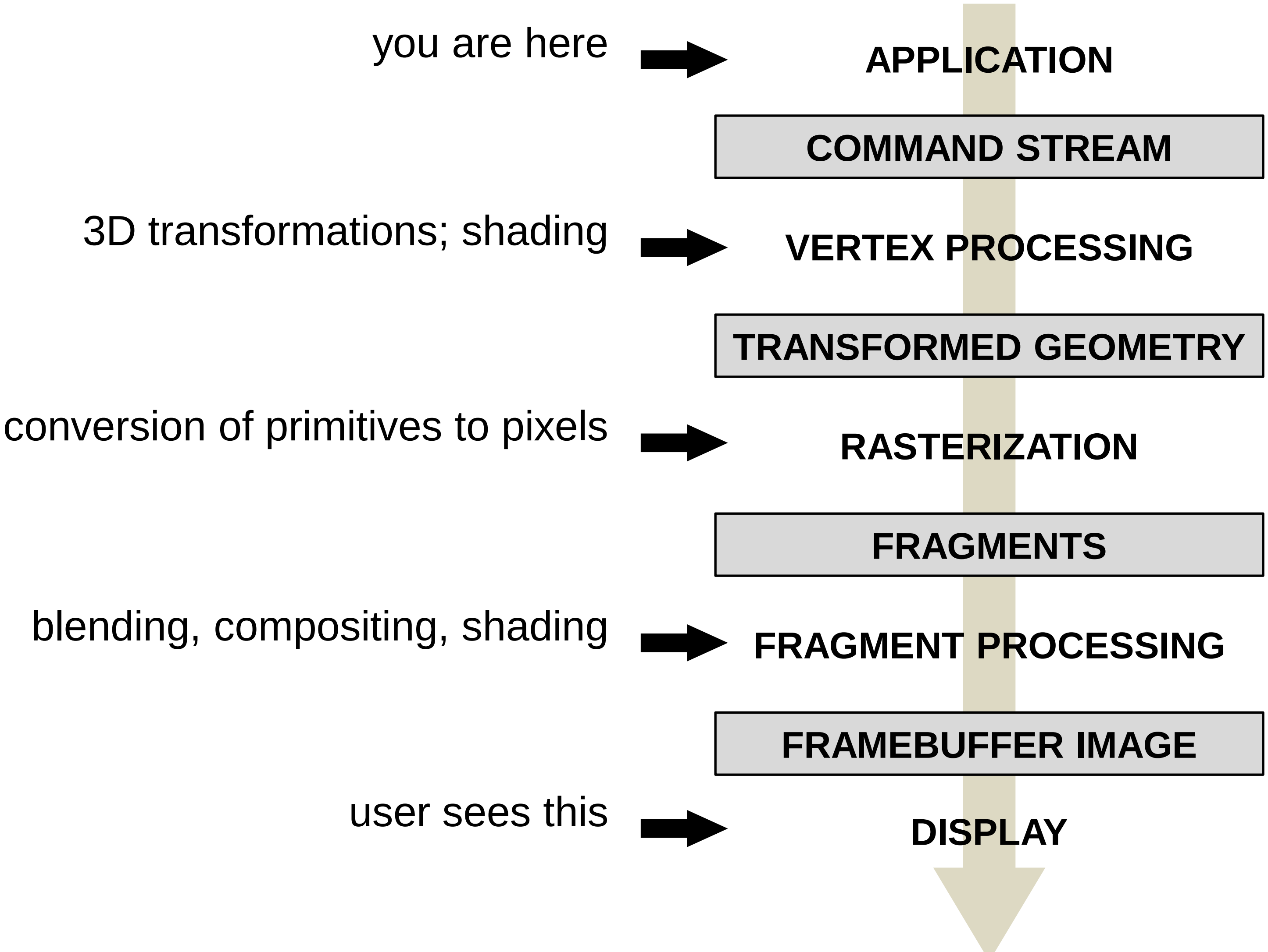
Rasterization

COMP 575/770
Spring 2013

The Rasterization Pipeline



Rasterization

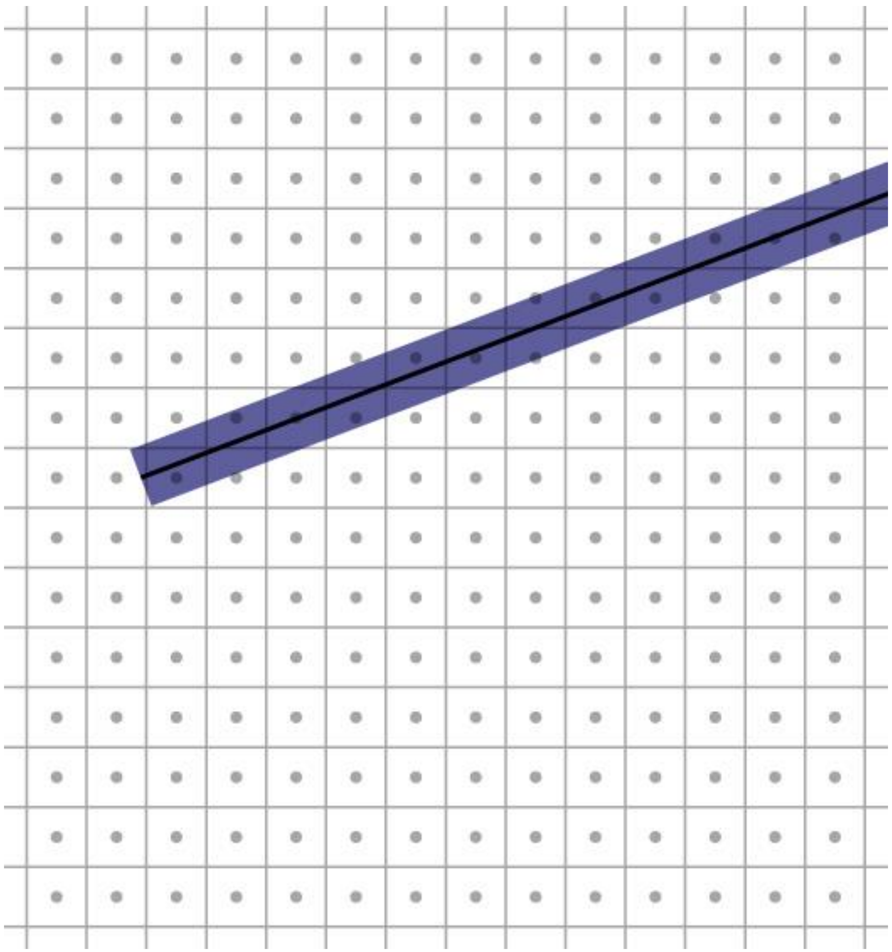
1. Project a primitive onto the screen
2. Find which pixels lie inside the projection
3. Interpolate **attributes** at each pixel
   - These are quantities that help in shading
4. Perform shading

Outline

- Overview
- Line Rasterization

Rasterizing Lines

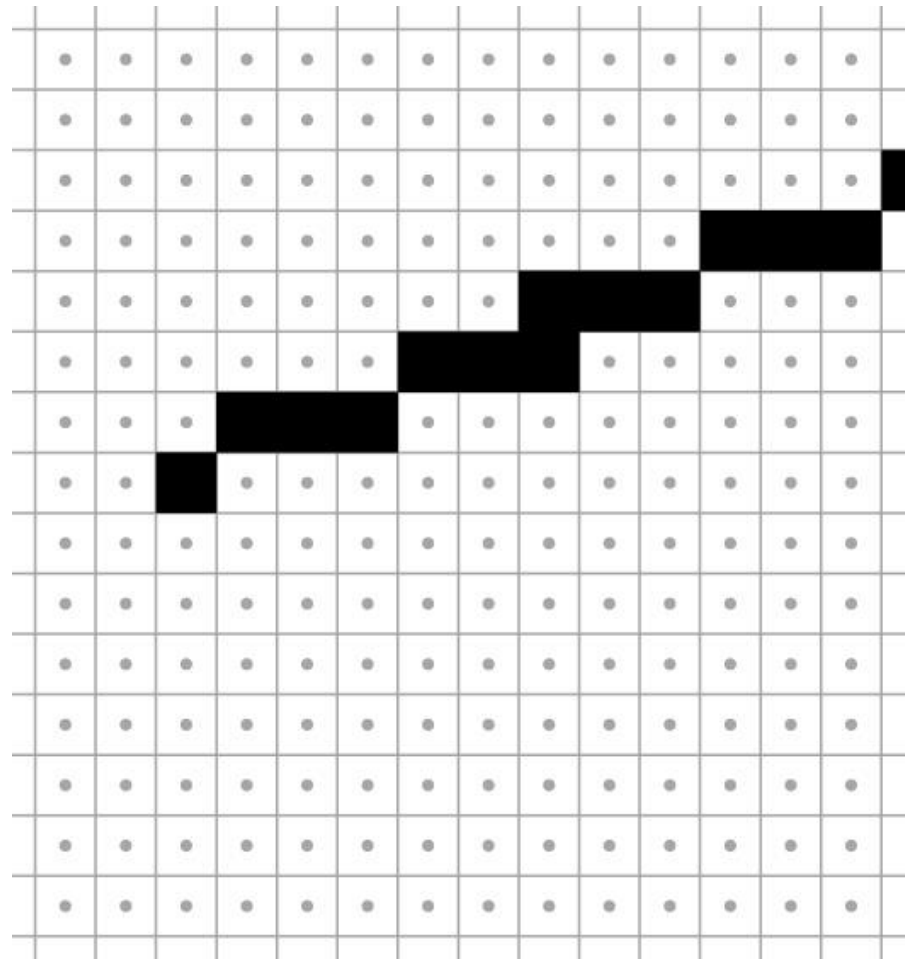
- Most pixels won't lie on the line!
- Approximate line using thin rectangle



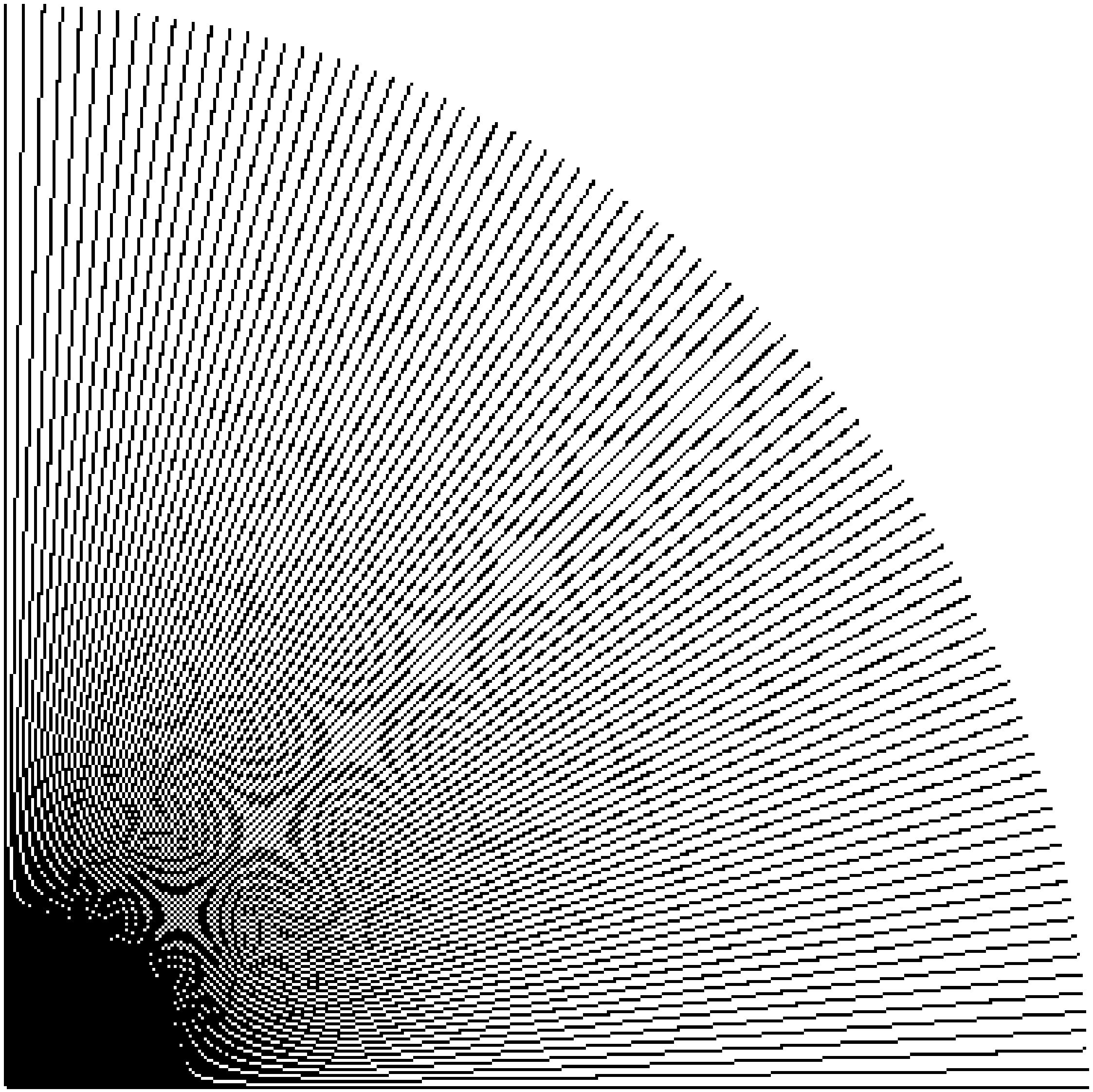
Rasterizing Lines

- Mark all pixels inside the thin rectangle?

- **Problem**: Sometimes marks adjacent pixels

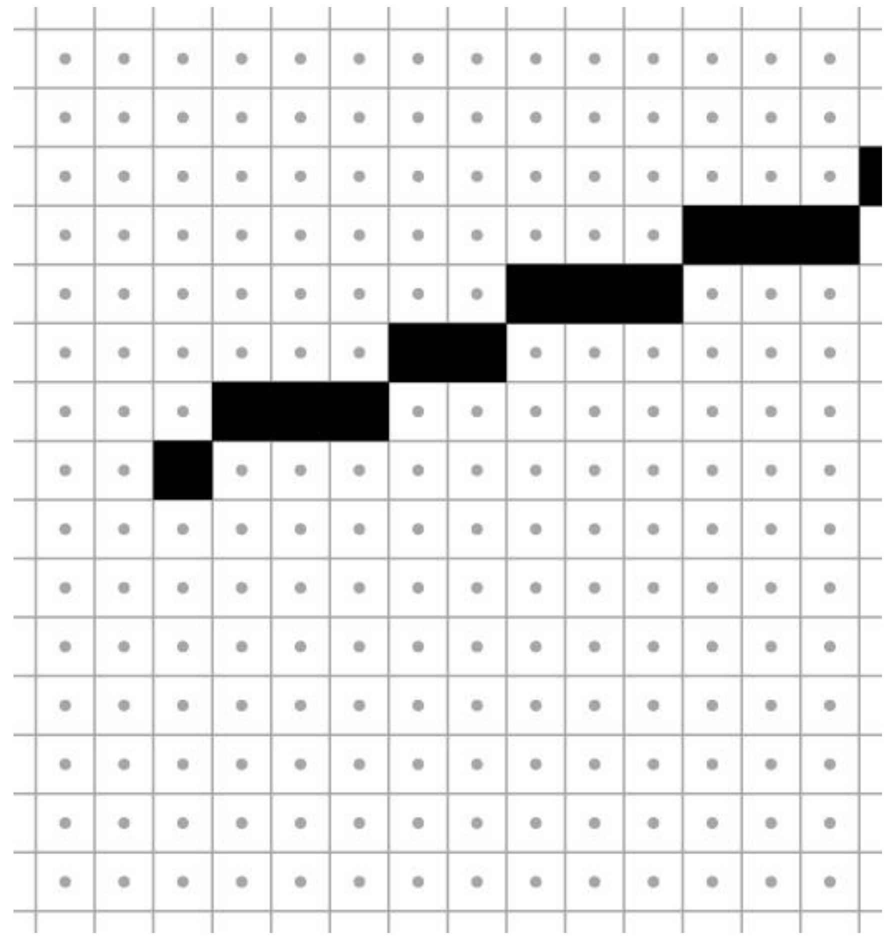


Rasterizing Lines

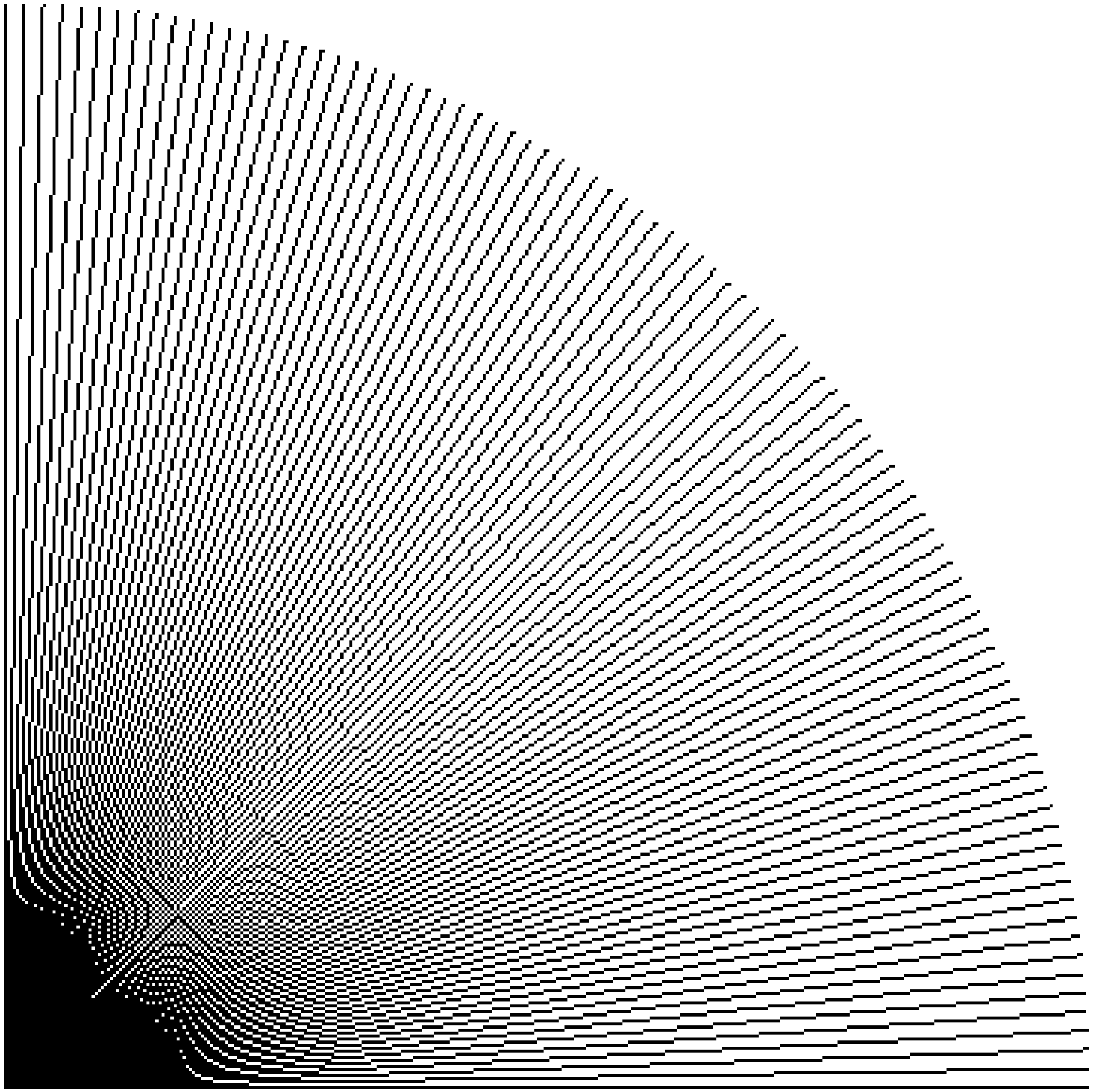


Midpoint Algorithm

- Mark only one pixel per column
  - The closest one
- Basically, line width defined parallel to pixel grid



Midpoint Algorithm



Midpoint Algorithm

- Slope-intercept form of line equation:

$$y = mx + b$$

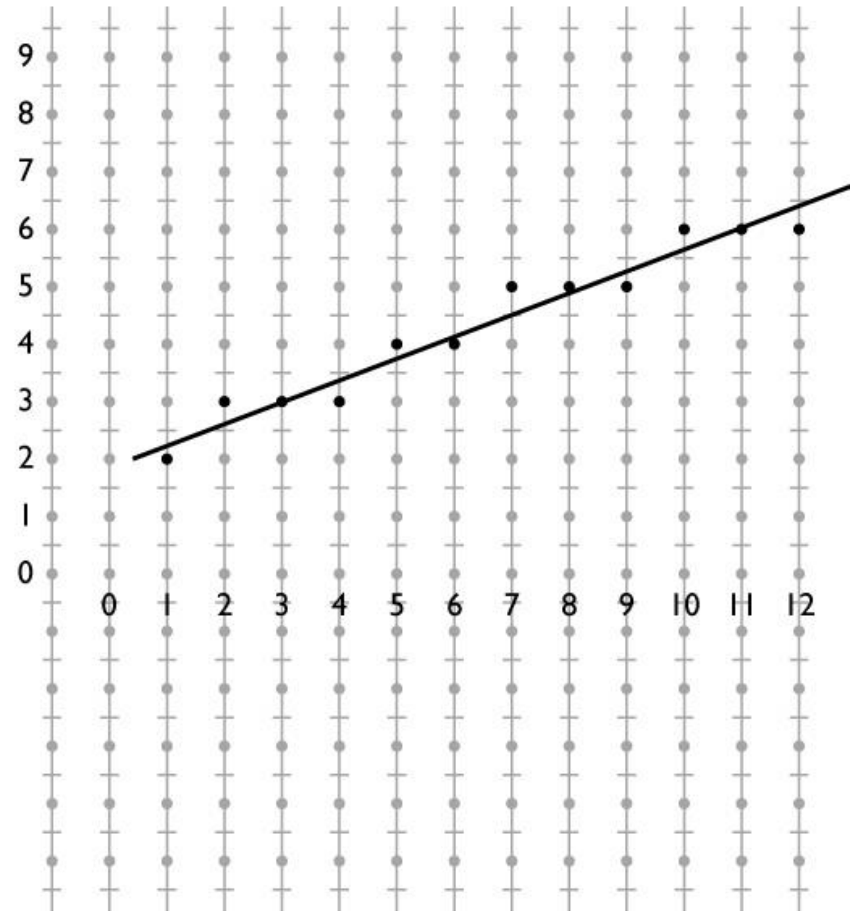
- We assume $m \in (0,1]$

- There are three other analogous cases:
  - $m \in (-\infty, -1]$
  - $m \in (-1,0]$
  - $m \in (1, \infty)$

Midpoint Algorithm

- Evaluate line equation per column
- Endpoints at $x_0 < x_1$

```
for x = ceil(x0) to floor(x1)
    y = m*x + b
    mark(x, round(y))
end
```

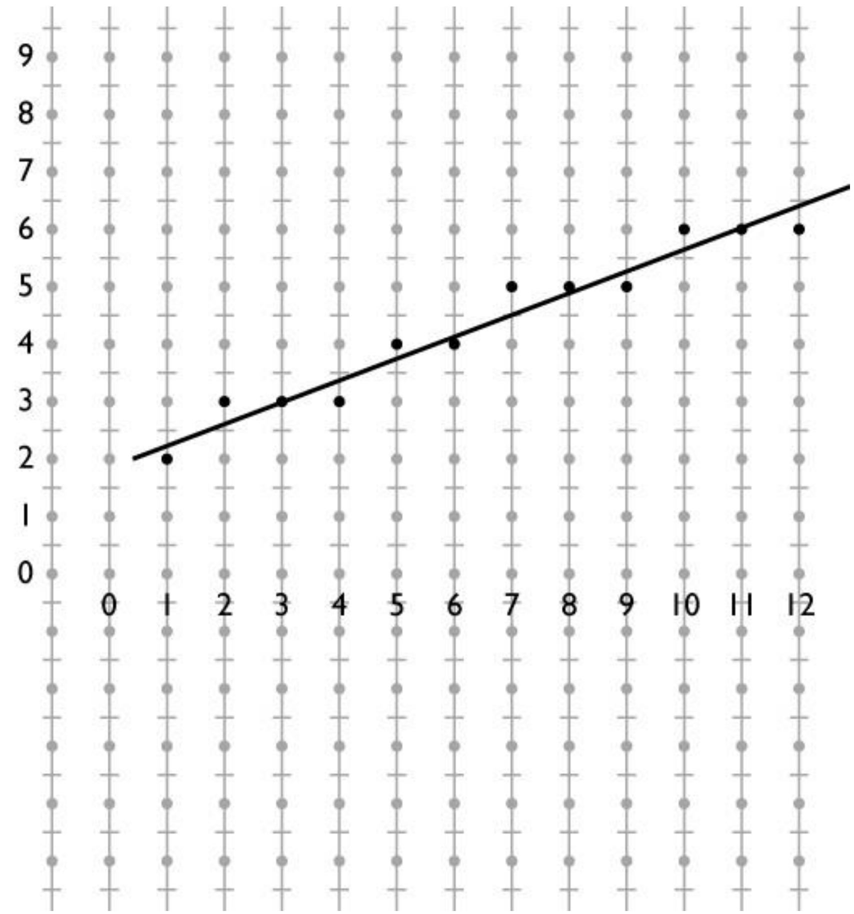


$$y = 1.91 + 0.37\,x$$

Midpoint Algorithm

- Evaluate line equation per column
- Endpoints at $x_0 < x_1$

```
x = ceil(x0)
while x < floor(x1)
    y = m*x + b
    mark(x, round(y))
    x += 1
end
```



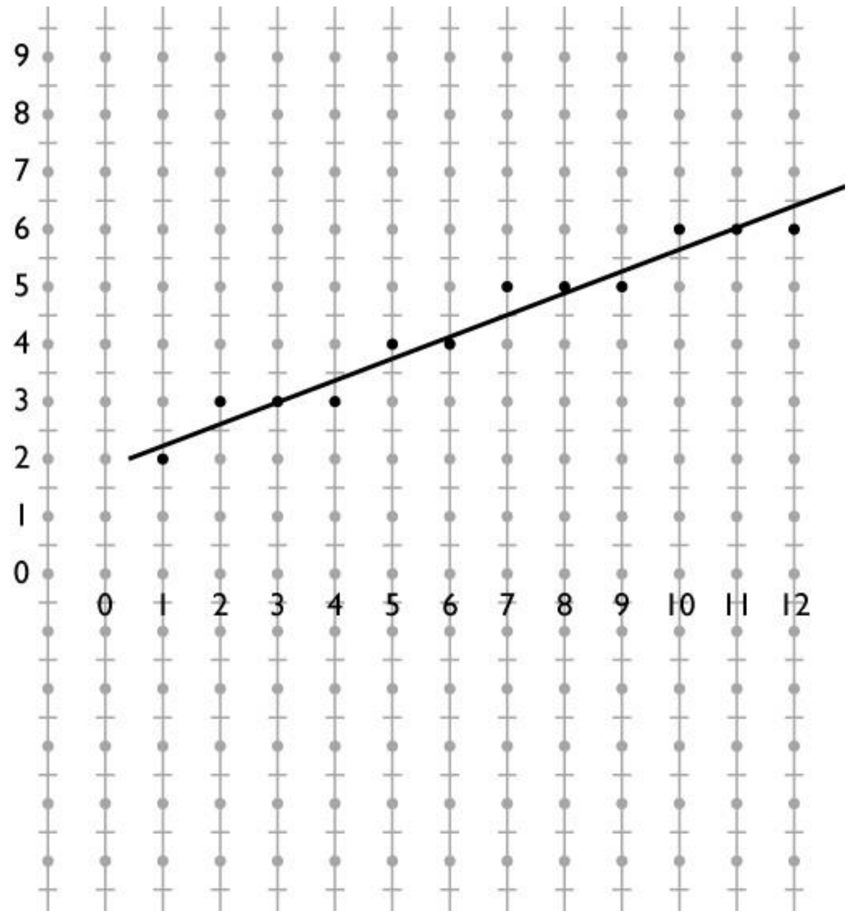
$$y = 1.91 + 0.37\,x$$

Optimized Midpoint Algorithm

- Two slow operations:
  - Multiply: y = m*x + b
  - Round: mark(x, round(y))
- y varies predictably:

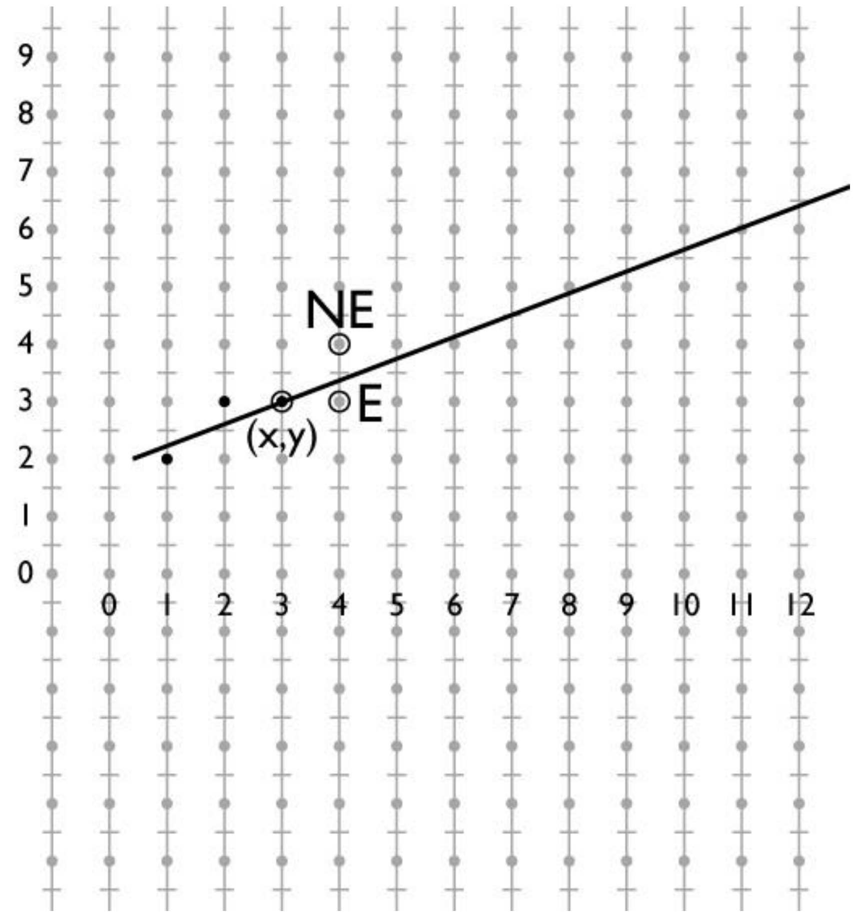
$$
\begin{aligned}
y(x+1) &= m(x+1) + b \\
&= y(x) + m
\end{aligned}
$$

```
x = ceil(x_0)
y = m*x + b
while x < floor(x_1)
    mark(x, round(y))
    x += 1
    y += m
end
```



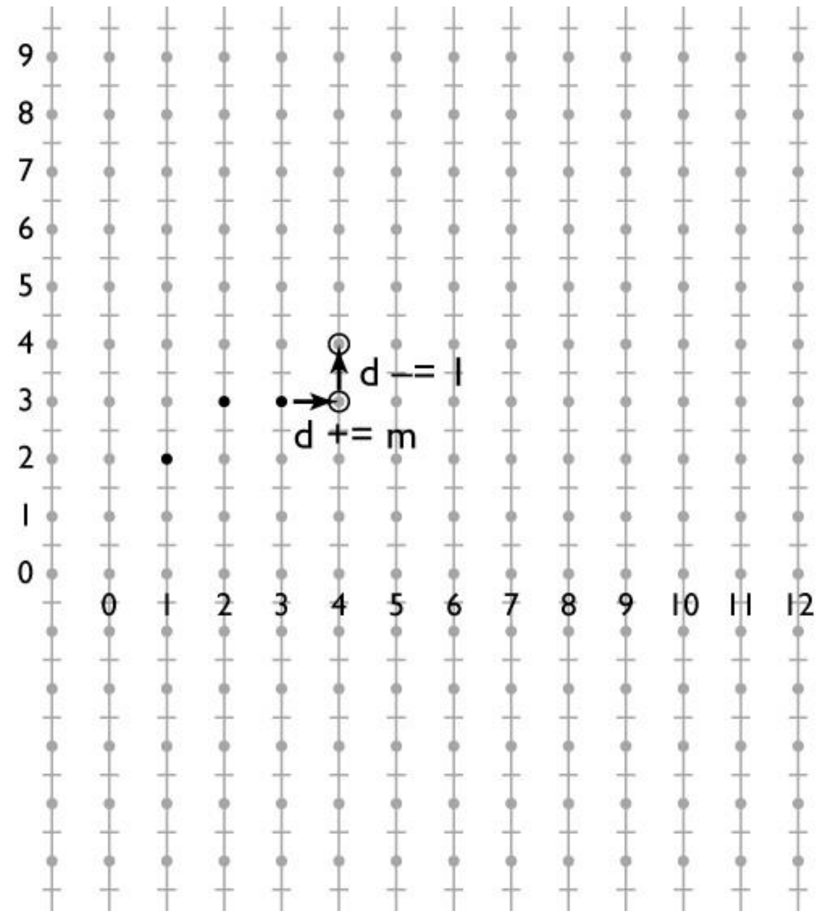
Optimized Midpoint Algorithm

- Only two options when moving to next column
- Which does line pass closer to?
- $d = m(x+1) + b - y$
- If $d > 0.5$, line is closer to NE
  - Otherwise, closer to E



Optimized Midpoint Algorithm

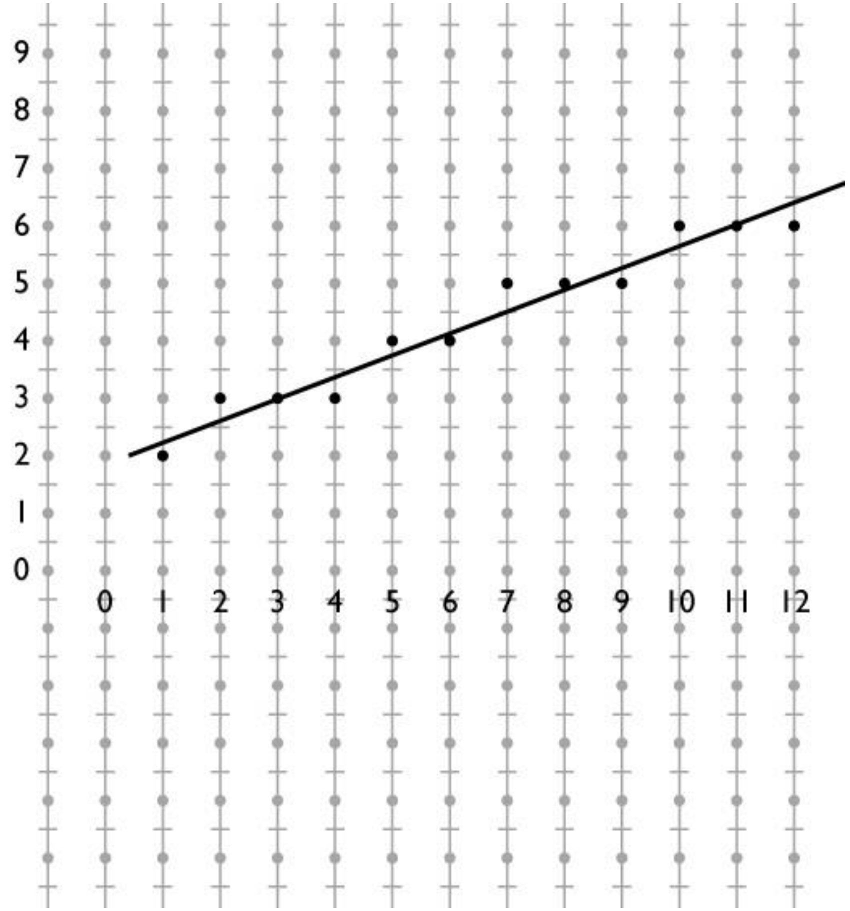
- Incrementally update d
- If we choose E:
  - Increment d by m
- If we choose NE:
  - Increment d by $m - 1$
- Approach also called **digital differential analyzer** (DDA)



$$d = m(x + 1) + b - y$$

Optimized Midpoint Algorithm

```
x = ceil(x0)
y = round(m*x + b)
d = m*(x+1) + b – y
while x < floor(x1)
    if d > 0.5
        y += 1
        d –= 1
    end
    x += 1
    d += m
    mark(x, y)
end
```



Outline

- Overview
- Line Rasterization
- Line Attributes

Attribute Interpolation

- **Attributes** are often attached to vertices/endpoints
  - E.g., color of hair drawn using lines
  - Want color to vary smoothly along line segments

- Linear interpolation:

$$f(x) = (1 - \alpha)y_0 + \alpha y_1$$
$$\alpha = \frac{x - x_0}{x_1 - x_0}$$

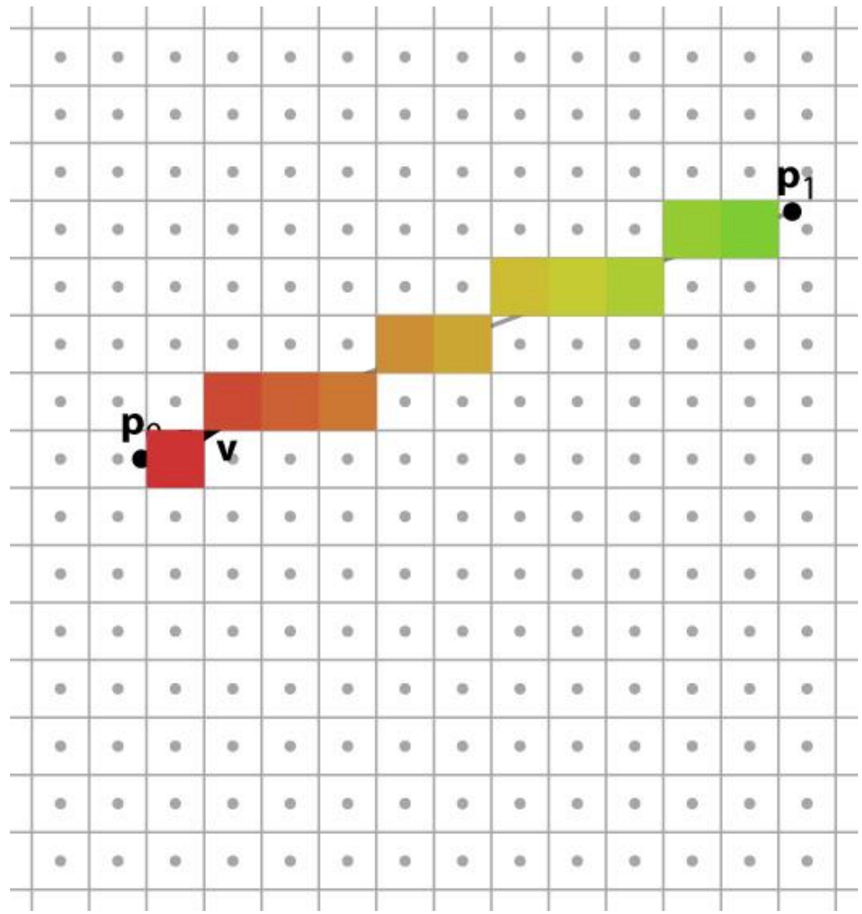
- α is the fraction of distance from (x_0, y_0) to (x_1, y_1)

Attribute Interpolation

- Suppose endpoint attributes are A_0 and A_1

  $$A(x) = A_0 + \alpha(A_1 - A_0)$$

- Since α is linear in x, can write an incremental expression



Outline

- Overview
- Line Rasterization
- Line Attributes
- Triangle Rasterization

Triangle Rasterization

- The most common primitive in most applications
  - Can represent any object using many triangles
  - A triangle always projects to a triangle

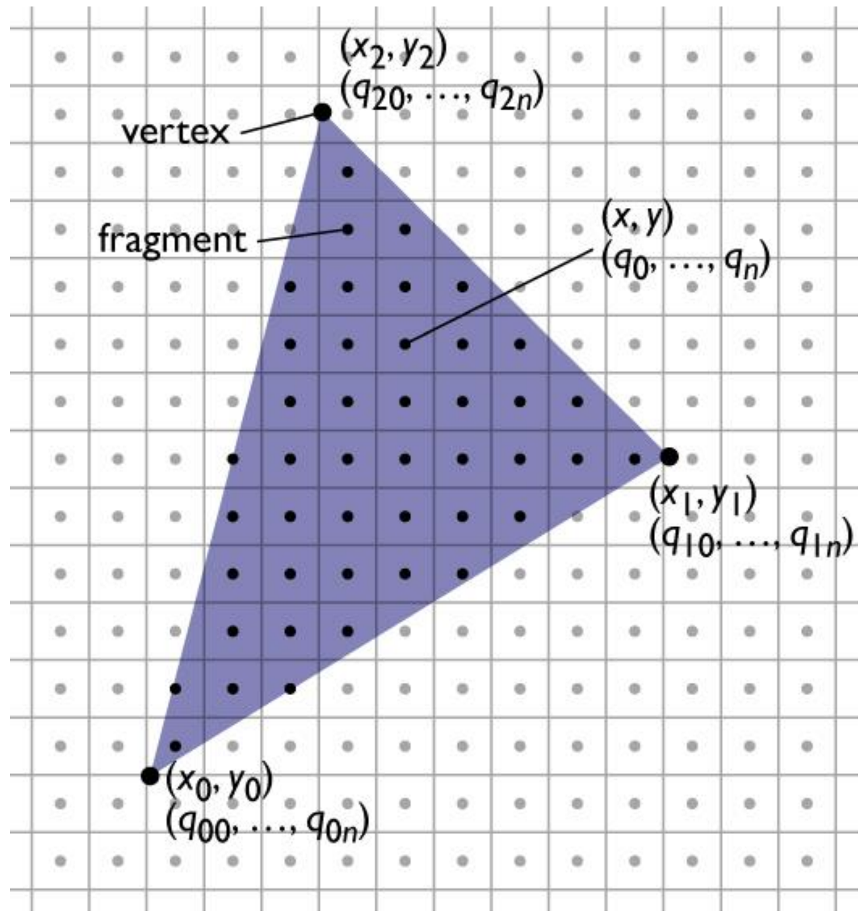
- Triangle represented by 3 vertices
  - $\mathbf{a} = (x_a, y_a)$, $\mathbf{b} = (x_b, y_b)$, and $\mathbf{c} = (x_c, y_c)$

- Need to figure out which pixels are inside the triangle

Bounding Rectangle

- Smallest rectangular portion of screen which contains triangle
- No pixels outside it could possibly be in the triangle

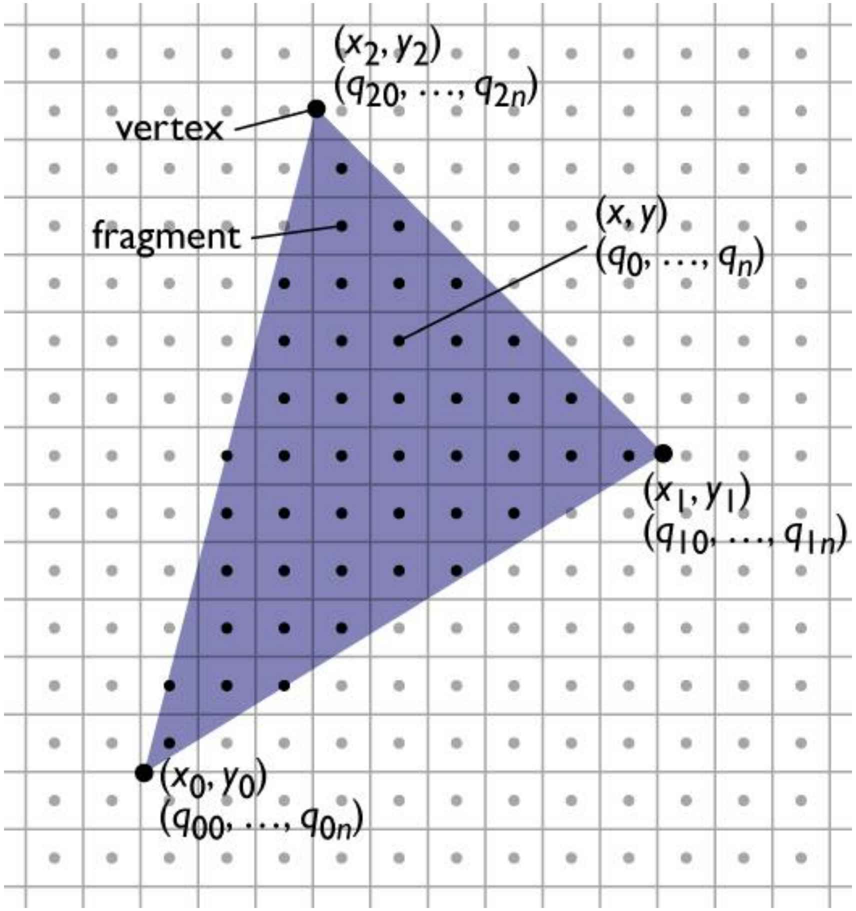
$$x_{min} = \text{floor}(\min(x_a, x_b, x_c))$$
$$x_{max} = \text{ceil}(\max(x_a, x_b, x_c))$$
$$y_{min} = \text{floor}(\min(y_a, y_b, y_c))$$
$$y_{max} = \text{ceil}(\max(y_a, y_b, y_c))$$



Triangle Rasterization

x_{min} = floor(min(x_a, x_b, x_c))
x_{max} = ceil(max(x_a, x_b, x_c))
y_{min} = floor(min(y_a, y_b, y_c))
y_{max} = ceil(max(y_a, y_b, y_c))

for y = y_{min} **to** y_{max}
    for x = x_{min} **to** x_{max}
        if (x, y) is in triangle
            mark(x, y)
        end
    end
end



Barycentric Coordinates

- A triangle is a **convex** shape
- Any point in the triangle is a **convex combination** of the triangle's vertices:

$$\mathbf{p} = \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c}$$

- Subject to:

$$\alpha + \beta + \gamma = 1$$

Barycentric Coordinates

- α, β, γ are called the **barycentric** coordinates of $\mathbf{p}$
- $\mathbf{p}$ lies inside the triangle if $\alpha \geq 0$, $\beta \geq 0$, and $\gamma \geq 0$
- Alternatively: $\beta \geq 0$, $\gamma \geq 0$, and $\beta + \gamma \leq 1$
- Main goal: determine β, γ

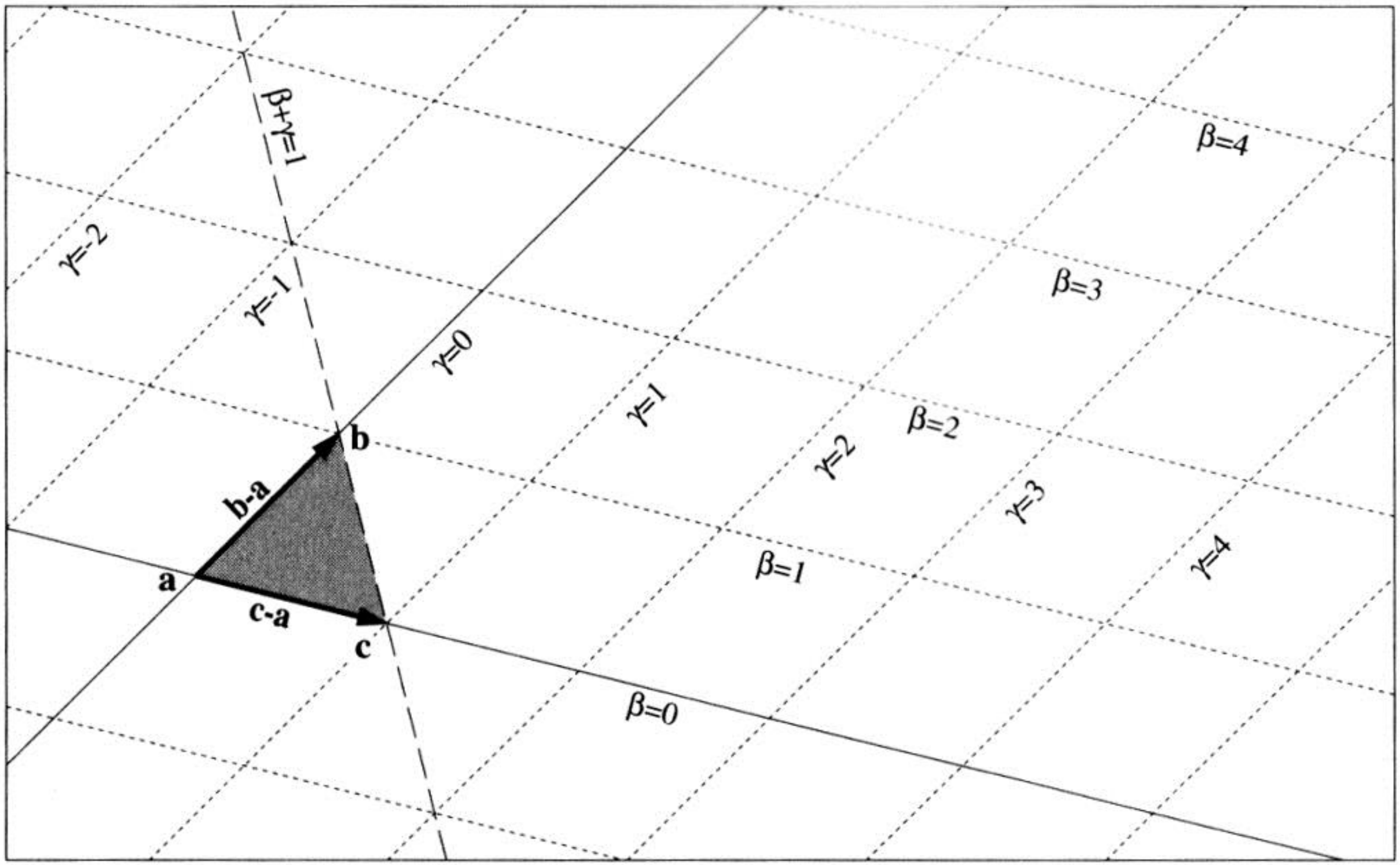
Barycentric Coordinates

- Rewriting:

$$\mathbf{p} = \mathbf{a} + \beta(\mathbf{b} - \mathbf{a}) + \gamma(\mathbf{c} - \mathbf{a})$$

- We've seen this sort of thing before!
- (β, γ) are coordinates of $\mathbf{p}$ in a different frame:
  - $\mathbf{a}$ is the origin
  - $(\mathbf{b} - \mathbf{a})$ and $(\mathbf{c} - \mathbf{a})$ are the axes

Barycentric Coordinates



Barycentric Coordinates

- Construct the “frame-to-canonical” matrix:

$$\begin{bmatrix} \mathbf{b}-\mathbf{a} & \mathbf{c}-\mathbf{a} & \mathbf{a} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \beta \\ \gamma \\ 1 \end{bmatrix} = \mathbf{F} \begin{bmatrix} \beta \\ \gamma \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- Invert to get the “canonical-to-frame” matrix:

$$\begin{bmatrix} \beta \\ \gamma \\ 1 \end{bmatrix} = \mathbf{F}^{-1} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{b}-\mathbf{a} & \mathbf{c}-\mathbf{a} & \mathbf{a} \\ 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Barycentric Coordinates

- The final answer is:

$$\beta = \frac{(y_a - y_c)x + (x_c - x_a)y + x_a y_c - x_c y_a}{(y_a - y_c)x_b + (x_c - x_a)y_b + x_a y_c - x_c y_a}$$

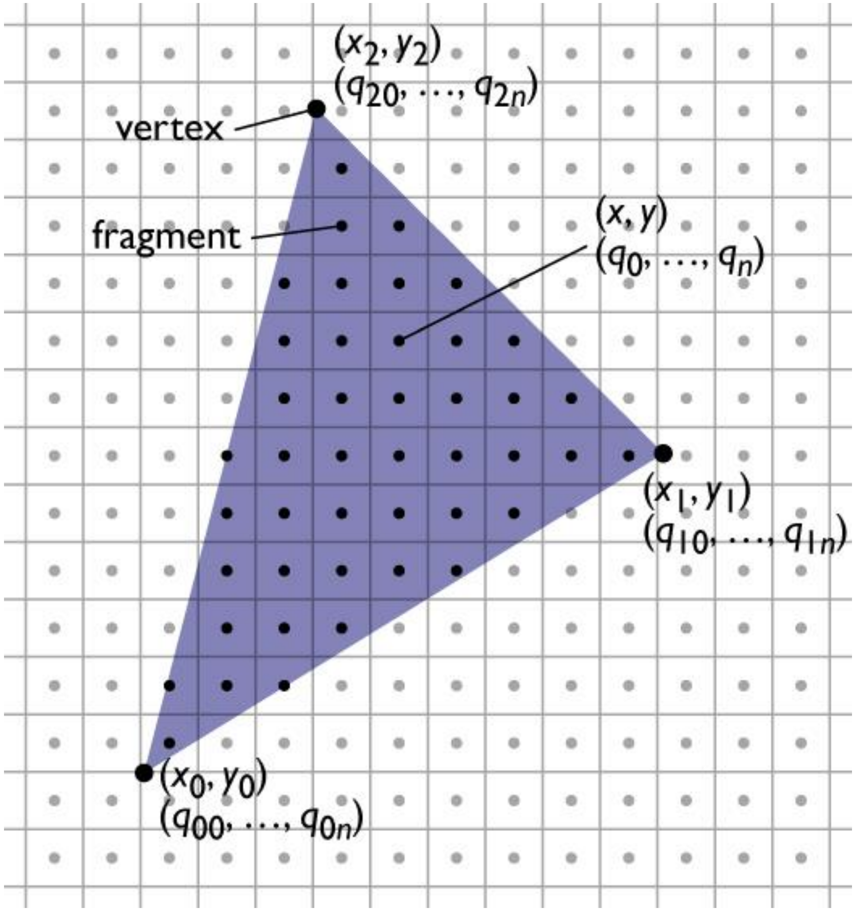
$$\gamma = \frac{(y_a - y_b)x + (x_b - x_a)y + x_a y_b - x_b y_a}{(y_a - y_b)x_c + (x_b - x_a)y_c + x_a y_b - x_b y_a}$$

$$\alpha = 1 - \beta - \gamma$$

Triangle Rasterization

$x_{min} = \text{floor}(\min(x_a, x_b, x_c))$
$x_{max} = \text{ceil}(\max(x_a, x_b, x_c))$
$y_{min} = \text{floor}(\min(y_a, y_b, y_c))$
$y_{max} = \text{ceil}(\max(y_a, y_b, y_c))$

for $y = y_{min}$ **to** y_{max}
    for $x = x_{min}$ **to** x_{max}
        compute β, γ given x, y
        if $\beta > 0$ **and** $\gamma > 0$ **and** $\beta+\gamma < 1$
            mark(x, y)
        end
    end
end



Optimized Triangle Rasterization

- β, γ are linear functions of x, y:

$$\beta = \frac{(y_a - y_c)x + (x_c - x_a)y + x_a y_c - x_c y_a}{(y_a - y_c)x_b + (x_c - x_a)y_b + x_a y_c - x_c y_a}$$

$$\gamma = \frac{(y_a - y_b)x + (x_b - x_a)y + x_a y_b - x_b y_a}{(y_a - y_b)x_c + (x_b - x_a)y_c + x_a y_b - x_b y_a}$$

$$\alpha = 1 - \beta - \gamma$$

- Rewriting:

$$\beta(x, y) = \beta_0 + \beta_x x + \beta_y y$$
$$\gamma(x, y) = \gamma_0 + \gamma_x x + \gamma_y y$$

Optimized Triangle Rasterization

- This gives the following recurrences:

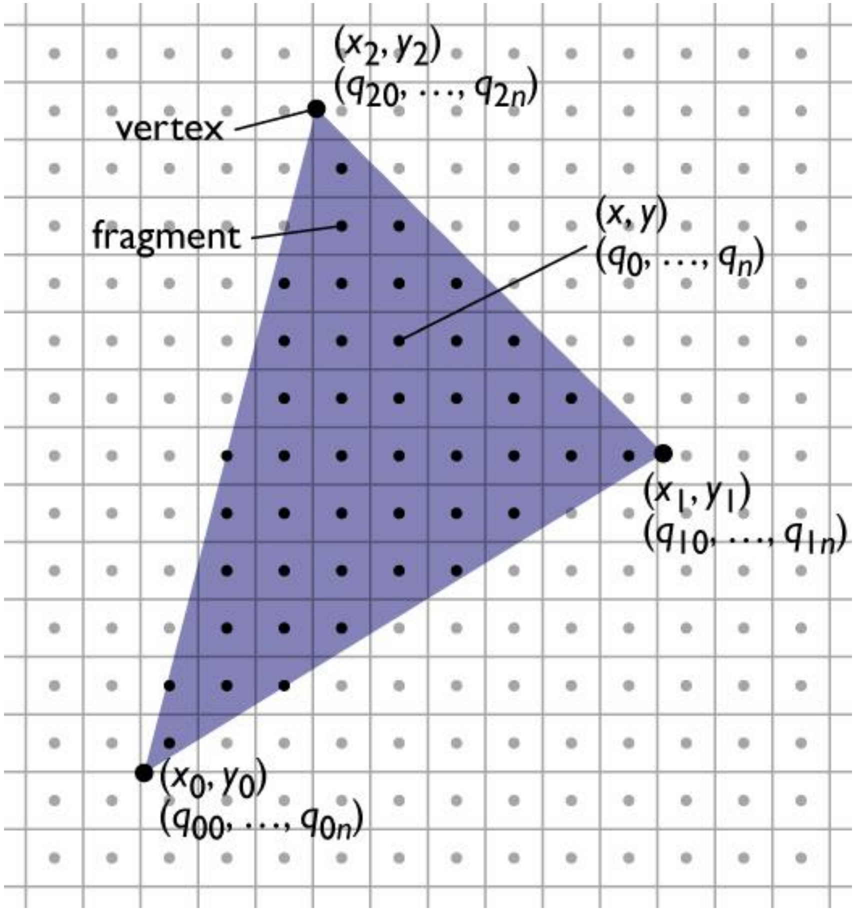
$$\beta(x+1,y)-\beta(x,y)=\beta_x$$
$$\beta(x,y+1)-\beta(x,y)=\beta_y$$
$$\gamma(x+1,y)-\gamma(x,y)=\gamma_x$$
$$\gamma(x,y+1)-\gamma(x,y)=\gamma_y$$

- So now we can write an incremental algorithm!

Optimized Triangle Rasterization

```
compute x_min, x_max, y_min, y_max
compute β, γ given x_min, y_min
n = (x_max - x_min)  + 1

for y = y_min to y_max
    for x = x_min to x_max
        if β > 0 and γ > 0 and β+γ < 1
            mark(x, y)
        end
        β += β_x
        γ += γ_x
    end
    β += β_y - n*β_x
    γ += γ_y - n*γ_x
end
```



Outline

- Overview
- Line Rasterization
- Line Attributes
- Triangle Rasterization
- Triangle Attributes

Attribute Interpolation

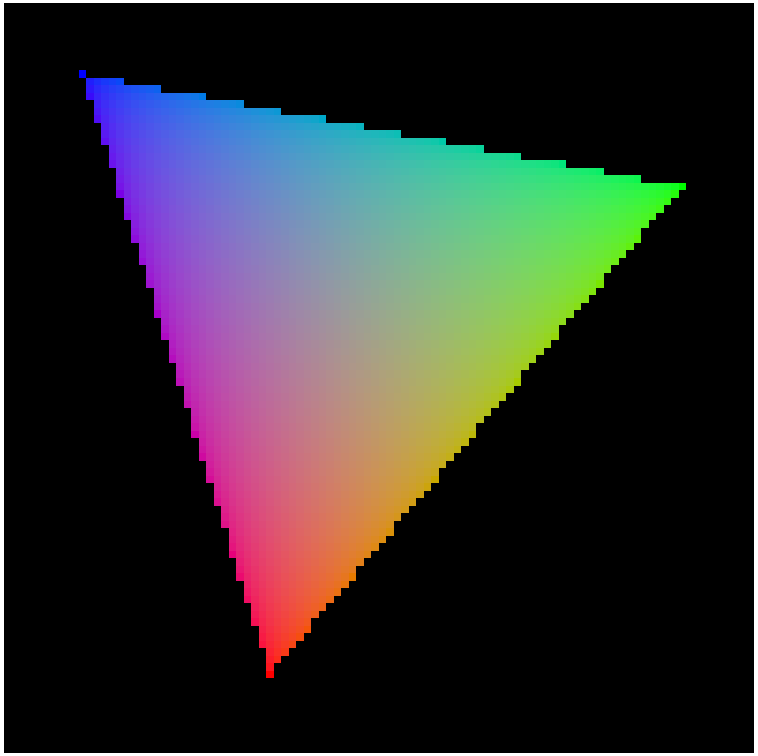
- Given attributes A_a, A_b, A_c at vertices $\mathbf{a}, \mathbf{b}, \mathbf{c}$
- Attribute at $\mathbf{p} = (x, y)$:

$$A(x, y) = A_a + \beta(A_b - A_a) + \gamma(A_c - A_a)$$

- β, γ are just the barycentric coordinates

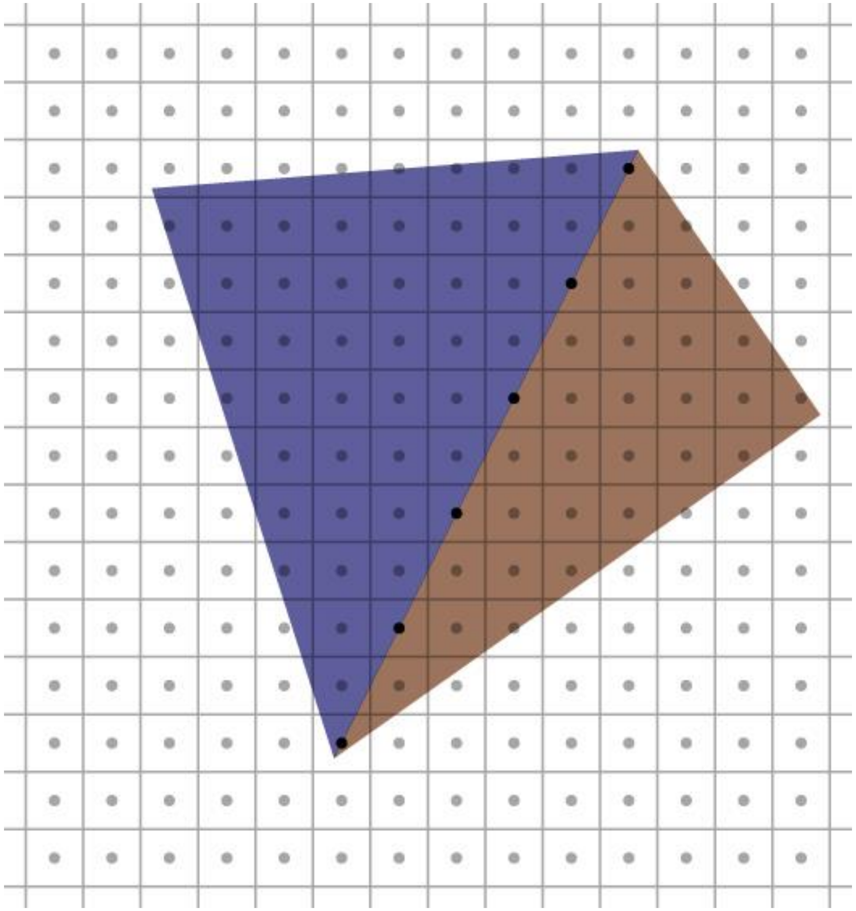
Attribute Interpolation

- Easy to incorporate into incremental algorithm
- Also called **Gouraud interpolation**
- Just one way of doing interpolation



Shared Edges

- Some pixels may lie exactly on an edge shared by two triangles
- What color to assign them?
- More than one way to do this

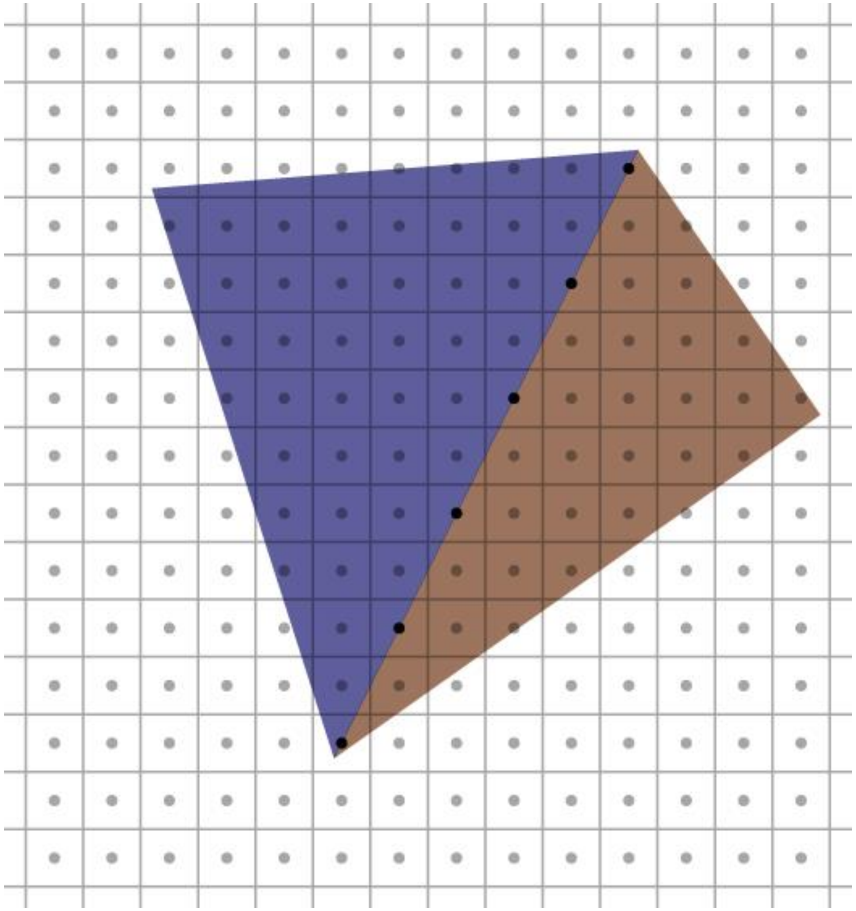


Shared Edges

- Pixel (x, y) is on an edge if:

  $\beta = 0$ or $\gamma = 0$ or
  $\beta + \gamma = 1$

- Ignore pixels on an edge when on top row or last column in a row
- OpenGL/Direct3D convention



http://msdn.microsoft.com/en-us/library/windows/desktop/bb147314(v=vs.85).aspx

Outline

- Overview
- Line Rasterization
- Line Attributes
- Triangle Rasterization
- Triangle Attributes
- Clipping

Clipping

- Rasterizer assumes triangle's pixels are on-screen
  - Bad things happen if triangle crosses near plane
- After applying perspective matrix, need to clip against canonical view volume
  - Clip triangle against planes $\{x, y, z\} = \pm 1$

Clipping

- 4 cases, based on which/how many vertices are inside the clipping plane:
  - **All inside**: retain triangle as-is
  - **All outside:** don’t draw triangle
  - **One inside, two outside:** one clipped triangle
  - **Two inside, one outside:** two clipped triangles