# Polygon Filling Algorithms

## Polygon Representation

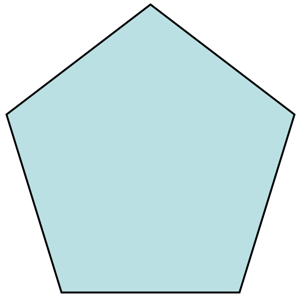
The polygon can be represented by listing its *n* vertices in an ordered list.

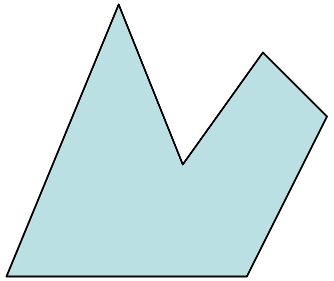
$$P = \{(x_1, y_1), (x_2, y_2), \ldots\ldots\ldots, (x_n, y_n)\}.$$

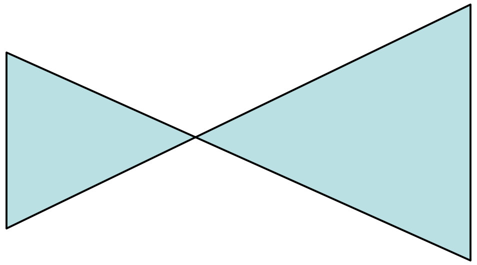
The polygon can be displayed by drawing a line between $(x_1, y_1)$, and $(x_2, y_2)$, then a line between $(x_2, y_2)$, and $(x_3, y_3)$, and so on until the end vertex. In order to close up the polygon, a line between $(x_n, y_n)$, and $(x_1, y_1)$ must be drawn.

# Polygon Fill Algorithm

- ***Different types of Polygons***
  - Simple Convex
  - Simple Concave
  - Non-simple : self-intersecting
  - With holes


Convex


Concave

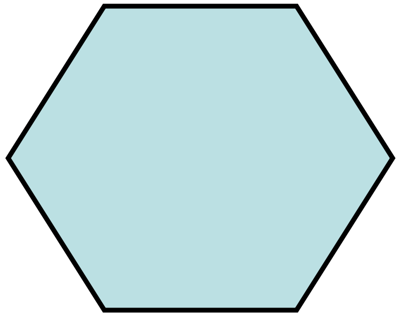

Self-intersecting

## Polygon Filling

Types of filling

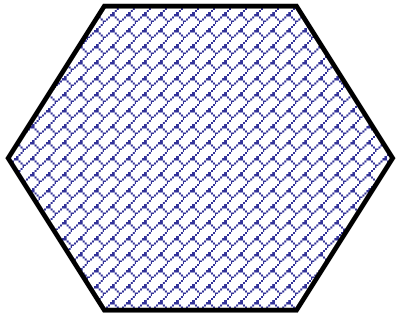
- **Solid-fill**

All the pixels inside the polygon's boundary are illuminated.



- **Pattern-fill**

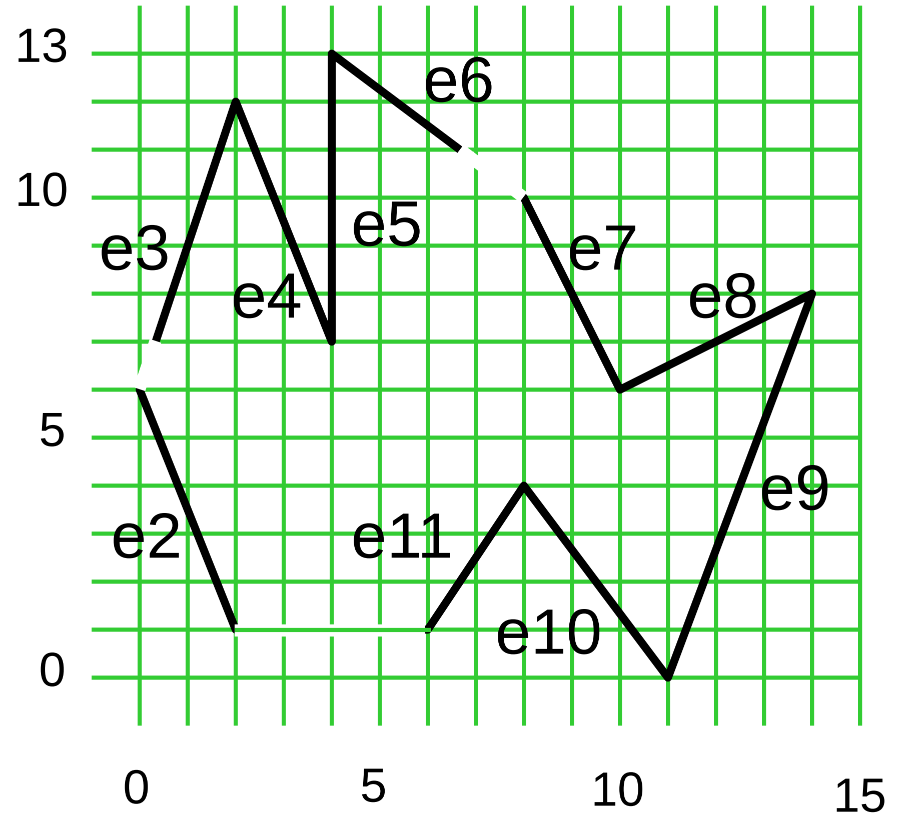
the polygon is filled with an arbitrary predefined pattern.



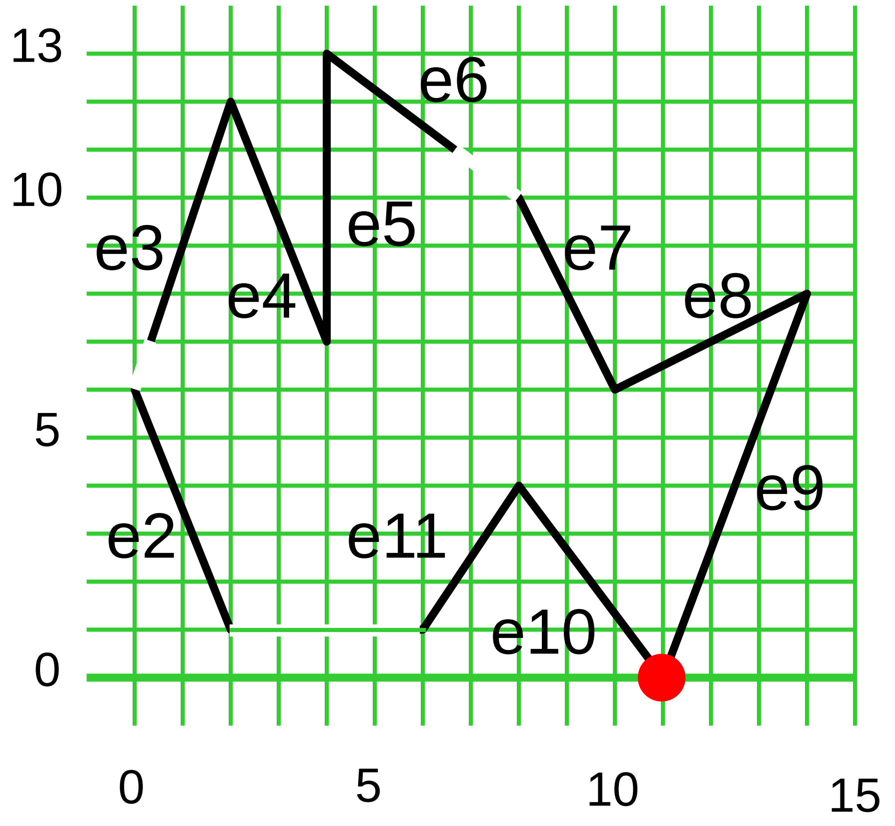
## Types of Polygon fill algorithm

- 1. Scan line algorithm
- 2. Boundary Fill algorithm
- 3. Flood fill algorithm

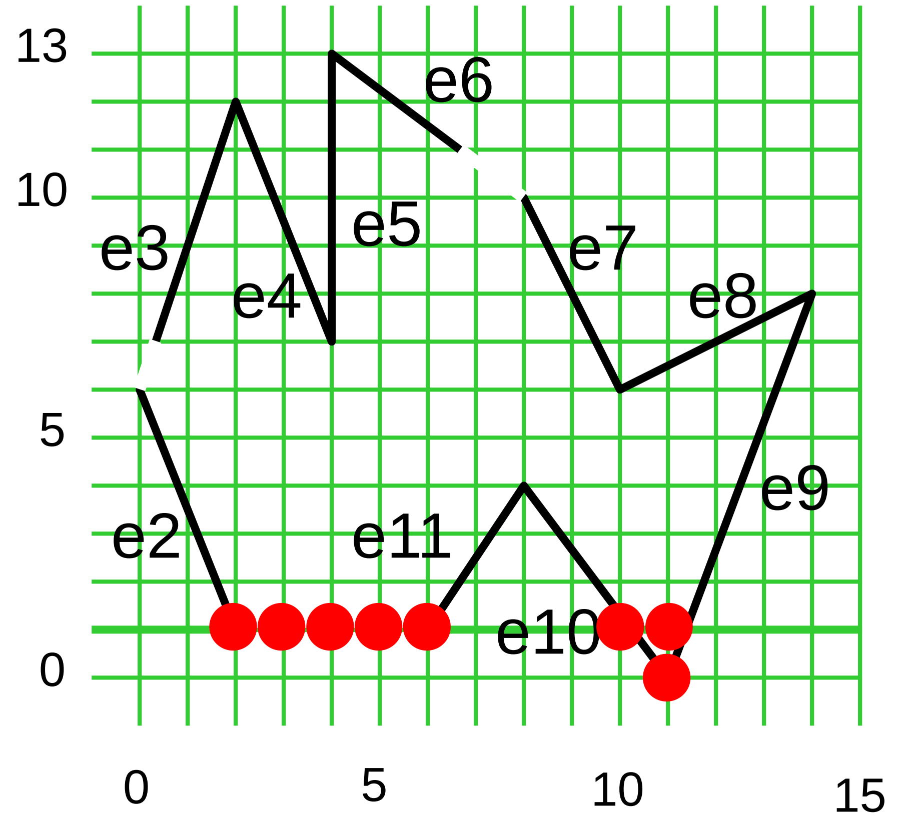
# Running the Algorithm



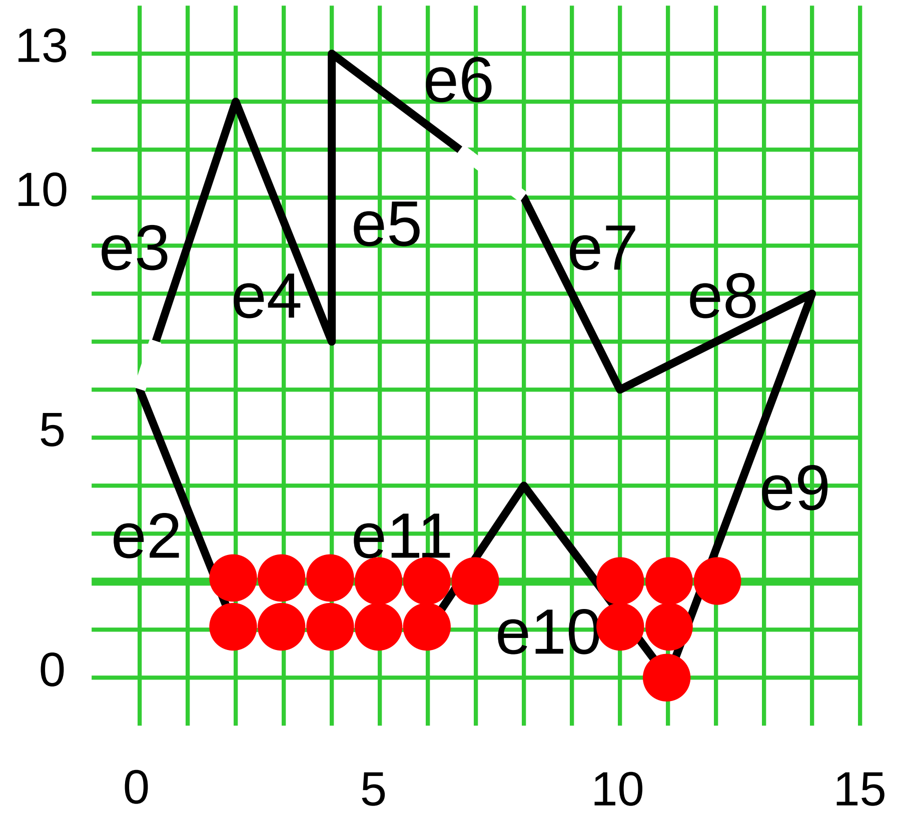
# SCAN LINE ALGORITHM



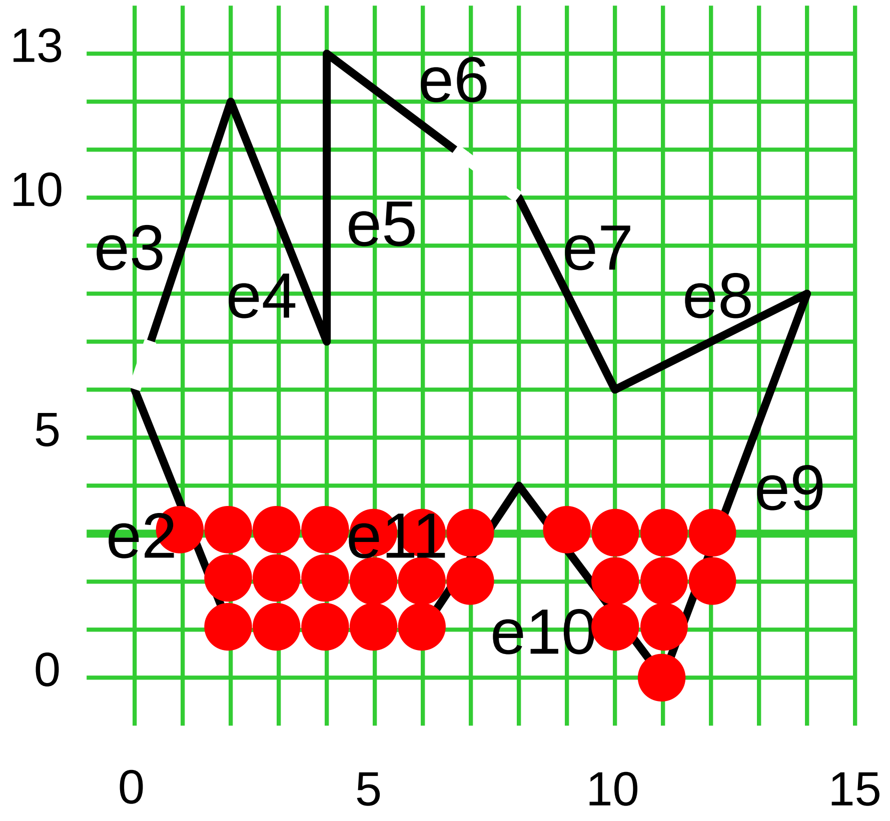
# Running the Algorithm



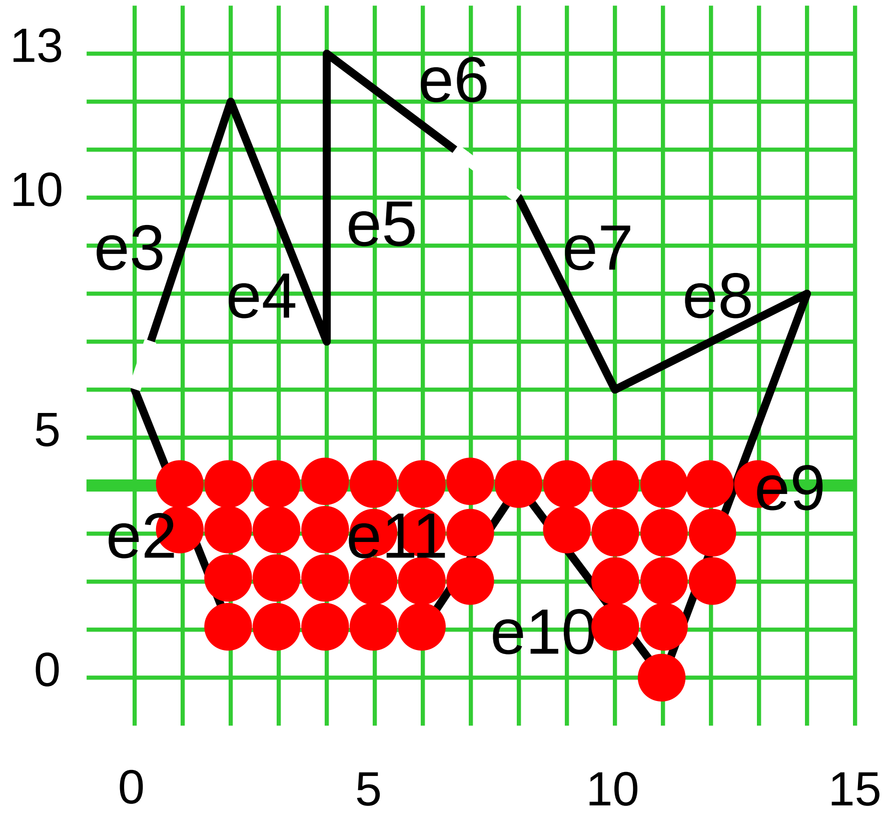
# Running the Algorithm



# Running the Algorithm

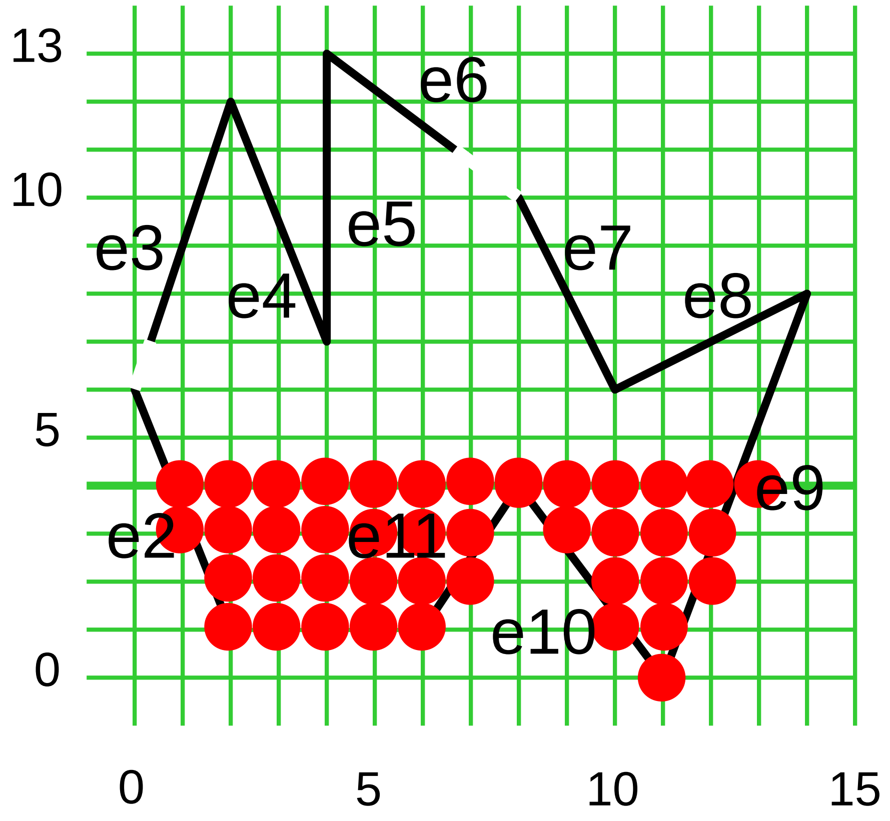


# Running the Algorithm

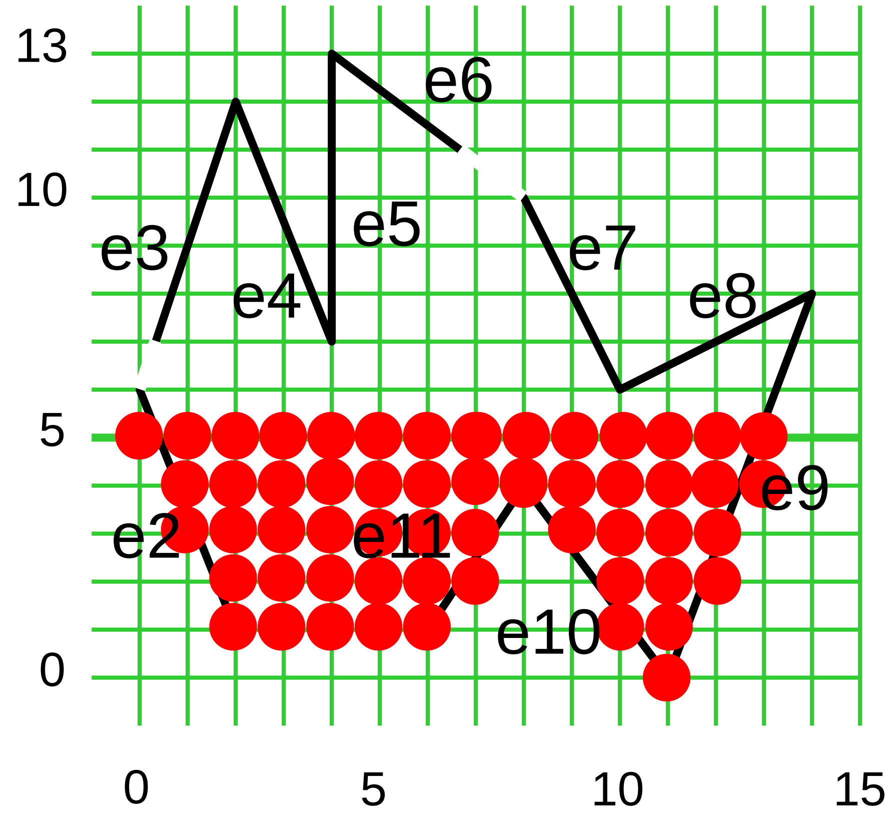


Remove these edges.

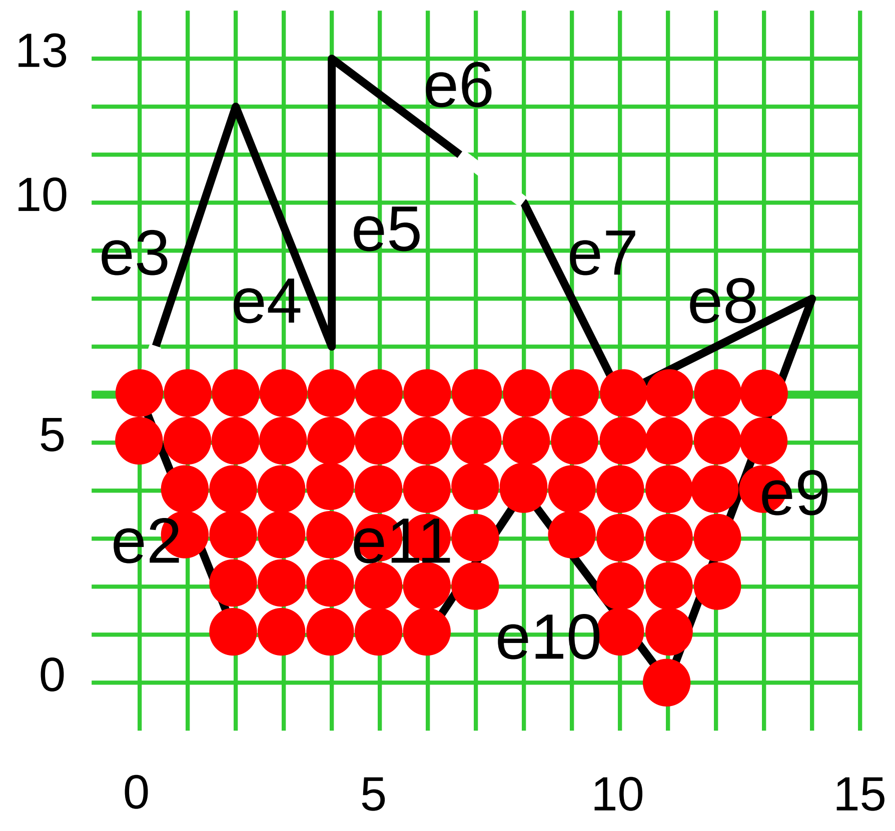
# Running the Algorithm



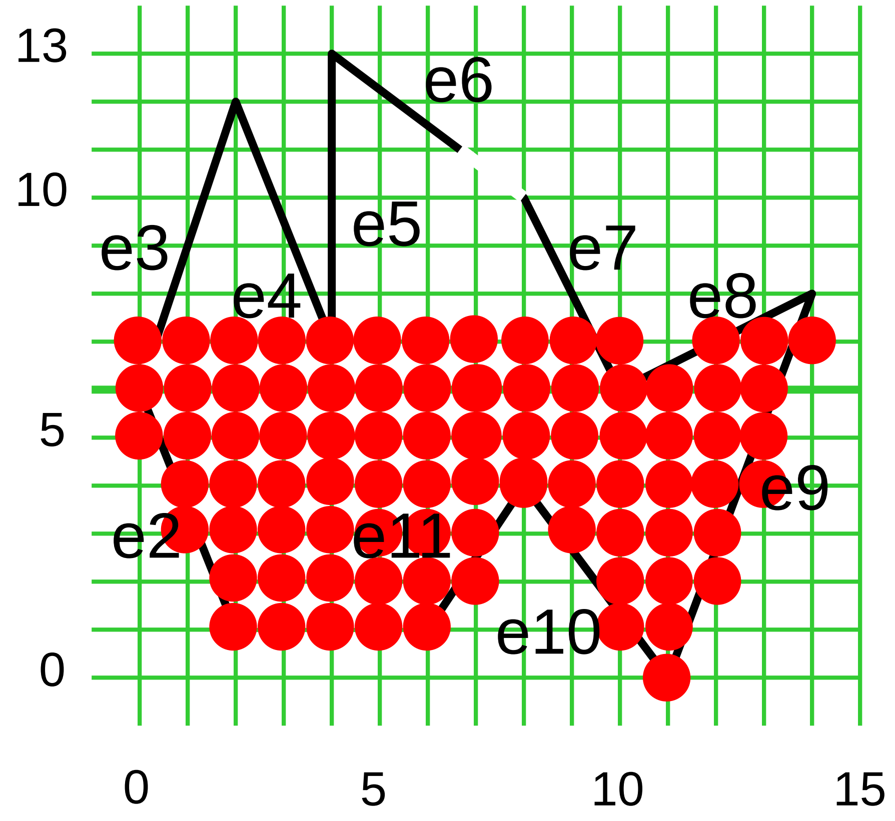
# Running the Algorithm



# Running the Algorithm



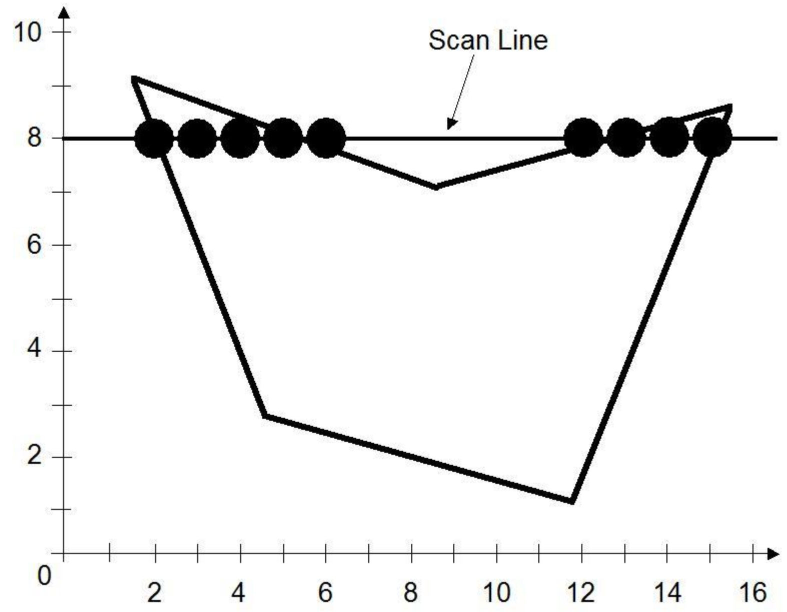
# Running the Algorithm



## The Scan-Line Polygon Fill Algorithm

The scan-line polygon-filling algorithm involves

- the **horizontal scanning** of the polygon from its **lowermost** to its **topmost** vertex,
- identifying which edges intersect the scan-line,
- and finally drawing the interior horizontal lines with the specified fill color. process.



## Example

- Consider the following polygon:

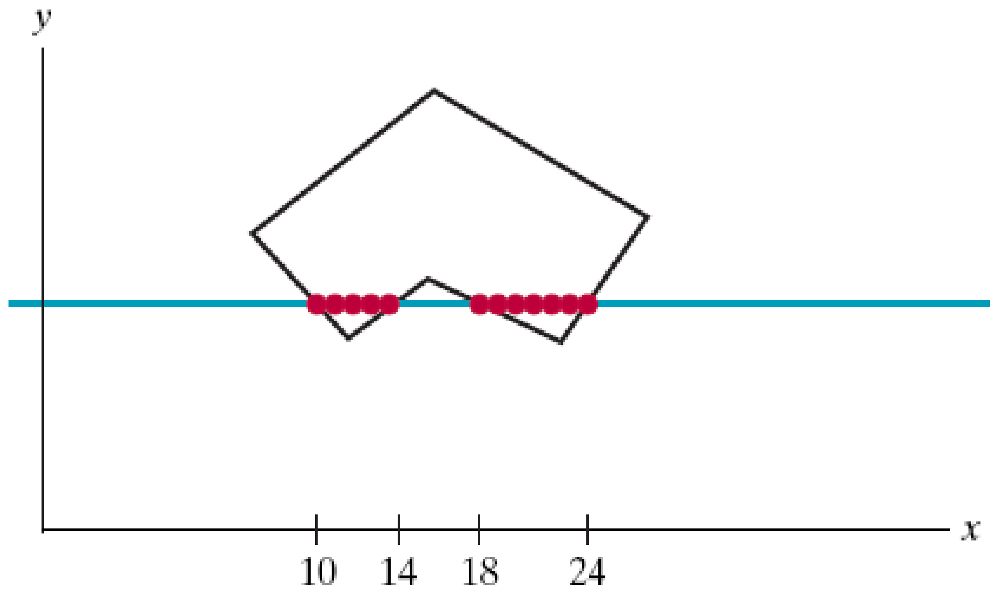


FIGURE 4–20 Interior pixels along a scan line passing through a polygon fill area.

## Example

- For each scan line that crosses the polygon, the edge intersections are sorted from left to right, and then the pixel positions between, and including, each intersection pair are set to the specified fill color.

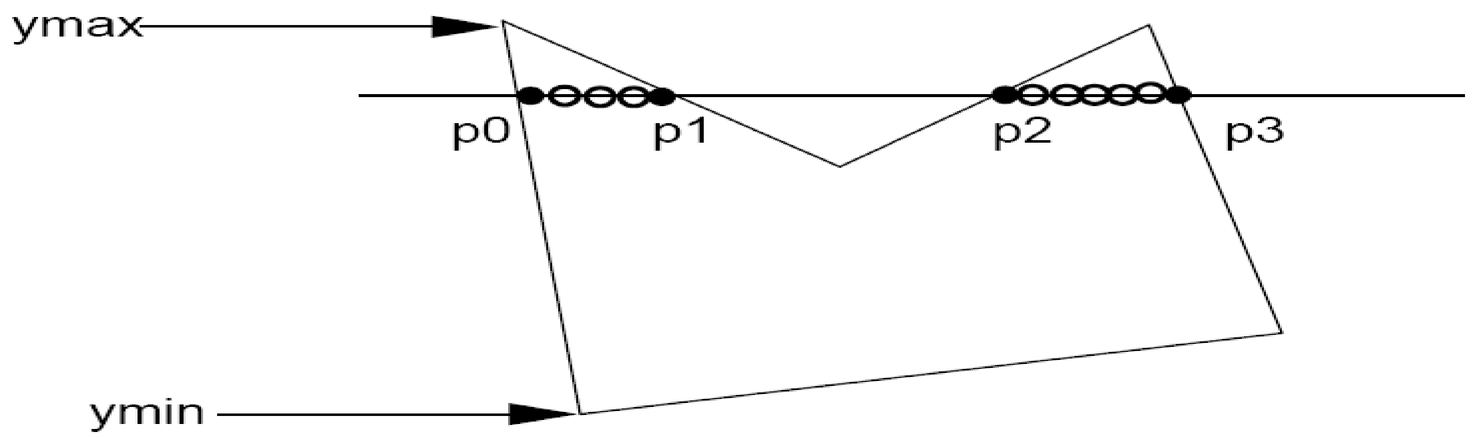
- In the previous Figure, the four pixel intersection positions with the polygon boundaries define two stretches of interior pixels.

# Scanline Fill Algorithm

- Intersect scanline with polygon edges
- Fill between pairs of intersections
- Basic algorithm:

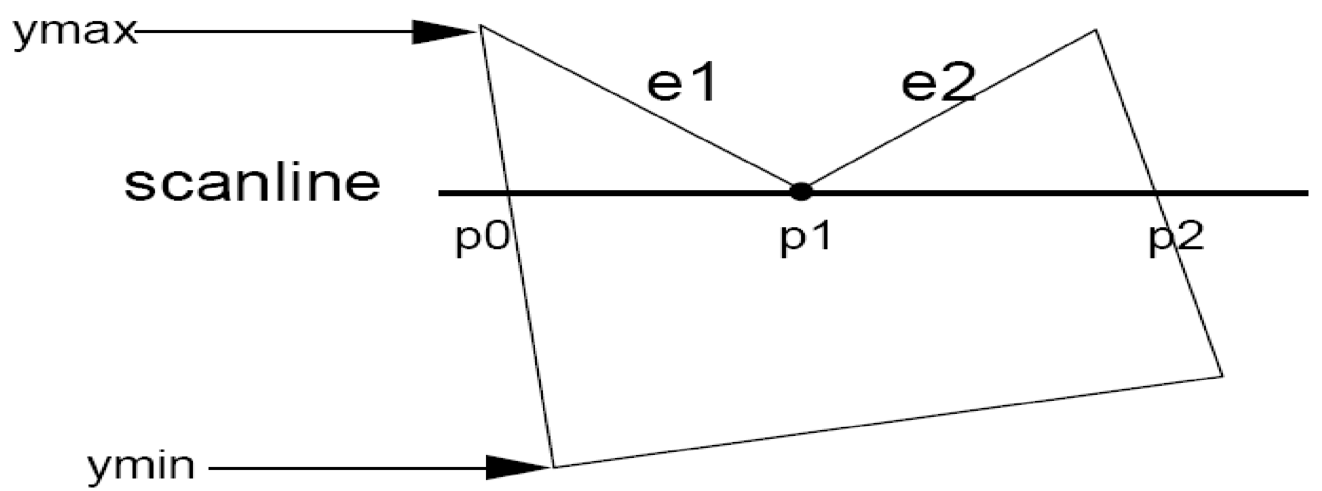
```
For y = ymin to ymax

  1) intersect scanline y with each edge
  2) sort interesections by increasing x
     [p0,p1,p2,p3]
  3) fill pairwise  (p0 -> p1, p2-> p3, ....)
```



## Special handling (cont'd)

c) Intersection is an edge end point



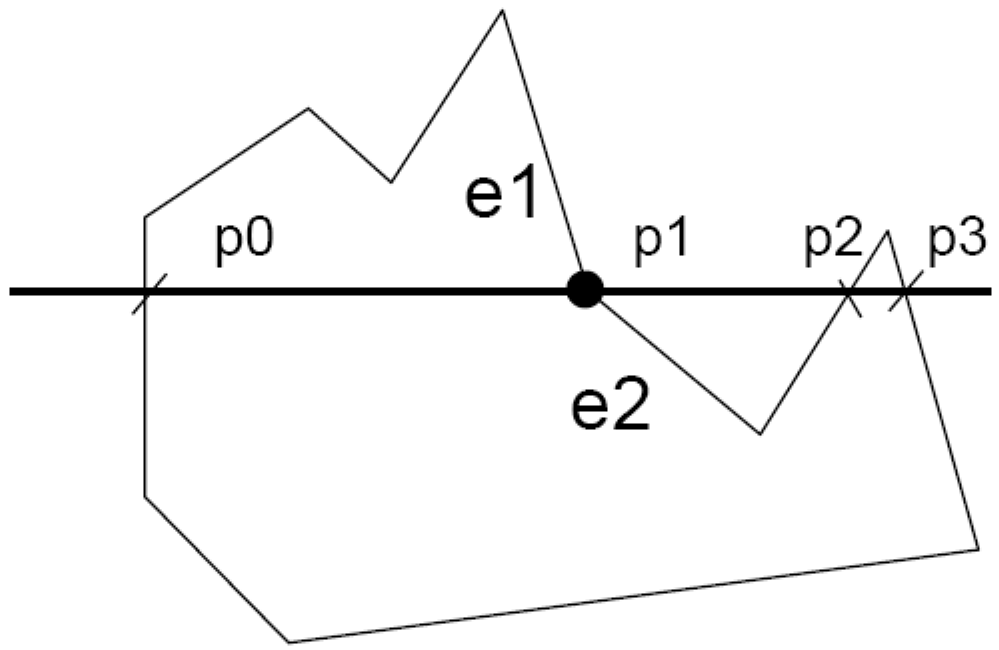
Intersection points: (p0, p1, p2) ???

–> (p0,p1,p1,p2) so we can still fill pairwise

–> In fact, if we compute the intersection of the scanline with edge e1 and e2 separately, we will get the intersection point p1 twice. Keep both of the p1.

## Special handling (cont'd)

c) Intersection is an edge end point (cont'd)



However, in this case we don't want to count p1 twice (p0,p1,p1,p2,p3), otherwise we will fill pixels between p1 and p2, which is wrong

## Polygon Fill Algorithm

- Consider the next Figure.
- It shows two scan lines that cross a polygon fill area and intersect a vertex.
- Scan line *y*’ intersects an even number of edges, and the two pairs of intersection points along this scan line correctly identify the interior pixel spans.
- But scan line *y* intersects five polygon edges.

# Polygon Fill Algorithm

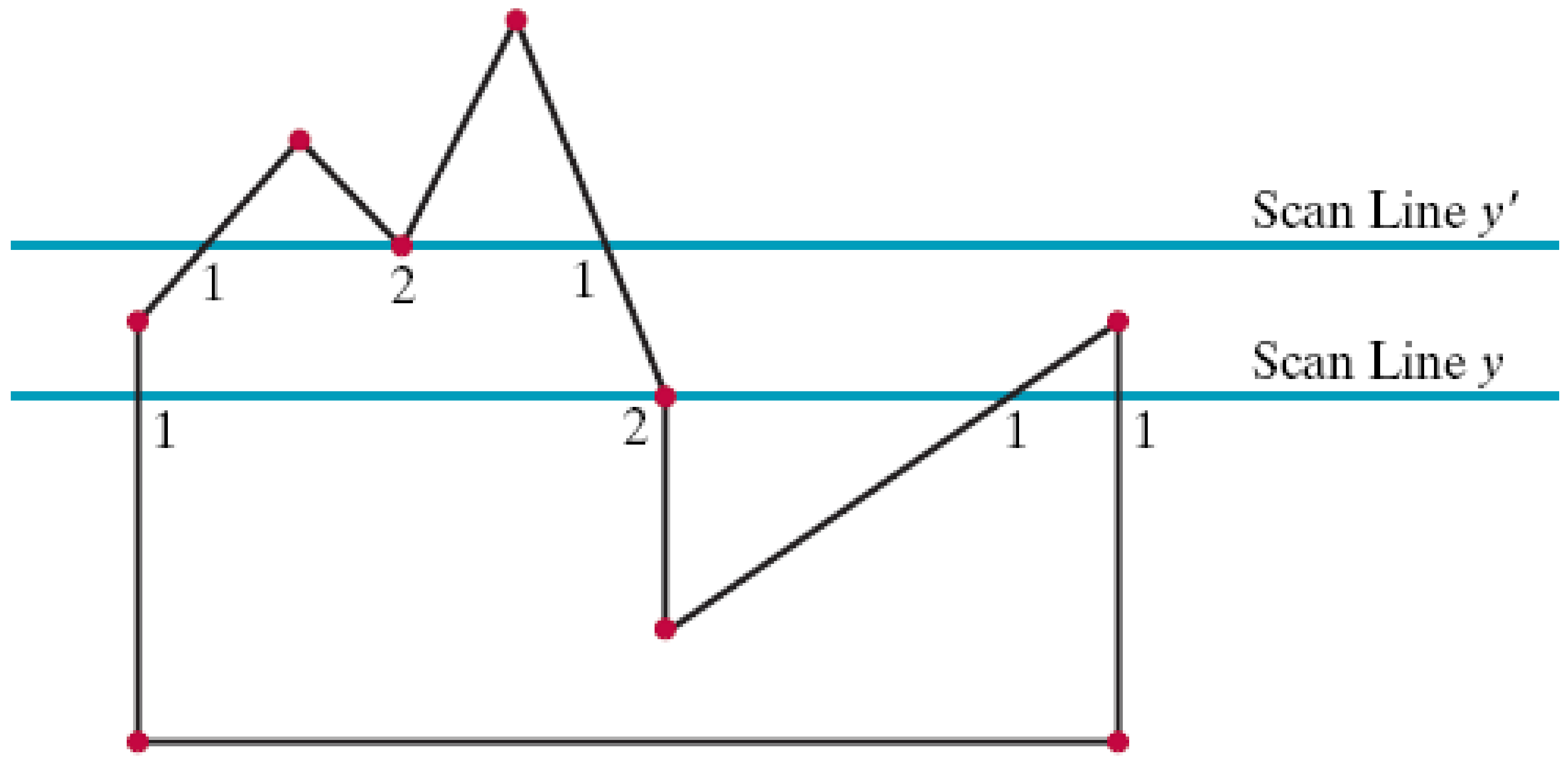
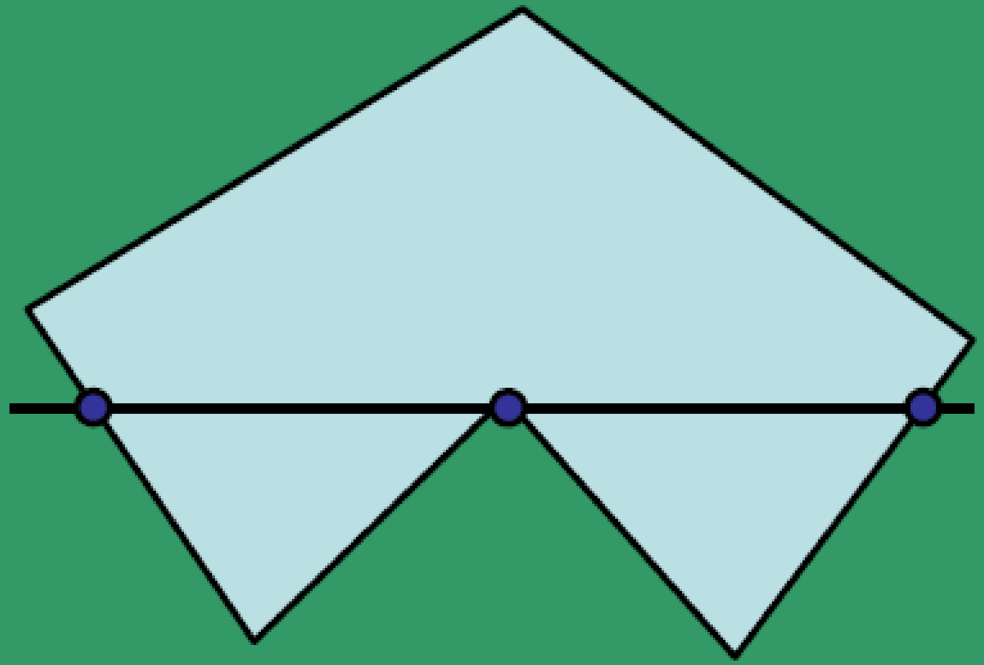


FIGURE 4–21 Intersection points along scan lines that intersect polygon vertices. Scan line $y$ generates an odd number of intersections, but scan line $y'$ generates an even number of intersections that can be paired to identify correctly the interior pixel spans.

## Polygon Fill Algorithm

- To identify the interior pixels for scan line *y*, we must count the vertex intersection as only one point.
- Thus, as we process scan lines, we need to distinguish between these two cases.

## Two different cases of scanlines passing through the vertex of a polygon



Case - I

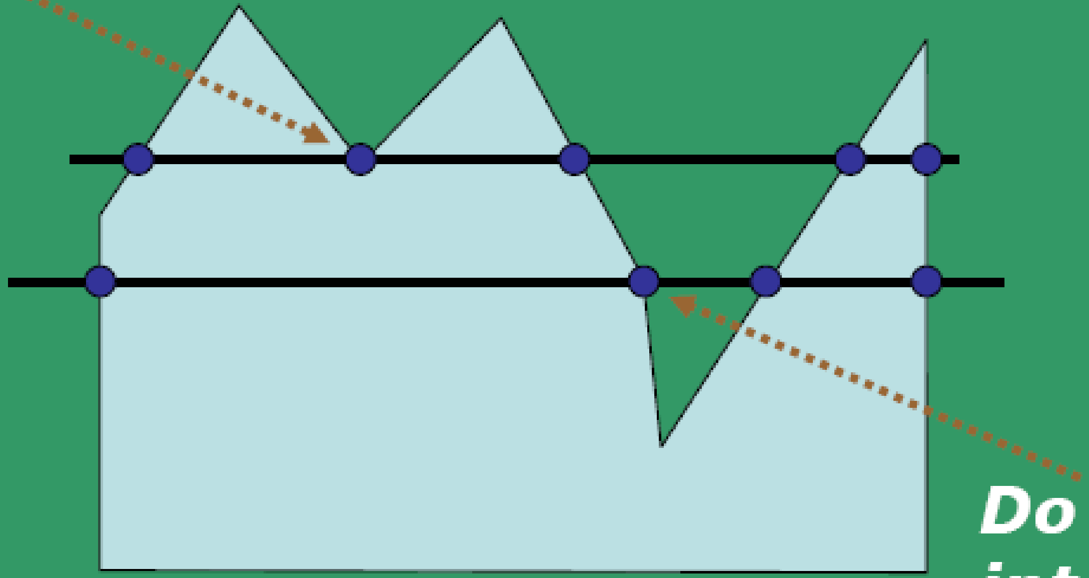
← *Add one more intersection:*

***3 -> 4***

## Case - II

***Add one more intersection:***

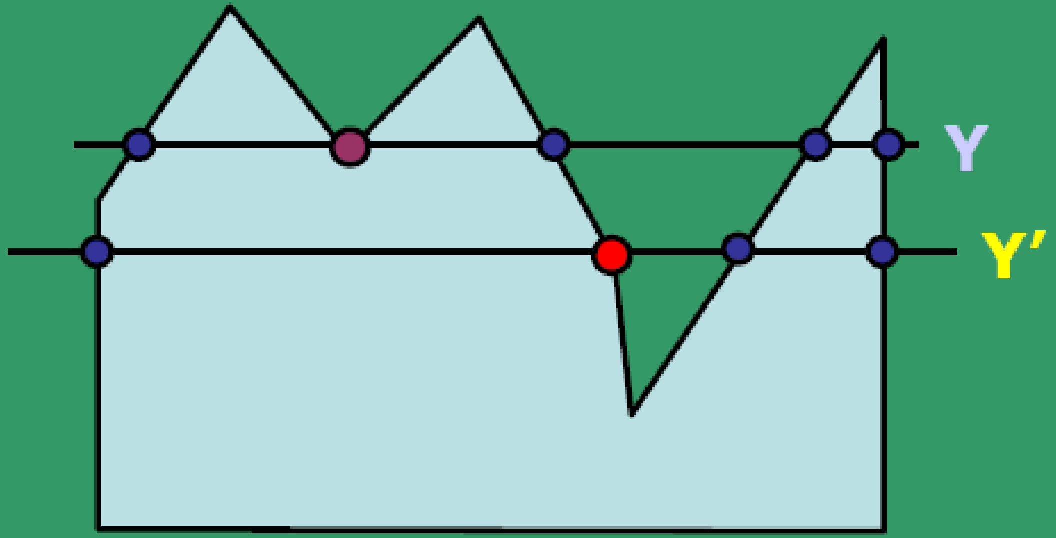
***5 -> 6;***



***Do not add intersection, keep 4;***

***HOW ??***

## What is the difference between the intersection of the scanlines Y and Y’, with the vertices?

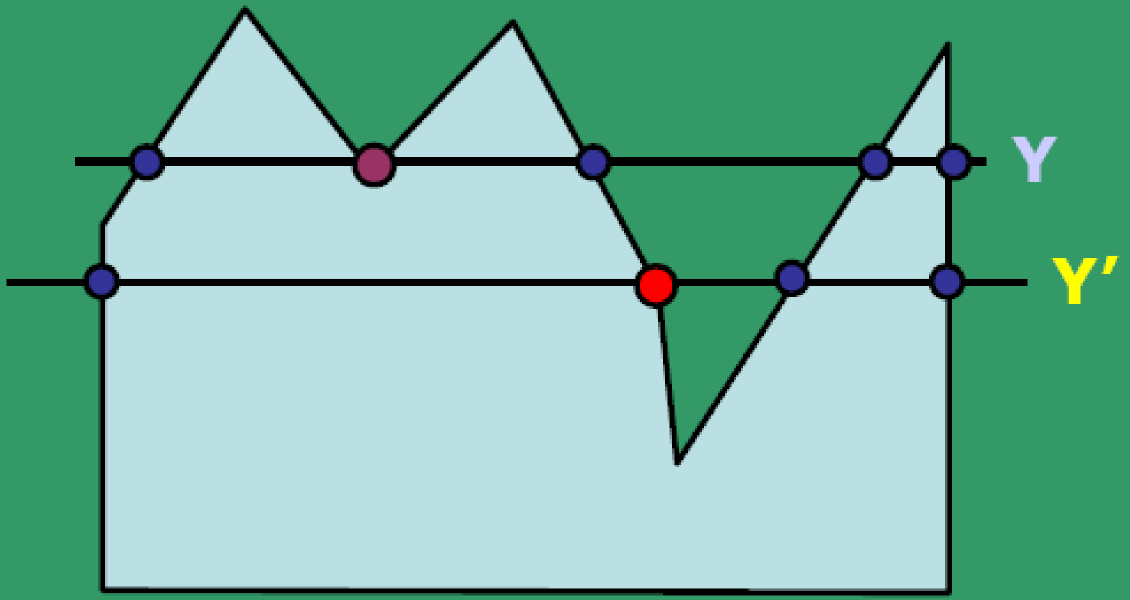


For Y, the edges at the vertex are on the same side of the scanline.

Whereas for Y’, the edges are on either/both sides of the vertex.

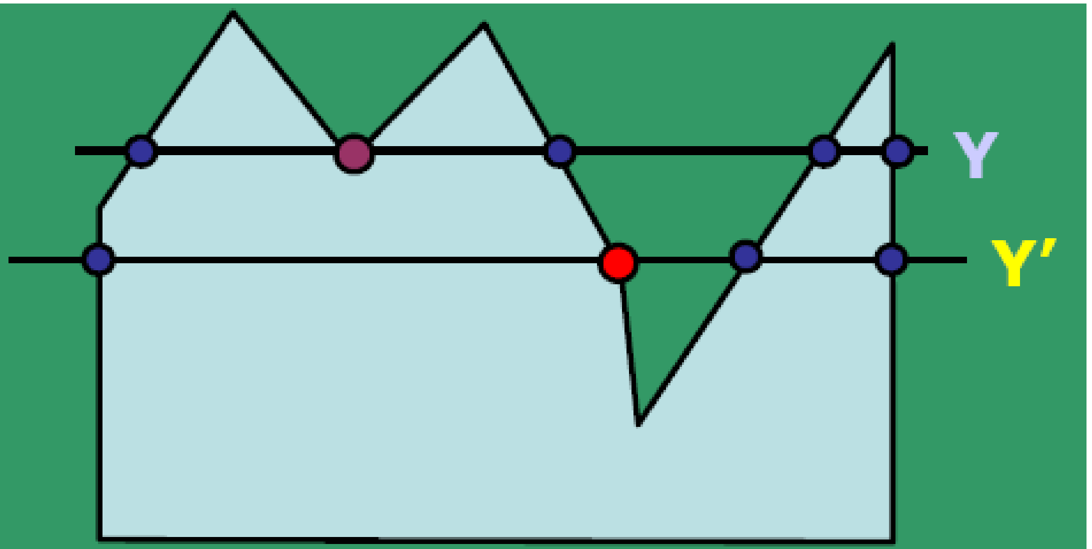
In this case, we require additional processing.

## Vertex counting in a scanline



- **Traverse** along the polygon boundary clockwise (or counter- clockwise) and
- **Observe** the *relative change in Y-value* of the edges on either side of the vertex (i.e. as we move from one edge to another).

## Vertex counting in a scanline



Check the condition:

If *end-point Y values* of two consecutive edges *monotonically increase or decrease*, count the middle vertex as a single intersection point for the scanline passing through it.

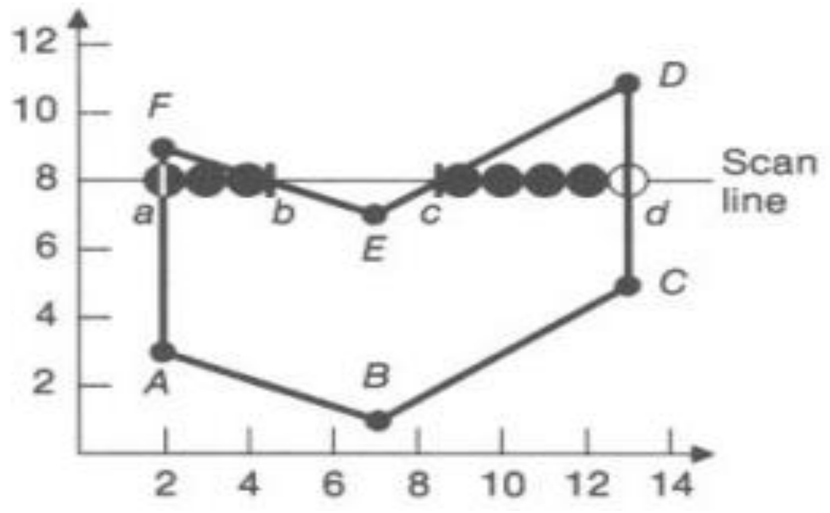
Else the shared vertex represents a *local maximum (or minimum)* on the polygon boundary. *Increment the intersection count*.

# Scan Line Algorithm

**Start with max y and move to min y or vice versa**

**<u>For each scan line:</u>**

- ❖ Find the intersections of the scan line with all edges of the polygon.
- ❖ Sort the intersections by increasing x coordinate.
- ❖ Fill in all pixels between pairs of intersections
- ❖ For scan line number 8 the sorted list of x-coordinates is (2, 4, 9, 13) Therefore draw line  b/w (2,4) & (9,13)



**Assume that we are taking scan line from min y and moving towards max y**

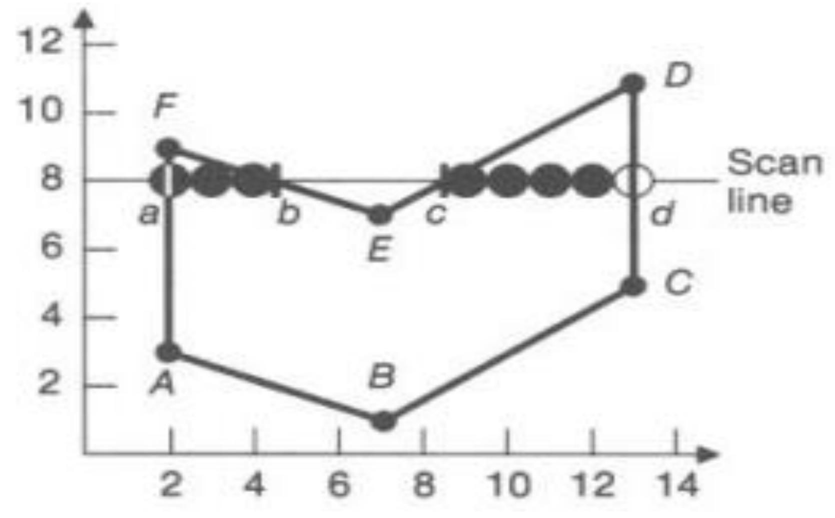
- We know that for every scan line we have to calculate x intersection with every polygon side. We can determine the next *x* intersection value as

- $x_{i+1}=x_i + 1/m$ where m is the slope of edge

- It is very easy to identify which polygon sides should be tested for x-intersection, if we sort edges in order of min *y* and find active edge list (Active edge list for scan line contains all the edges crossed by that scan line ).

- Two pointers are used to indicate the beginning and ending of active edge list. As the scan line move up the picture the beginning and end pointers also moved up to eliminate the edges which end above the current scan line and include the new edges which now start on or above the current scan line .

# Edge List and Active Edge List



12
10
8
6
4
2
F
D
Scan line
a
b
c
d
E
C
A
B
2
4
6
8
10
12
14