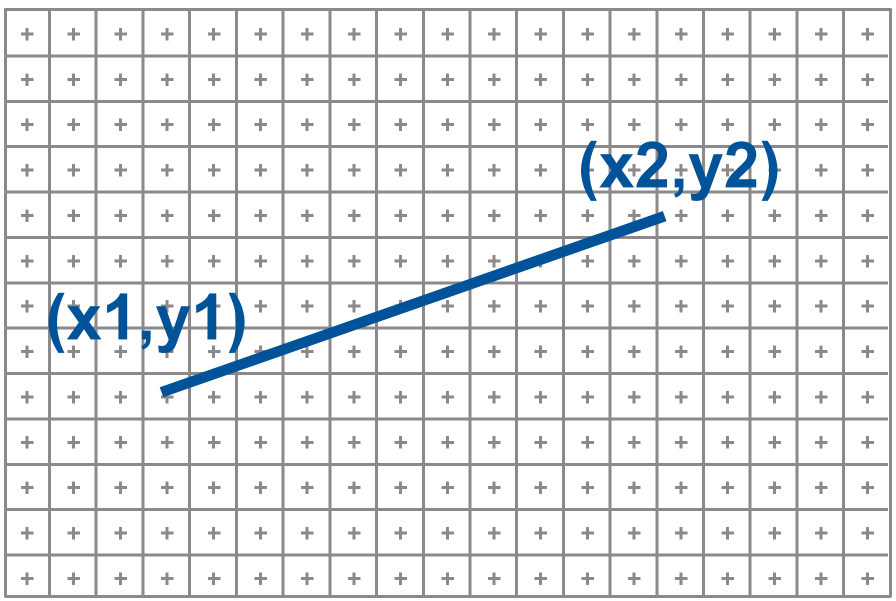
Framebuffer Model

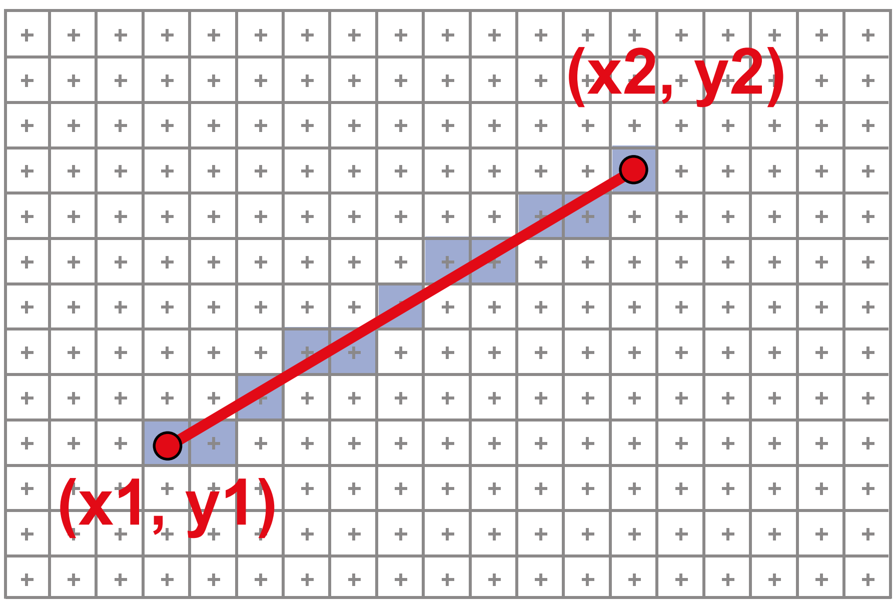
- Raster Display: 2D array of picture elements (pixels)
- Pixels individually set/cleared (greyscale, color)
- Window coordinates: pixels centered at integers




```
glBegin(GL_LINES)
glVertex3f(...)
glVertex3f(...)
glEnd();
```

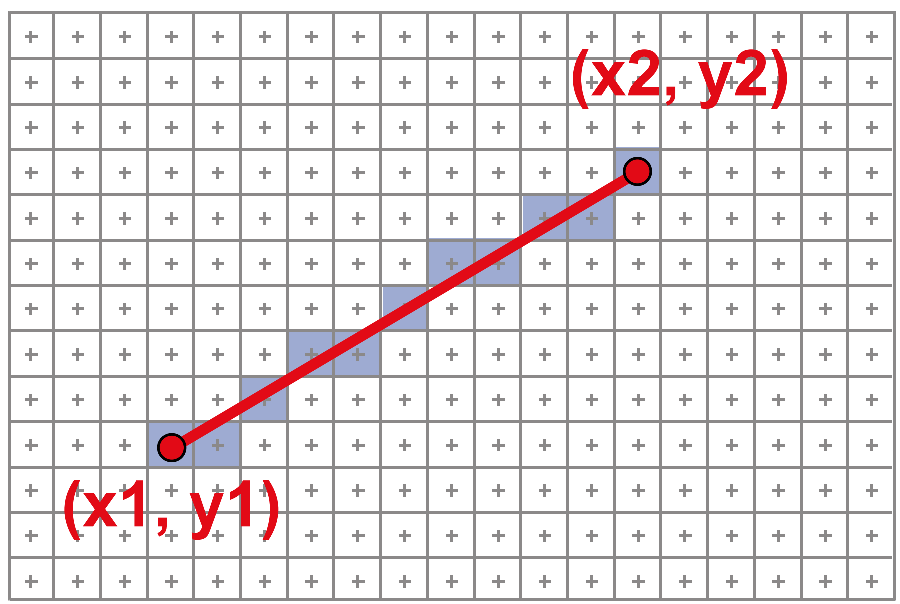
Scan Converting 2D Line Segments

- Given:
  - Segment endpoints (integers x1, y1; x2, y2)
- Identify:
  - Set of pixels (x, y) to display for segment



Line Rasterization Requirements

- Transform continuous primitive into discrete samples
- Uniform thickness & brightness
- Continuous appearance
- No gaps
- Accuracy
- Speed



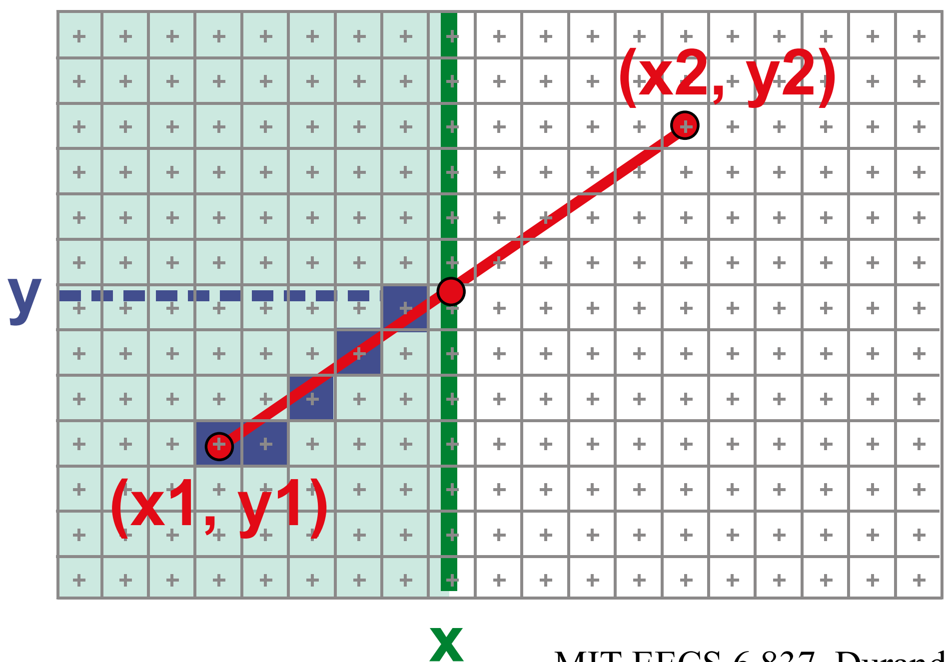
Naive Line Rasterization Algorithm

- Simply compute y as a function of x
  - Conceptually: move vertical scan line from x1 to x2
  - What is the expression of y as function of x?
  - Set pixel (x, round (y(x)))

$$y = y1 + \frac{x - x1}{x2 - x1}(y2 - y1)$$

$$= y1 + m(x - x1)$$

$$m = \frac{dy}{dx}$$

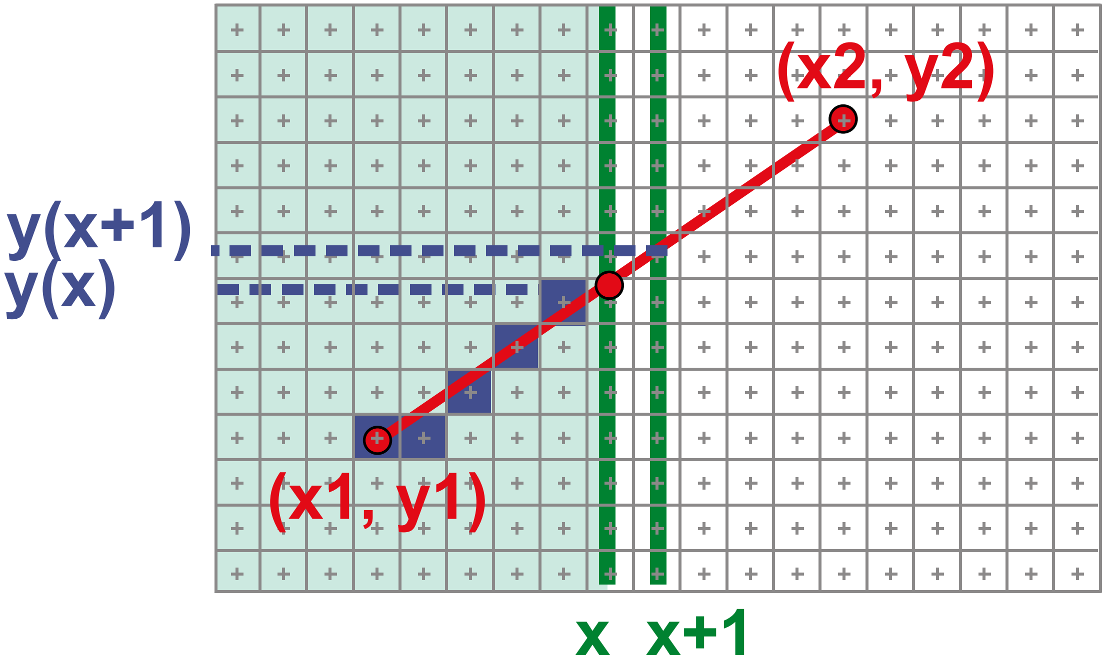


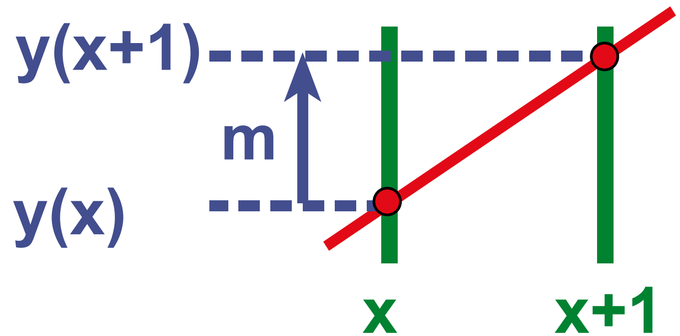
Efficiency

- Computing y value is expensive

$$y = y1 + m(x - x1)$$

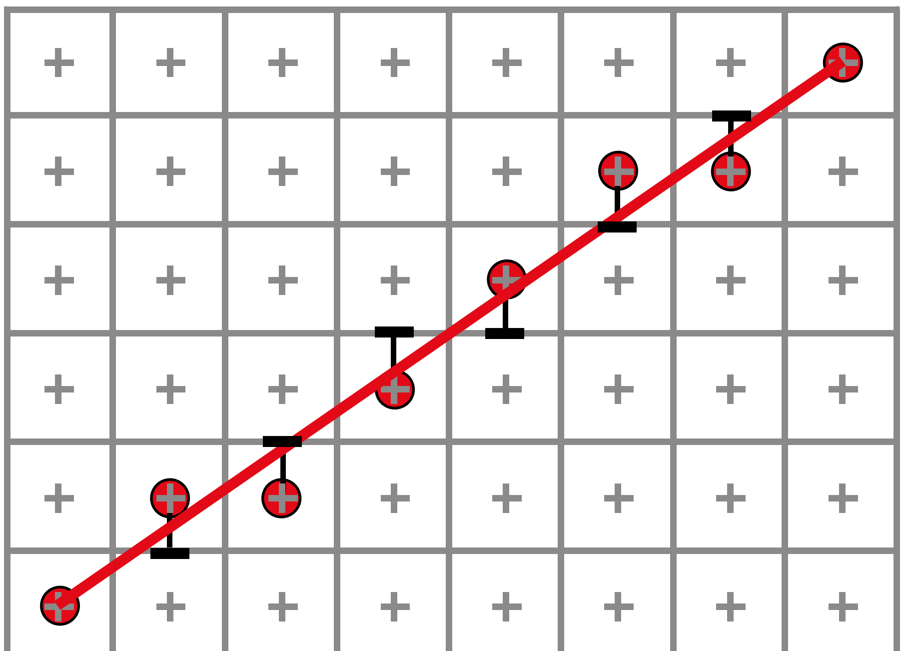
- Observe: $y \mathrel{+}= m$ at each x step ($m = dy/dx$)





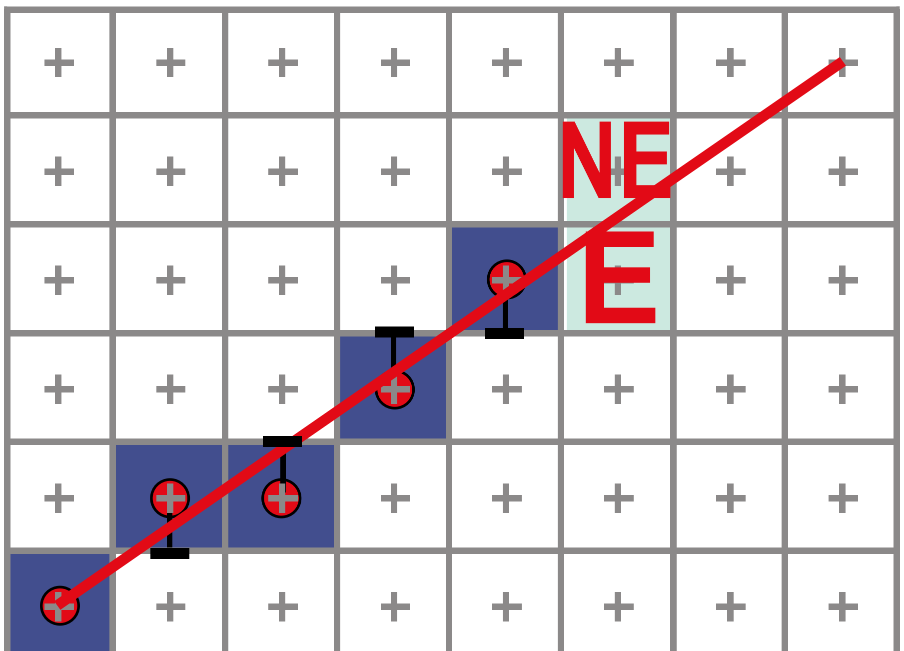
Bresenham's Algorithm (DDA)

- Select pixel vertically closest to line segment
  - intuitive, efficient,
    pixel center always within 0.5 vertically
- Same answer as naive approach



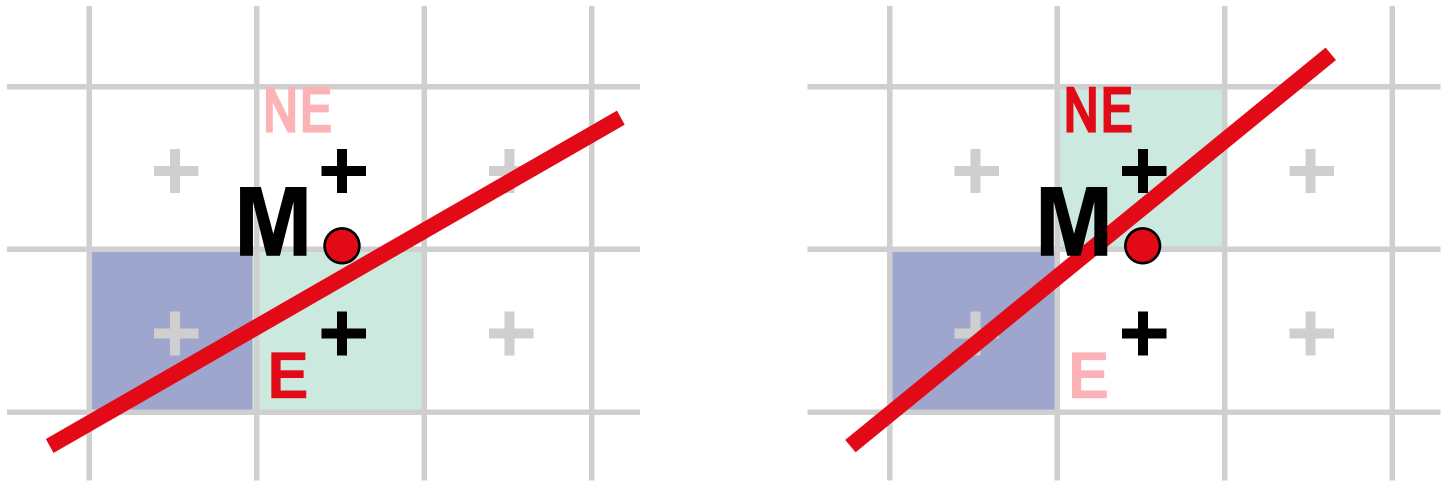
Bresenham's Algorithm (DDA)

- Observation:
  - If we're at pixel P (x_p, y_p), the next pixel must be either E (x_p+1, y_p) or NE (x_p, y_p+1)
  - Why?



Bresenham Step

- Which pixel to choose: E or NE?
  - Choose E if segment passes below or through middle point M
  - Choose NE if segment passes above M

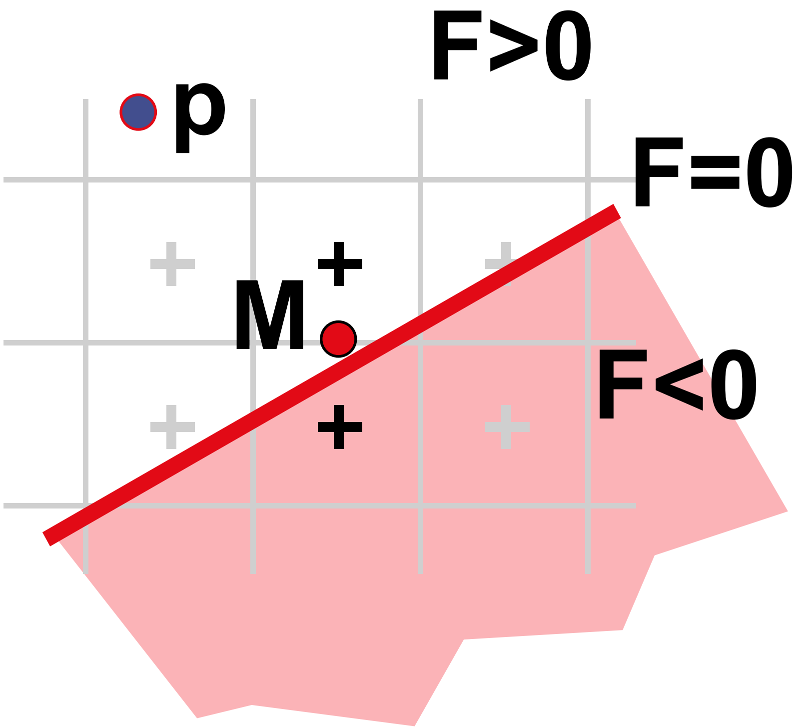


Bresenham Step

- Use *decision function* D to identify points underlying line L:

  $D(x, y) = y-mx-b$

  – positive above L
  – zero on L
  – negative below L

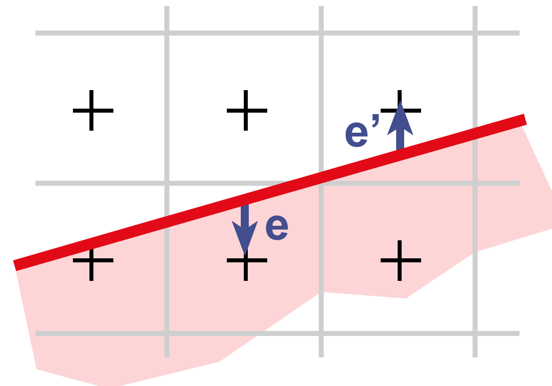


$D(p_x, p_y) =$ vertical distance from point to line

Bresenham's Algorithm (DDA)

- Decision Function:

  $D(x, y) = y - mx - b$



- Initialize:

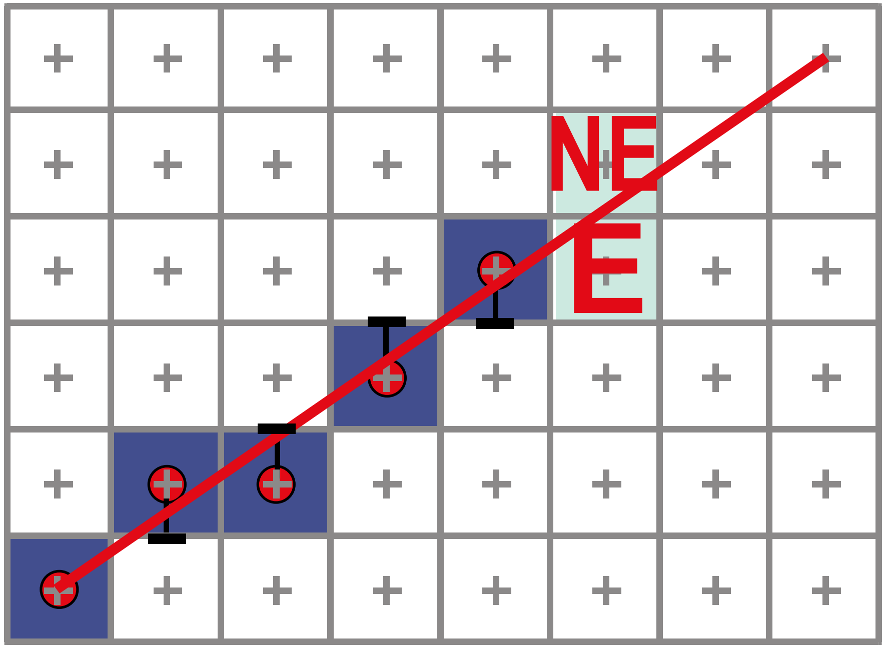
  error term $e = -D(x,y)$

- On each iteration:

  update x: $x' = x+1$
  update e: $e' = e + m$
  if ($e \leq 0.5$): $y' = y$ (choose pixel E)
  if (e > 0.5): y' = y + (choose pixel NE) e' = e - 1

Summary of Bresenham

- initialize *x, y, e*
- for (x = x1; $x \leq$ x2; x++)
  - plot (x,y)
  - update *x, y, e*



- Generalize to handle all eight octants using symmetry
- Can be modified to use only integer arithmetic