Rasterization

CS418 Computer Graphics

John C. Hart

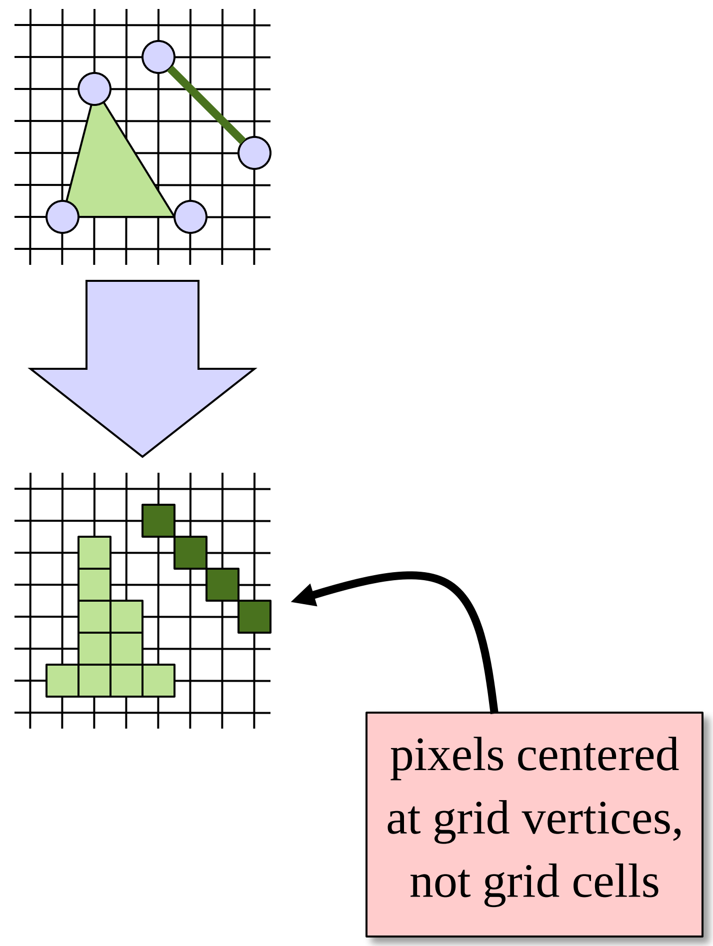
Rasterization

Converts

- lines and triangles
- with floating point vertices
- in viewport (screen) coordinates

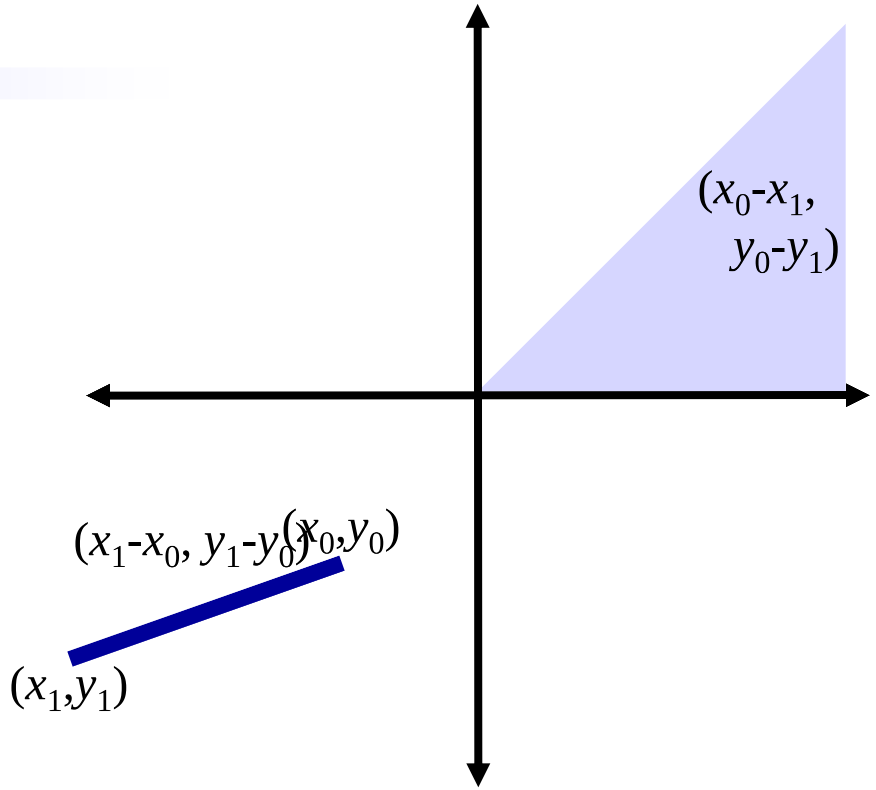
into

- pixels
- with integer coordinates
- in viewport (screen) coordinates



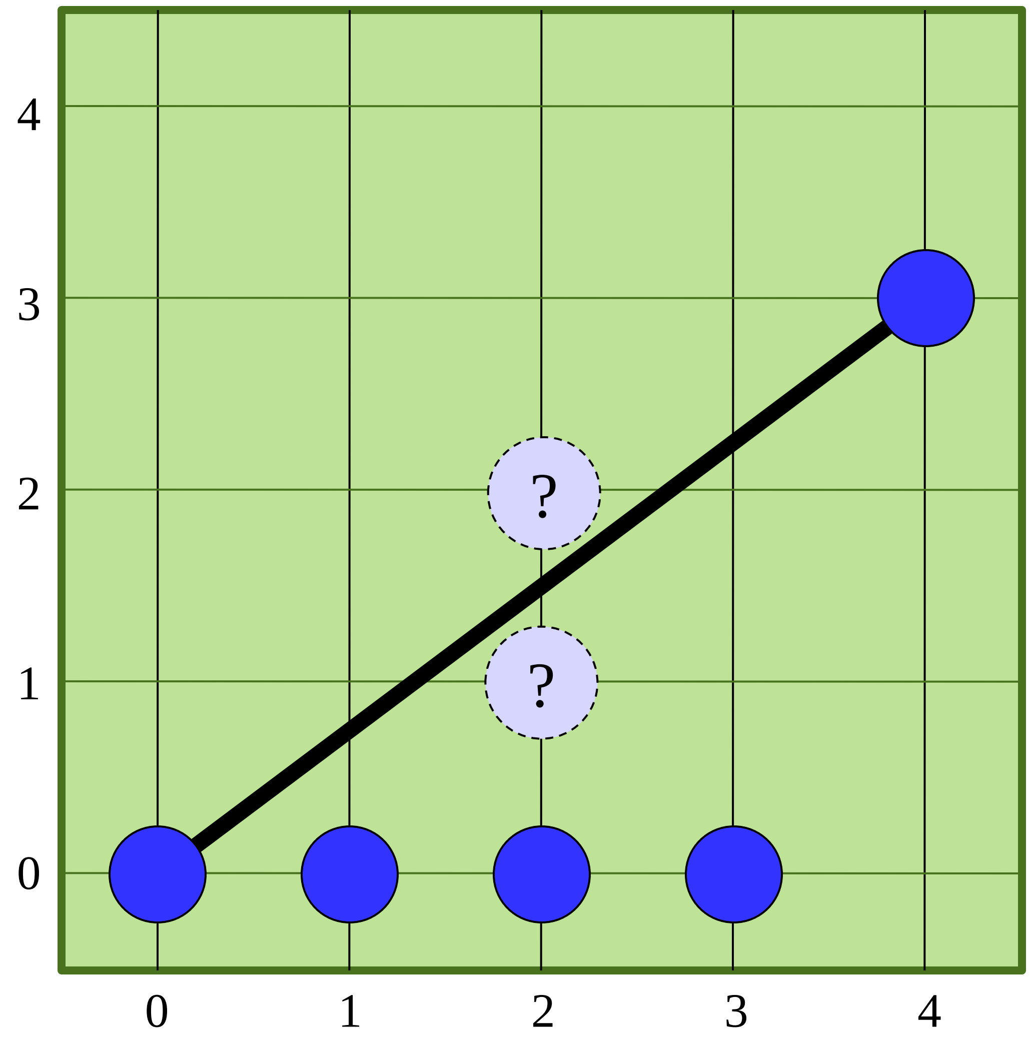
Line Rasterization

- Need to rasterize lines between any two clipped screen points, from (x_0,y_0) to (x_1,y_1)
- Only rasterize lines from the origin to a point in the first octant
- Translate start vertex to origin
- Reflect end vertex into first octant



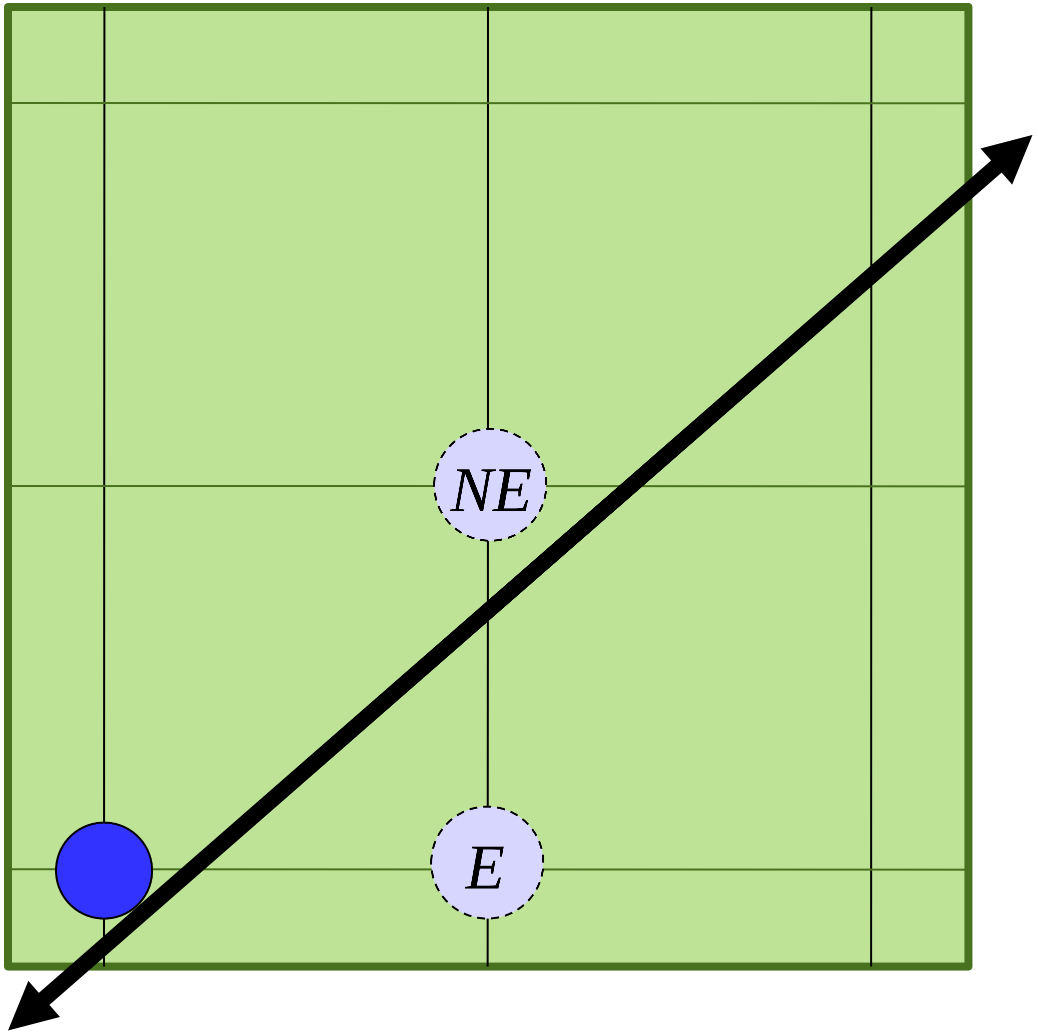
Line Rasterization

- How to rasterize a line from (0,0) to (4,3)
- Pixel (0,0) and (4,3) easy
- One pixel for each integer x-coordinate
- Pixel's y-coordinate closest to line
- If line equal distance between two pixels, pick on arbitrarily but consistently



Midpoint Algorithm

- Which pixel should be plotted next?
  - East?
  - Northeast?



Midpoint Algorithm

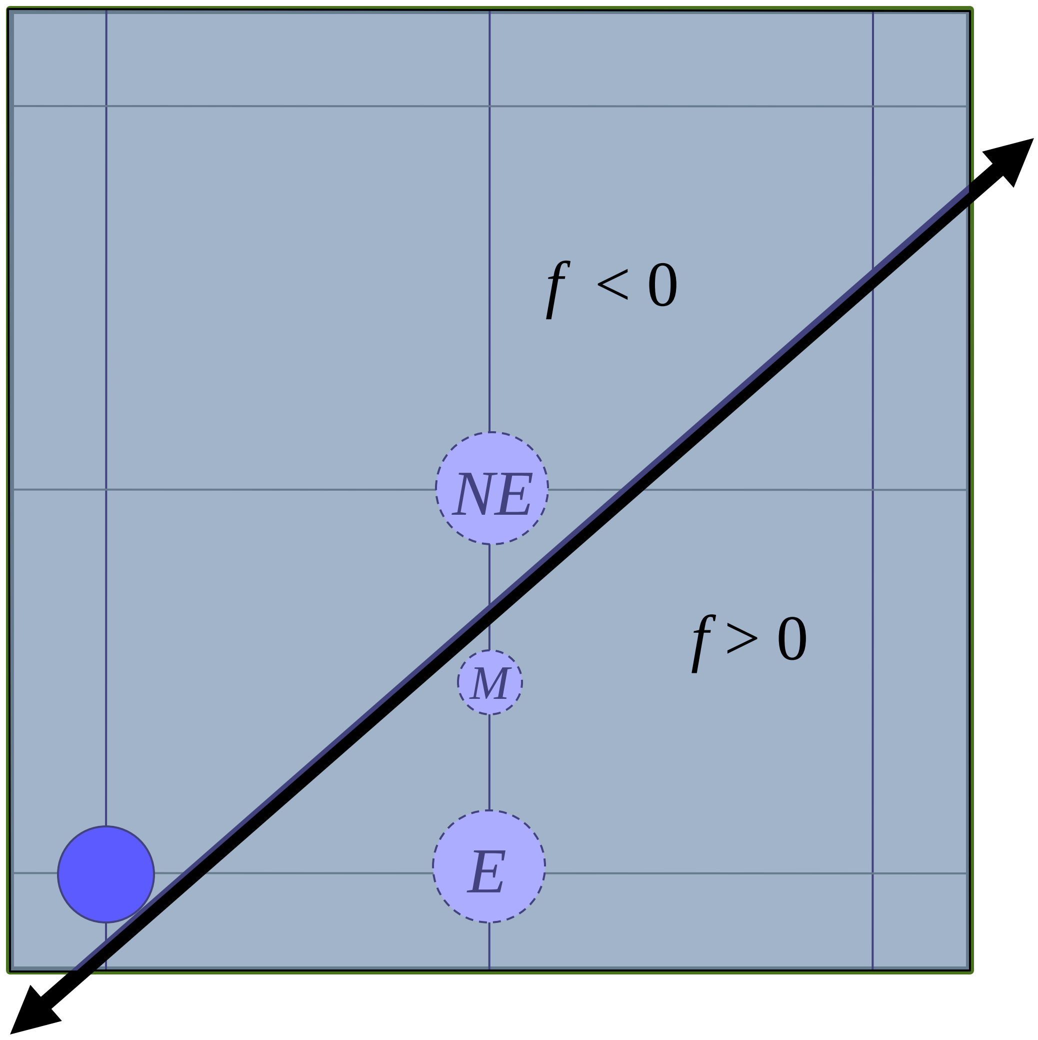
- Which pixel should be plotted next?
  - East?
  - Northeast?
- Line equation

$$y = mx + b$$

$$m = (y_1 - y_0)/(x_1 - x_0)$$

$$b = y_0 - mx_0$$

$$f(x,y) = mx + b - y$$



Midpoint Algorithm

- Which pixel should be plotted next?
  - East?
  - Northeast?
- Line equation

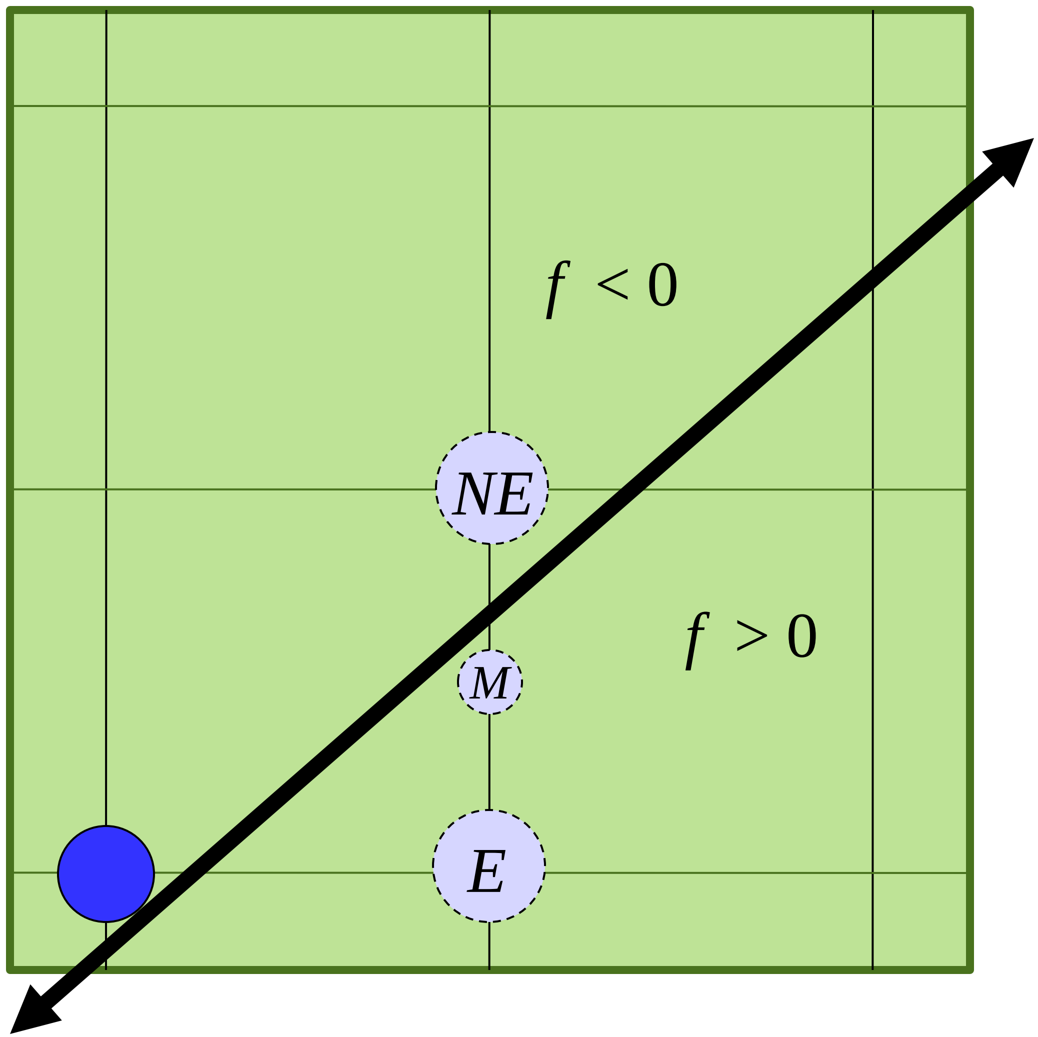
$$y = mx + b$$

$$m = (y_1 - y_0)/(x_1 - x_0)$$

$$b = y_0 - mx_0$$

$$f(x,y) = mx + b - y$$

- $f(M) \geq 0 \rightarrow NE$
- $f(M) < 0 \rightarrow E$



Pixel Increments

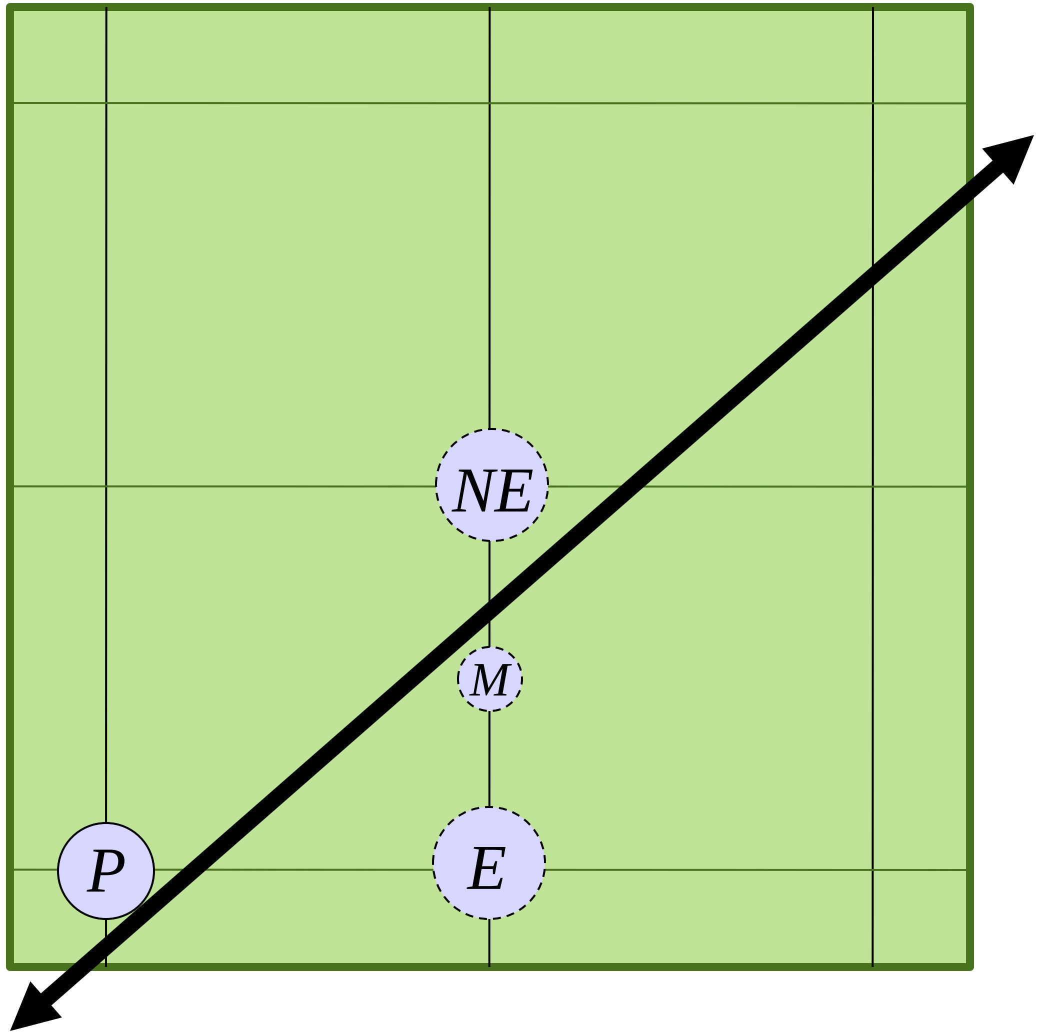
$f(x,y) = mx + b - y$

$M = P + (1,½)$

$$
\begin{aligned}
f(M) &= f(x+1, y+½) \\
&= m(x+1) + b - (y+½) \\
&= mx + m + b - y - ½ \\
&= mx + b - y + m - ½ \\
&= f(P) + m - ½
\end{aligned}
$$

$f(0,0) = b$

$= 0$ if line starts at origin



Midpoint Increments

$f(M) = f(P) + m - ½$

$f(M_E) = f(x+2, y+½)$

$= m(x+2) + b - (y+½)$

$= f(P) + 2m - ½$

$= f(M) + m$

$f(M_{NE}) = f(x+2, y+1½)$

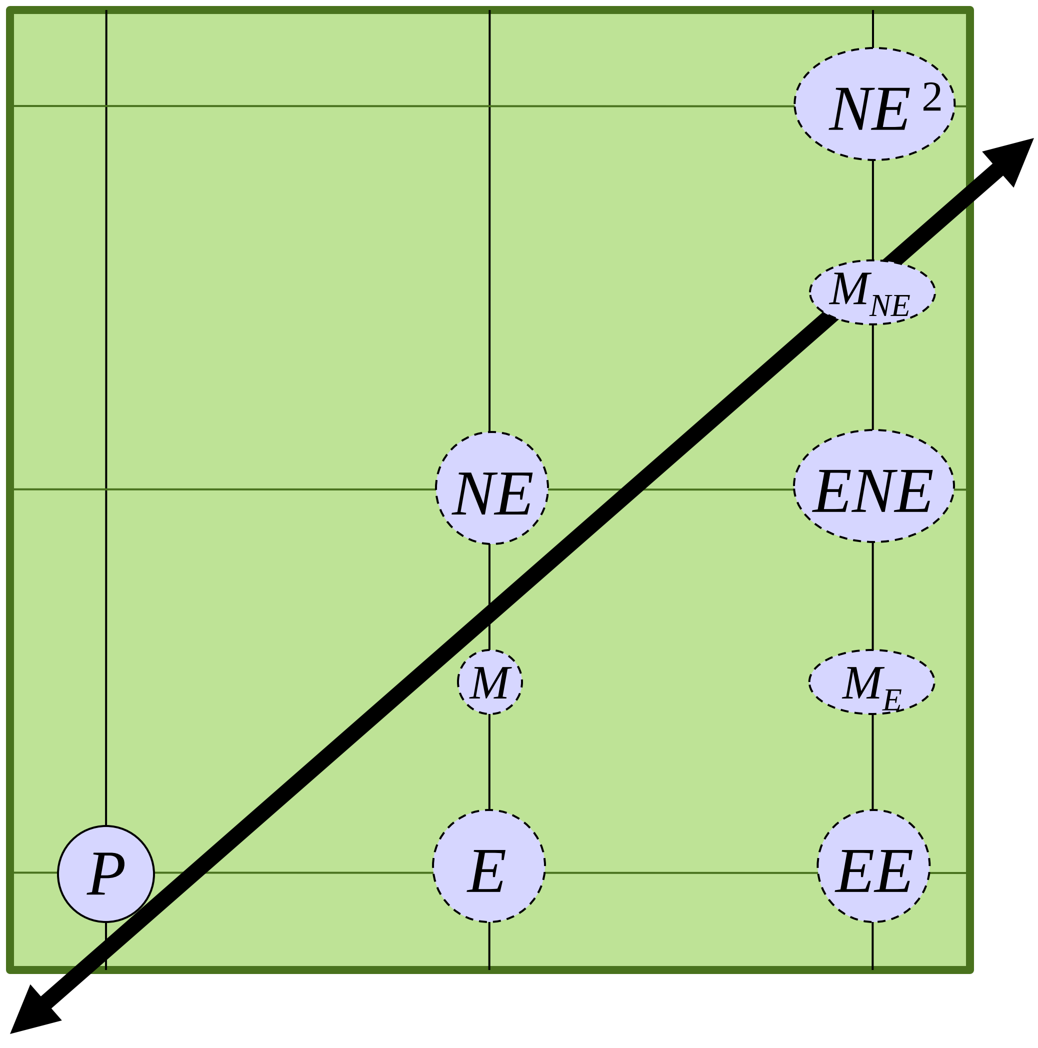
$= m(x+2) + b - (y+1½)$

$= f(P) + 2m - 1½$

$= f(M) + m - 1$

$f(1, ½) = m + b - ½$

$= m - ½$ if line starts at origin



Integer Math

$f(M_E) = f(M) + m$

$f(M_{NE}) = f(M) + m - 1$

$f(1, ½) = m + b - ½$

$b = 0$

$m = (y_1 - y_0)/(x_1 - x_0)$

$= \Delta y/\Delta x$

$\Delta x\, f(M_E) = \Delta x\, f(M) + \Delta y$

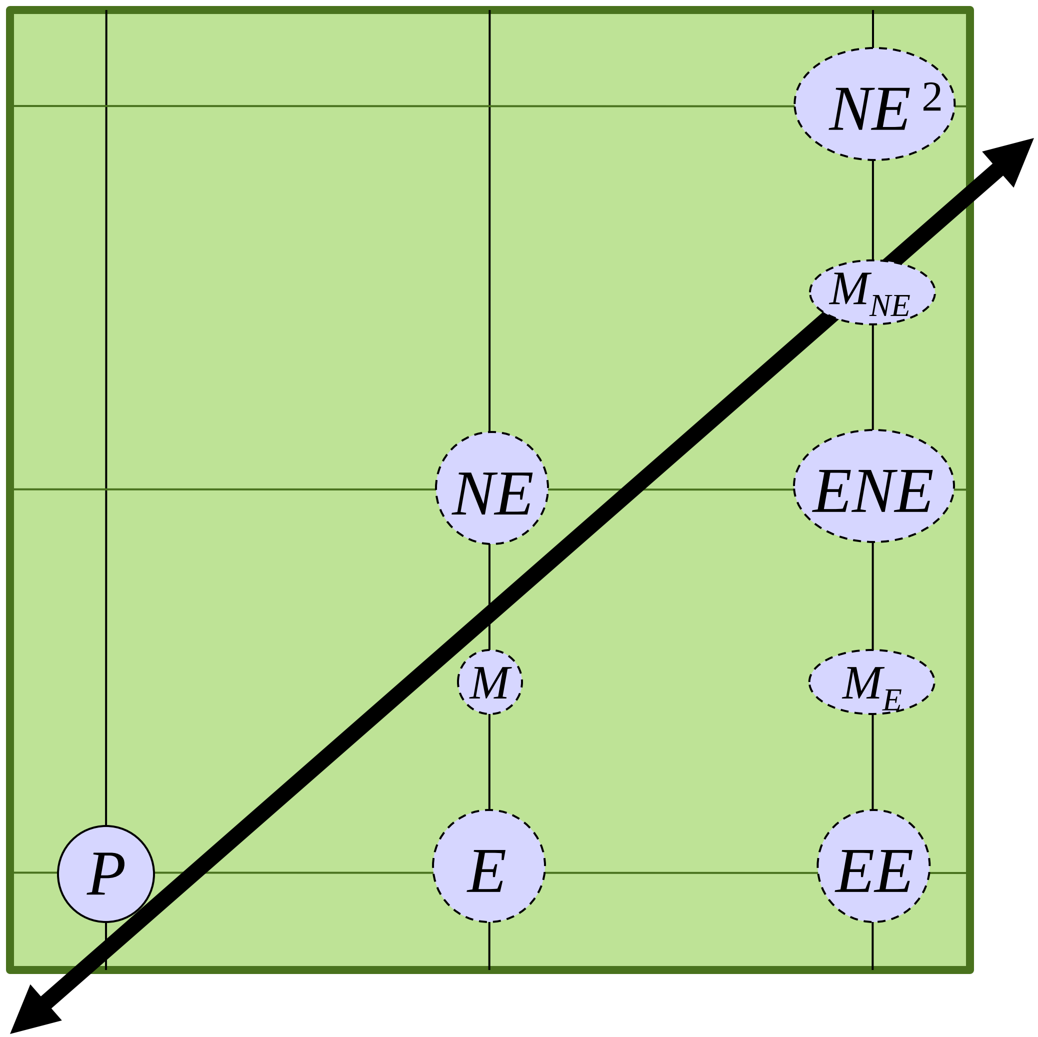
$\Delta x\, f(M_{NE}) = \Delta x\, f(M) + \Delta y - \Delta x$

$\Delta x\, f(1, ½) = \Delta y - ½\, \Delta x$

$2\Delta x f(M_E) = 2\Delta x\, f(M) + 2\Delta y$

$2\Delta x f(M_{NE}) = 2\Delta x\, f(M)+2\Delta y-2\Delta x$

$2\Delta x f(1, ½) = 2\Delta y - \Delta x$



Integer Math

$2\Delta x f(M_E) = 2\Delta x\, f(M) + 2\Delta y$

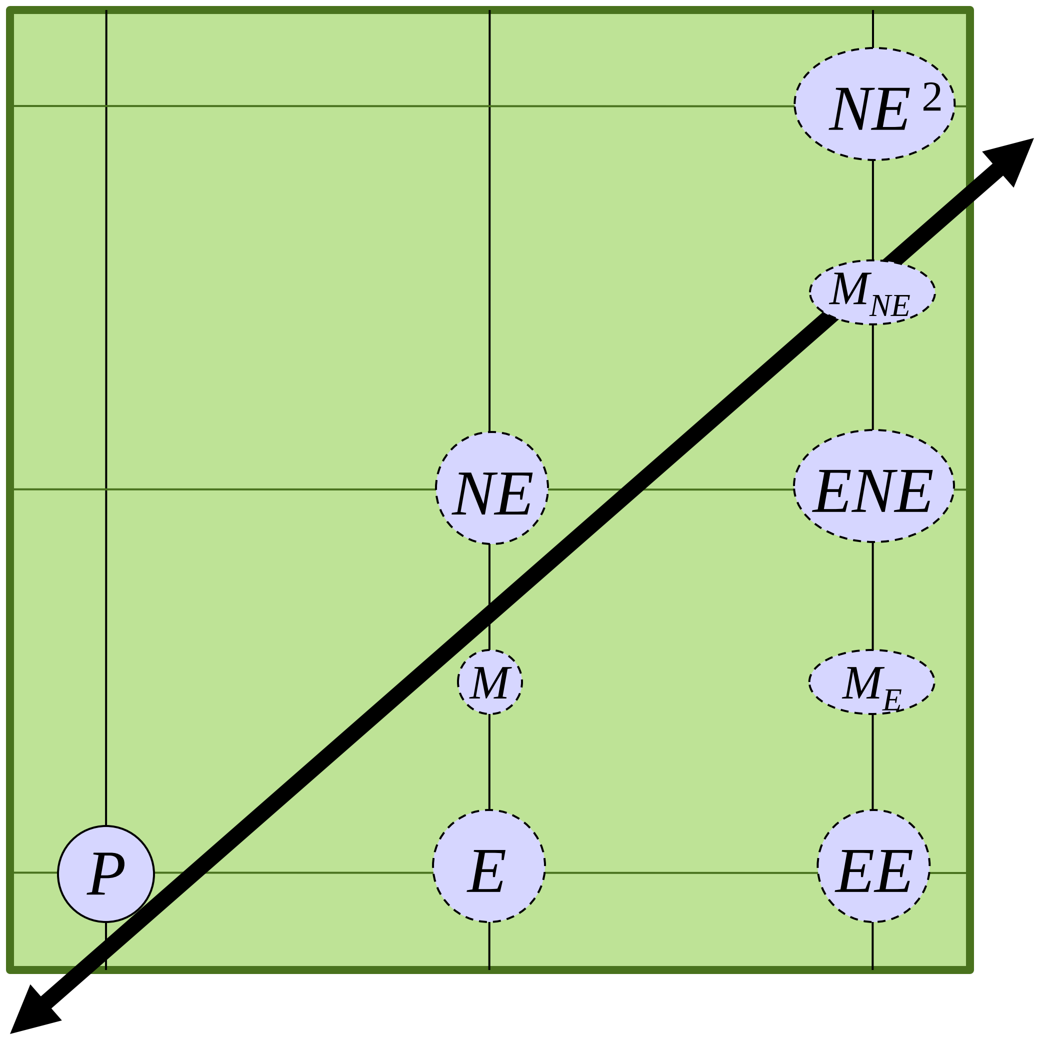
$2\Delta x f(M_{NE}) = 2\Delta x\, f(M) + 2\Delta y - 2\Delta x$

$2\Delta x f(1, ½) = 2\Delta y - \Delta x$

$F(M_E) \quad = F(M) + 2\Delta y$

$F(M_{NE}) = F(M) + 2\Delta y - 2\Delta x$

$F(1,½) \quad = 2\Delta y - \Delta x$



Integer Math

The Bresenham Line Algorithm

$$F(M_E) = F(M) + 2\Delta y$$

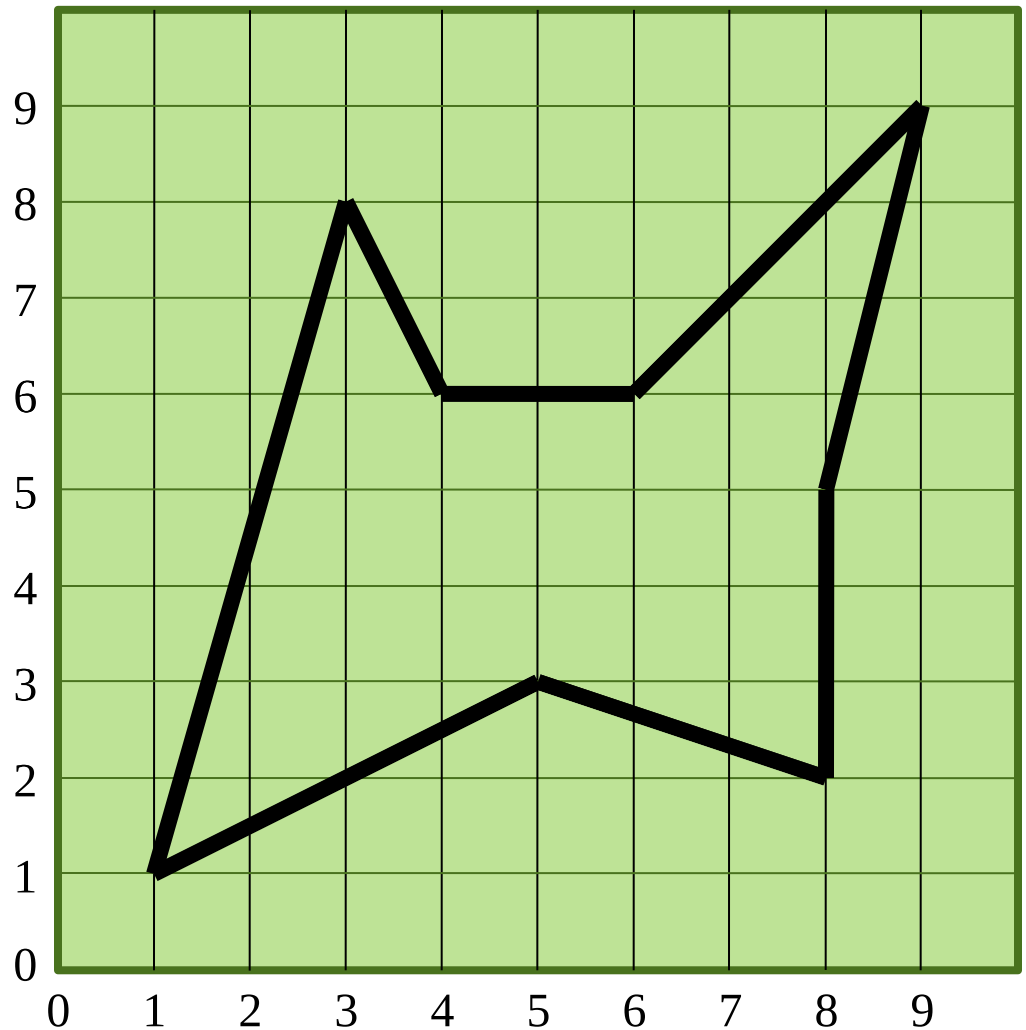
$$F(M_{NE}) = F(M) + 2\Delta y - 2\Delta x$$

$$F(1,½) = 2\Delta y - \Delta x$$

```
line(int x0,int y0,int x1,int y1)
{
  int dx = x1 - x0;
  int dy = y1 - y0;

  int F = 2*dy - dx;

  int dFE = 2*dy;
  int dFNE = 2*dy - 2*dx;

  int y = y0;
  for (int x = x0, x < x1; x++) {
    plot(x,y);
    if (F < 0) {
      F += dFE;
    } else {
      F += dFNE; y++;
    }
  }
}
```

Polygon Rasterization

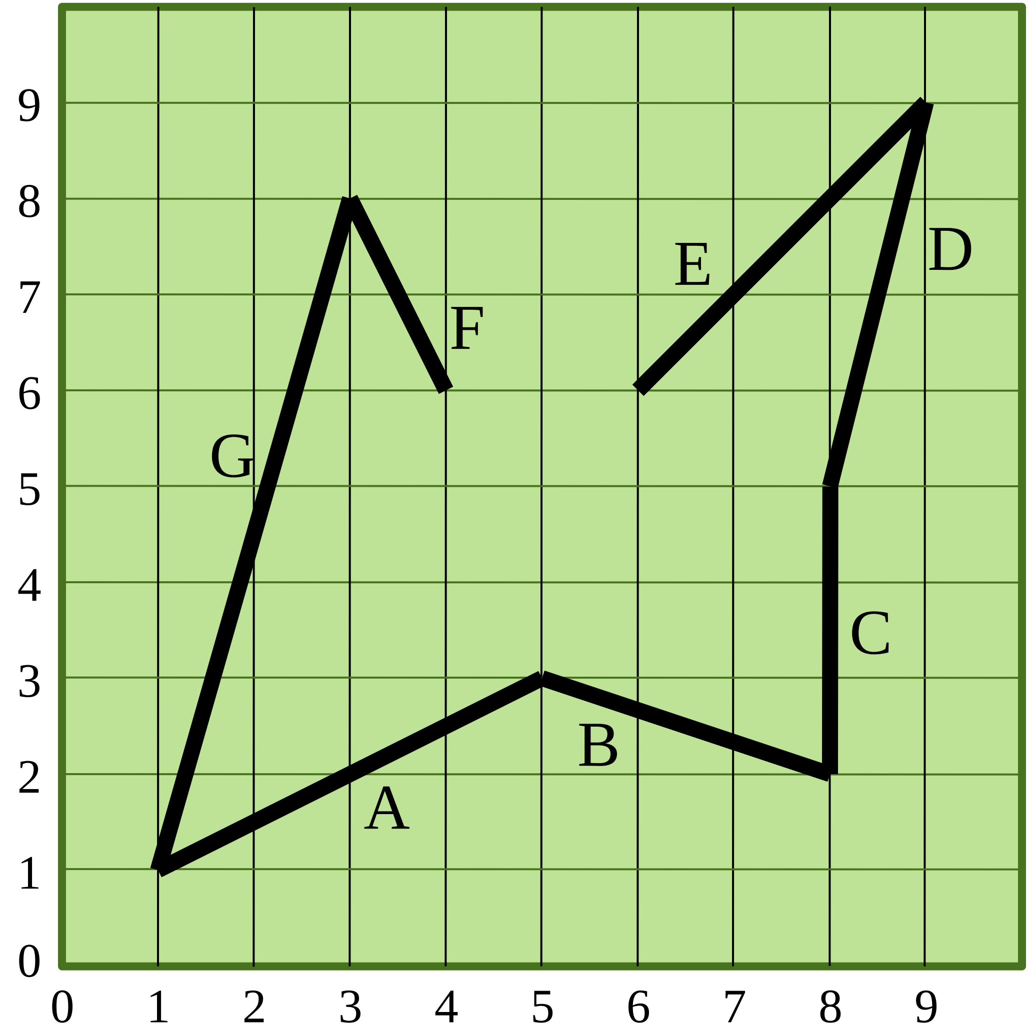
- Ignore horizontal lines



Polygon Rasterization

- Ignore horizontal lines
- Sort edges by smaller y coordinate

Edge	ymin
A	1
G	1
B	2
C	2
D	5
E	6
F	6

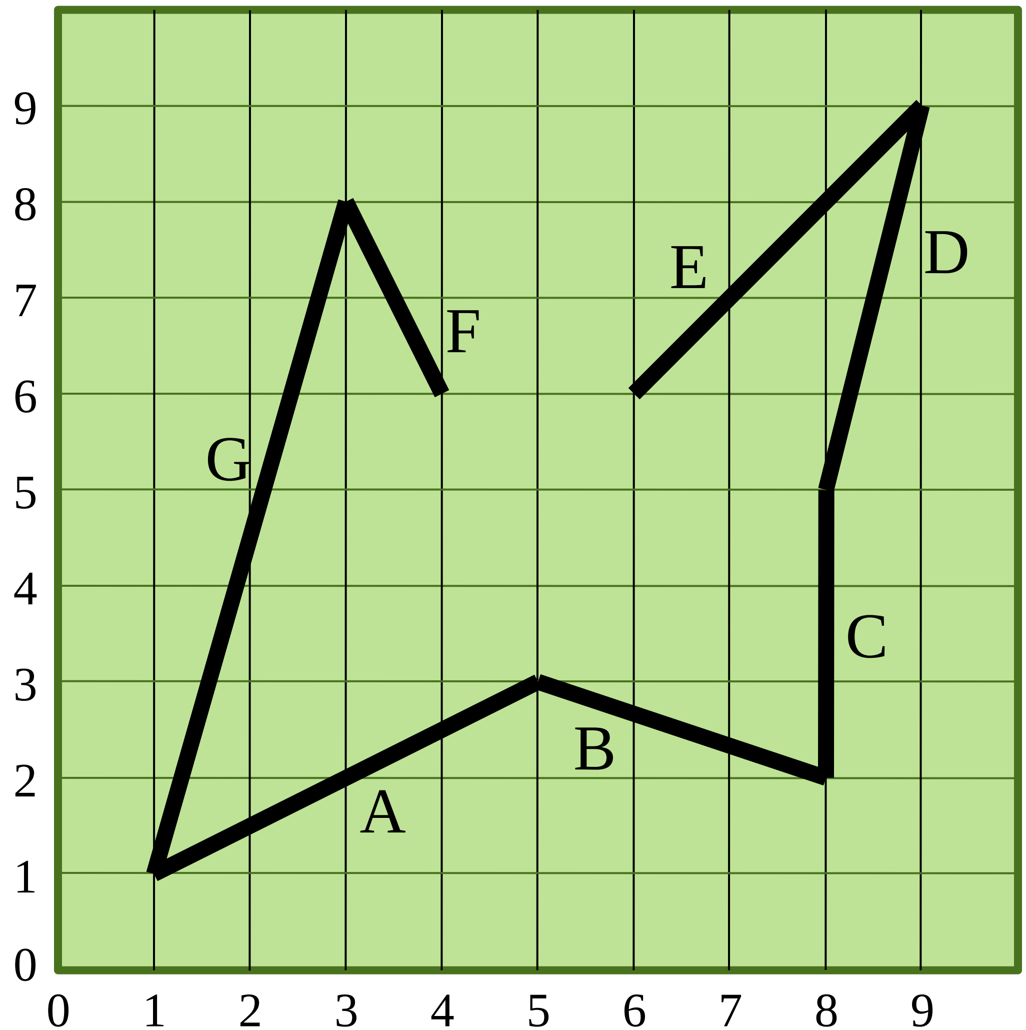


Polygon Rasterization

Edge	x	dx/dy	ymax

- For each scanline…
- Add edges where y = ymin
- Sorted by x
- Then by dx/dy

Edge	ymin
A	1
G	1
B	2
C	2
D	5
E	6
F	6



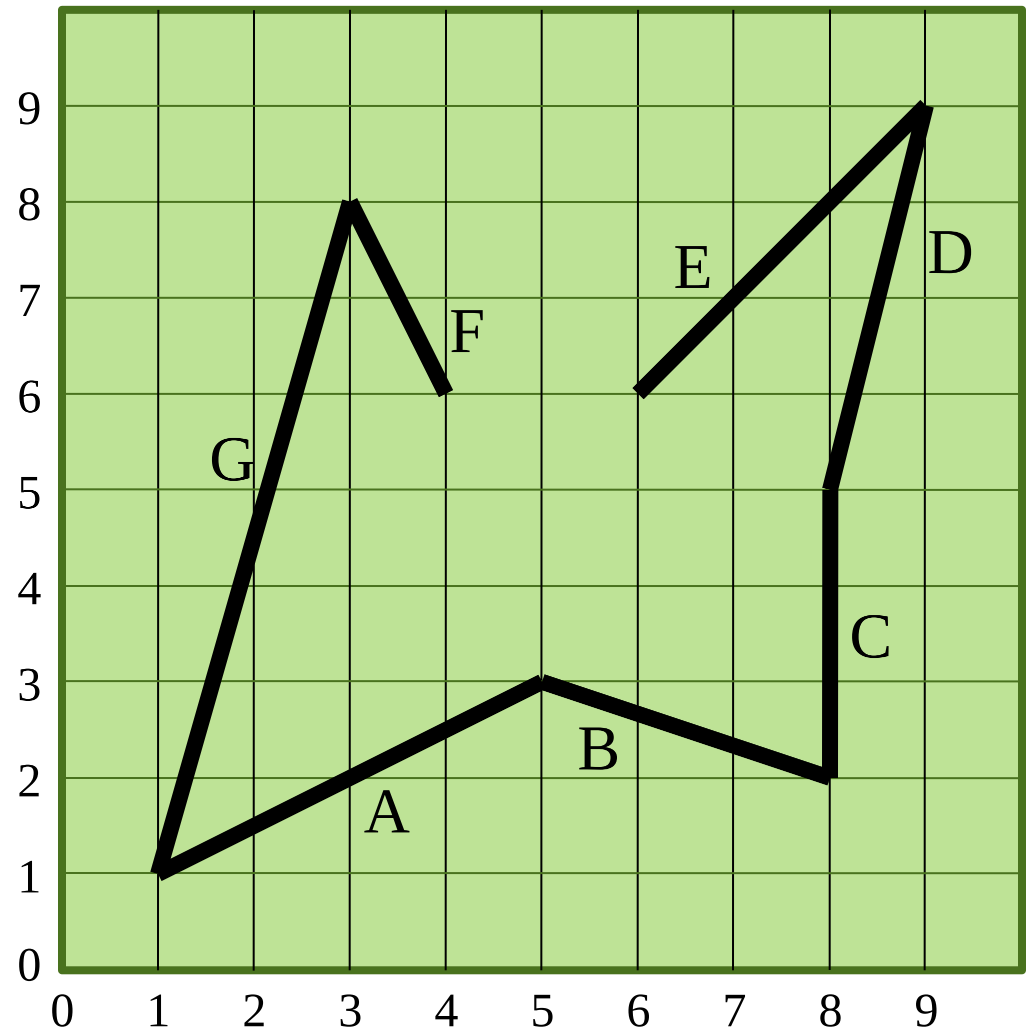
Polygon Rasterization

Edge	x	dx/dy	ymax

Plotting rules for when segments lie on pixels

1. Plot lefts
2. Don’t plot rights
3. Plot bottoms
4. Don’t plot tops

Edge	ymin
A	1
G	1
B	2
C	2
D	5
E	6
F	6

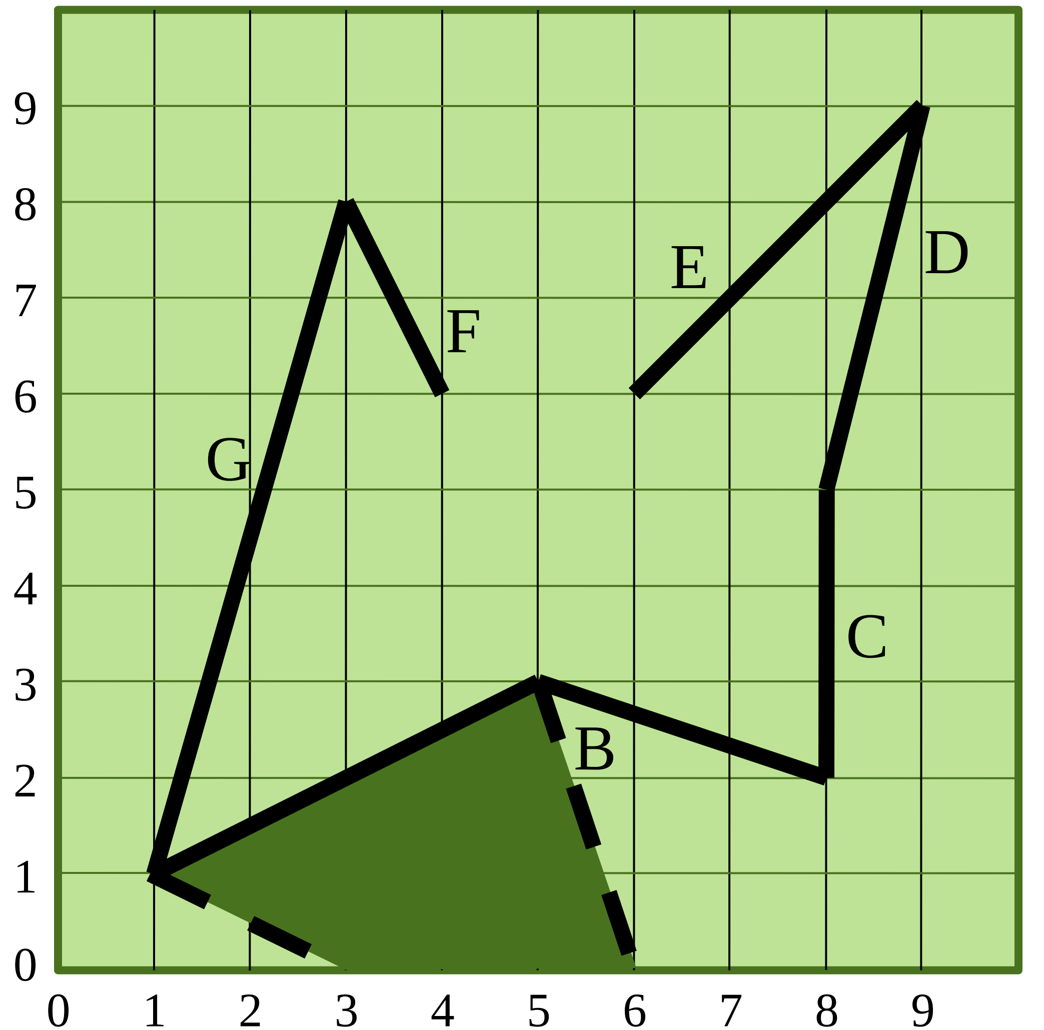


Polygon Rasterization

Edge	x	dx/dy	ymax
G	1	2/7	8
A	1	4/2	3

- y = 1
- Delete y = ymax edges
- Update x
- Add y = ymin edges
- For each pair x_0,x_1, plot from ceil(x_0) to ceil(x_1) – 1

Edge	ymin
A	1
G	1
B	2
C	2
D	5
E	6
F	6

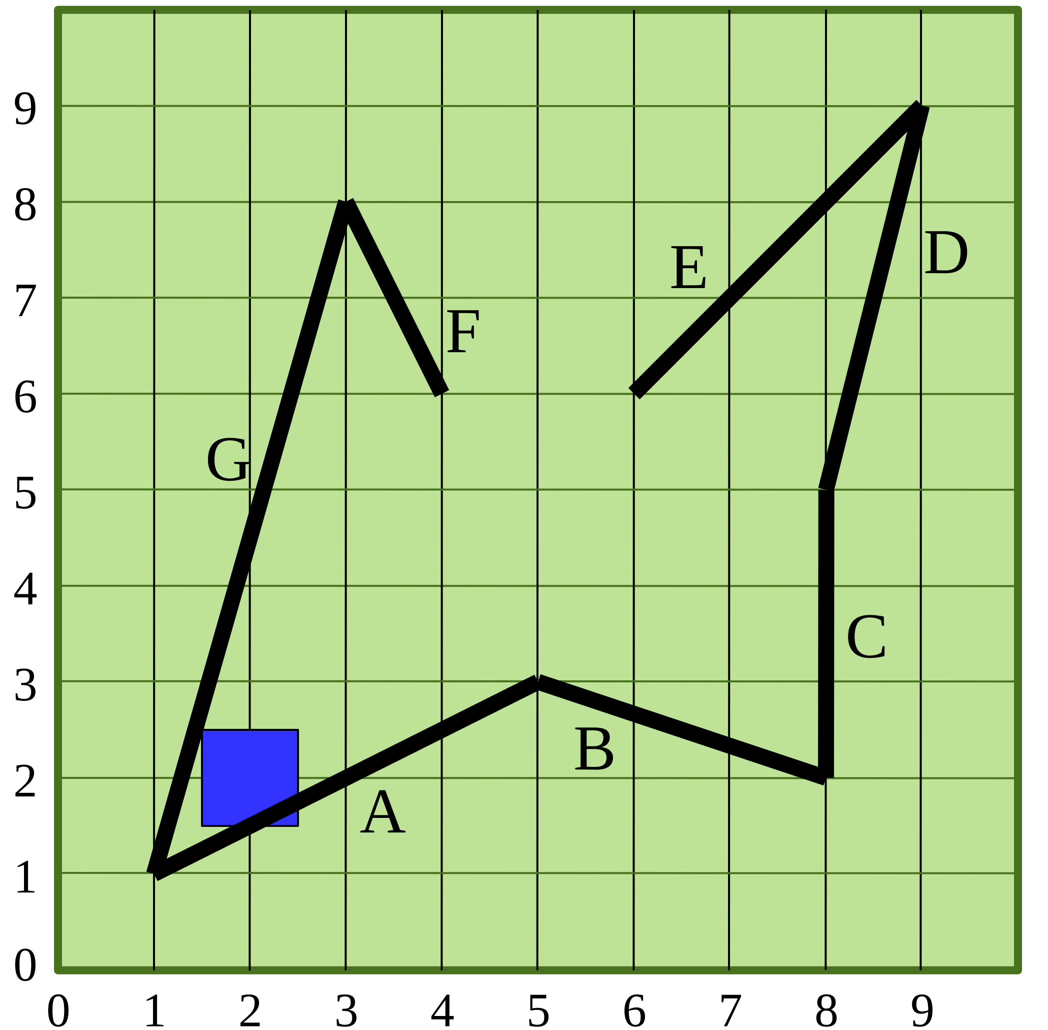


Polygon Rasterization

Edge	x	dx/dy	ymax
G	1 2/7	2/7	8
A	3	4/2	3
B	8	-3/1	3
C	8	0/3	5

- y = 2
- Delete y = ymax edges
- Update x
- Add y = ymin edges
- For each pair x_0,x_1, plot from ceil(x_0) to ceil(x_1) – 1

Edge	ymin
A	1
G	1
B	2
C	2
D	5
E	6
F	6

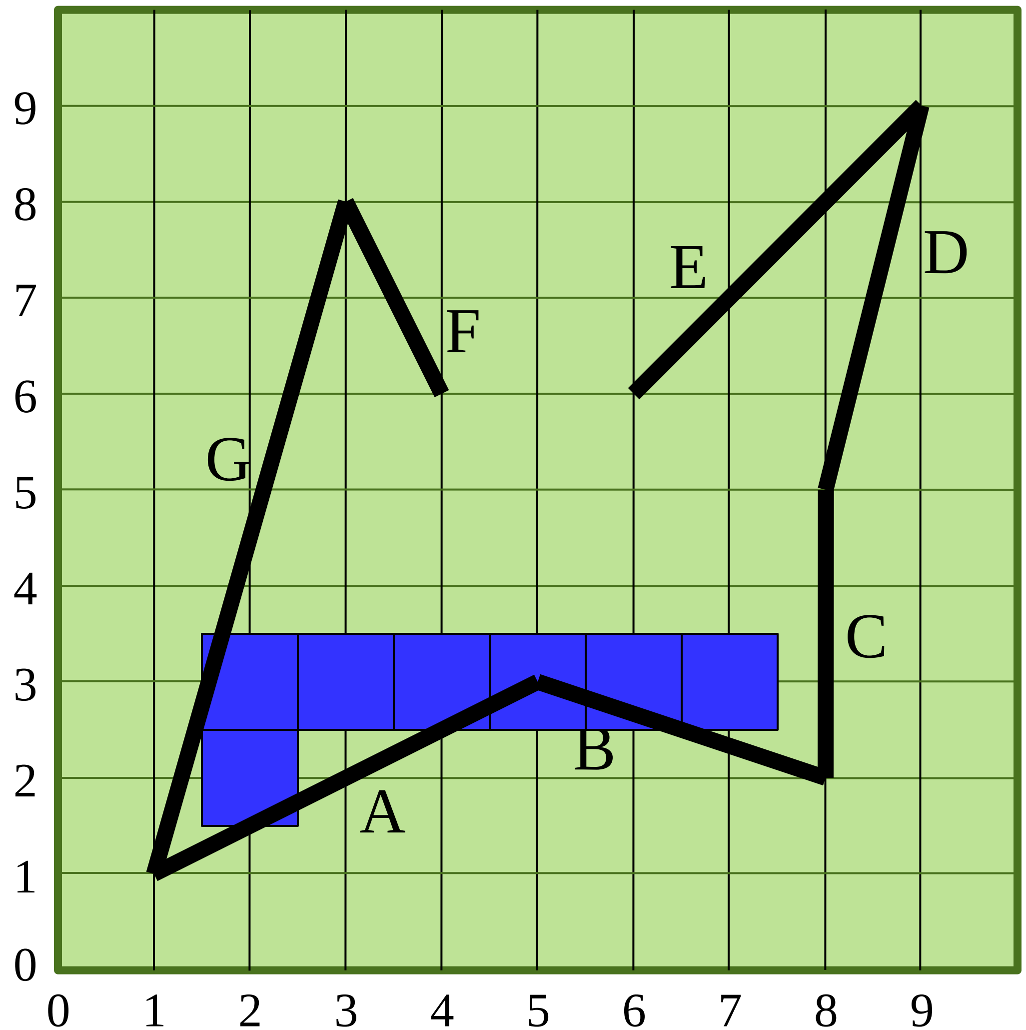


Polygon Rasterization

Edge	x	dx/dy	ymax
G	1 4/7	2/7	8
C	8	0/3	5

- y = 3
- Delete y = ymax edges
- Update x
- Add y = ymin edges
- For each pair x_0, x_1, plot from ceil(x_0) to ceil(x_1) – 1

Edge	ymin
A	1
G	1
B	2
C	2
D	5
E	6
F	6

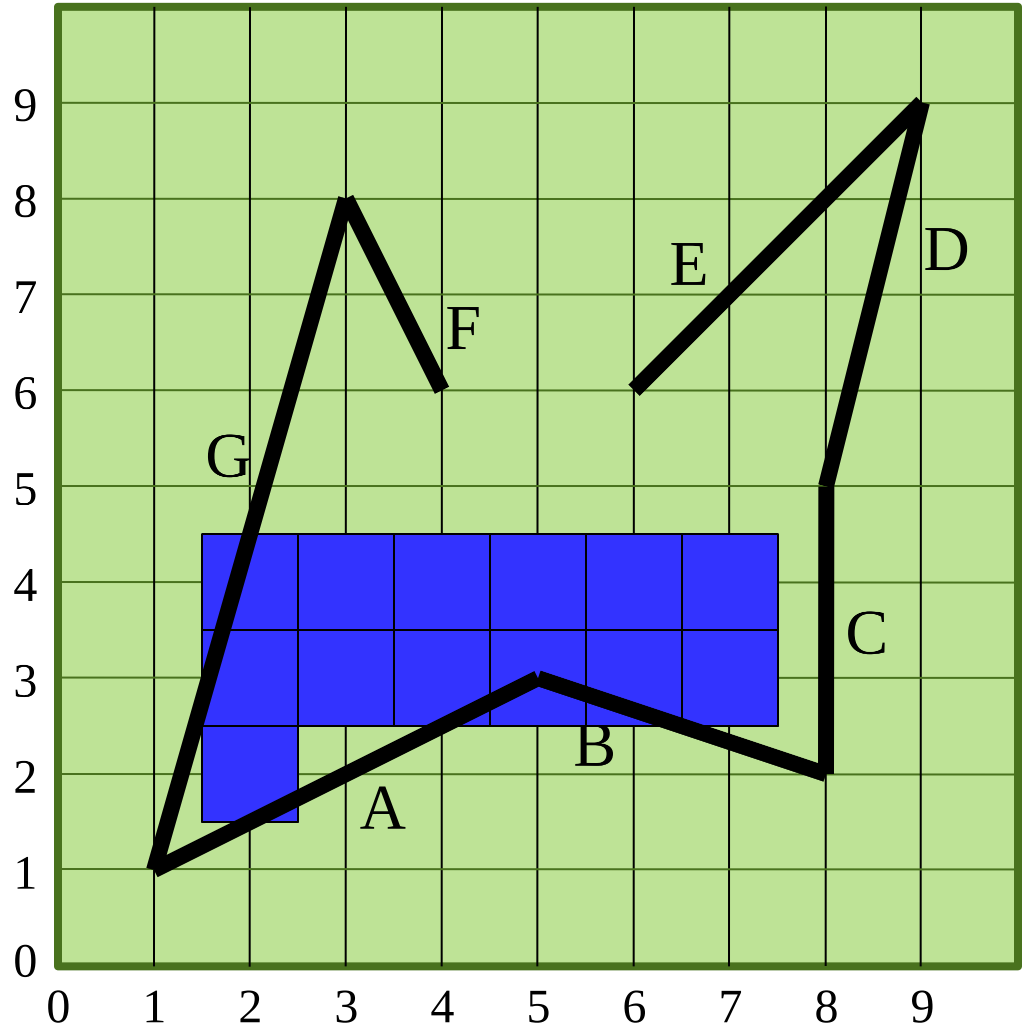


Polygon Rasterization

Edge	x	dx/dy	ymax
G	1 6/7	2/7	8
C	8	0/3	5

- y = 4
- Delete y = ymax edges
- Update x
- Add y = ymin edges
- For each pair x_0,x_1, plot from ceil(x_0) to ceil(x_1) – 1

Edge	ymin
A	1
G	1
B	2
C	2
D	5
E	6
F	6

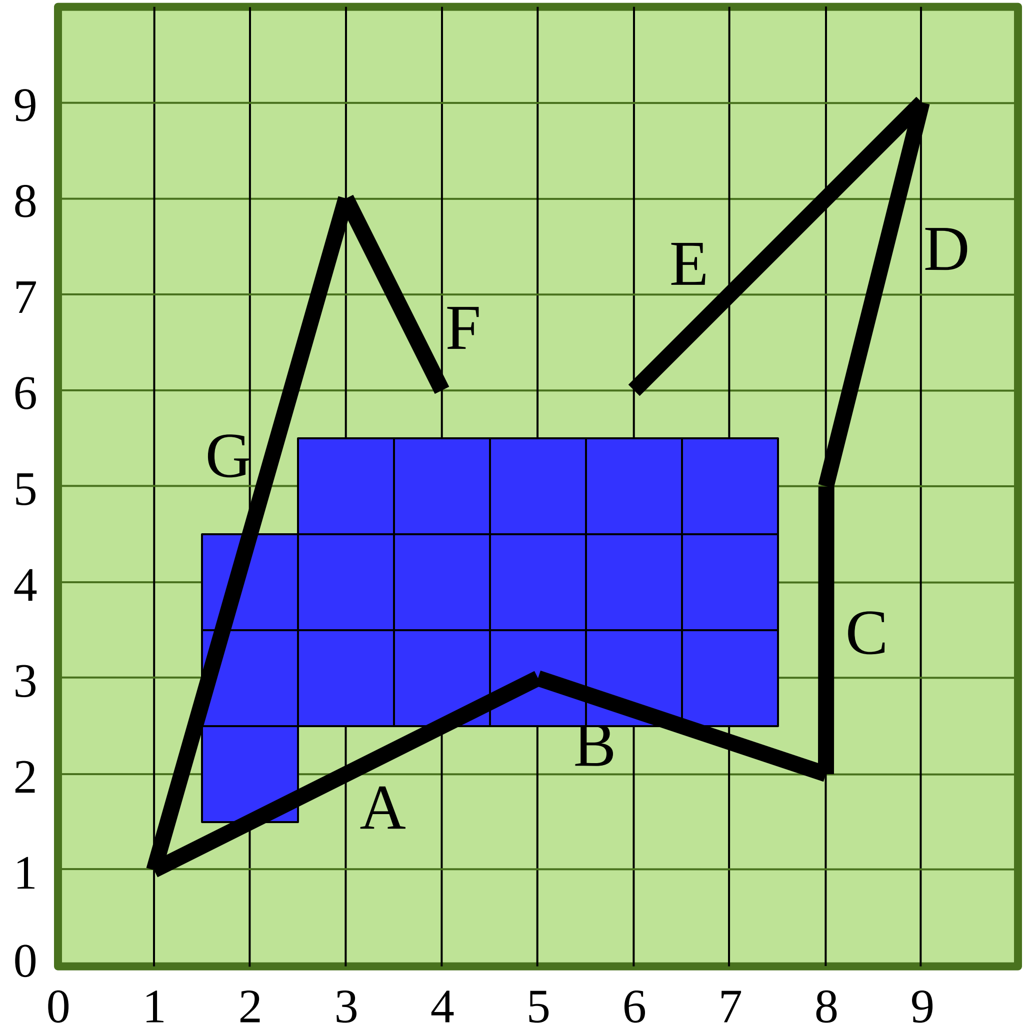


Polygon Rasterization

Edge	x	dx/dy	ymax
G	2 1/7	2/7	8
D	8	1/4	9

- y = 5
- Delete y = ymax edges
- Update x
- Add y = ymin edges
- For each pair x_0,x_1, plot from ceil(x_0) to ceil(x_1) – 1

Edge	ymin
A	1
G	1
B	2
C	2
D	5
E	6
F	6

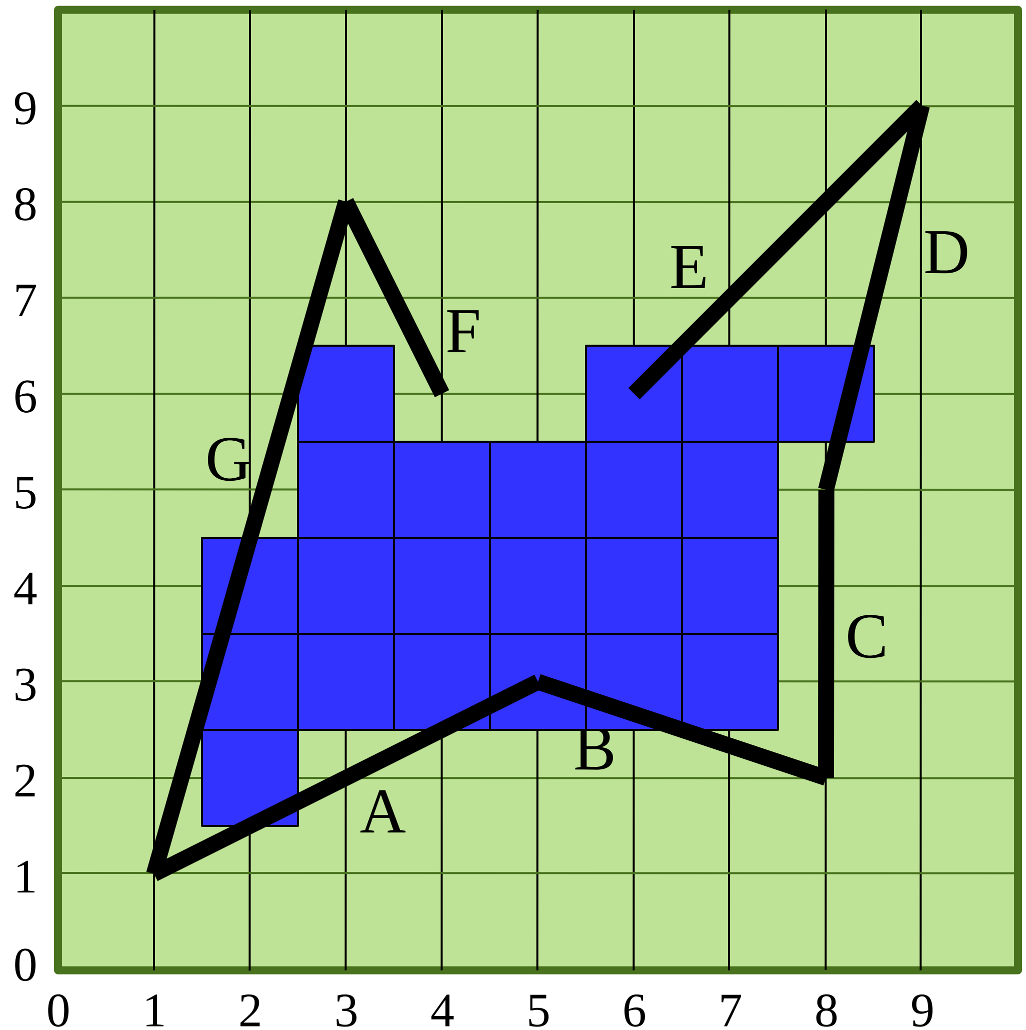


Polygon Rasterization

Edge	x	dx/dy	ymax
G	2 3/7	2/7	8
F	4	-1/2	8
E	6	1/1	9
D	8 1/4	1/4	9

- y = 6
- Delete y = ymax edges
- Update x
- Add y = ymin edges
- For each pair x_0, x_1, plot from ceil(x_0) to ceil(x_1) – 1

Edge	ymin
A	1
G	1
B	2
C	2
D	5
E	6
F	6

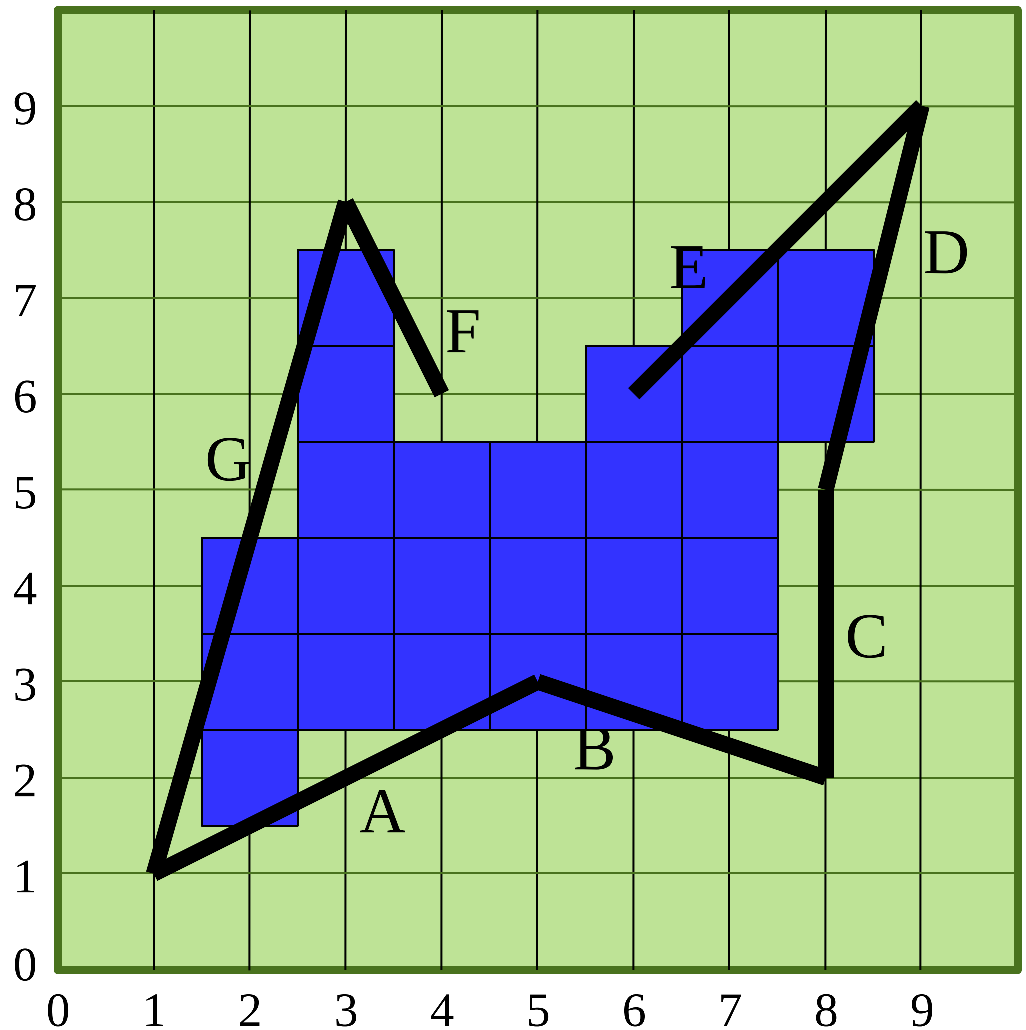


Polygon Rasterization

Edge	x	dx/dy	ymax
G	2 5/7	2/7	8
F	3 1/2	-1/2	8
E	7	1/1	9
D	8 2/4	1/4	9

- y = 7
- Delete y = ymax edges
- Update x
- Add y = ymin edges
- For each pair x_0,x_1, plot from ceil(x_0) to ceil(x_1) – 1

Edge	ymin
A	1
G	1
B	2
C	2
D	5
E	6
F	6

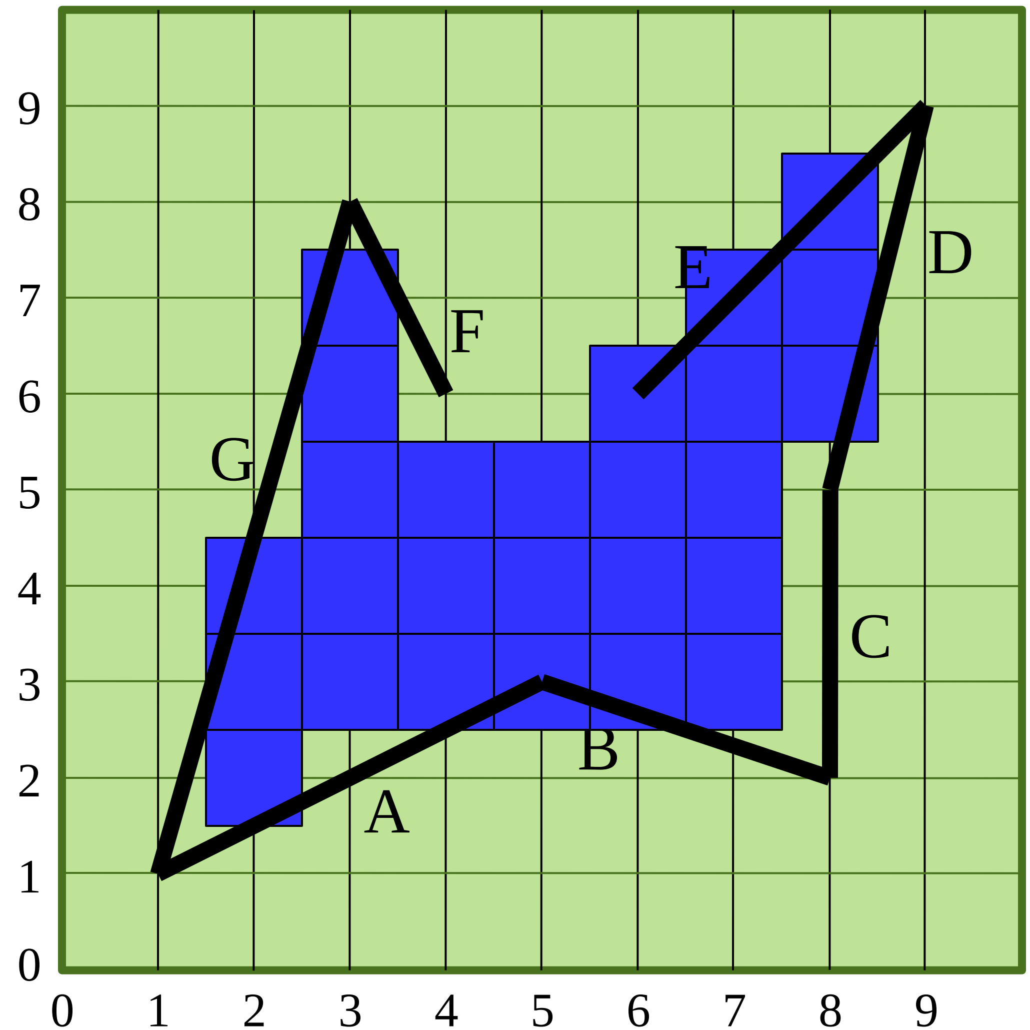


Polygon Rasterization

Edge	x	dx/dy	ymax
E	8	1/1	9
D	8 3/4	1/4	9

- y = 8
- Delete y = ymax edges
- Update x
- Add y = ymin edges
- For each pair x_0, x_1, plot from ceil(x_0) to ceil(x_1) – 1

Edge	ymin
A	1
G	1
B	2
C	2
D	5
E	6
F	6

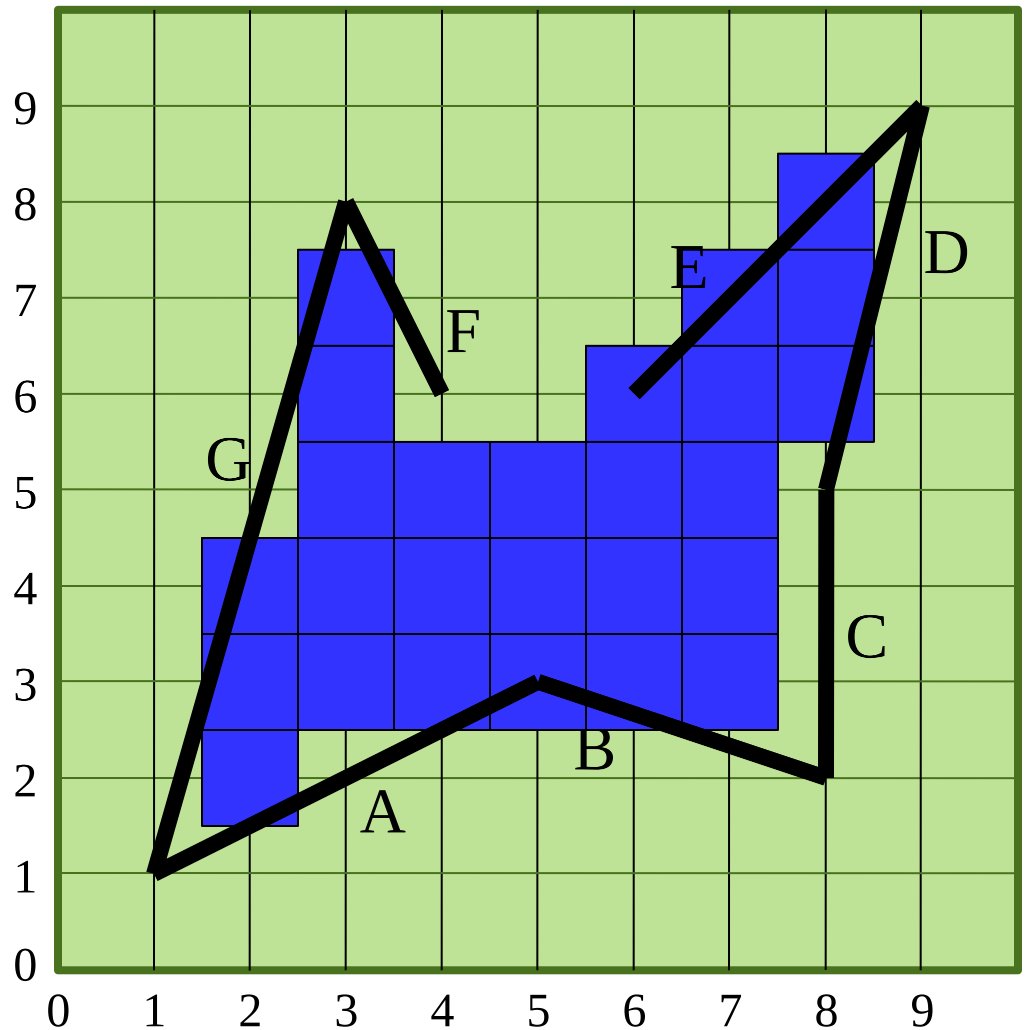


Polygon Rasterization

Edge	x	dx/dy	ymax

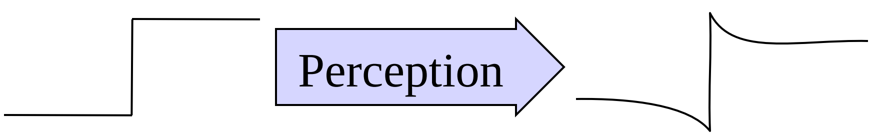
- y = 9
- Delete y = ymax edges
- Update x
- Add y = ymin edges
- For each pair x_0,x_1, plot from ceil(x_0) to ceil(x_1) – 1

Edge	ymin
A	1
G	1
B	2
C	2
D	5
E	6
F	6



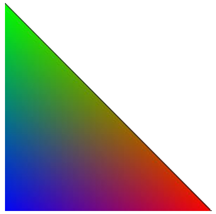
Gouraud Interpolation

- Flat shading
  - Per face normals
  - Color jumps across edge
  - Human visual perception accentuates edges



- Smooth shading
  - Per vertex normals
  - Colors similar across edge
  - Edges become harder to discern

Gouraud Interpolation



- Keep track of R, G, B at edge endpoints
- Compute dR/dy, dG/dy and dB/dy per edge
- Compute dR/dx, dG/dx and dB/dx at each scanline
- Color each pixel

  R += dR/dx

  G += dG/dx

  B += dB/dx