Introduction to Computer Graphics

7. Rasterization

I-Chen Lin
National Yang Ming Chiao Tung University

Textbook: E.Angel, D. Shreiner Interactive Computer Graphics, 6th Ed., Pearson
Ref: D.D. Hearn, M. P. Baker, W. Carithers, Computer Graphics with OpenGL, 4th Ed., Pearson

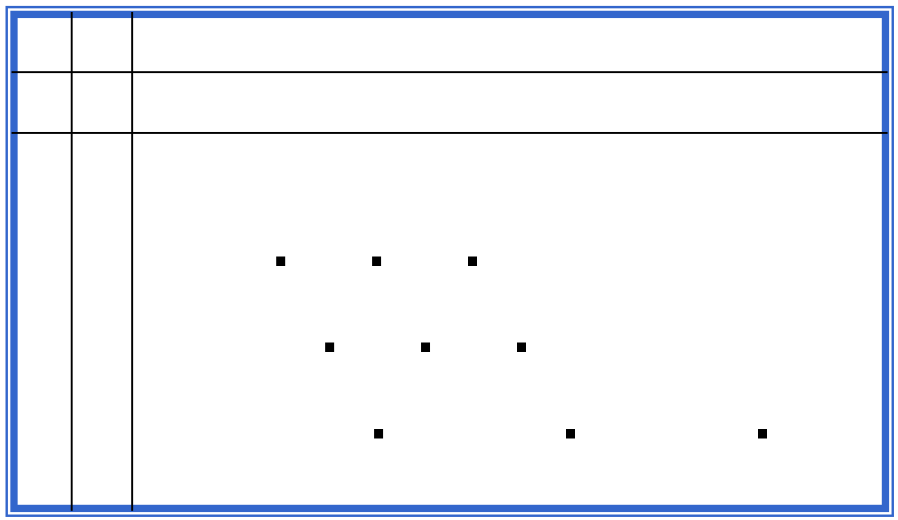
Intended Learning Outcomes

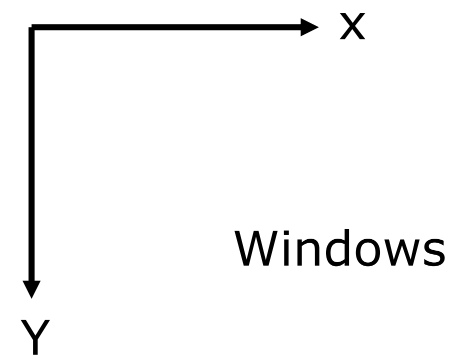
- On completion of this chapter, a student will be able to:
  - **Identify** the key issues of line and circle drawing.
  - **Compare** the primary line and circle drawing algorithms.
  - **Explain** the key issues of polygonal area filling.
  - **Describe** the primary methods for polygon filling.

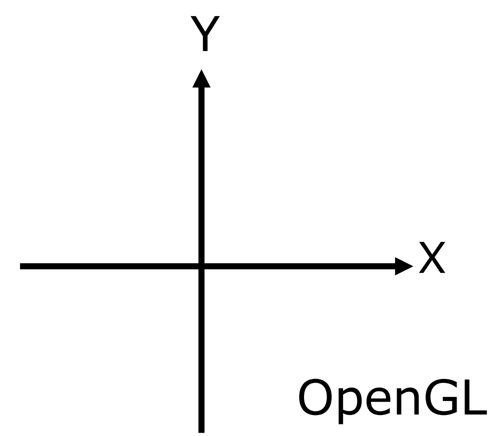
Outline

- Draw primitives in discrete screen space.
- 2D graphics primitives
  - Line drawing
  - Circle drawing
- Area filling
  - Polygons
    - Convex polygons
    - General polygons

Discrete Video Screen







- Assigning pixel values by
  - Functions:
    - e.g. SetPixel(x, y, color)
  - Buffer or arrary:
    - e.g. FrameBuf[x][y] = color

How to Draw Primitives ?

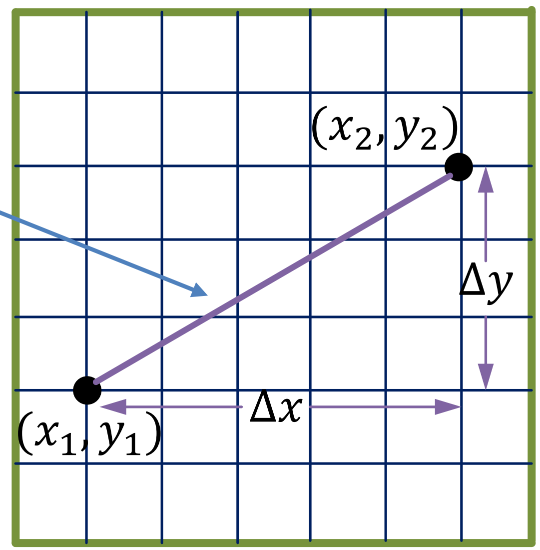
- From math representation to screen.
- In addition to “brute-force”, how to improve the efficiency of computation or memory usage.
- Primitives
  - Lines
  - Circles
  - Curves
  - ……

Line-Drawing Algorithms

- Start with a line segment in window coordinates with integer values for endpoints.

$$\mathbf{y = mx + h}$$

$$\boldsymbol{m} = \frac{\Delta y}{\Delta x}$$



DDA Algorithm

- Digital Differential Analyzer
  - Line $y=mx+h$ satisfies differential equation.

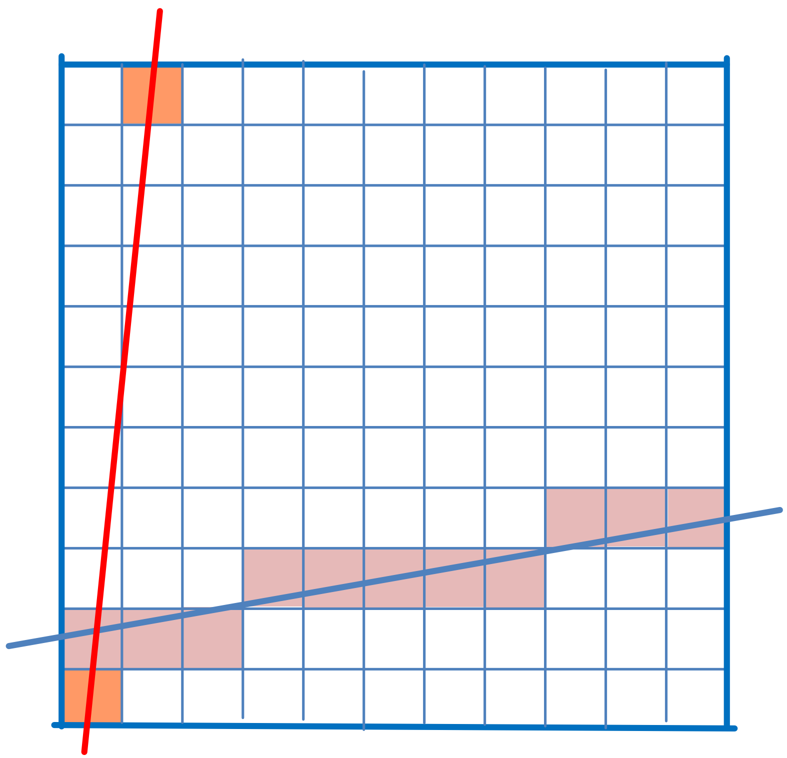
$$\frac{dy}{dx} = m = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1}$$

- Along scan line $\Delta x = 1$

```
For(x=x1; x<=x2, x++) {
    y+=m;
    write_pixel(x, round(y), line_color)
}
```

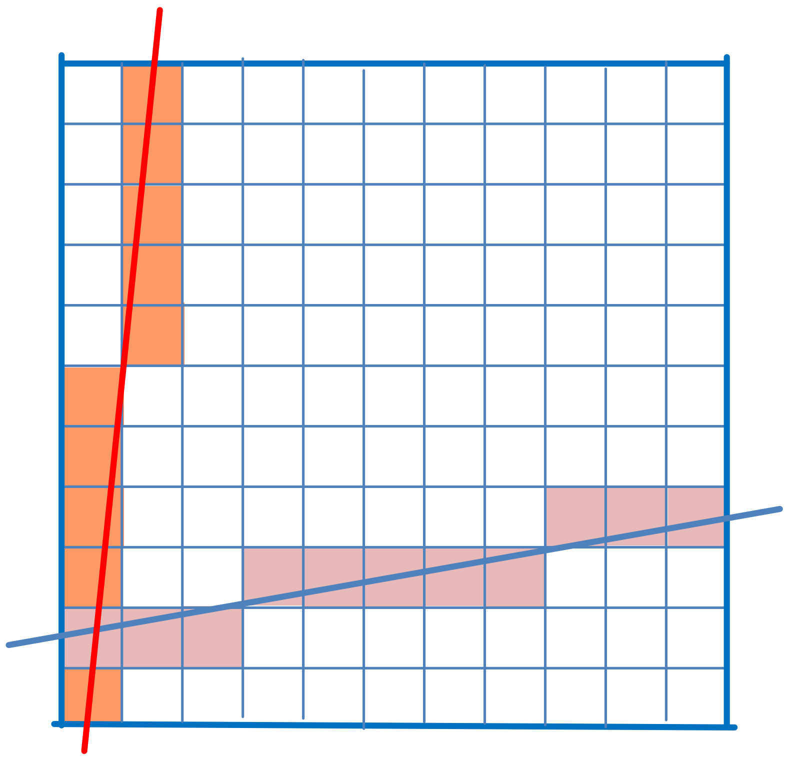
Problem

- DDA = for each *x* plot pixel at closest *y*.
  - Problems for steep lines



Using Symmetry

- Use for $1 \geq m \geq 0$
- For $m > 1$, swap roles of x and y
  - For each y, plot closest x



Bresenham’s Algorithm

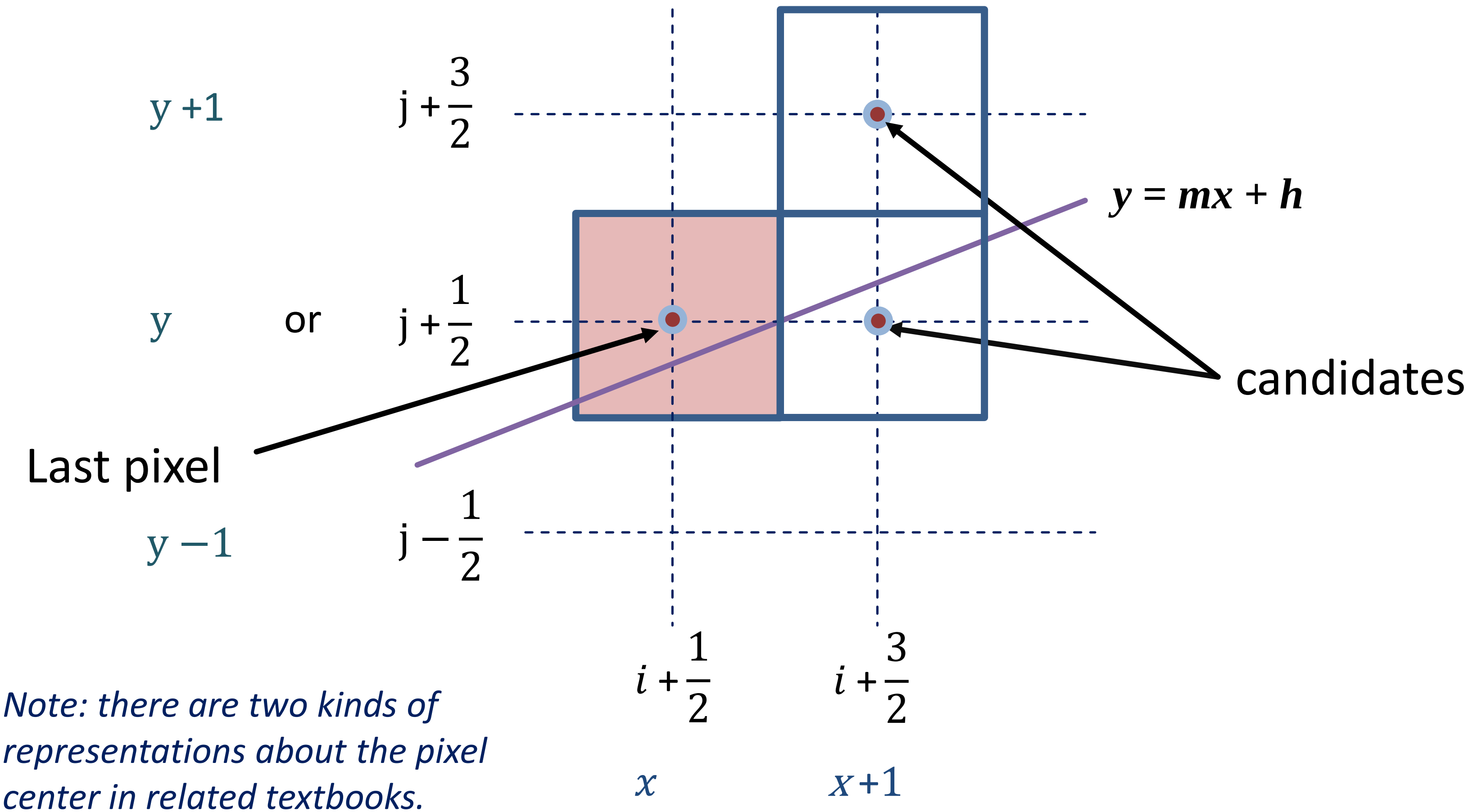
- DDA requires one floating point addition per step.
- Bresenham’s algorithm eliminates all fp.

- Consider only $1 \geq m \geq 0$
  - Handing other cases by symmetry
- Assume pixel centers are at half integers.

- Characteristics:
  - If we start at a pixel that has been written, there are only two candidates for the next pixel

Candidate Pixels

- $1 \geq m \geq 0$



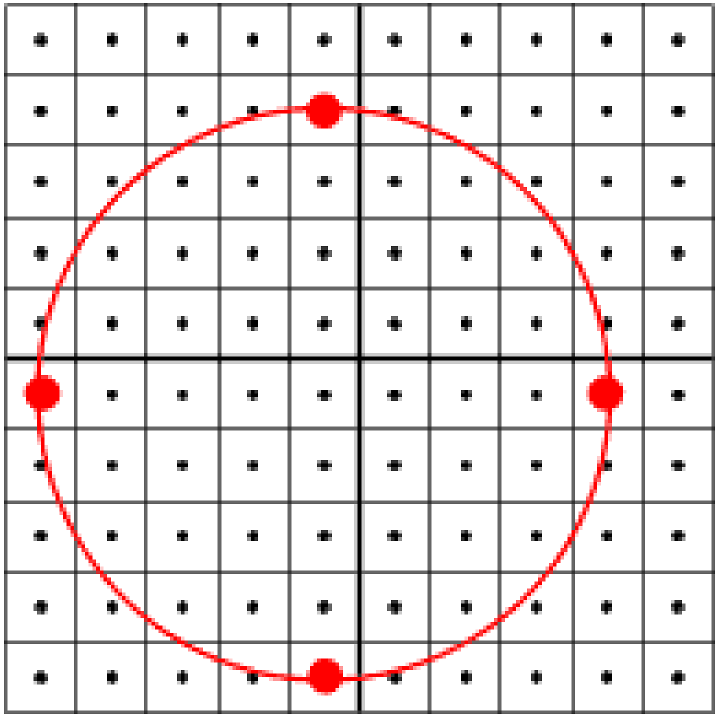
Note: there are two kinds of representations about the pixel center in related textbooks.

Bresenham's Algorithm

```
function line(x0, x1, y0, y1)
    int deltax := abs(x1 - x0)
    int deltay := abs(y1 - y0)
    real error := 0
    real deltaerr := deltay ÷ deltax
    int y := y0
    for x from x0 to x1
    {  plot(x,y)
       error := error + deltaerr
       if error ≥ 0.5
       {  y := y + 1
          error := error - 1.0 }
    }
```

```
function line(x0, x1,y0, y1)
int deltax := abs(x1 - x0)
int deltay := abs(y1 - y0)
int s_error := 0                //Scaled error
int s_deltaerr := deltay        //Scaled deltaerr
int y := y0
for x from x0 to x1
{     plot(x,y)
      s_error := s_error + s_deltaerr
      if 2×s_error ≥ deltax
      {  y := y + 1
         s_error := s_error - deltax }
}
```

Circle-drawing Algorithms



Ref: http://www.cs.umbc.edu/~rheingan/435/index.html

Circle-drawing Algorithm 1

for each x, y

{ if ($| x^2 + y^2 - r^2 | <= \varepsilon$)

SetPixel (x, y) }

Do any issue occur with this algorithm?

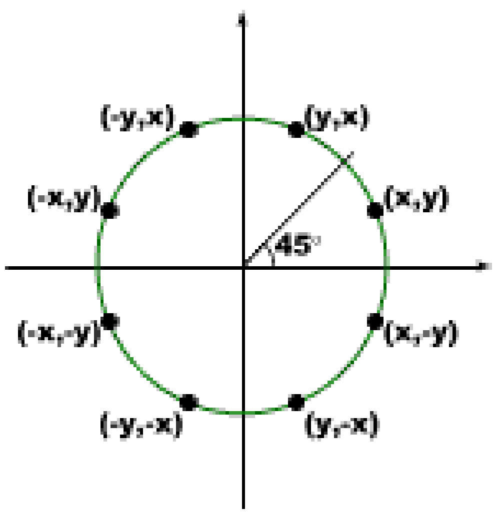
Circle-drawing Algorithm 2

```
for θ in [0~360 degree ]
{  x = r cos(θ)
   y = r sin(θ)
   SetPixel ( x, y ) }
```

How about this one?

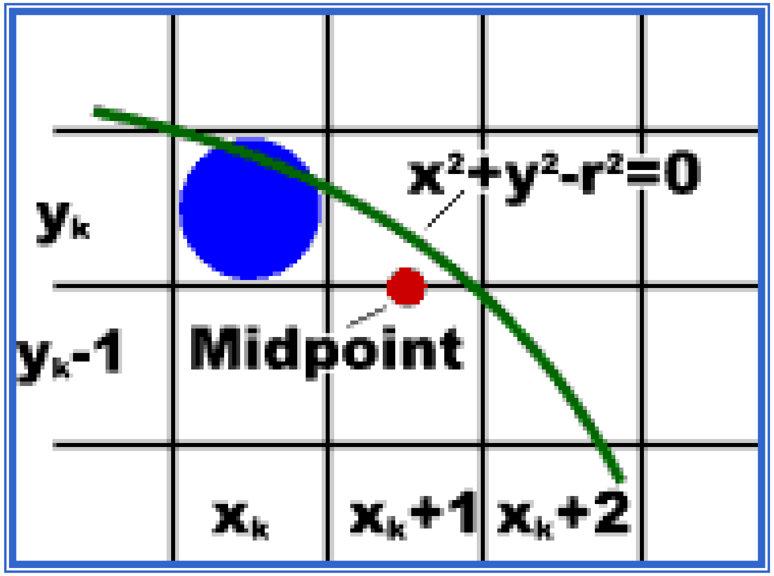
Midpoint Circle Algorithm

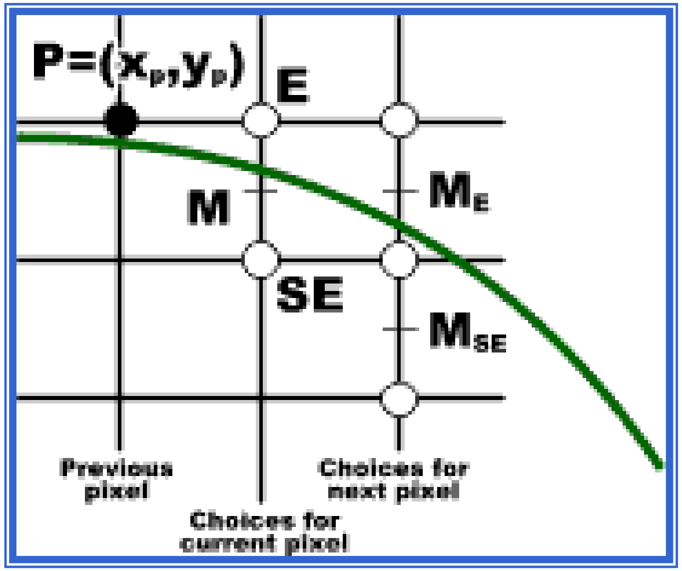
- Can we utilize the similar idea in Bresenham's line-drawing algorithm ?
  - Check only the next candidates.
  - Use symmetry and simple decision rules.



Symmetry of a Circle

Midpoint Circle Algorithm (cont.)



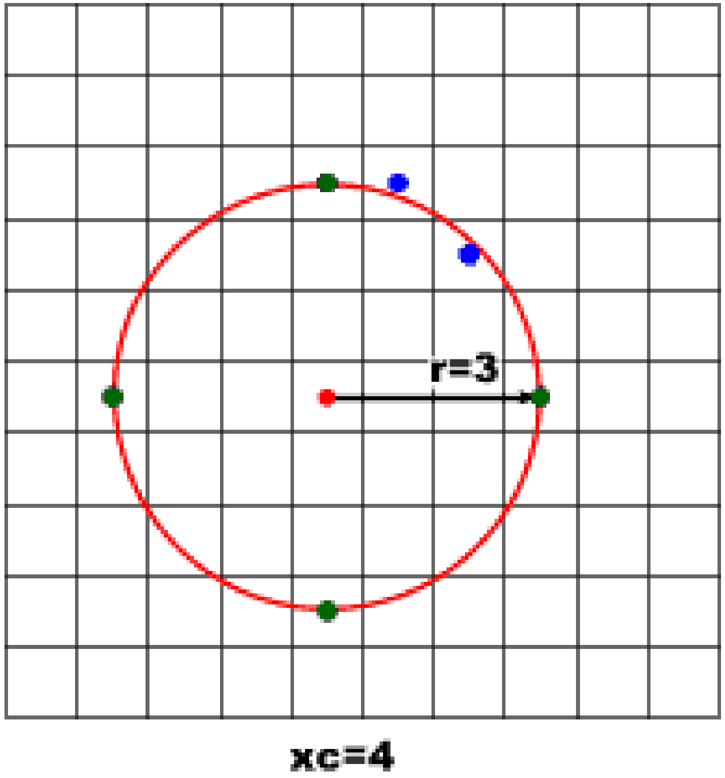


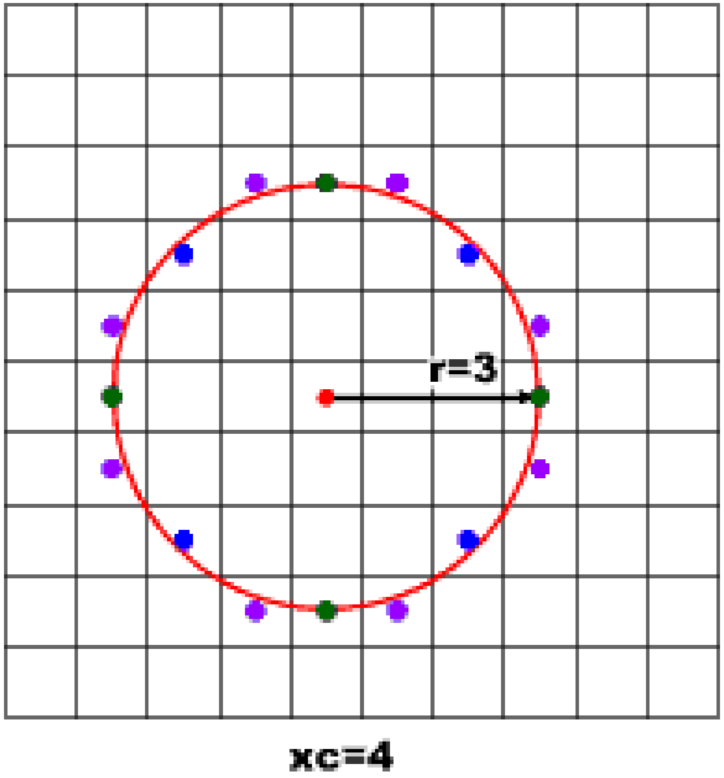
f(x,y) = $x^2 + y^2 - R^2$
f(x,y) > 0 => point outside circle
f(x,y) < 0 => point inside circle

f(M) > 0 => choose SE
f(M) < 0 => choose E

$$P_k = f_{circ}(x_k + 1, y_k - ½)$$

Midpoint Circle Algorithm (cont.)





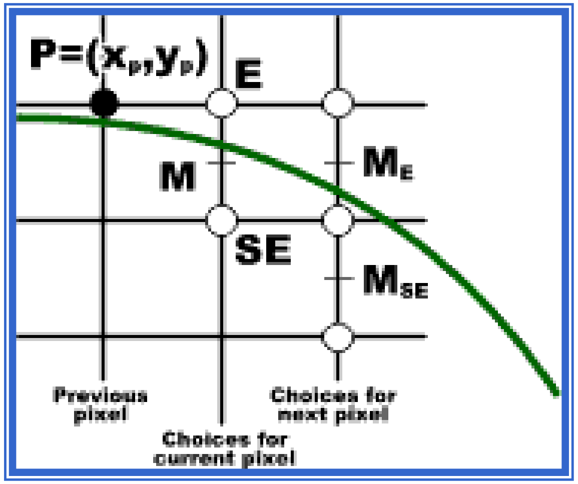
Midpoint Circle Algorithm

- Given the starting point (0,r), the computation is more efficient.

  $p_0 = f_{circle}(1, r\text{-}1/2)$

  $= 1+ (r\text{-}1/2)^2\text{-}r^2$

  $= 5/4 – r$



- For each x position,

  $p_k = f_{circle} (x_k +1, y_k\text{-}1/2) = (x_k +1)^2 + (y_k -1/2)^2 – r^2,$

  If $p_k < 0$, choose E, $(x_{k+1} = x_k+1, y_{k+1} = y_k)$

  $p_{k+1} = f_{circle}(x_{k+1}+1, y_{k+1}\text{-}1/2) = [(x_k+1)+1]^2 + (y_k -1/2)^2 –r^2$

  $= p_k + 2x_k + 3 =p_k + 2x_{k+1} + 1$

  If $p_k > 0$, choose SE, $(x_{k+1} = x_k+1, y_{k+1} = y_k\text{-}1)$

  $p_{k+1} = f_{circle}(x_{k+1}+1, y_{k+1}\text{-}1/2) = [(x_k+1)+1]^2 + (y_k -1/2\text{-}1)^2 –r^2$

  $= p_k + 2x_k - 2y_k + 5 = p_k + 2x_{k+1} - 2y_{k+1} +1$

Midpoint Circle Algorithm (cont.)

Summary of the algorithm:

- Given the starting point (0,r),

  Initialization,

  $P_0 = 5/4 – r$

  At each x position,

  if($p_k < 0$)

  the next point is (x_{k+1}, y_k)

  $p_{k+1} = p_k + 2x_{k+1} + 1$

  else

  the next point is (x_{k+1}, y_k-1)

  $p_{k+1} = p_k + 2x_{k+1} + 1 – 2y_{k+1}$

Other Primitives

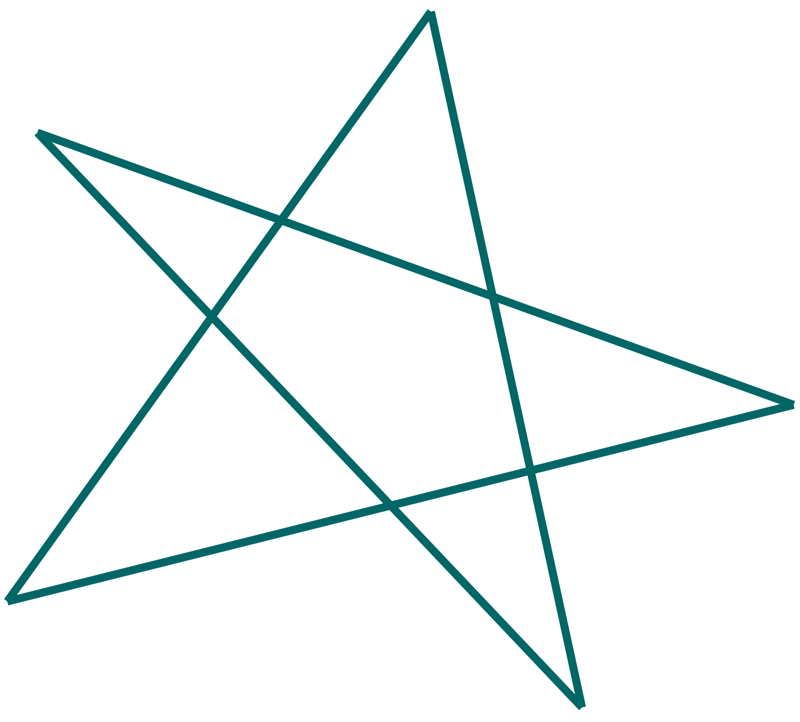
- The same concept can be extended to other primitives.
  - Ellipse, polynomials, splines, etc.

2D Polygon Filling

- Recall:
  - In computer graphics, we usually use polygons to approximate complex surfaces.
- Let’s focus on the polygon filling !

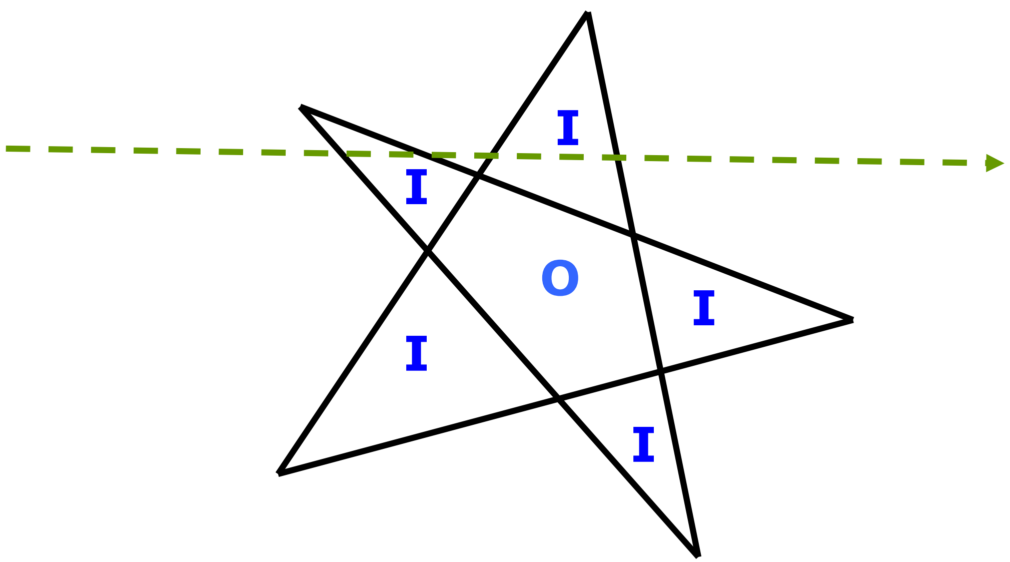
General Polygons

- Inside or Outside are not obvious
  - It’s not obvious when the polygon intersects itself.



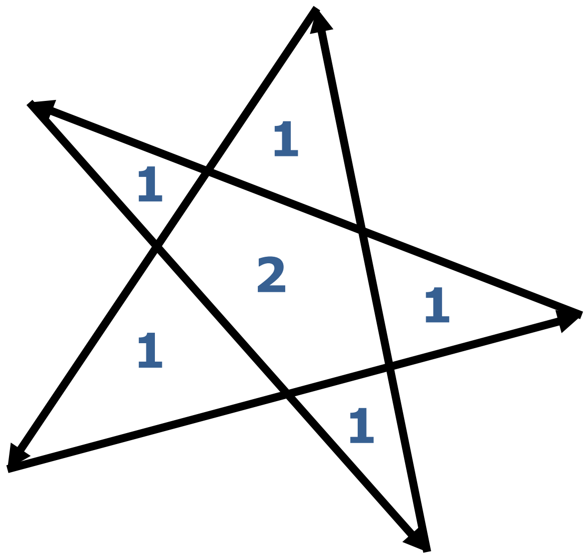
Inside or Outside

- Odd-even rule :
  - Draw a ray to infinity and count the number of edges that cross it.
  - Even → outside; odd → inside
  - usually used for polygon rasterization



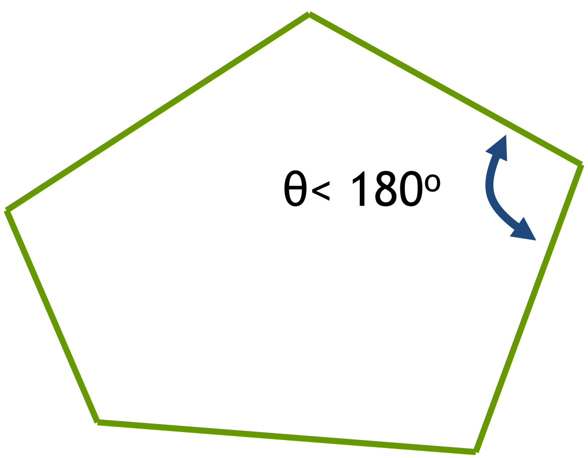
Inside or Outside

- Non-zero winding rule
  - trace around the polygon, count the number of times the point is circled (+1 for clockwise, -1 for counter clockwise).
  - Non-zero winding counts = inside

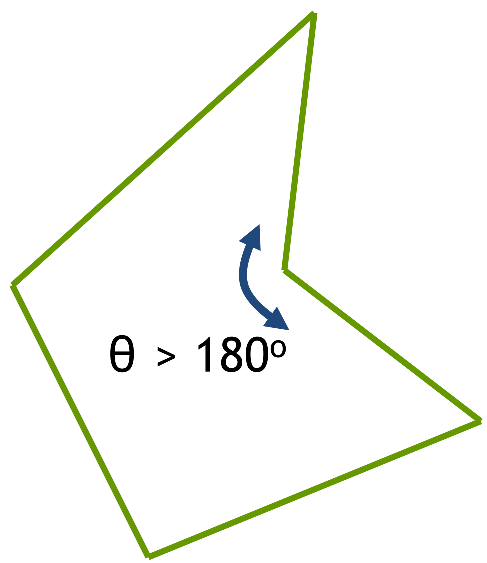


Concave vs. Convex

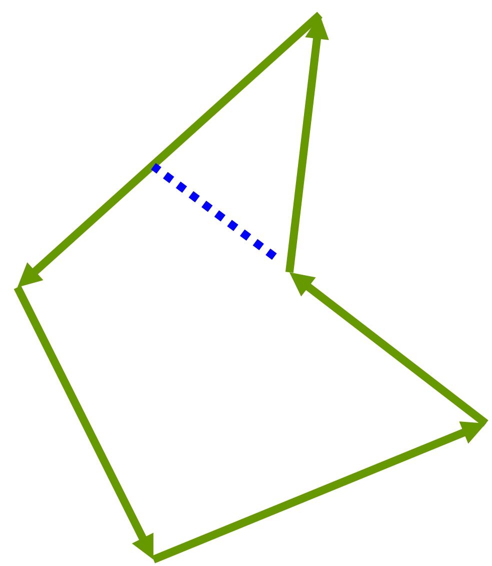
- We prefer dealing with "simpler" polygons.
- Convex (easy to break into triangles)



convex

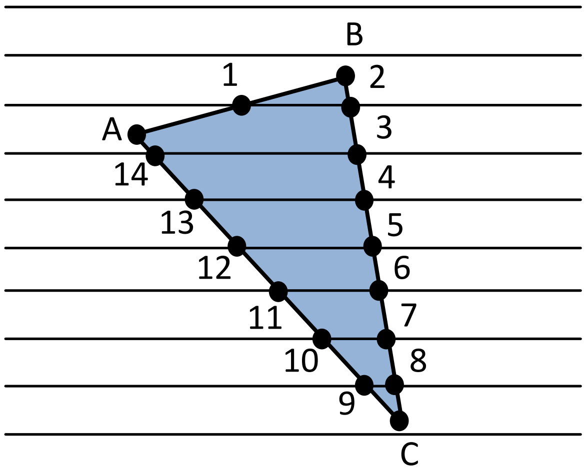


concave

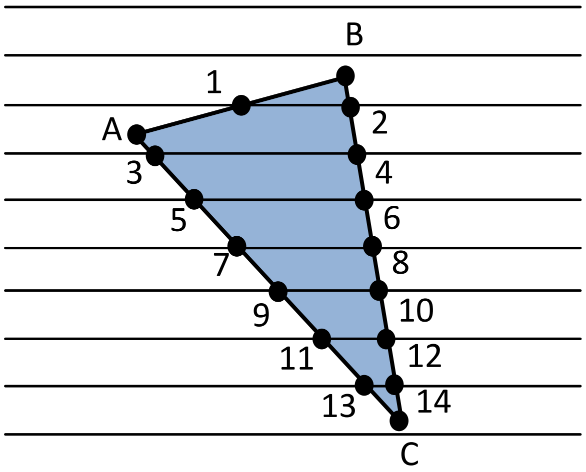


Polygon Filling by Scan Lines

- Fill by maintaining a data structure of all intersections of polygons with scan lines
  - Sort the scan lines
  - Fill each span

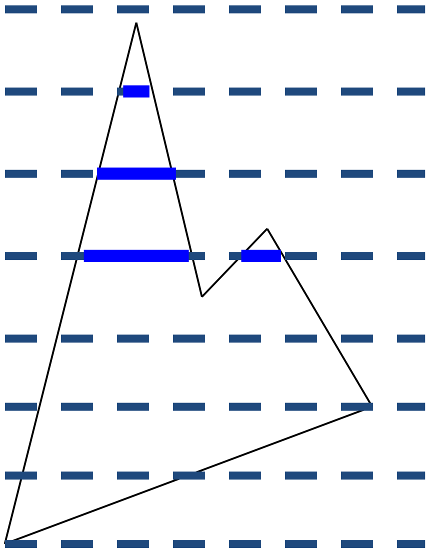


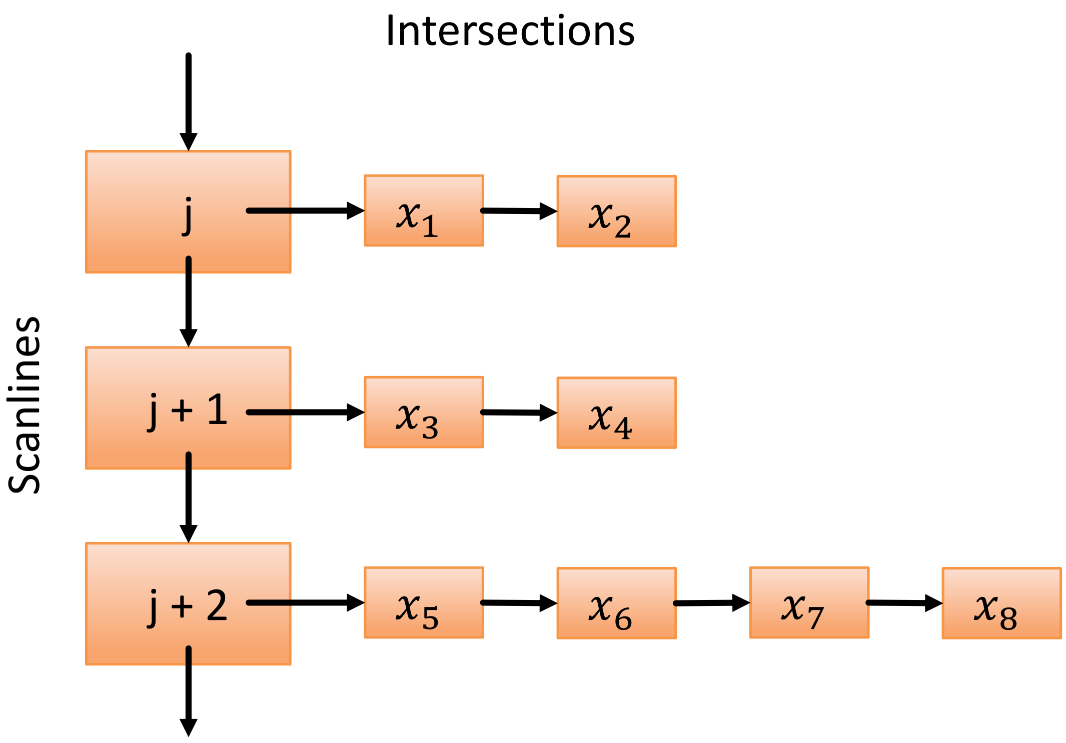
vertex order generated by vertex list



desired order

Data Structure for General Cases





Applying the odd-even rule

The End of Chapter 7