CMP205: Computer Graphics

Lecture 2: Raster Algorithms

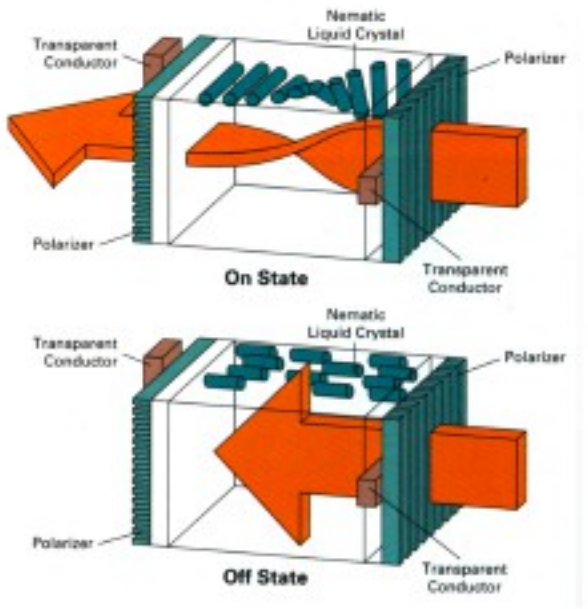
Mohamed Alaa El-Dien Aly
Computer Engineering Department
Cairo University
Fall 2012

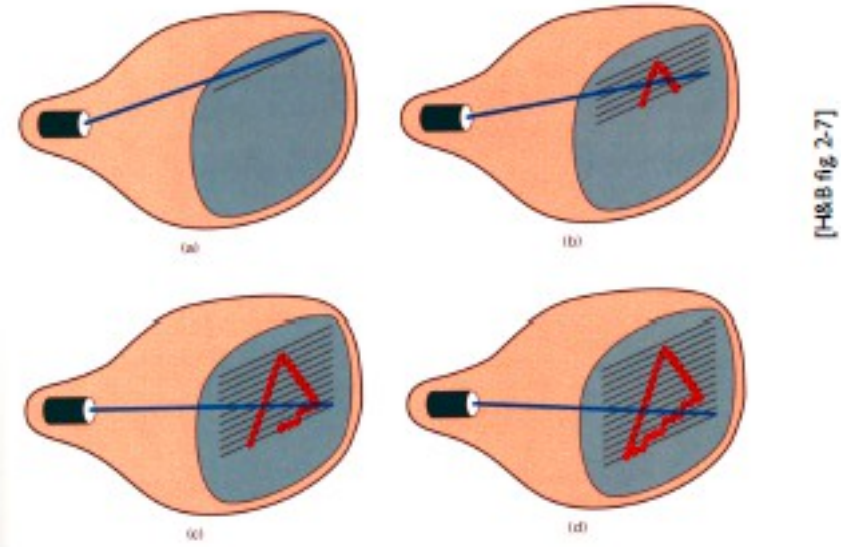
Agenda

- Raster Displays
- Gamma Correction
- RGB Color & Alpha Channel
- Line Drawing
- Triangle Rasterization
- Anti-aliasing

Raster Displays

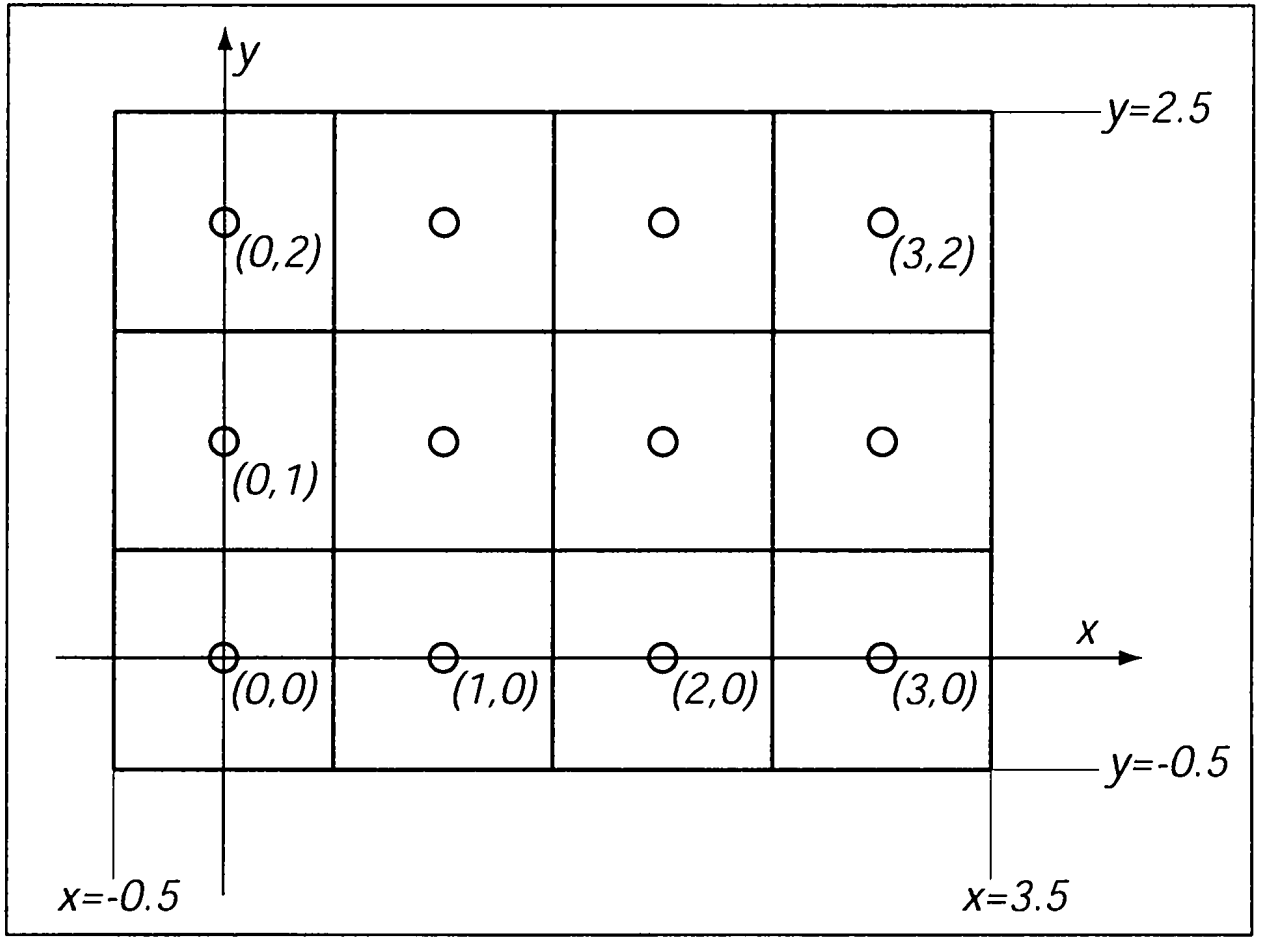
- LCD, CRT, Projector





Pixels

- Square Grid
- Resolution



Display Intensity

- Pixel values
  - 0 → black (pixel off)
  - 1 → white (pixel fully on)
  - 0.5 → halfway gray
- Quantization e.g. 8 bits
  - 0 → black
  - 255 → white

Gamma Correction

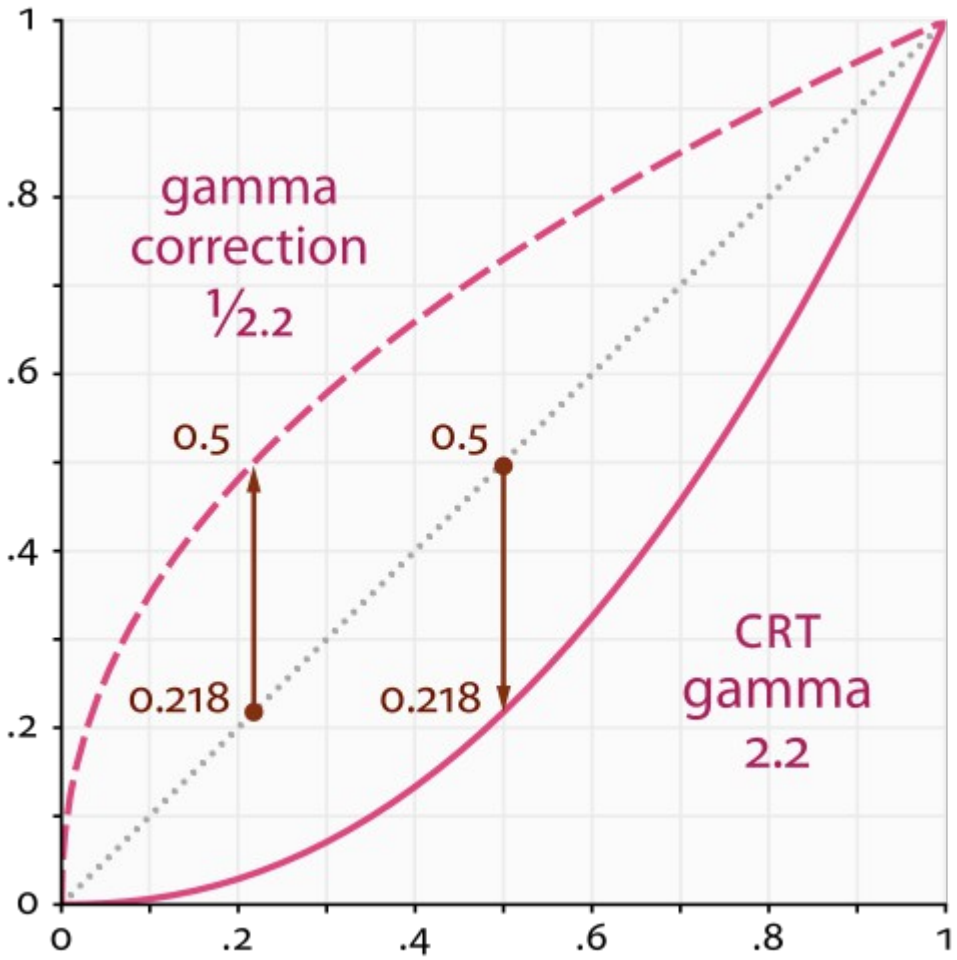
Nonlinearity

$$I_{out} = I_{max} \times I_{\text{in}}^{\gamma}$$

$$\gamma = 2.2$$
$$I_{\text{in}} = 0 \rightarrow I_{out} = 0$$
$$I_{\text{in}} = 0.5 \rightarrow I_{out} = 0.218$$
$$I_{\text{in}} = 1 \rightarrow I_{out} = 1$$

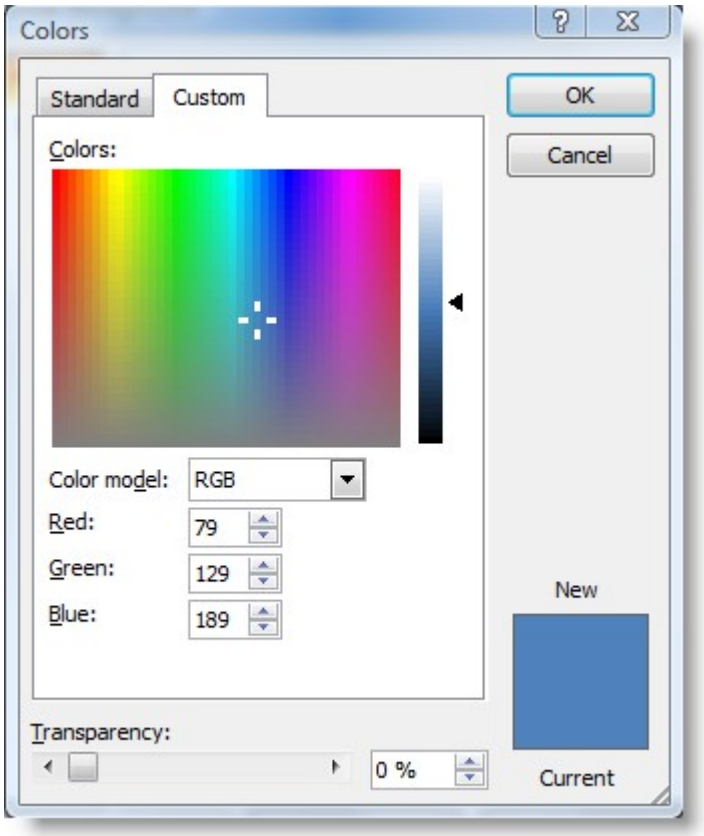
Correction

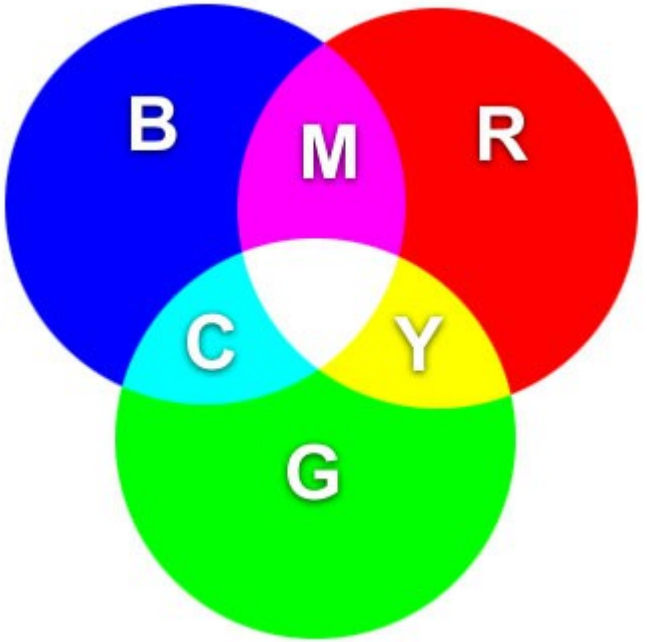
$$I_{out} = I_{max} \times \left(I_{\text{in}}^{1/\gamma} \right)^{\gamma}$$
$$= I_{max} \times I_{\text{in}}$$



RGB Color

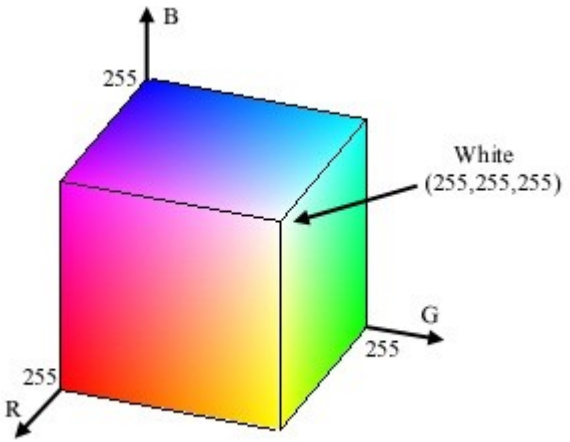
- Red, Green, Blue Color
- Additive Property





RGB Colors

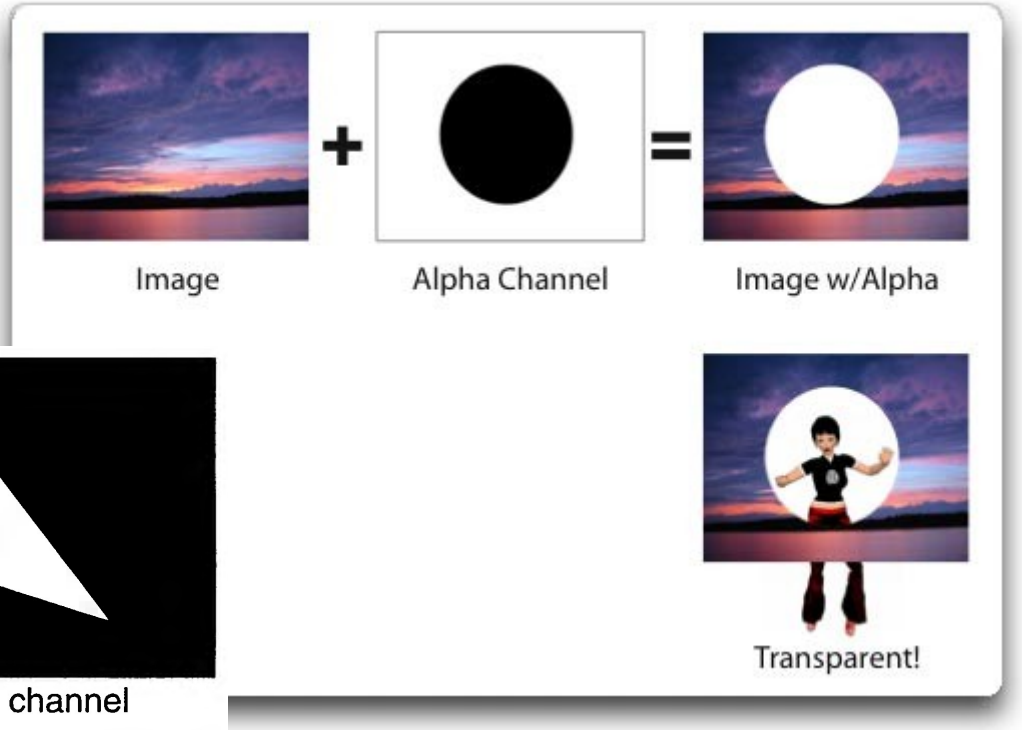
- 3 dimensional vectors (r, g, b)
  - Black = (0, 0, 0)
  - Red = (1, 0, 0)
  - …
  - White = (1, 1, 1)

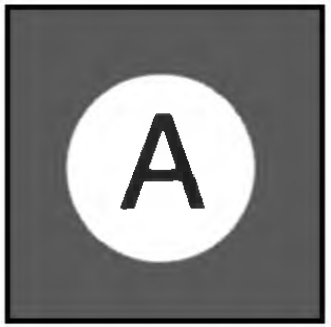


Alpha Channel

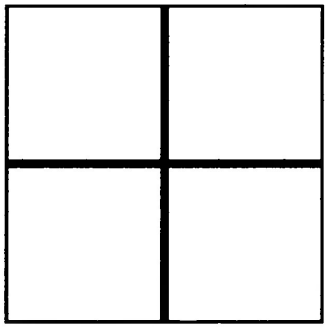
- Handles (partially) transparent objects

$$c = \alpha c_f + (1-\alpha) c_b$$





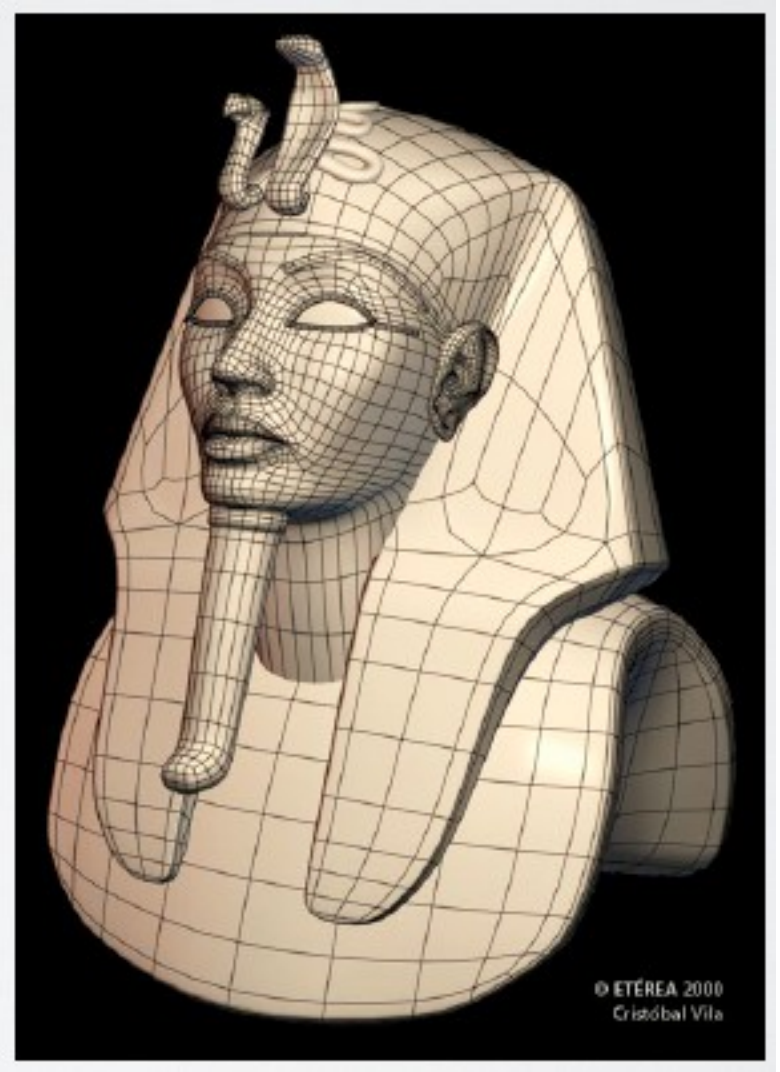
background RGB


foreground RGB

α channel

Line Drawing

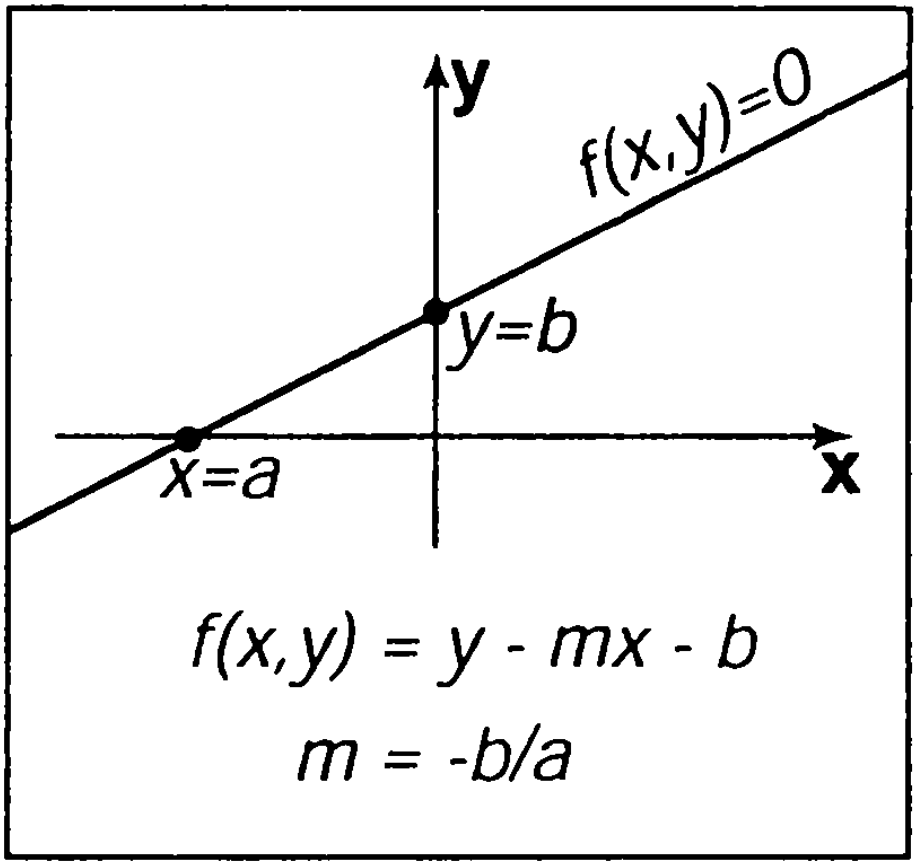
- Basic Operation



Line Representation

- Slope-Intercept
- Problem?
  - Vertical lines!
- Solution?

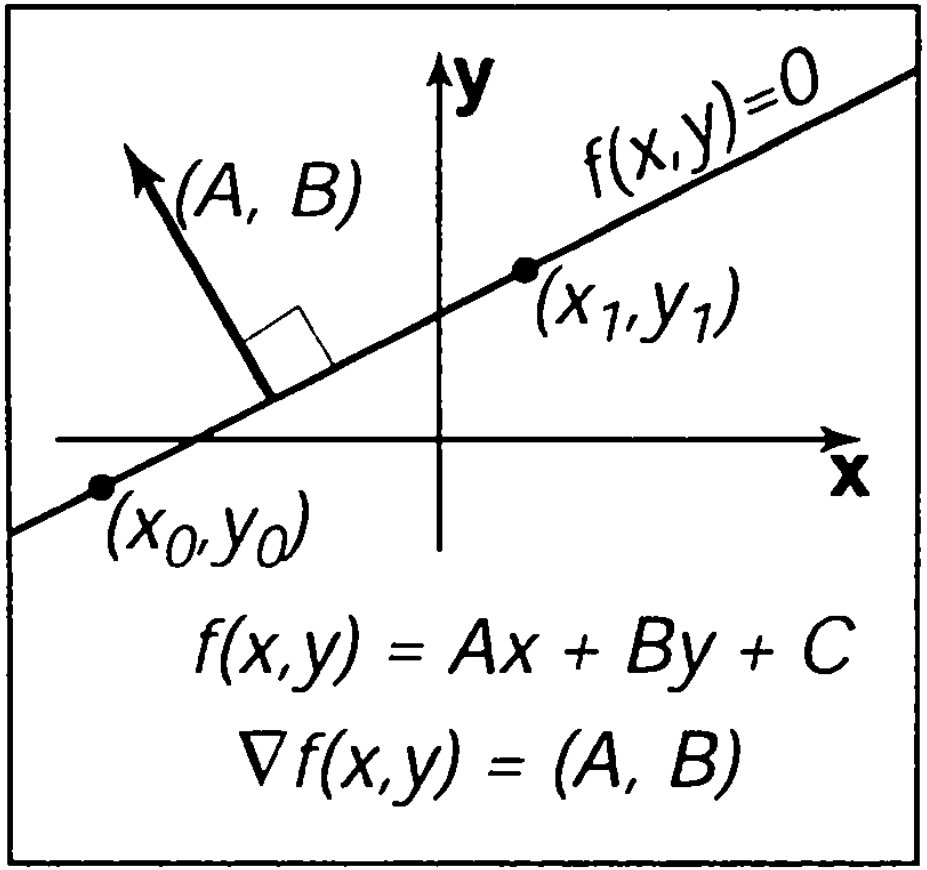
$$y = mx + b$$



Line Representation

- Implicit Form
- Derivation

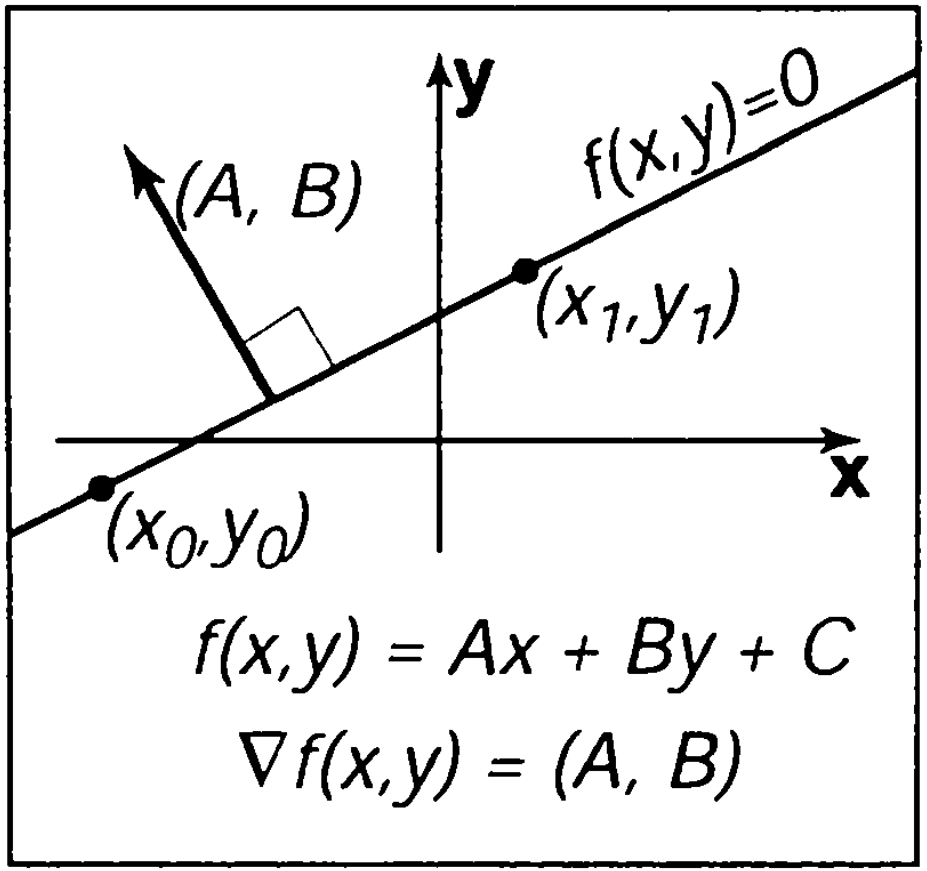
$$f(x, y) = Ax + By + C = 0$$



$$A = y_0 - y_1$$
$$B = x_1 - x_0$$
$$C = x_0 y_1 - x_1 y_0$$

Line Representation

- Implicit Form $f(x, y) = Ax + By + C = 0$
- Can always go back to slope-intercept form



$$A = y_0 - y_1$$
$$B = x_1 - x_0$$
$$C = x_0 y_1 - x_1 y_0$$

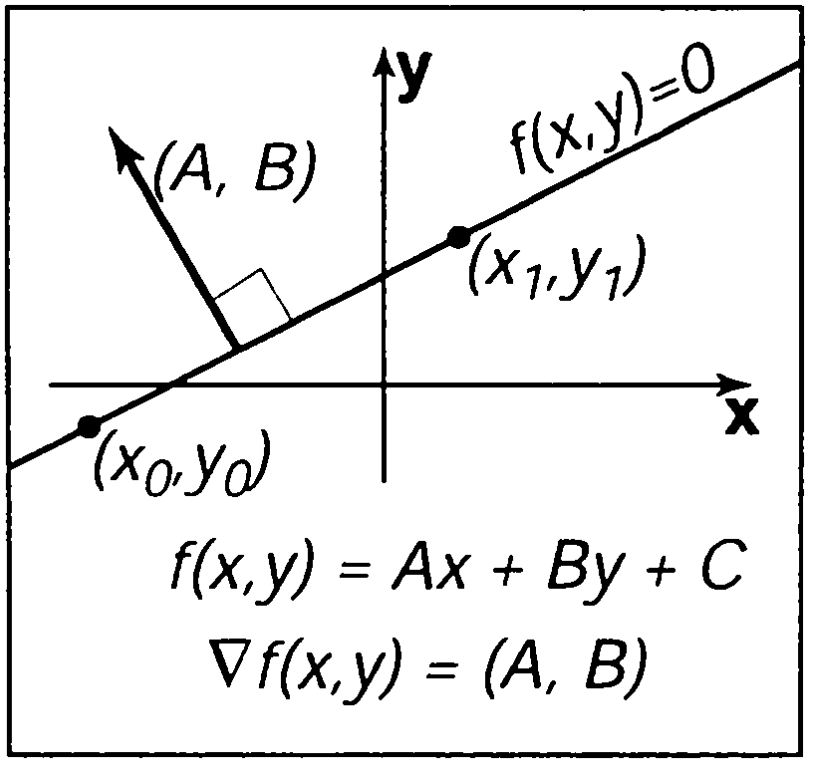
Line Representation

$$f(x, y) = Ax + By + C = 0$$

$f(x, y) = 0 \rightarrow$ On the line

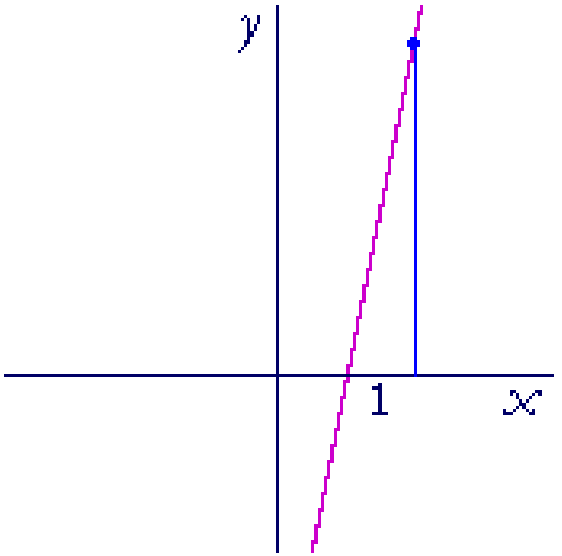
$f(x, y) > 0 \rightarrow$ Above the line

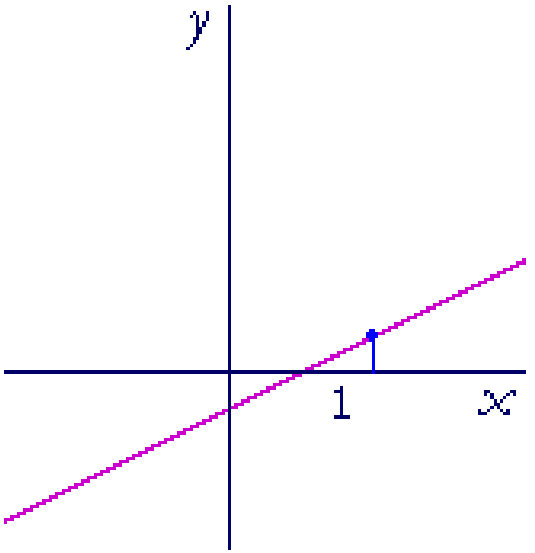
$f(x, y) < 0 \rightarrow$ Below the line



Midpoint Algorithm

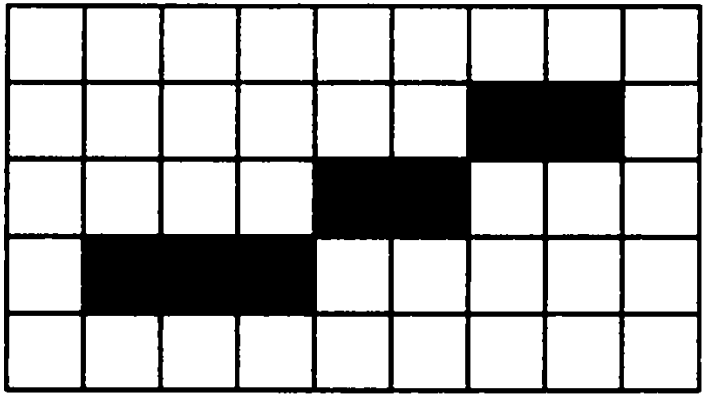
- Line Drawing Algorithm
- Uses the Implicit Equation
- Similar to Bresenham's Algorithm
- Integer end-points
- Four Cases:
  - $0 < m \leqslant 1$
  - $1 < m < \infty$
  - $-1 < m \leqslant 0$
  - $-\infty < m \leqslant -1$



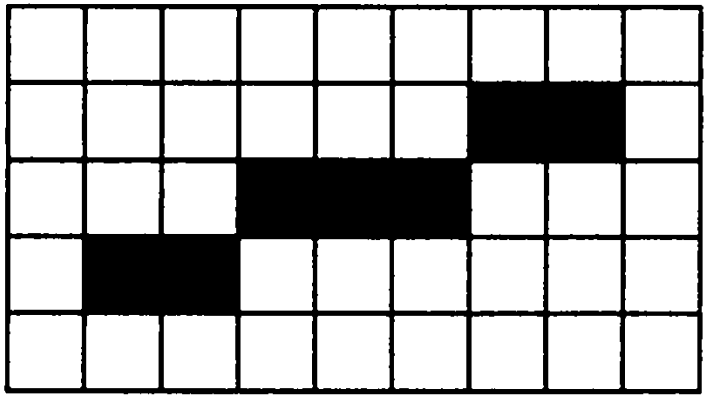


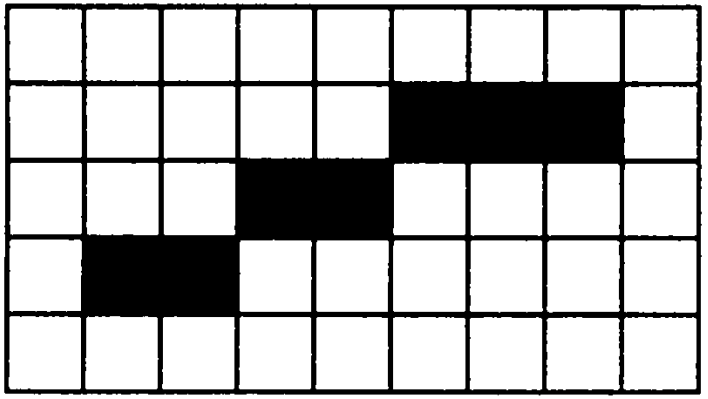
Midpoint Algorithm

- First case:
  - more “run” than “rise”
  - Assume moving right



- Which one is better?

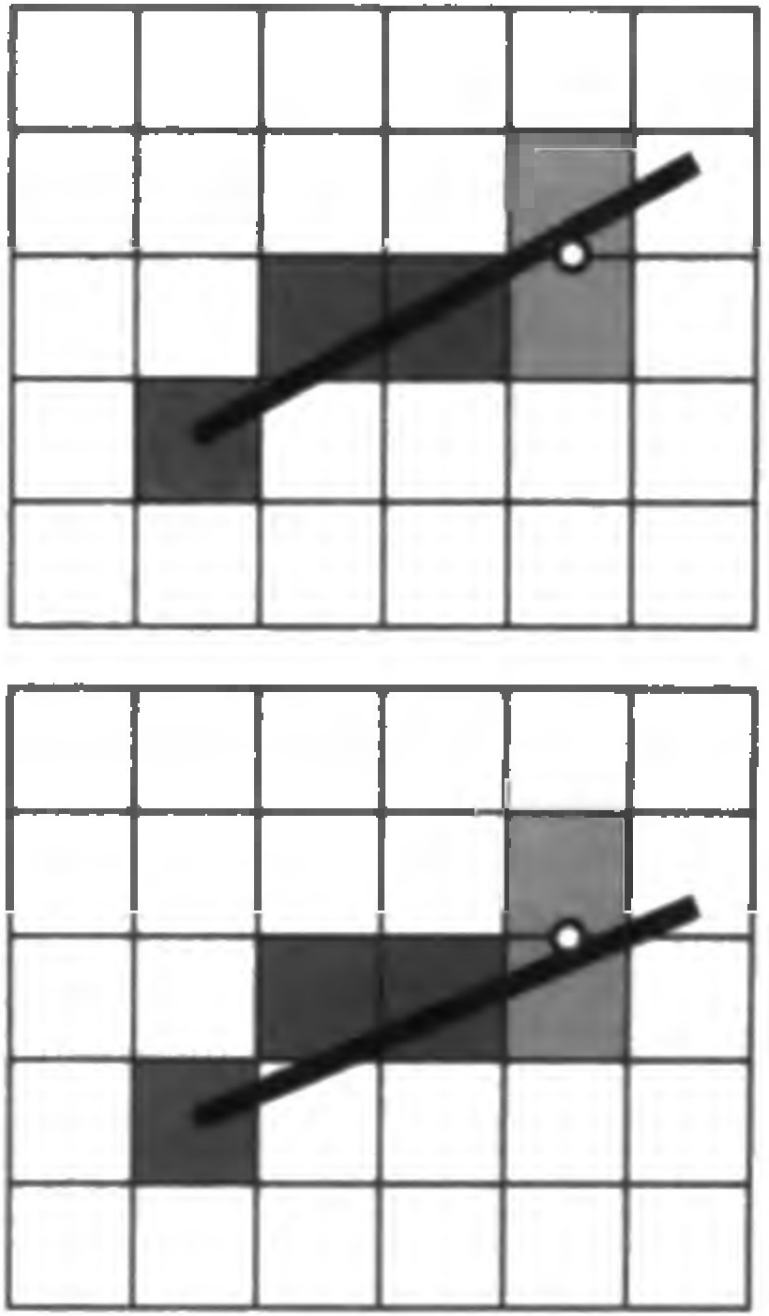




Midpoint Algorithm

- First version

$$y = y_0$$
$$\text{for } x = x_0 : x_1$$
$$\quad draw(x, y)$$
$$\quad \text{if } f(x+1, y+0.5) < 0$$
$$\quad\quad y = y+1$$



- Optimizations?

Midpoint Algorithm

- Optimizations
  - Incremental Calculations
  - Integer Operations

$$
\begin{aligned}
&y = y_0 \\
&\text{for } x = x_0 : x_1 \\
&\quad draw(x, y) \\
&\quad \text{if } f(x+1, y+0.5) < 0 \\
&\quad\quad y = y+1
\end{aligned}
$$

Midpoint Algorithm

- Incremental Calculations
  - Note that:

$$f(x, y) = (y_0 - y_1)x + (x_1 - x_0)y + C = 0$$

  - Which implies that:

$$f(x+1, y) = f(x, y) + (y_0 - y_1)$$
$$f(x+1, y+1) = f(x, y) + (y_0 - y_1) + (x_1 - x_0)$$

Midpoint Algorithm

- Second Pass

$$
\begin{aligned}
&y = y_0 \\
&d = f\left(x_0 + 1, y_0 + 0.5\right) \\
&\text{for } x = x_0 : x_1 \\
&\quad draw\left(x, y\right) \\
&\quad \text{if } d < 0 \\
&\quad\quad y = y + 1 \\
&\quad\quad d = d + \left(x_1 - x_0\right) + \left(y_0 - y_1\right) \\
&\quad \text{else} \\
&\quad\quad d = d + \left(y_0 - y_1\right)
\end{aligned}
$$

Midpoint Algorithm

- Integer Operations
  - Note that:

$$f(x, y) = 2\mathrm{f}(x, y) = 0$$

- Which implies:

$$d = 2\mathrm{f}(x_0 + 1, y_0 + 0.5)$$

$$d = 2(y_0 - y_1)(x_0 + 1) + (x_1 - x_0)(2\mathrm{y}_0 + 1) + 2\mathrm{x}_0 y_1 - 2 x_1 y_0$$

$\downarrow$

Midpoint Algorithm

- Third Pass

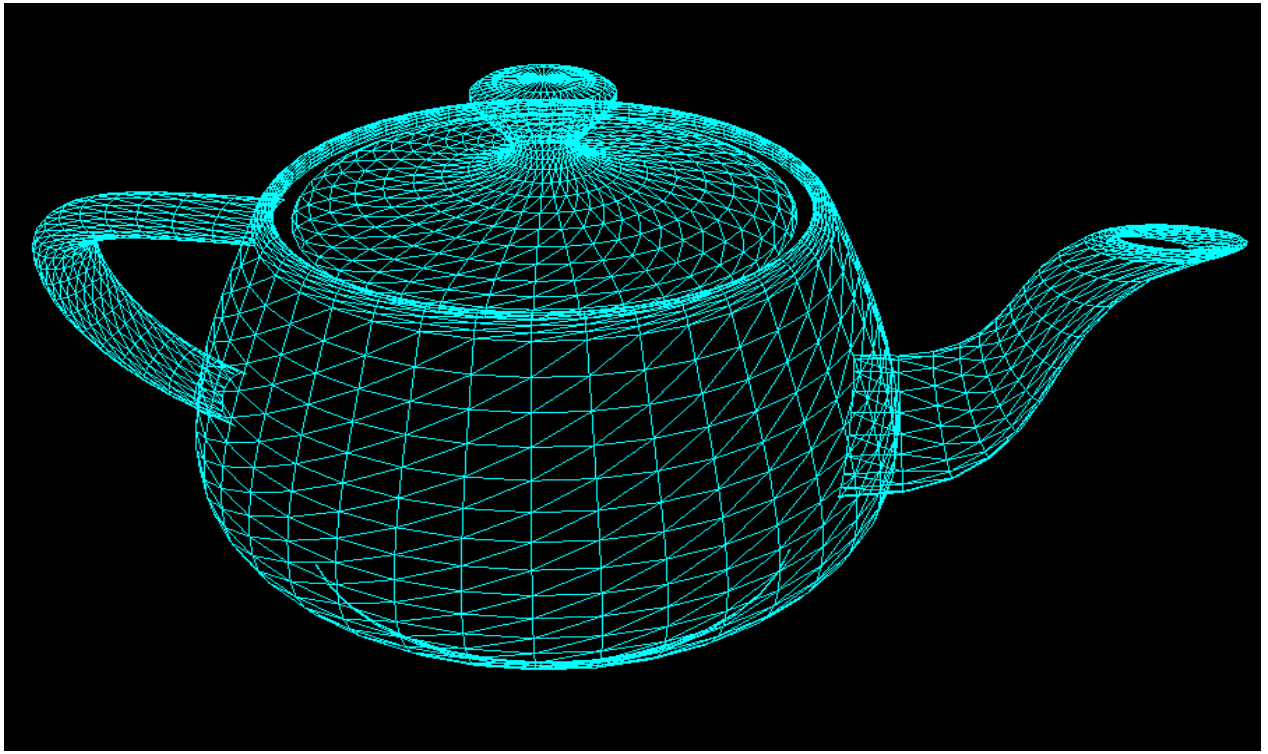
$$y = y_0$$
$$d = 2(y_0 - y_1)(x_0 + 1) + (x_1 - x_0)(2y_0 + 1) + 2x_0 y_1 - 2x_1 y_0$$
for $x = x_0 : x_1$
  $draw(x, y)$
  if $d < 0$
    $y = y + 1$
    $d = d + 2(x_1 - x_0) + 2(y_0 - y_1)$
  else
    $d = d + 2(y_0 - y_1)$

Midpoint Algorithm

- Homework
  - Other three cases
  - Read about Bresenham's Algorithm

Triangle Rasterization

- Primitive shape
- Used for modeling and shading surfaces
- Assign properties to vertices and interpolate



Barycentric Coordinates

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

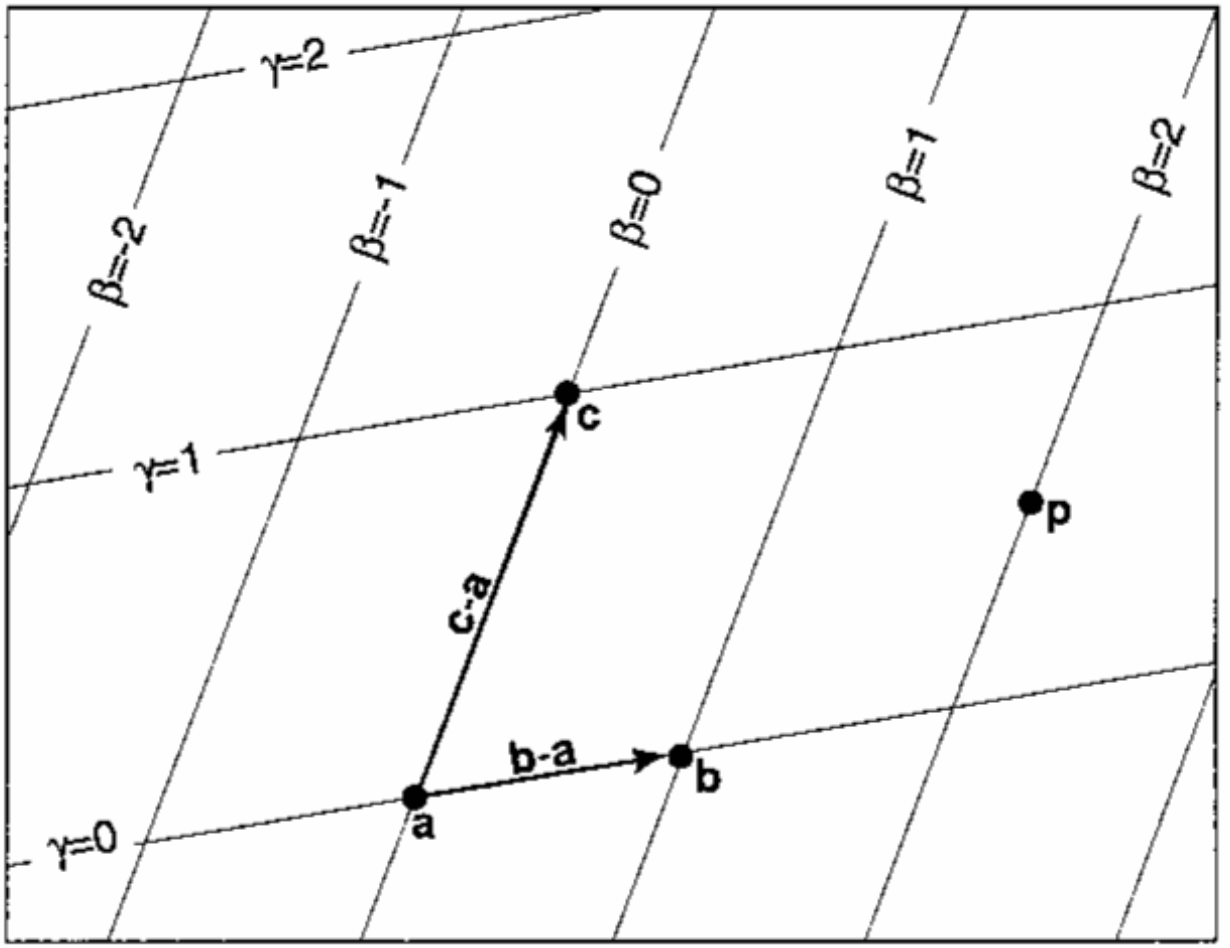
$$p = (1-\beta-\gamma)a + \beta b + \gamma c$$

$$p(\alpha,\beta,\gamma) = \alpha a + \beta b + \gamma c$$

$$s.t. \quad \alpha+\beta+\gamma = 1$$

Color Interpolation:

$$c_p = \alpha c_a + \beta c_b + \gamma c_c$$



Barycentric Coordinates

Inside Triangle iff:

$0 < \alpha < 1$

$0 < \beta < 1$

$0 < \gamma < 1$

On edge ab if:

$0 \leqslant \alpha \leqslant 1$

$0 \leqslant \beta \leqslant 1$

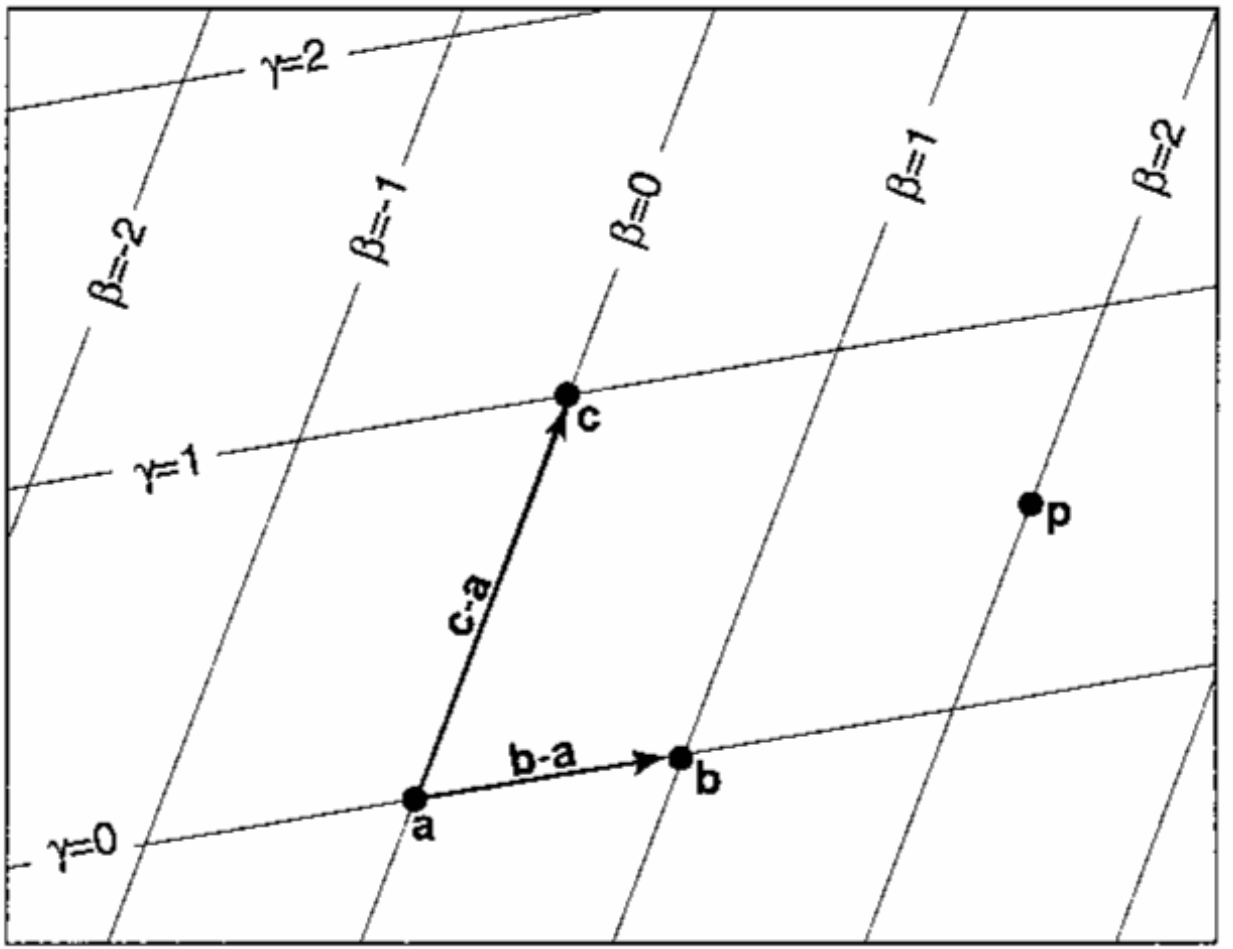
$\gamma = 0$

On vertex a if:

$\alpha = 1$

$\beta = 0$

$\gamma = 0$



$$p(\alpha, \beta, \gamma) = \alpha a + \beta b + \gamma c$$

$$s.t. \quad \alpha + \beta + \gamma = 1$$

Computing Barycentric Coordinates

- Algebraic Solution
  - 2 equations in 2 unknowns

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

$$\begin{bmatrix} x_b - x_a & x_c - x_a \\ y_b - y_a & y_c - y_a \end{bmatrix} \begin{bmatrix} \beta \\ \gamma \end{bmatrix} = \begin{bmatrix} x_p - x_a \\ y_p - y_a \end{bmatrix}$$

Computing Barycentric Coordinates

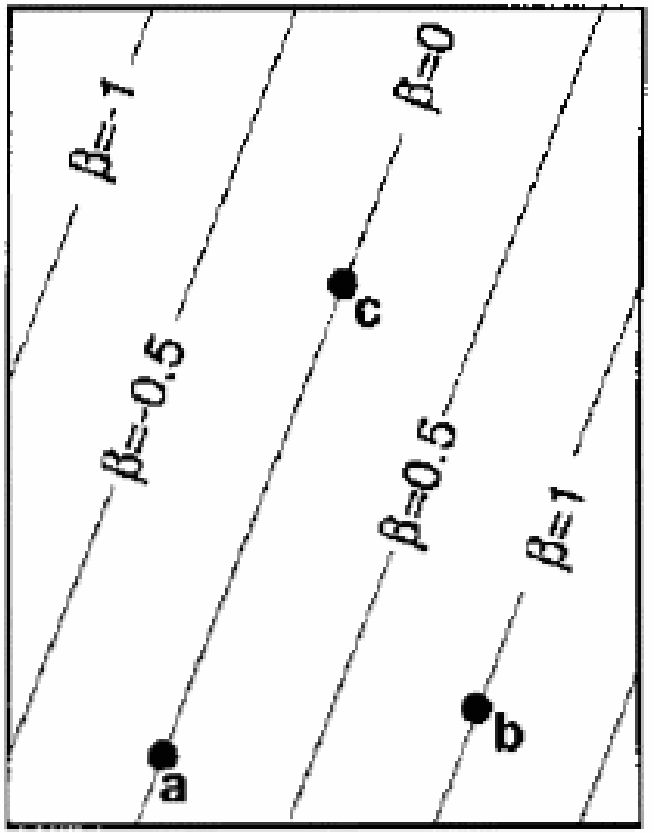
- Geometric Solution

Implicit Function: $\beta = f_{ac}(x, y)$

want:
$\beta = 0 \rightarrow$ on line ac
$\beta = 1 \rightarrow$ at point b

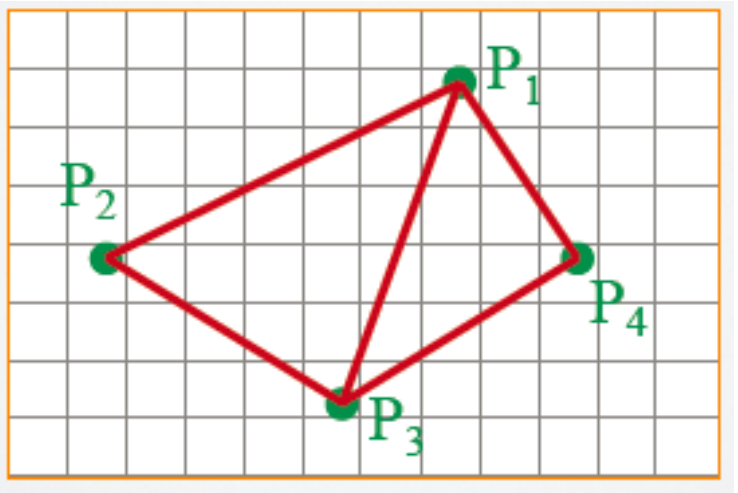
$$\beta = \frac{f_{ac}(x, y)}{f_{ac}(x_b, y_b)}$$

$$f_{ac}(x, y) = (y_a - y_c)x + (x_c - x_a)y + x_a y_c - x_c y_a$$



Triangle Rasterization

- Fast
- Accurate
- No gaps
- No order dependency
- Approach
  - Compute Barycentric Coordinates
  - If inside triangle
    - Interpolate color
    - Draw
- Edges? Later



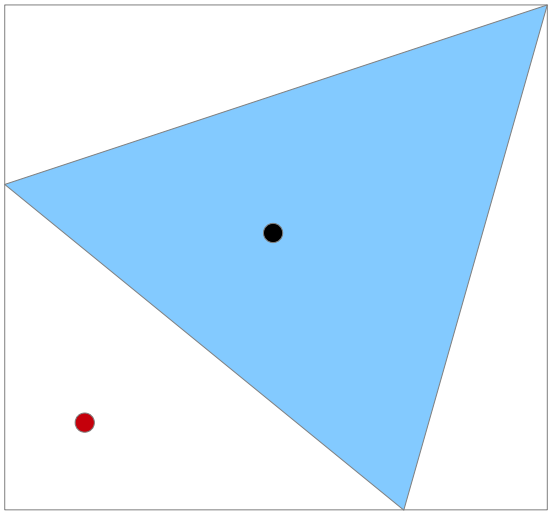
Triangle Rasterization

```
for all x do
  for all y do
    compute (alpha, beta, gamma) for (x,y)

    // Inside?
    if (alpha in (0,1) AND
        beta in (0,1) AND
        gamma in (0,1)) then
      c = alpha*c0 + beta*c1 + gamma*c2
      drawpixel(x,y) with color c
```

Optimizations?

Triangle Rasterization

Bounding Rectangles

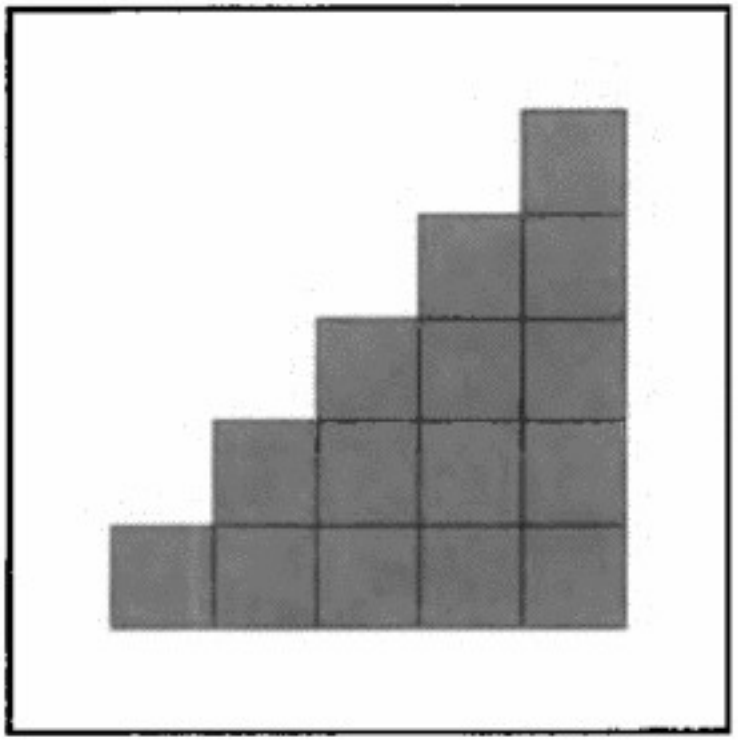


Triangle Rasterization

```
// Bounding rectangle
(xmin, xmax) = minmax(x0, x1, x2)
(ymin, ymax) = minmax(y0, y1, y2)

for x=xmin:xmax do
  for y=ymin:ymax do
    alpha = f12(x,y) / f12(x0,y0)
    beta = f20(x,y) / f20(x1,y1)
    gamma = f01(x,y) / f01(x2,y2)

    // Inside?
    if (alpha>0 AND beta>0 AND gamma>0)
      c = alpha*c0 + beta*c1 + gamma*c2
      drawpixel(x,y) with color c
```

More Optimizations?

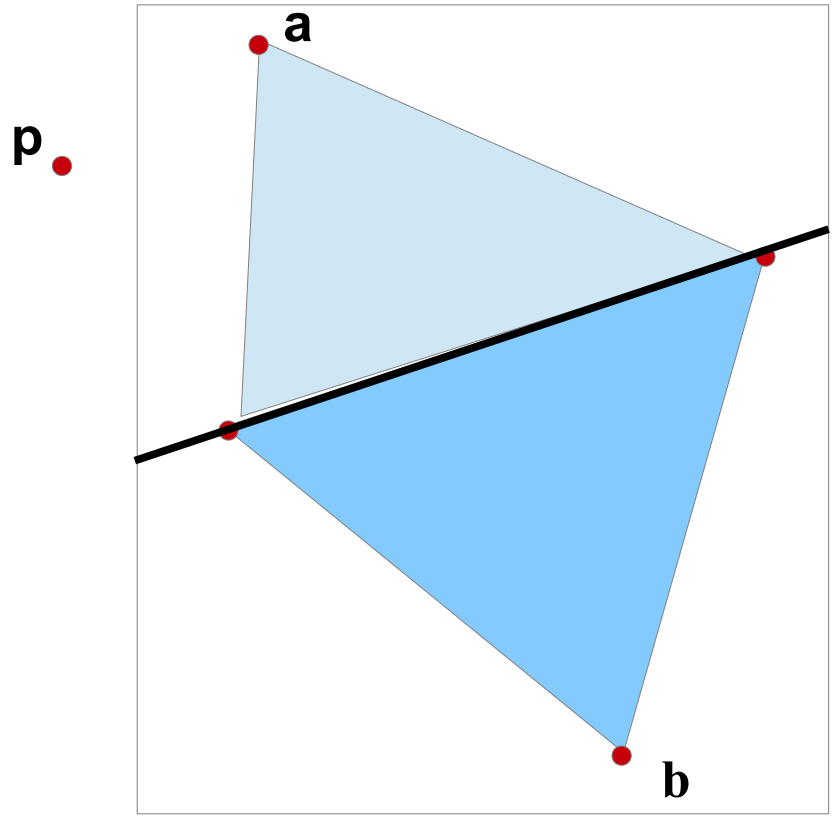
Triangle Rasterization



					0.00 1.00 0.00	
				0.25 0.75 0.00	0.25 1.00 0.00	
			0.50 0.50 0.00	0.50 0.75 0.00	0.50 1.00 0.00	
		0.75 0.25 0.00	0.75 0.50 0.00	0.75 0.75 0.00	0.75 1.00 0.00	
	1.00 0.00 0.00	1.00 0.25 0.00	1.00 0.50 0.00	1.00 0.75 0.00	1.00 1.00 0.00	

Triangle Rasterization

Points on adjacent edges

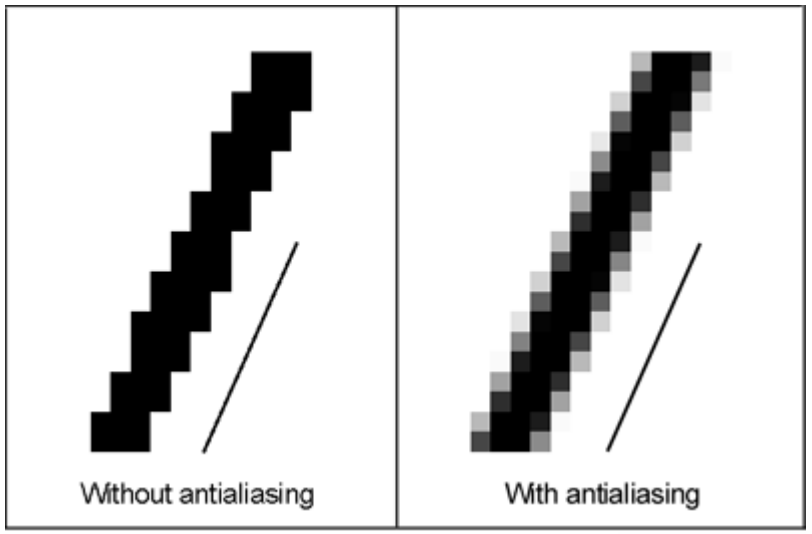


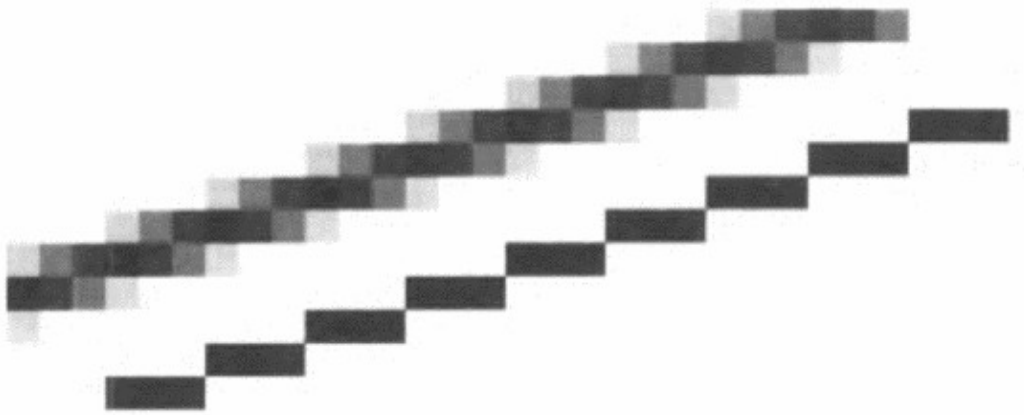
$$f(a) \times f(p) > 0 ?$$

Triangle Rasterization

```
// Bounding rectangle
(xmin, xmax) = minmax(x0, x1, x2)
(ymin, ymax) = minmax(y0, y1, y2)

f_alpha = f12(x0,y0)
f_beta = f20(x1,y1)
f_gamma = f01(x2,y2)

for x=xmin:xmax do
  for y=ymin:ymax do
    alpha = f12(x,y) / f_alpha
    beta = f20(x,y) / f_beta
    gamma = f01(x,y) / f_gamma

    // Inside or on edge?
    if (alpha>=0 AND beta>=0 AND gamma>=0) then
      if (alpha>0 OR f_alpha * f12(-1,-1) > 0) AND
         (beta>0 OR f_beta * f20(-1,-1) > 0) AND
         (gamma>0 OR f_gamma * f01(-1,-1) > 0)) then
        c = alpha*c0 + beta*c1 + gamma*c2
        drawpixel(x,y) with color c
```

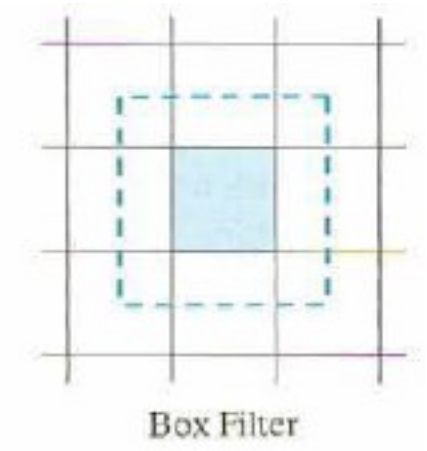
Antialiasing

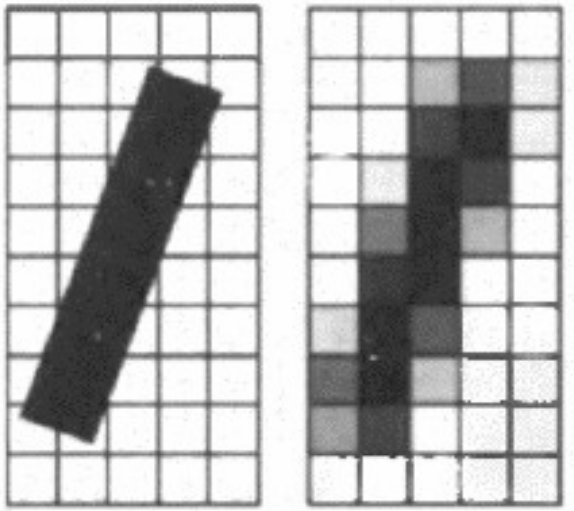




Antialiasing

- Draw line on higher resolution
- “Box filter” to subsample





Recap

- Raster Displays
- Alpha Channels
- Gamma Correction
- Line Drawing
- Triangle Rasterization
- Antialiasing

Homework #1

- C++ program to implement Midpoint Algorithm
- Due in 1 week