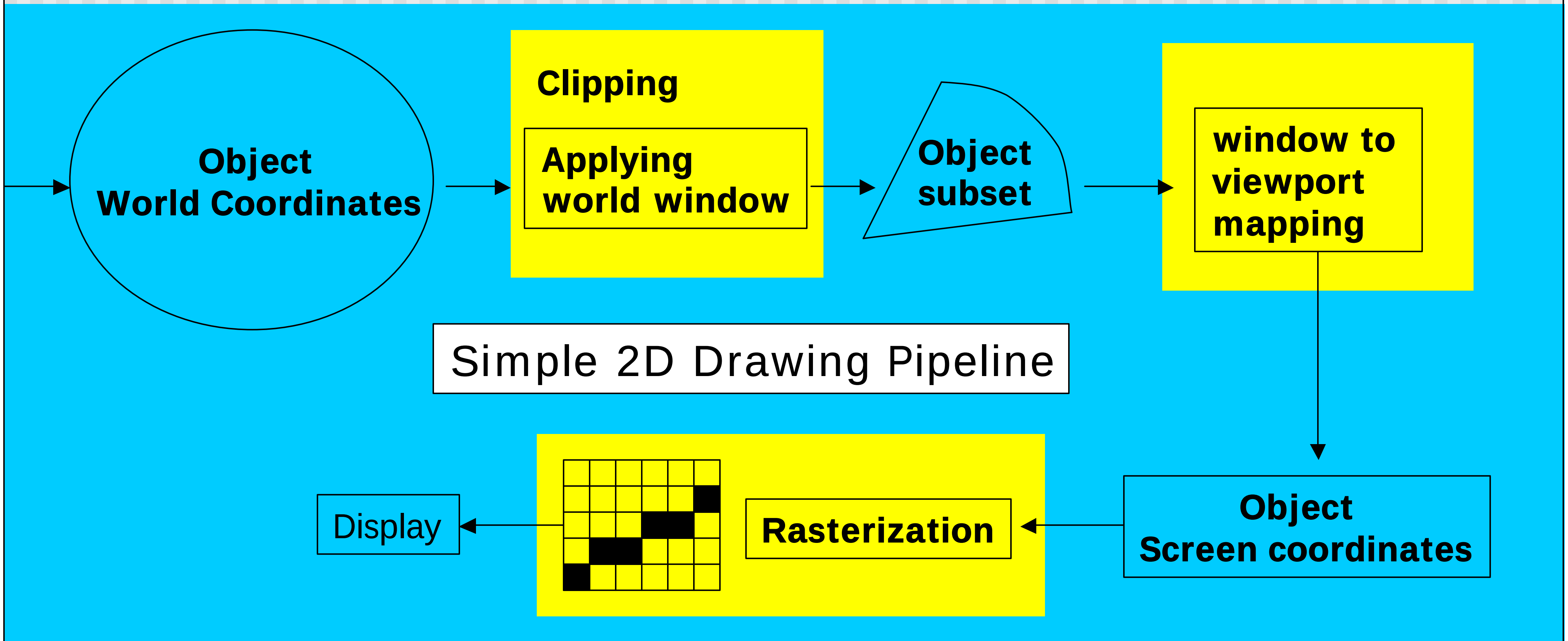
CS 543: Computer Graphics
Lecture 9 (Part I): Raster Graphics Part 1

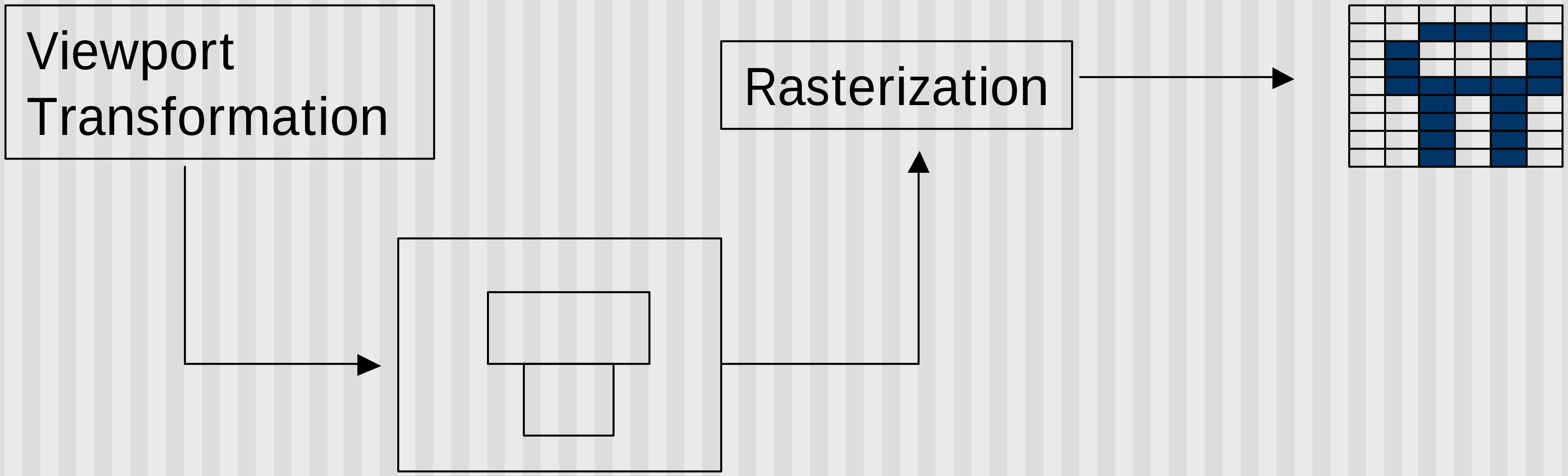
Emmanuel Agu

2D Graphics Pipeline



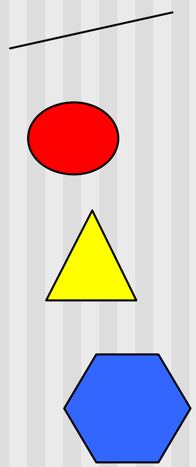
Rasterization (Scan Conversion)

- Convert high-level geometry description to pixel colors in the frame buffer
- Example: given vertex x,y coordinates determine pixel colors to draw line
- Two ways to create an image:
  - Scan existing photograph
  - Procedurally compute values (rendering)



Rasterization

- A fundamental computer graphics function
- Determine the pixels' colors, illuminations, textures, etc.
- Implemented by graphics hardware
- Rasterization algorithms
  - Lines
  - Circles
  - Triangles
  - Polygons



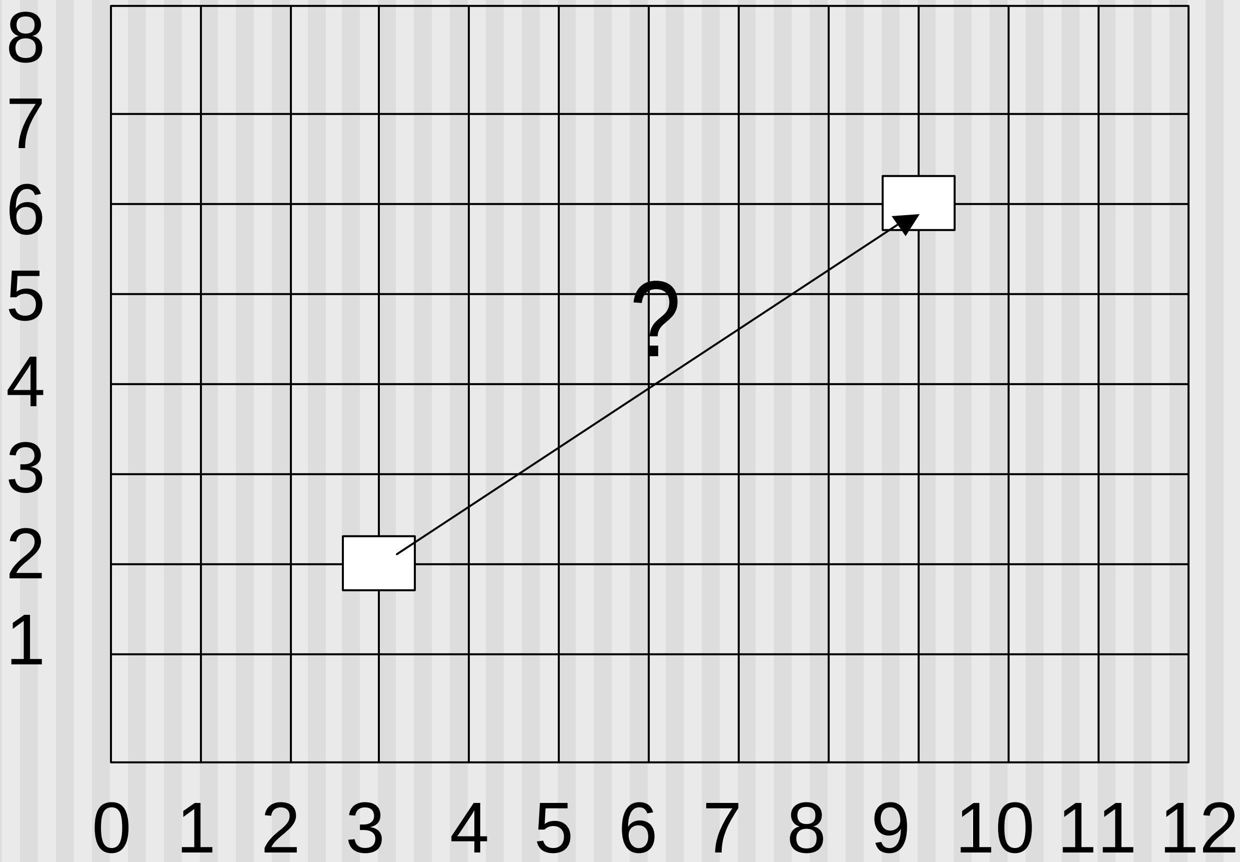
Rasterization Operations

- Drawing lines on the screen
- Manipulating pixel maps (pixmaps): copying, scaling, rotating, etc
- Compositing images, defining and modifying regions
- Drawing and filling polygons
  - Previously glBegin(GL_POLYGON), etc
- Aliasing and antialiasing methods

Line drawing algorithm

- Programmer specifies (x,y) values of end pixels
- Need algorithm to figure out which intermediate pixels are on line path
- Pixel (x,y) values constrained to integer values
- Actual computed intermediate line values may be floats
- Rounding may be required. E.g. computed point (10.48, 20.51) rounded to (10, 21)
- Rounded pixel value is off actual line path (jaggy!!)
- Sloped lines end up having jaggies
- Vertical, horizontal lines, no jaggies

Line Drawing Algorithm



Line: (3,2) -> (9,6)

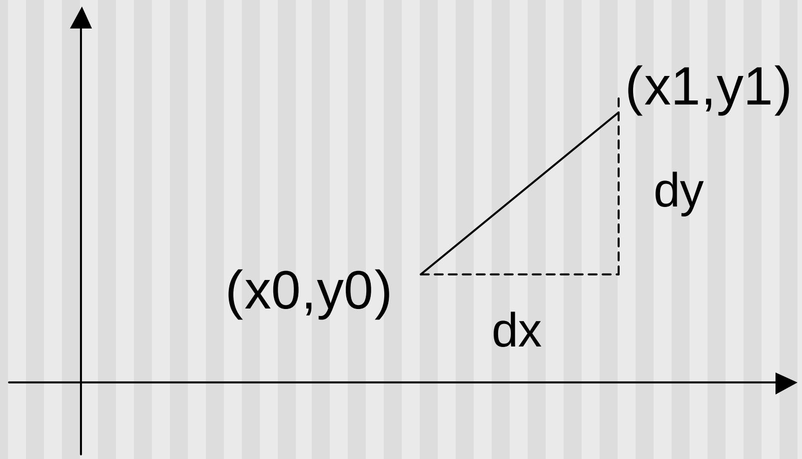
Which intermediate pixels to turn on?

Line Drawing Algorithm

- Slope-intercept line equation
  - y = mx + b
  - Given two end points (x0,y0), (x1, y1), how to compute m and b?

$$m = \frac{dy}{dx} = \frac{y1 - y0}{x1 - x0}$$

$$b = y0 - m * x0$$



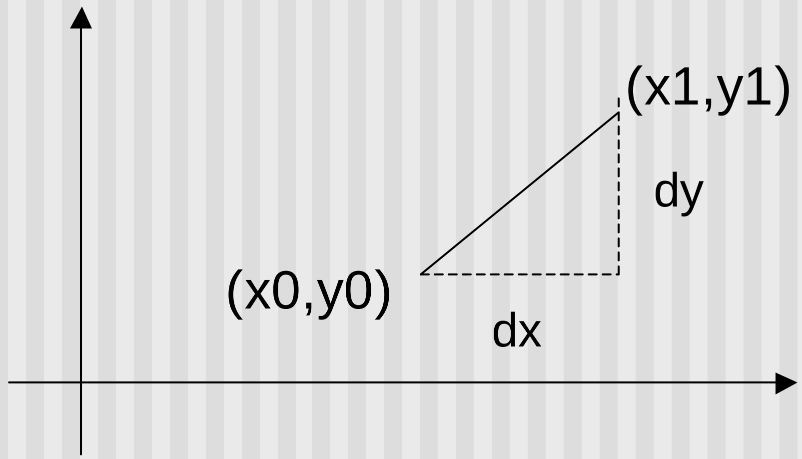
Line Drawing Algorithm

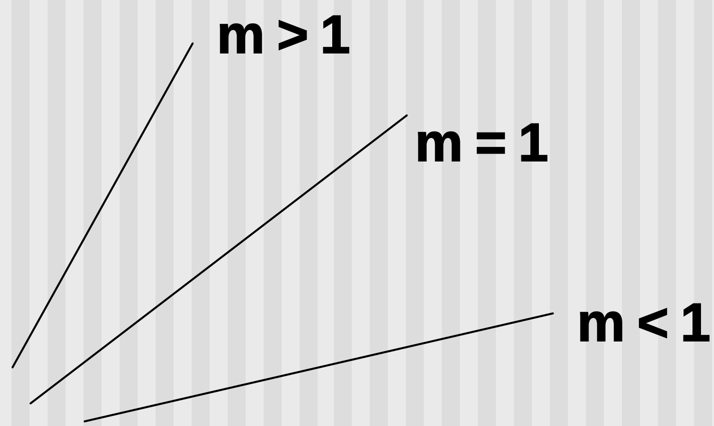
- Numerical example of finding slope m:
- (Ax, Ay) = (23, 41), (Bx, By) = (125, 96)

$$m = \frac{By - Ay}{Bx - Ax} = \frac{96 - 41}{125 - 23} = \frac{55}{102} = 0.5392$$

Digital Differential Analyzer (DDA): Line Drawing Algorithm

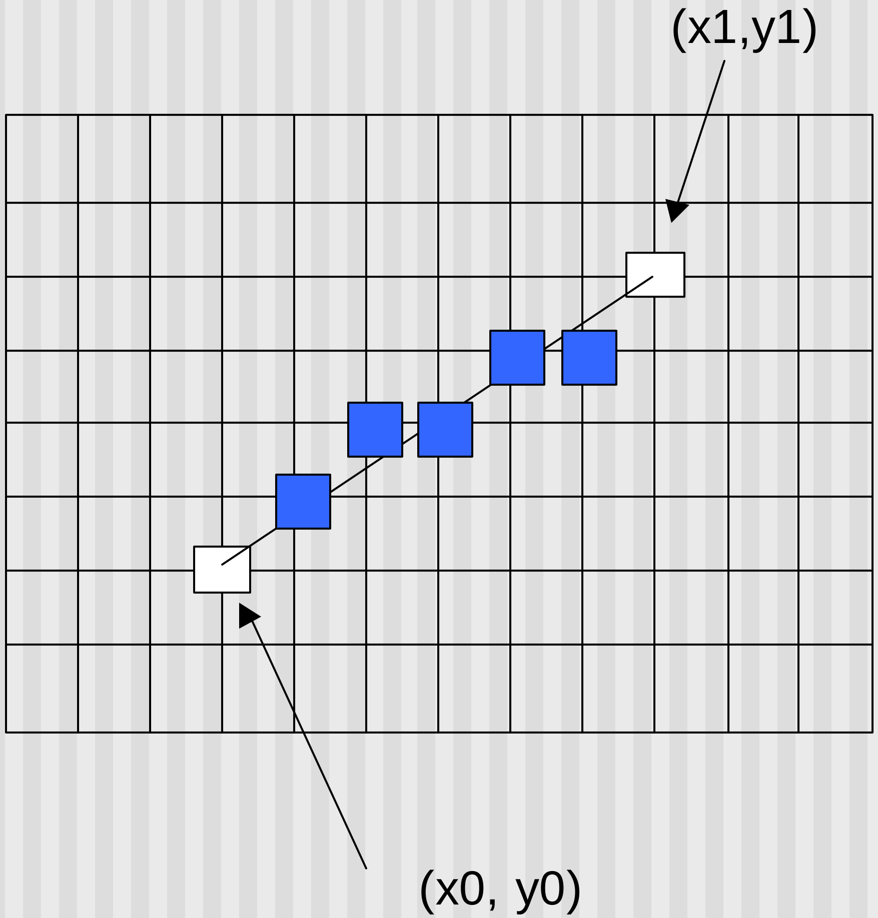
- Walk through the line, starting at (x0,y0)
- Constrain x, y increments to values in [0,1] range
- Case a: x is incrementing faster (m < 1)
    - Step in x=1 increments, compute and round y
- Case b: y is incrementing faster (m > 1)
    - Step in y=1 increments, compute and round x





DDA Line Drawing Algorithm (Case a: m < 1)

$$y_{k+1} = y_k + m$$



x = x0 y = y0

Illuminate pixel (x, round(y))

x = x0 + 1 y = y0 + 1 * m

Illuminate pixel (x, round(y))

x = x + 1 y = y + 1 * m

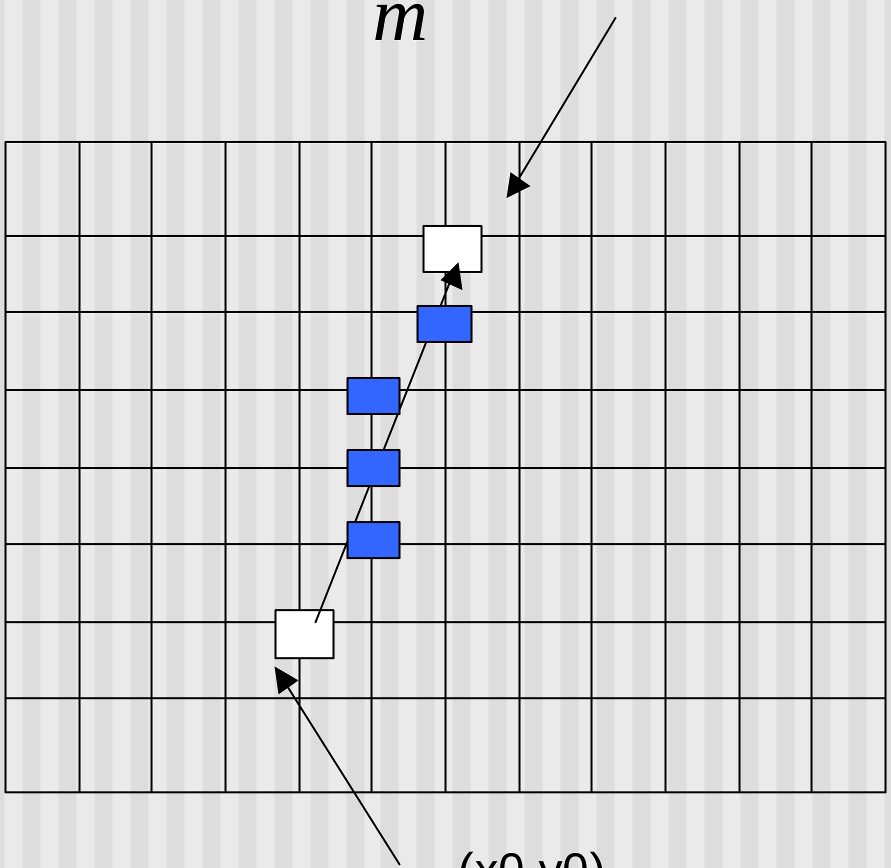
Illuminate pixel (x, round(y))

…

Until x == x1

DDA Line Drawing Algorithm (Case b: m > 1)

$$x_{k+1} = x_k + \frac{1}{m}$$



x = x0 y = y0

Illuminate pixel (round(x), y)

y = y0 + 1 x = x0 + 1 * 1/m

Illuminate pixel (round(x), y)

y = y + 1 x = x + 1 /m

Illuminate pixel (round(x), y)

...

Until y == y1

DDA Line Drawing Algorithm Pseudocode

```
compute m;
if m < 1:
{
   float y = y0;          // initial value
   for(int x = x0;x <= x1; x++, y += m)
               setPixel(x, round(y));
}
else // m > 1
{
   float x = x0;          // initial value
   for(int y = y0;y <= y1; y++, x += 1/m)
               setPixel(round(x), y);
}
```

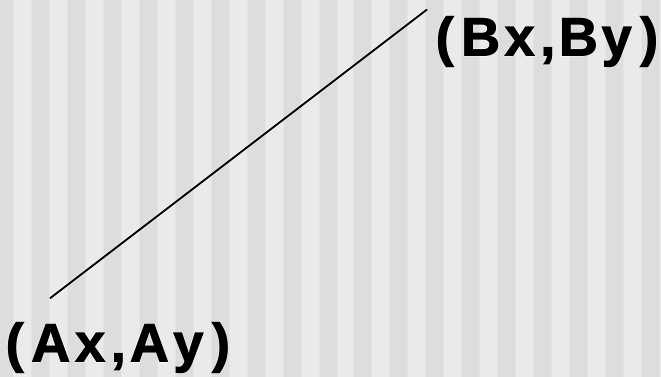
- Note: **`setPixel(x, y)`** writes current color into pixel in column x and row y in frame buffer

Line Drawing Algorithm Drawbacks

- DDA is the simplest line drawing algorithm
  - Not very efficient
  - Round operation is expensive
- Optimized algorithms typically used.
  - Integer DDA
  - E.g.Bresenham algorithm (Hill, 10.4.1)
- Bresenham algorithm
  - Incremental algorithm: current value uses previous value
  - Integers only: avoid floating point arithmetic
  - Several versions of algorithm: we'll describe midpoint version of algorithm

Bresenham's Line-Drawing Algorithm

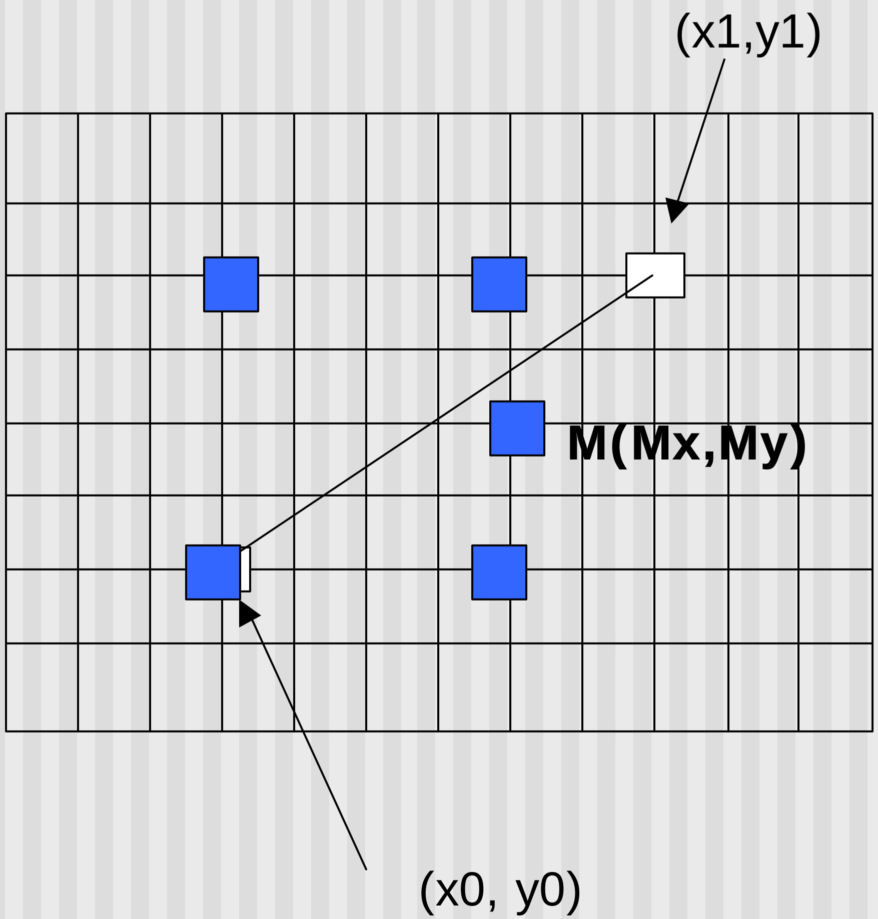
- Problem: Given endpoints (Ax, Ay) and (Bx, By) of a line, want to determine best sequence of intervening pixels
- First make two simplifying assumptions (remove later):
  - (Ax < Bx) and
  - (0 < m < 1)
- Define
  - Width W = Bx – Ax
  - Height H = By - Ay



Bresenham's Line-Drawing Algorithm

- Based on assumptions:
  - W, H are +ve
  - H < W
- As x steps in +1 increments, y incr/decr by <= +/–1
- y value sometimes stays same, sometimes increases by 1
- Midpoint algorithm determines which happens

Bresenham's Line-Drawing Algorithm



What Pixels to turn on or off?

Consider pixel midpoint M(Mx, My)

M = (x0 + 1, Y0 + ½)

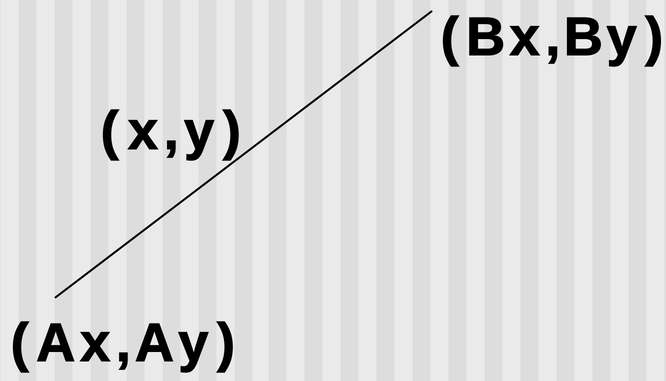
Build equation of line through and compare to midpoint

If midpoint is above line, y stays same
If midpoint is below line, y increases + 1

...

Bresenham's Line-Drawing Algorithm

- Using similar triangles:



$$\frac{y - Ay}{x - Ax} = \frac{H}{W}$$

- $H(x - Ax) = W(y - Ay)$
- $-W(y - Ay) + H(x - Ax) = 0$

- Above is ideal equation of line through (Ax, Ay) and (Bx, By)
- Thus, any point (x,y) that lies on ideal line makes eqn = 0
- Double expression (to avoid floats later), and give it a name,

$$F(x,y) = -2W(y - Ay) + 2H(x - Ax)$$

Bresenham's Line-Drawing Algorithm

- So, $F(x,y) = -2W(y - Ay) + 2H(x - Ax)$
- Algorithm, If:
  - $F(x, y) < 0$, (x, y) above line
  - $F(x, y) > 0$, (x, y) below line
- Hint: $F(x, y) = 0$ is on line
- Increase y keeping x constant, $F(x, y)$ becomes more negative

Bresenham's Line-Drawing Algorithm

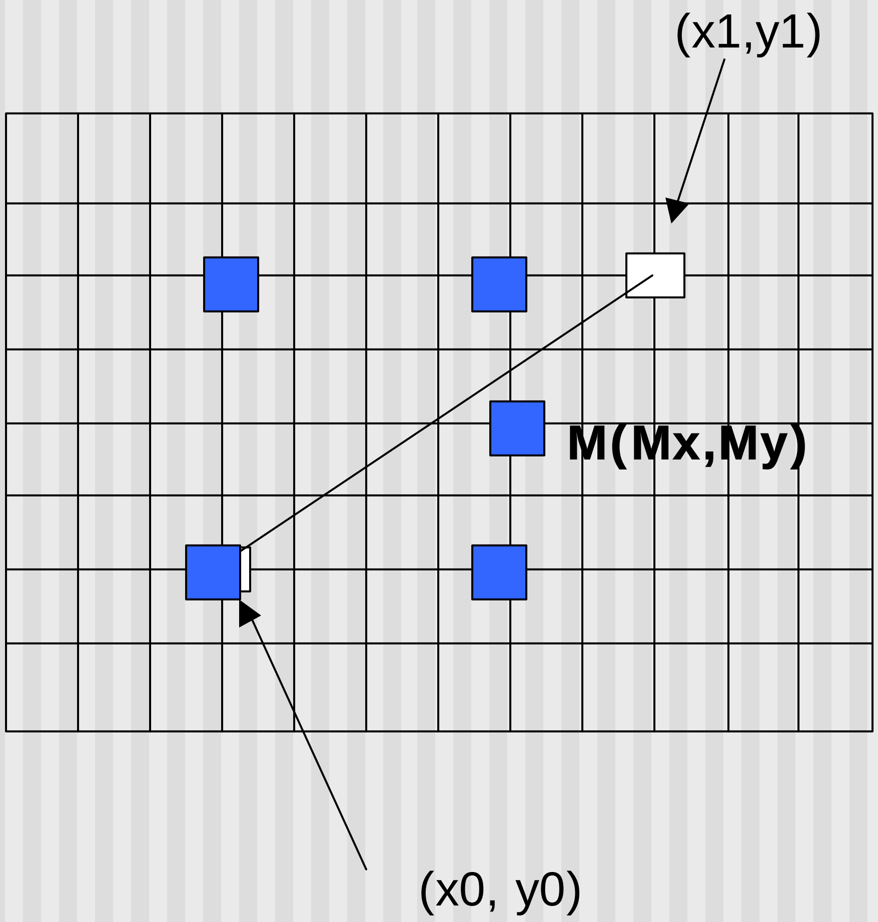
- Example: to find line segment between (3, 7) and (9, 11)

  $F(x,y) = -2W(y - Ay) + 2H(x - Ax)$

  $= (-12)(y - 7) + (8)(x - 3)$

- For points on line. E.g. (7, 29/3), F(x, y) = 0
- A = (4, 4) lies below line since F = 44
- B = (5, 9) lies above line since F = -8

Bresenham's Line-Drawing Algorithm



What Pixels to turn on or off?

Consider pixel midpoint M(Mx, My)

M = (x0 + 1, Y0 + ½)

If F(Mx,My) < 0, M lies above line, shade lower pixel (same y as before)

If F(Mx,My) > 0, M lies below line, shade upper pixel

...

Can compute F(x,y) incrementally

Initially, midpoint M = (Ax + 1, Ay + ½)

F(Mx, My) = -2W(y – Ay) + 2H(x – Ax)

= 2H – W

Can compute F(x,y) for next midpoint incrementally

If we increment x + 1, y stays same, compute new F(Mx,My)

F(Mx, My) += 2H

If we increment x +1, y + 1

F(Mx, My) -= 2(W – H)

Bresenham's Line-Drawing Algorithm

```
Bresenham(IntPoint a, InPoint b)
{ // restriction: a.x < b.x and 0 < H/W < 1
   int y = a.y, W = b.x – a.x, H = b.y – a.y;
   int F = 2 * H – W;    // current error term
   for(int x = a.x; x <= b.x; x++)
   {
     setpixel at (x, y); // to desired color value
        if F < 0
             F = F + 2H;
        else{
             Y++,  F = F  + 2(H – W)
        }
   }
}
```

- Recall: F is equation of line

Bresenham's Line-Drawing Algorithm

- Final words: we developed algorithm with restrictions $0 < m < 1$ and $Ax < Bx$
- Can add code to remove restrictions
  - To get the same line when $Ax > Bx$ (swap and draw)
  - Lines having $m > 1$ (interchange x with y)
  - Lines with $m < 0$ (step x++, decrement y not incr)
  - Horizontal and vertical lines (pretest a.x = b.x and skip tests)
- Important: **Read Hill 10.4.1**

References

- Hill, chapter 10