CSE-170 Computer Graphics

Lecture 13

Rasterization

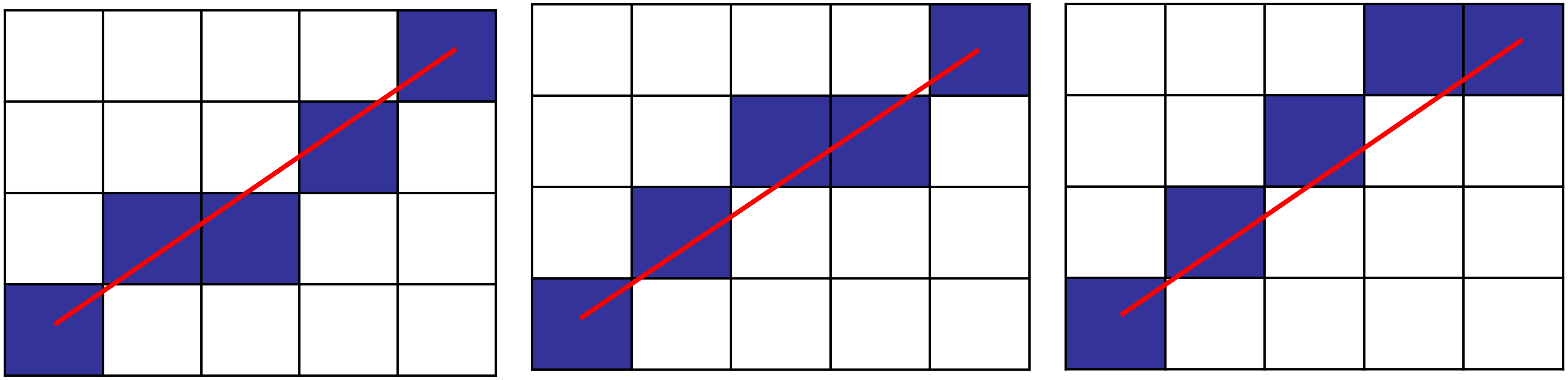
Dr. Renato Farias
rfarias2@ucmerced.edu

Rasterization

- Also known as *scan-conversion*
- All primitives considered by a graphics system have to be rasterized at some point
  - Lines
  - Lines with thickness
  - Polygonal lines with thickness
  - Triangles
  - Polygons
  - Circles
  - etc

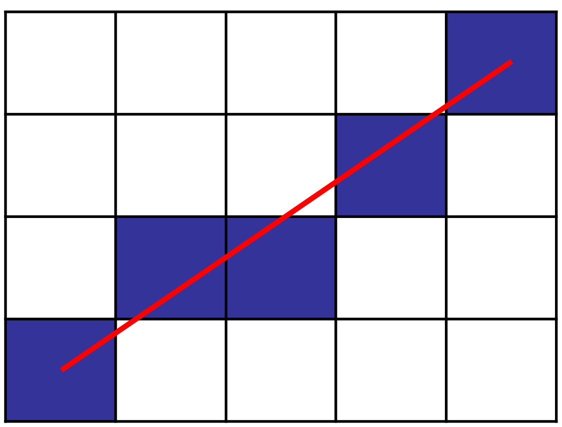
Scan Converting Lines

- Determine the sequence of pixels that lie as close to the ideal line as possible
  - No gaps, best approximation, consistency, etc.



Scan Converting Lines

- Simplest approach (example considers m in [0,1])
  - Given endpoints, compute slope $m = \Delta y/\Delta x$
  - Increment x by 1, starting with leftmost point
  - Calculate $y_i = mx_i + B$
  - Paint pixel (x_i, round(y_i))



INEFFICIENT: too many floating-point operations!

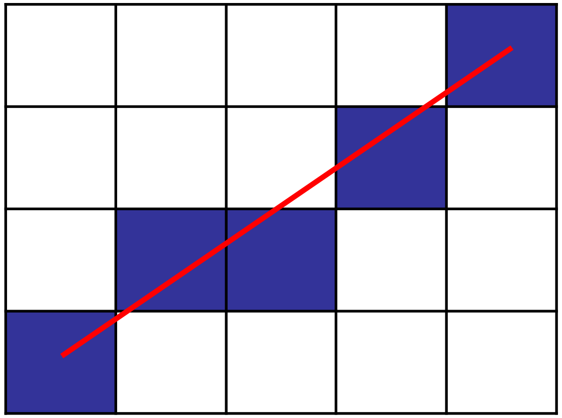
(Note: for simplicity we consider the line as going from left to right and up but extending to any quadrant is easy)

Scan Converting Lines

- Incremental Algorithm / DDA *(digital differential analyzer)*

```
void line ( int x0, int y0, int x1, int y1 )
   int x;
   float deltay = y1 - y0;
   float deltax = x1 - x0;
   float m = deltay / deltax;
   float y = y0;

   for ( x = x0; x <= x1; x++ )
    { paint ( x, round(y) );  // round(y): int(y+0.5f), y>0
      y = y + m;
    }
```



- Ok for most (short) lines, but can accumulate error

- Needs floating-point operations

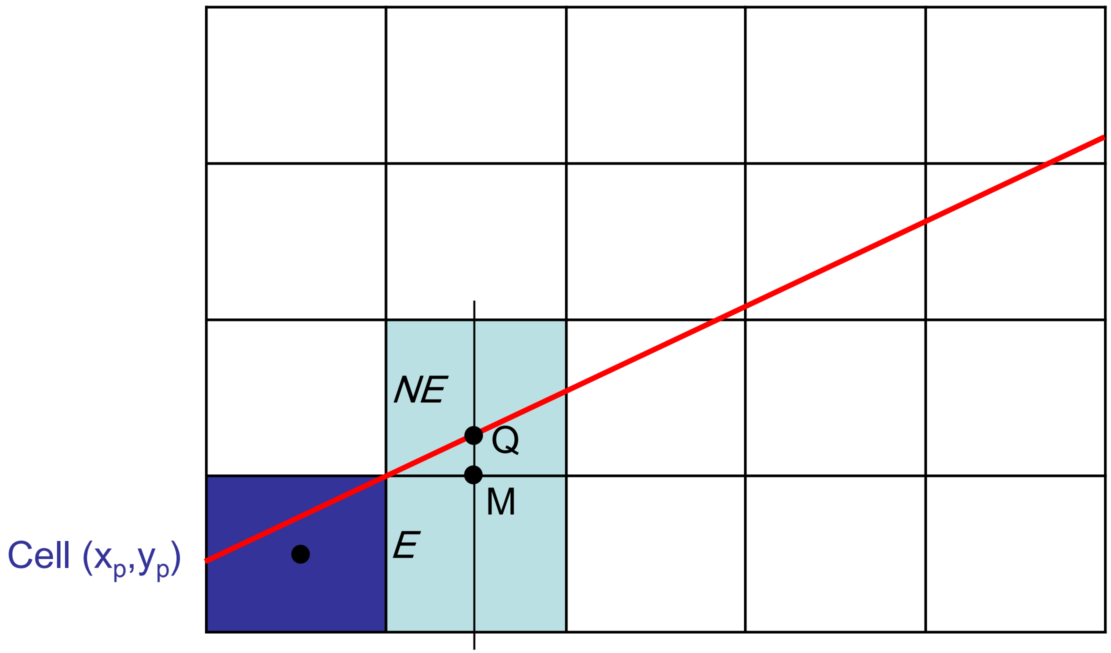
Bresenham

- Bresenham (1965)
  - Classic algorithm using only integer arithmetic
  - No round function, incremental calculation
  - Applicable as well to circles, but not conics
  - Best fit, minimizes error (dist. to true shape)

- Extension/variation: Midpoint algorithm
  - For lines and circles it selects the same pixels as Bresenham
  - Handles conics

Midpoint

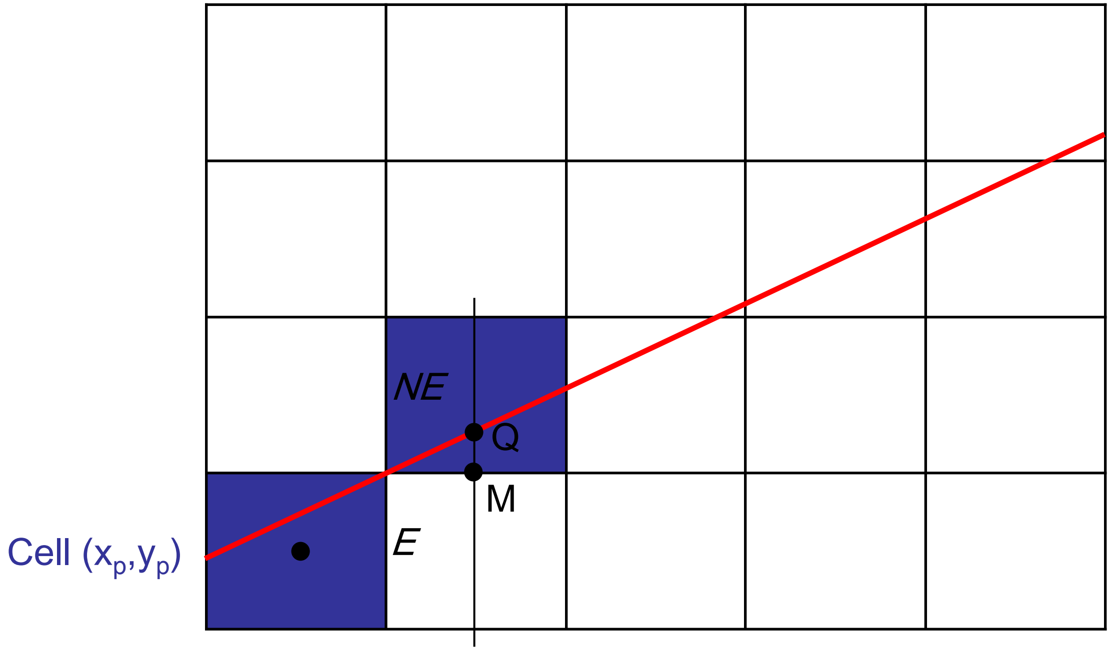
- Midpoint Line Algorithm (example considers *m* in [0,1])
  - Test: on which side of the midpoint M does the line lie?
  - If above, *NE* cell is chosen, otherwise *E* cell is chosen



M = (x_p+1, y_p+0.5)

Midpoint

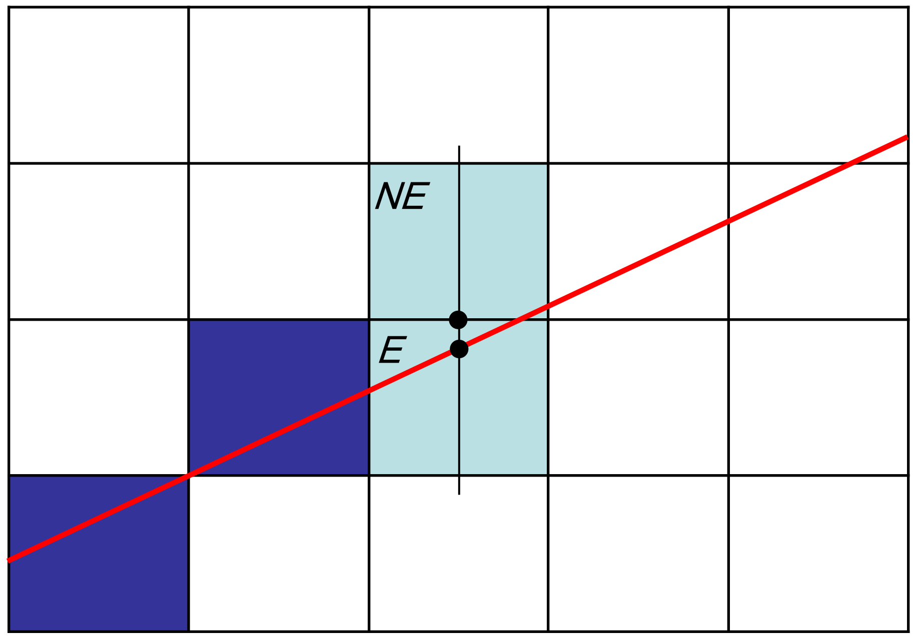
- *NE* chosen



$M = (x_p+1,\ y_p+0.5)$

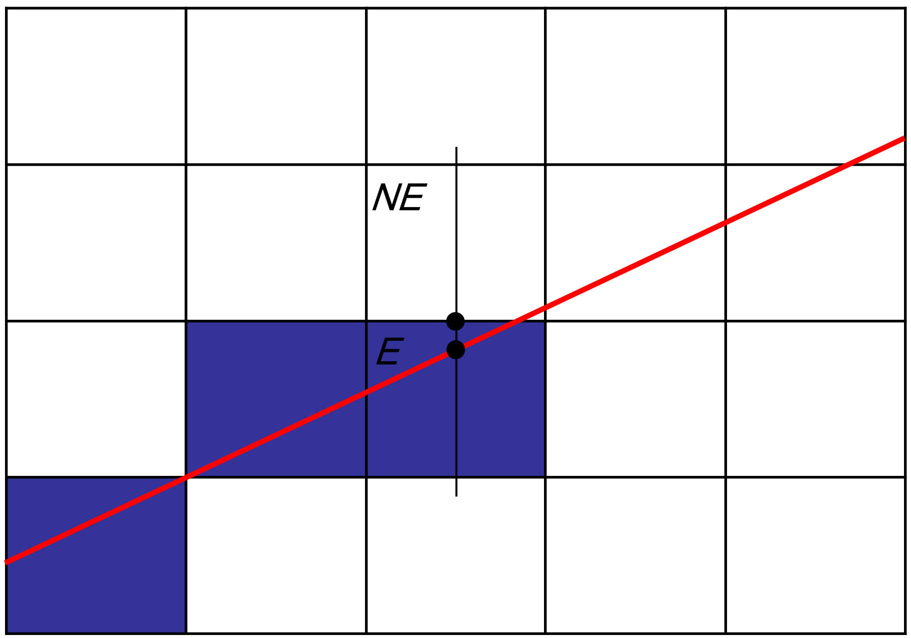
Midpoint

- Test *E* and *NE*



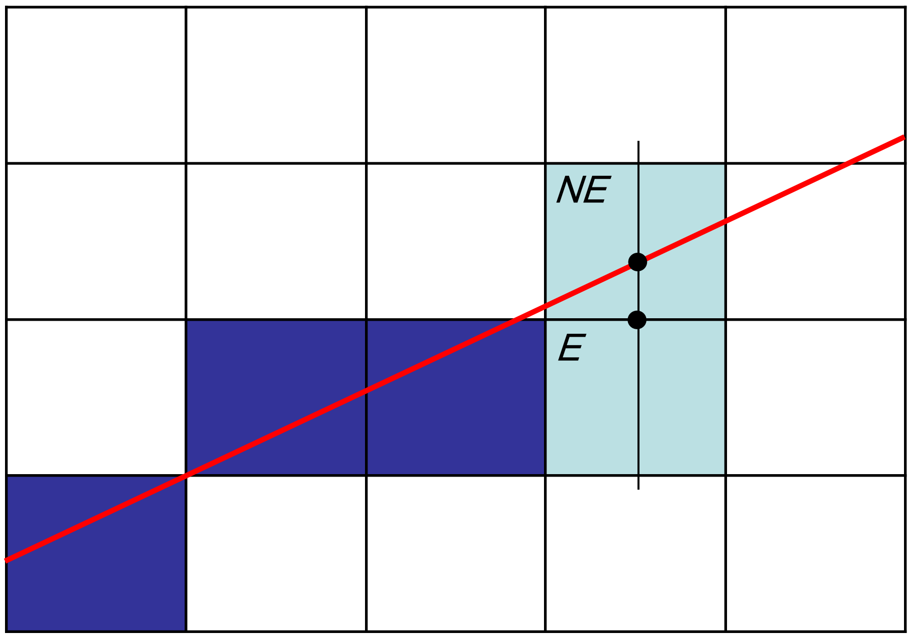
Midpoint

- *E* chosen



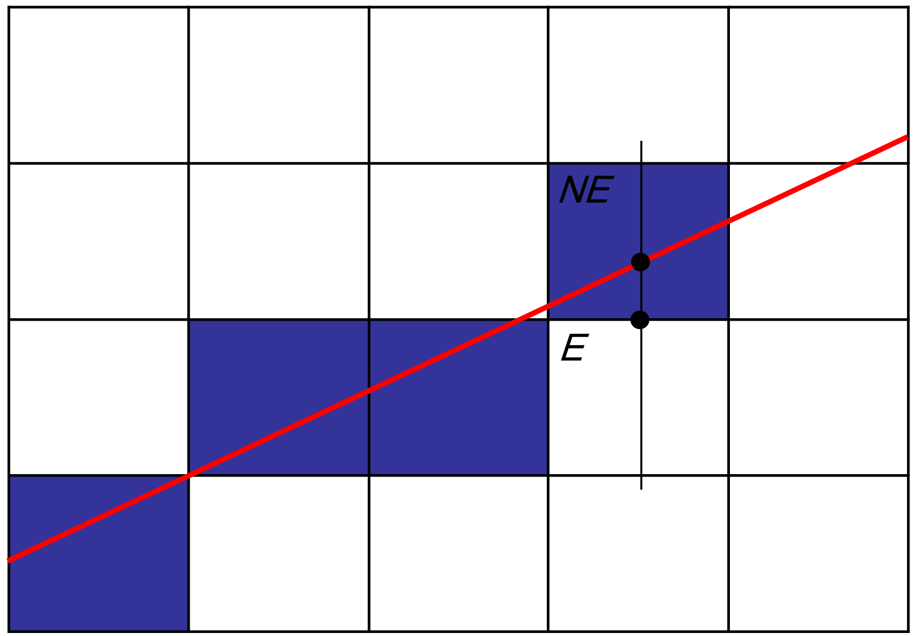
Midpoint

- Test *E* and *NE*



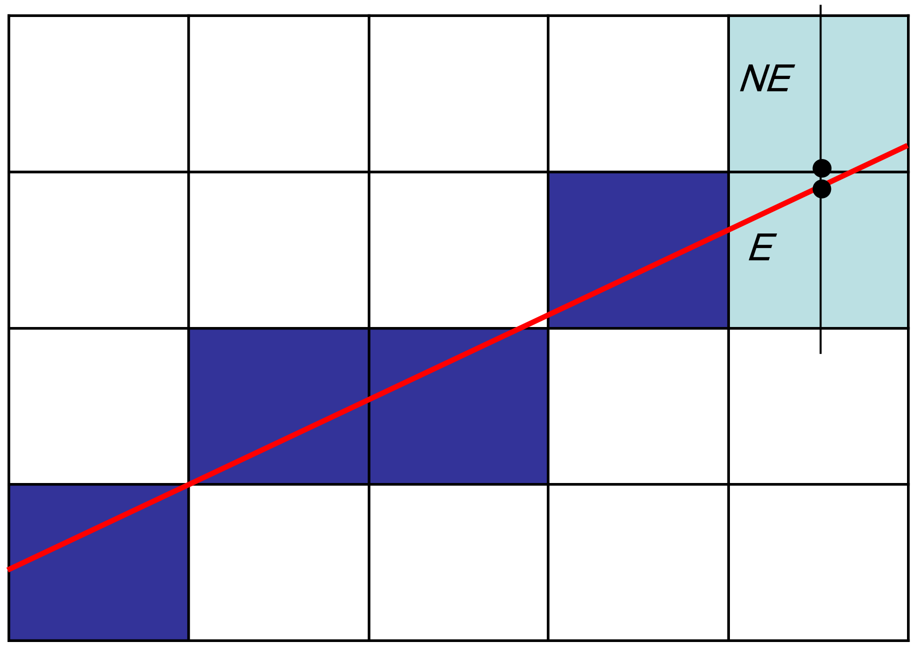
Midpoint

- *NE* chosen



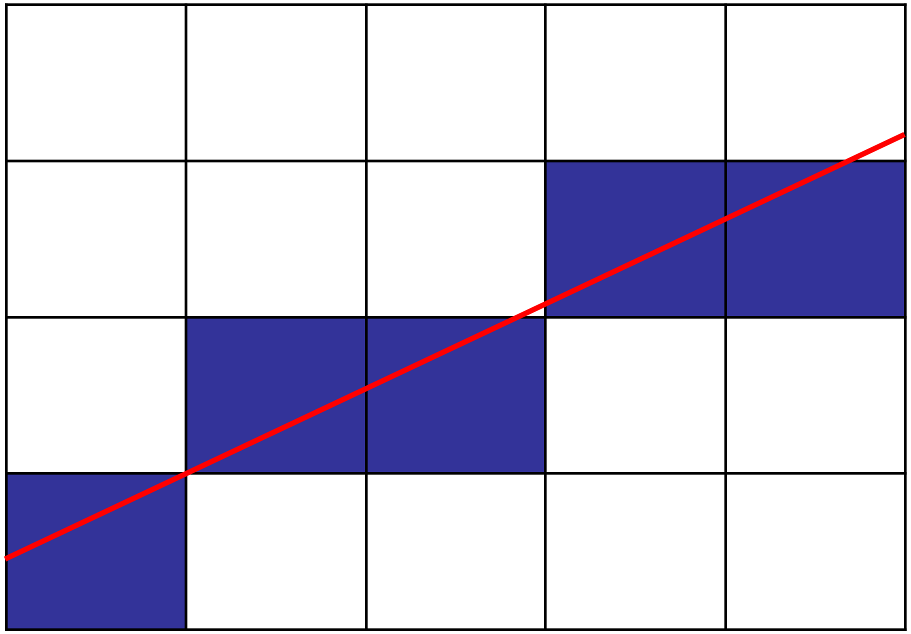
Midpoint

- Test *E* and *NE*



Midpoint

- *E* chosen



Midpoint Test

- Midpoint test
  - Implicit line: $F(x,y)$: $ax+by+c=0$

  - Let $\Delta x = x1-x0$, $\Delta y = y1-y0$

    $y = (\Delta y/\Delta x)x + B$ (B is the Y intercept, for ex., $B = y1-m*x1$)

    $F(x,y)$: $\Delta y\, x - \Delta x\, y + B\Delta x = 0$

    $F(x,y)$ is:

    0, on the line
    >0, for points below the line
    <0, for points above the line

    midpoint criterion:

    evaluate sign of $F(x_p+1, y_p+0.5)$

Midpoint Test

F(x,y): $ax+by+c=0$, d is our decision variable:

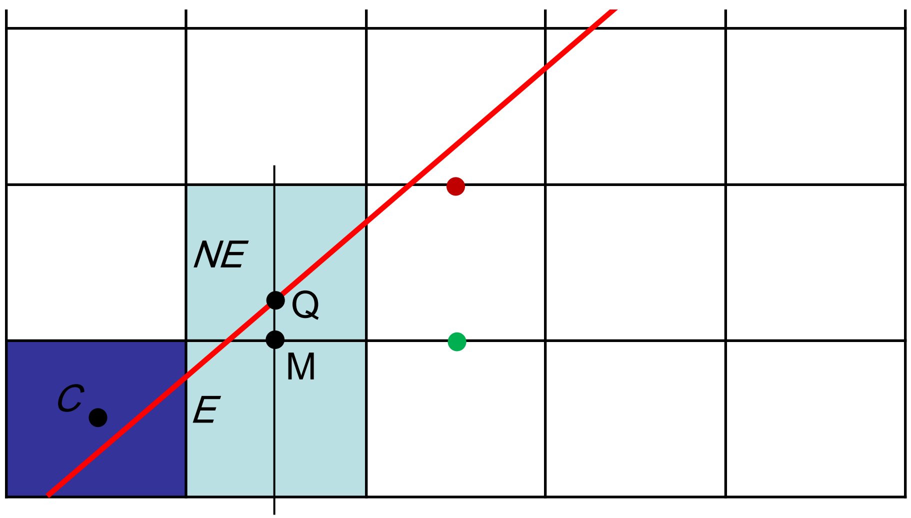
$d_{cur} = F(x_p+1, y_p+0.5) = a(x_p+1)+b(y_p+0.5)+c$

If *E* is taken:

$d_{next} = a(x_p+2)+b(y_p+0.5)+c = d_{cur} + a$ *(green point)*

If *NE* is taken:

$d_{next} = a(x_p+2)+b(y_p+1.5)+c = d_{cur} + a + b$ *(red point)*



Prior cell C (x_p,y_p)

$M = (x_p+1, y_p+0.5)$

Midpoint Test

Start : (x_0, y_0) 1^{st} midpoint: $(x_0+1, y_0+0.5)$

$$F(x_0+1, y_0+0.5) = a(x_0+1)+b(y_0+0.5)+c$$
$$= ax_0 + by_0 + c + a + b/2$$
$$= F(x_0, y_0) + a + b/2$$
$$\Rightarrow d_{start} = dy - dx/2$$

To eliminate the fraction, we multiply F by 2:

$$\Rightarrow d_{start} = 2dy - dx$$

If *E* is taken:

$$d_{next} = d_{cur} + 2a$$

If *NE* is taken:

$$d_{next} = d_{cur} + 2(a + b)$$

reminder: $a=\Delta y$, $b=-\Delta x$

Midpoint Algorithm

```
void midpointline ( int x0, int y0, int x1, int y1 )

int deltax = x1-x0;
int deltay = y1-y0;
int d = deltay+deltay - deltax; // initial value of d (2dy-dx)
int incE = deltay+deltay;       // increment to move to E (2dy)
int incNE = deltay+deltay-deltax-deltax; // inc to move to NE (2dy-2dx)
x = x0;
y = y0;

paint ( x, y );                 // first point

while ( x<x1 )
 { if ( d<0 )
    { d = d + incE;             // great, only integer arithmetic !!!
      x = x + 1;
    }
   else
    { d = d + incNE;
      x = x + 1;
      y = y + 1;
    }
   paint ( x, y );              // paint current point
 }
```

Issues

- Endpoint order
  - Ensure that p0,p1 and p1,p0 generates same pixels:
    - Change choice used when d=0, or
    - Switch endpoints to ensure same result
- So far, we considered integer endpoints
  - Closest pixel from real points can be used
  - Additional care needed when drawing clipped lines, to ensure the slope remains the same

Other primitives

- Now that we know how to efficiently scan-convert lines
  - Same principles can be used to scan-convert other primitives
    - Polylines
    - Rectangles
    - Polygons
    - etc.

Scan Converting Circles

- Circle has eight-way symmetry
  - CirclePaint (x, y)

    Paint (x, y);

    Paint (y, x);

    Paint (y, -x);

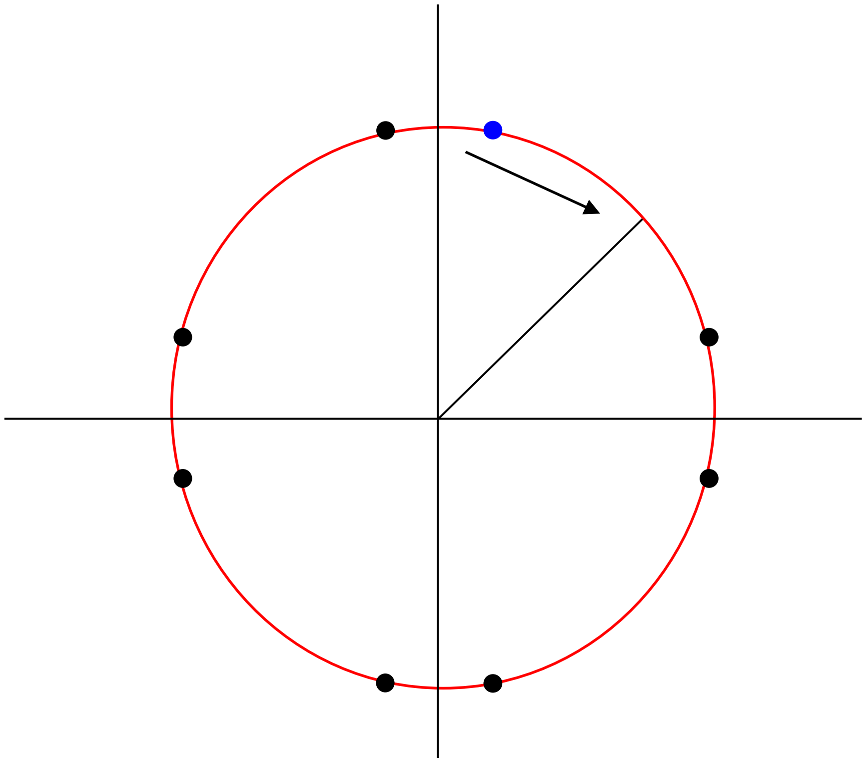
    Paint (x, -y);

    Paint (-x, -y);

    Paint (-y, -x);

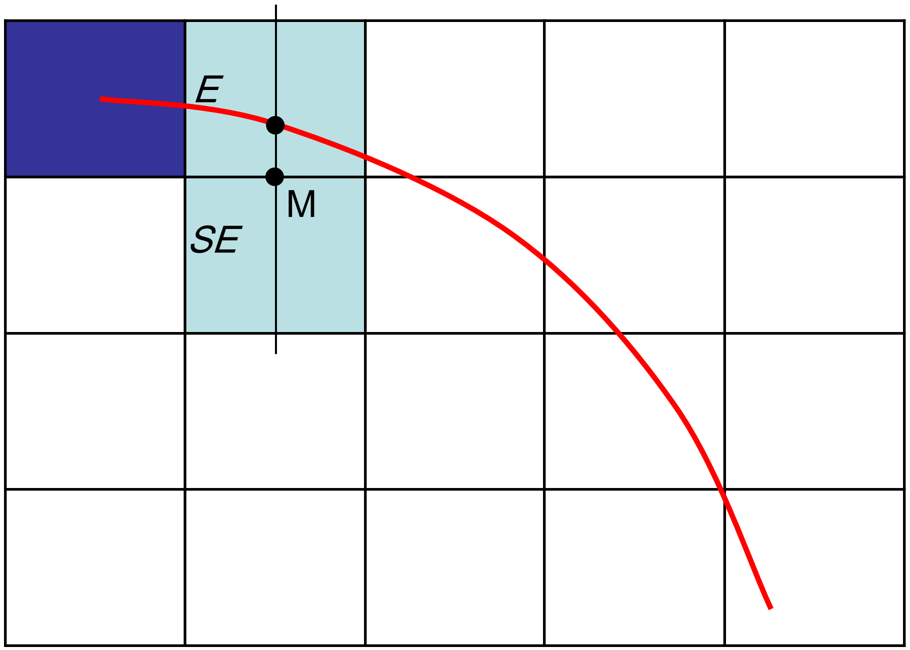
    Paint (-y, x);

    Paint (-x, y);



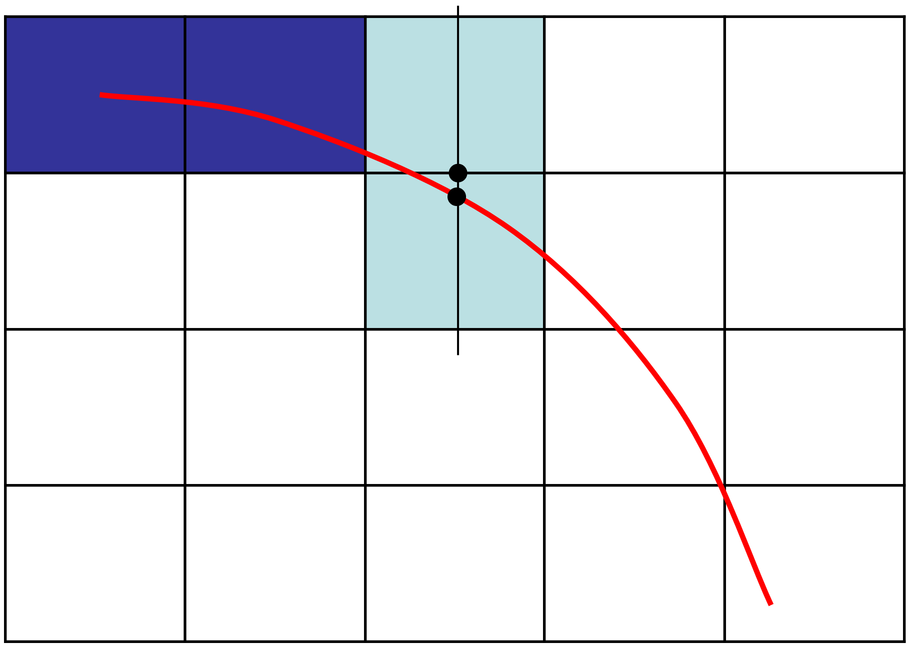
Midpoint Circle Algorithm

- Exactly same logic as the midpoint line algorithm!



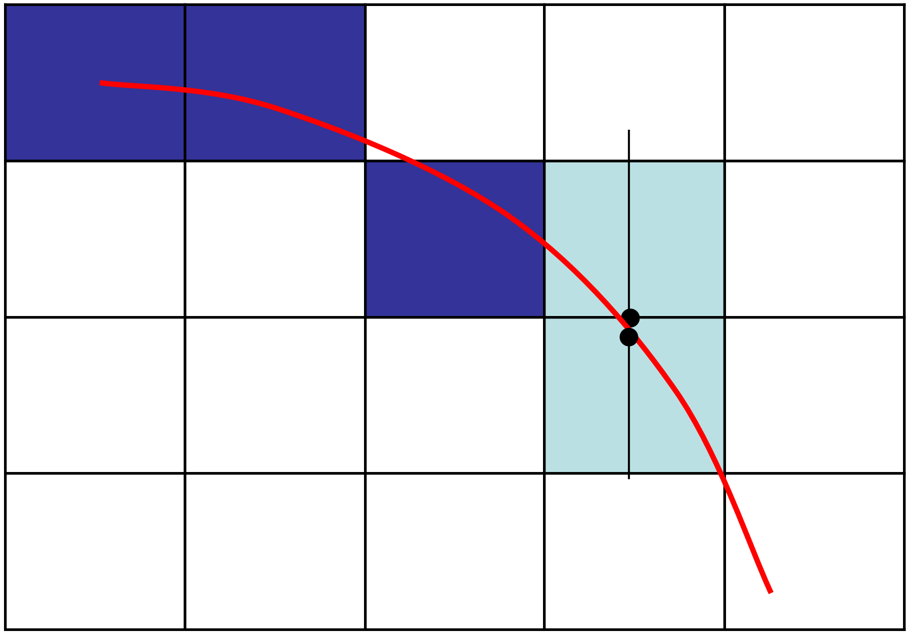
Midpoint Circle Algorithm

- Exactly same logic as the midpoint line algorithm!



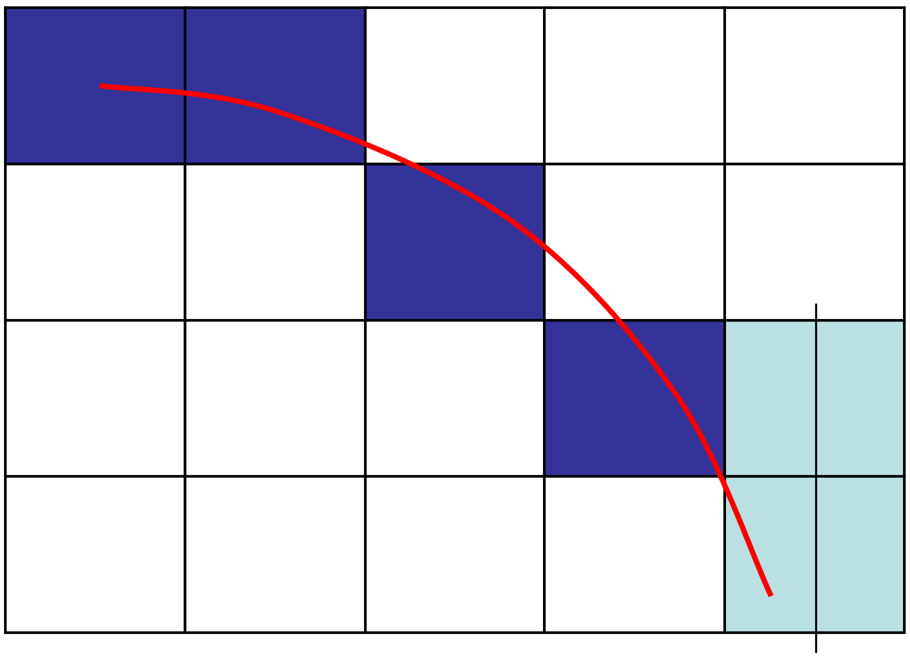
Midpoint Circle Algorithm

- Exactly same logic as the midpoint line algorithm!



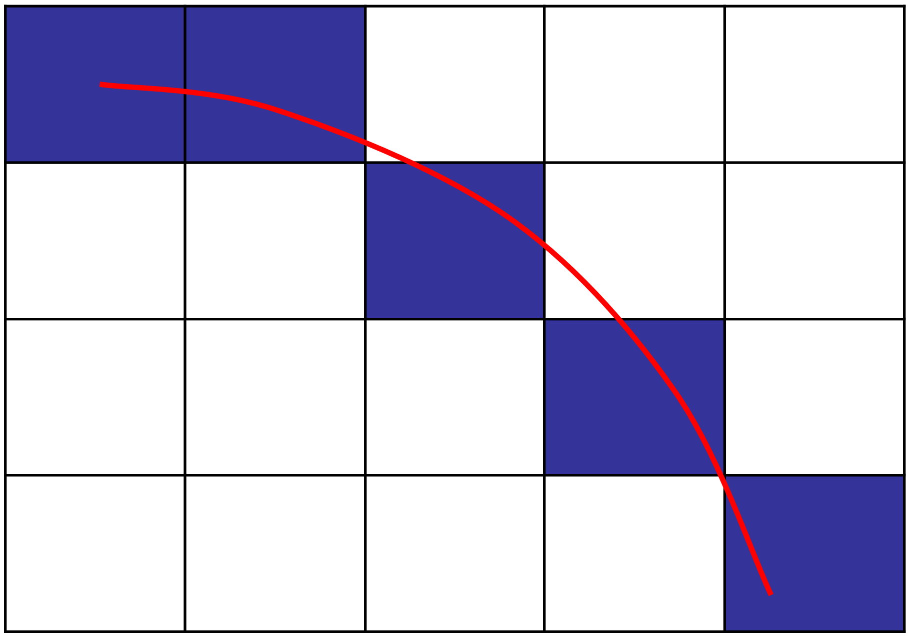
Midpoint Circle Algorithm

- Exactly same logic as the midpoint line algorithm!



Midpoint Circle Algorithm

- Exactly same logic as the midpoint line algorithm!

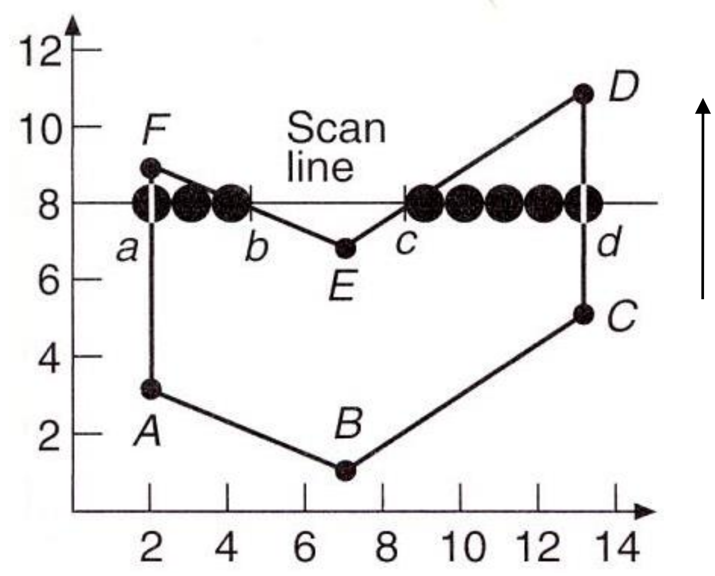


Note: this arc is just for illustration, it is not a true circle octant!

Filling Primitives

Scan Converting Polygons

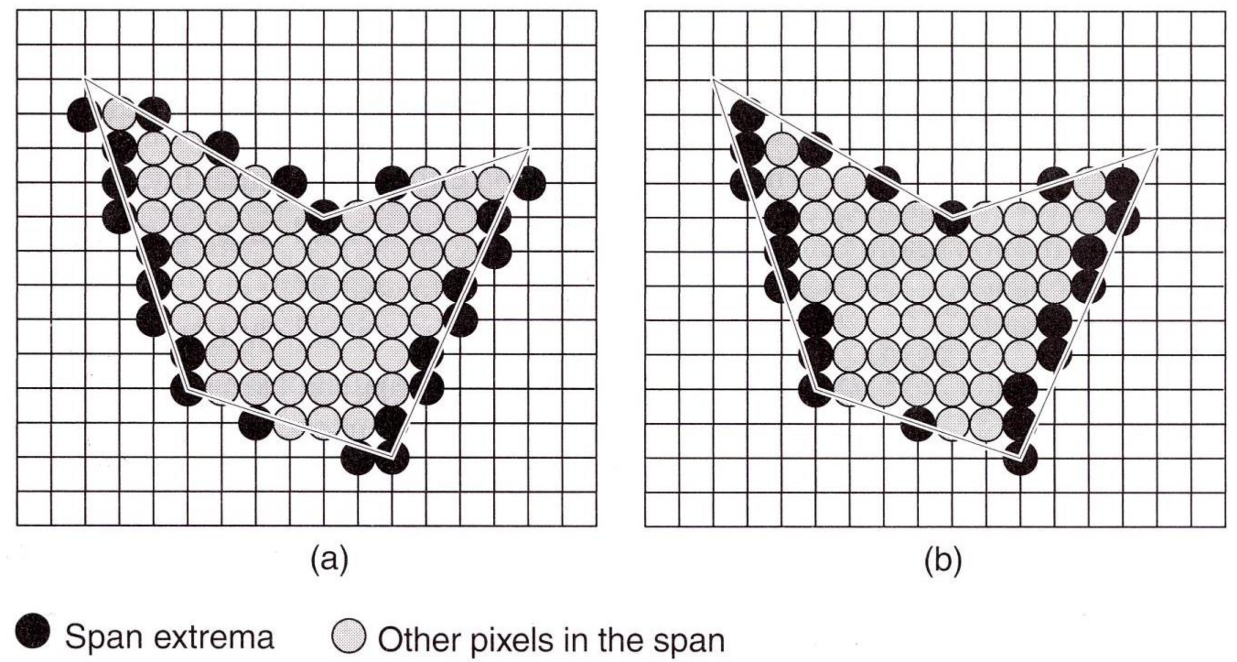
- Scan Line
  – Computes spans that lie between left and right edges of the polygon
  – Handles convex and concave polygons



Scan Converting Polygons

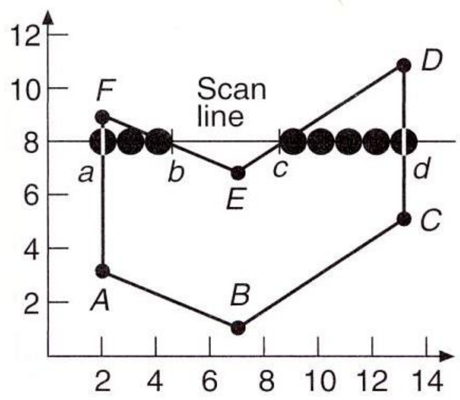
- Scan Line
  - Example of spans

    (Note: here, pixels are the intersection points in the grids, and not the grid cells)



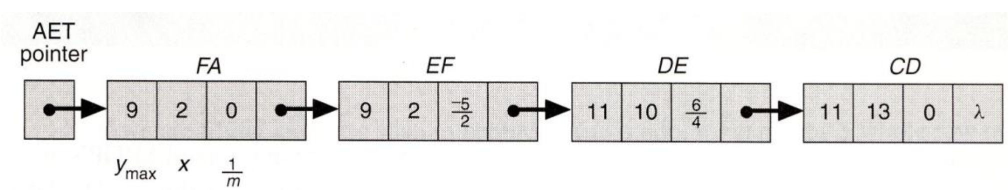
Scan Converting Polygons

- Scan Line
  - Same midpoint technique is used to calculate and update the extremes of the spans
    - No need to calculate analytically the intersections between the polygon edges and the scan line
  - Spans are filled in 3 step process
    - Find scan line intersections with polygon edges
    - Sort intersections by increasing x
    - Fill pixels using the odd-parity rule:
      - Initially even
      - invert on each intersection
        - » draw when odd
        - » do not draw when even

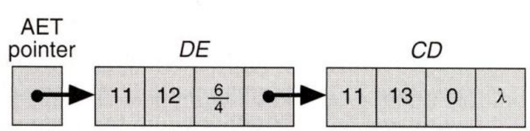


Scan Converting Polygons

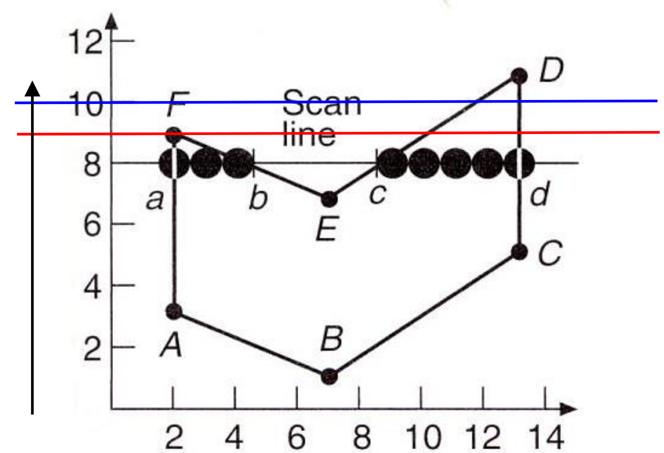
- Data Structure
  - Active-Edge Table (AET)
    - Edges sorted on their x intersection values
    - Edges are inserted/removed as the scan line traverses the polygon



(a) scan line 9



(b) scan line 10



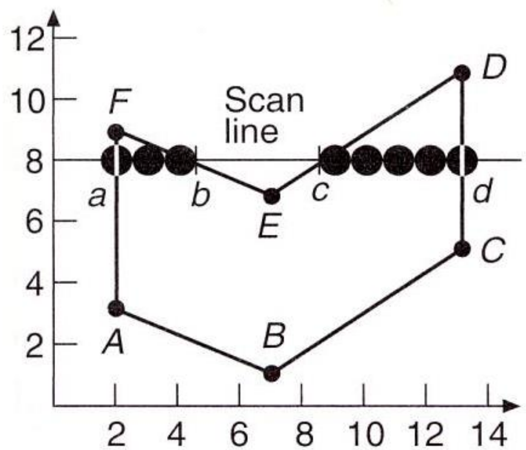
Scan Converting Polygons

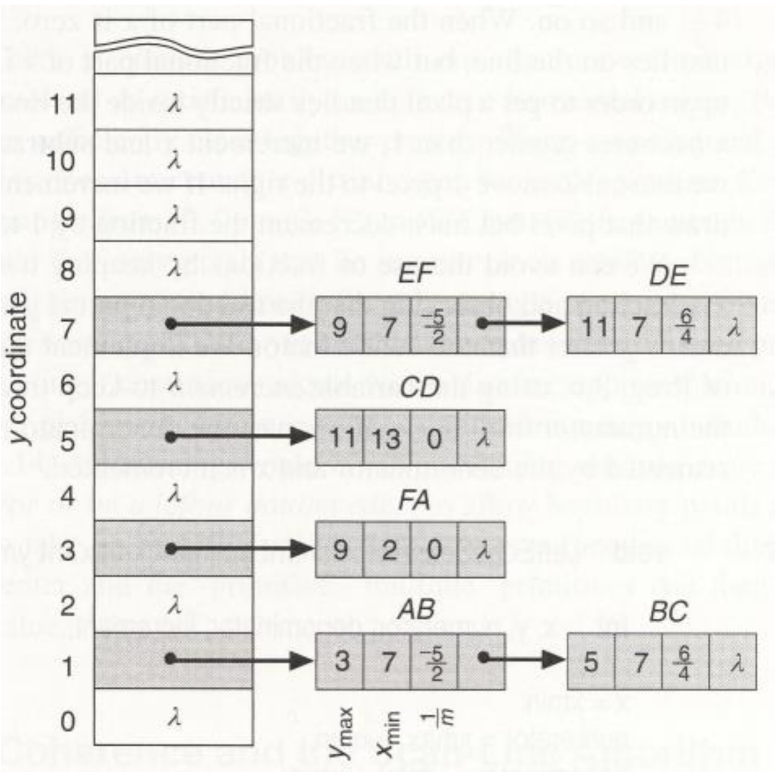
- To make addition of edges to the EAT efficient, a global Edge Table ET containing all edges sorted by their smallest y coordinate is used
  - Use bucket sort, buckets are the number of scan lines
  - Within each bucket, edges are in increasing x order of the lower endpoint

Scan Converting Polygons

– Edge Table:

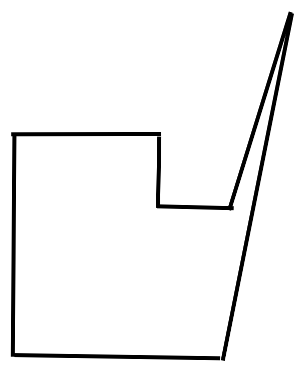
- 1 Bucket for each scan line, sorted by smallest y

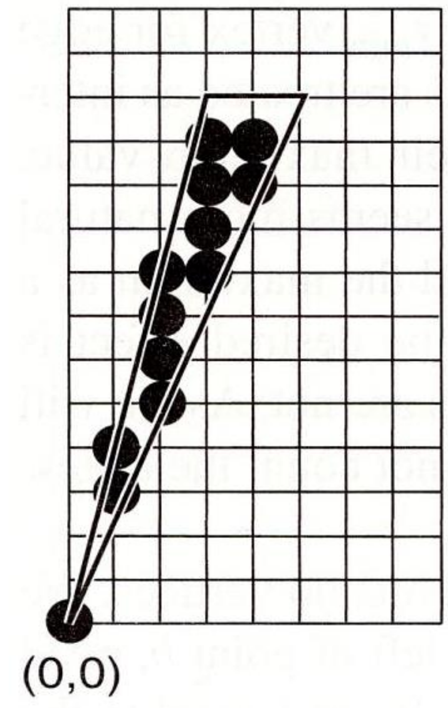




Scan Converting Polygons

- Issues
  - – Horizontal edges
  - – Slivers
  - – Calculating the intersections
  - – Exploiting edge coherence
  - – etc.



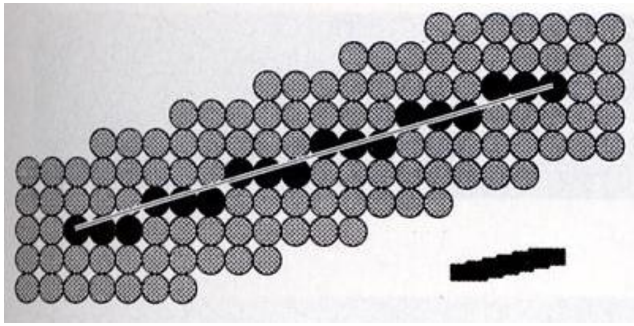


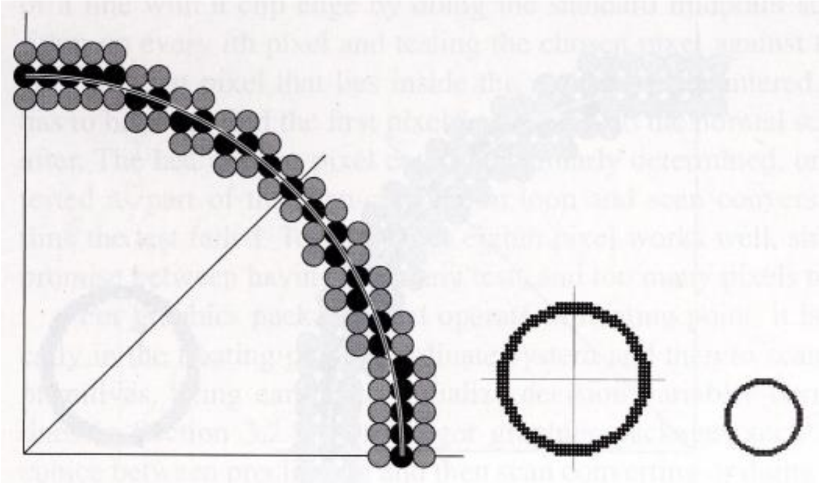
Convex Polygons

- It is simpler to deal with convex polygons
  - Easier management of scan lines
  - Triangles even simpler (OpenGL case)
- How to decompose arbitrary polygons in convex pieces?
  - Scan line algorithm for trapezoidal decomposition
  - Polygon triangulation methods
    - Optimal method is $O(n)$
  - Simplest approach: triangulation by “ear cuts”
    - $O(n^2)$

Related Topics

- Thick primitives
  - Effects of using different “pens”
- Connections between thick lines
  - Round
  - Sharp, etc





Related Topics

- Pattern filling
  - Several options here, but no support in OpenGL 4

- Antialiasing
  - All systems support it, but sometimes it is preferable to not do antialiasing
    - Ex: some fonts are sharper without antialiasing