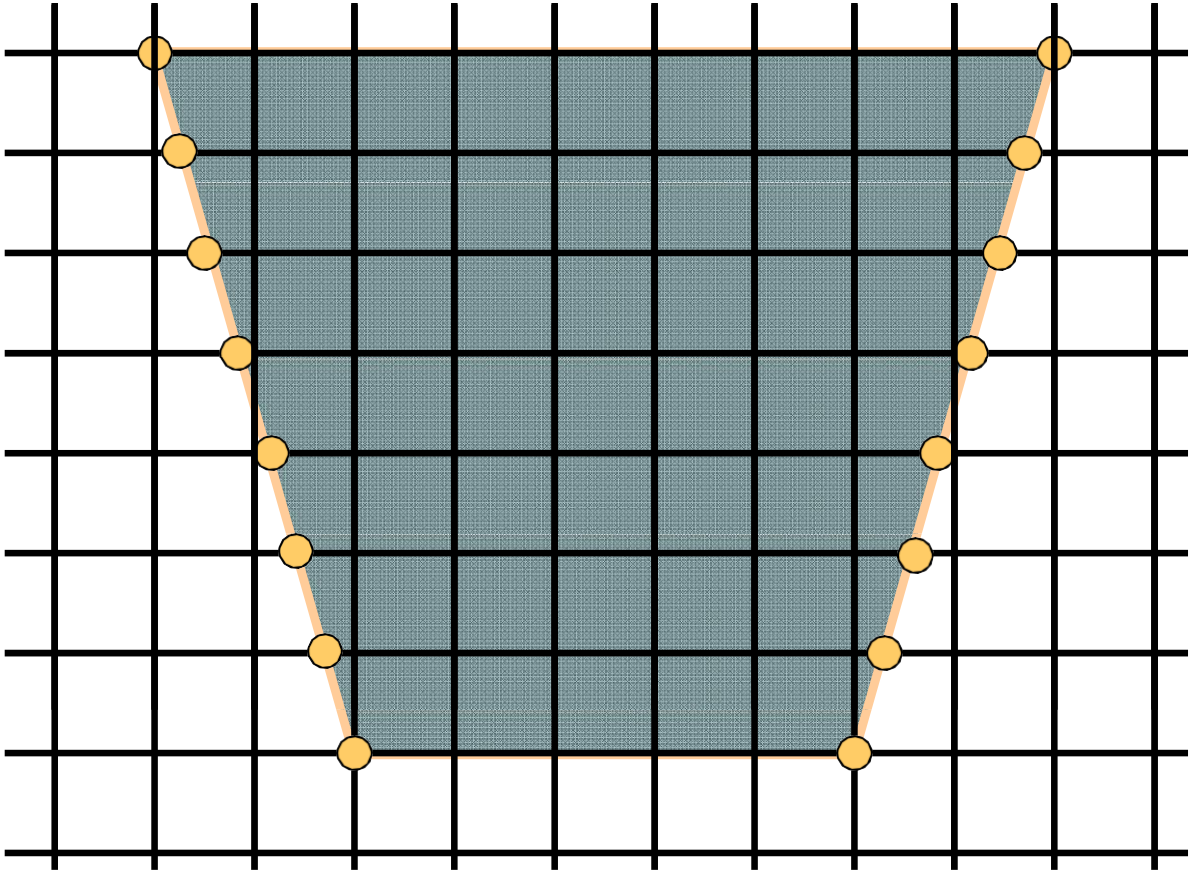
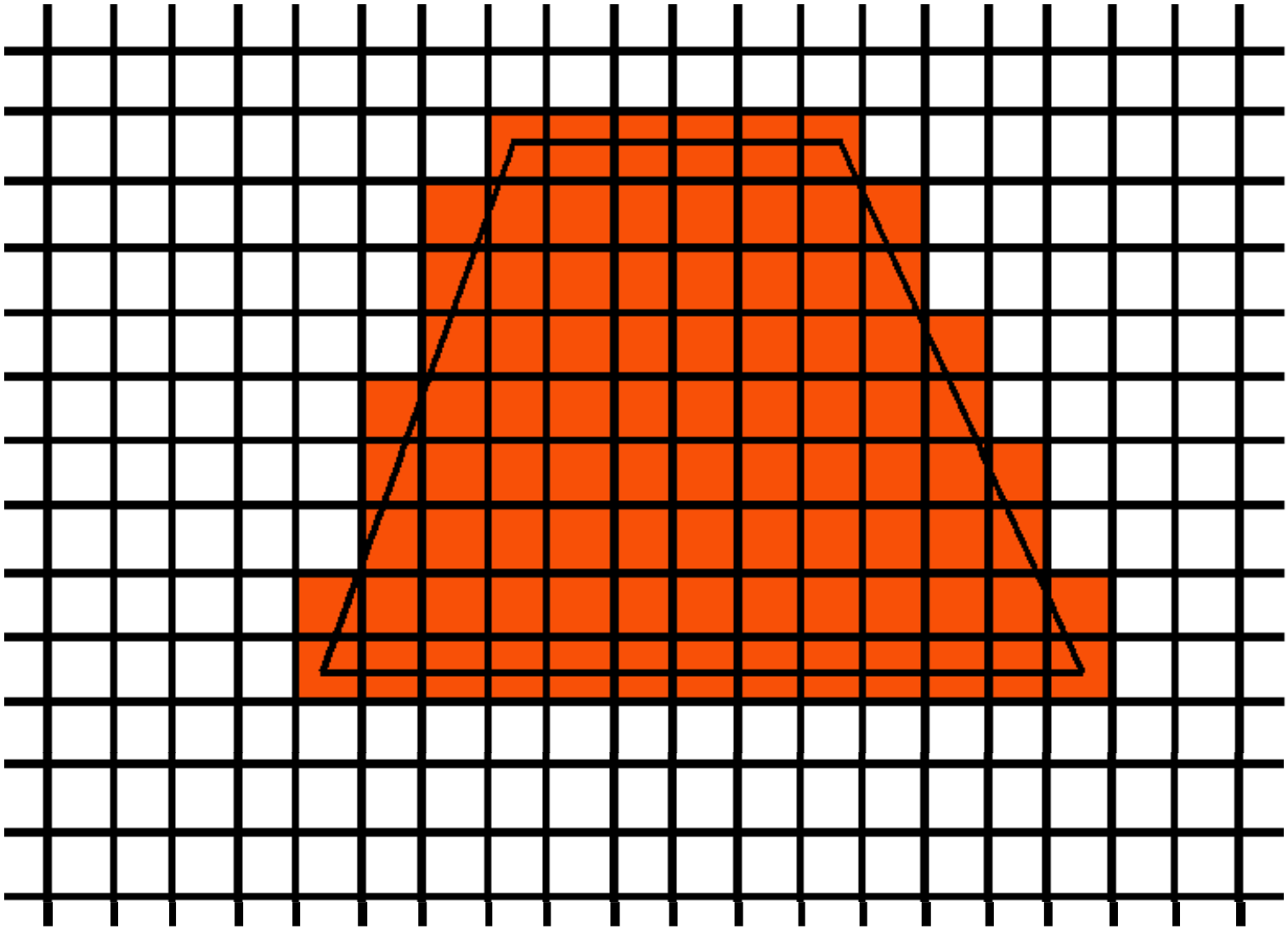
# POLYFILL-
# SCAN CONVERSION
# OF A POLYGON

**Pixels are not at the center of the grid, but at the intersection of two orthogonal scan lines (on the grid intersection points).**



# SCAN CONVERSION - POLYFILL



# SCANLINE POLYFILL ALGORITHM

- Steps (conceptual):

- Find minimum enclosed rectangle

- No. of scanlines = $Y_{max} - Y_{min} + 1$

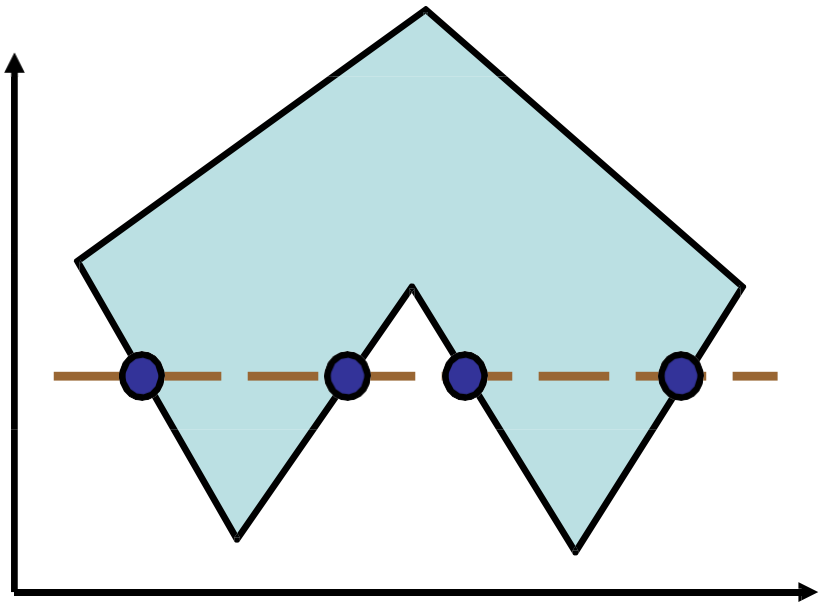
- For each scanline do

  - **Obtain intersection points of scanline with polygon edges.**

  - **Sort intersections from left to right**

- **Form pairs of intersections from the list§**
- **Fill within pairs**
- **Intersection points are updated for each scanline**
- **Stop when scanline has reached $Y_{max}$**

**§ - Intersections at vertices require special handling**

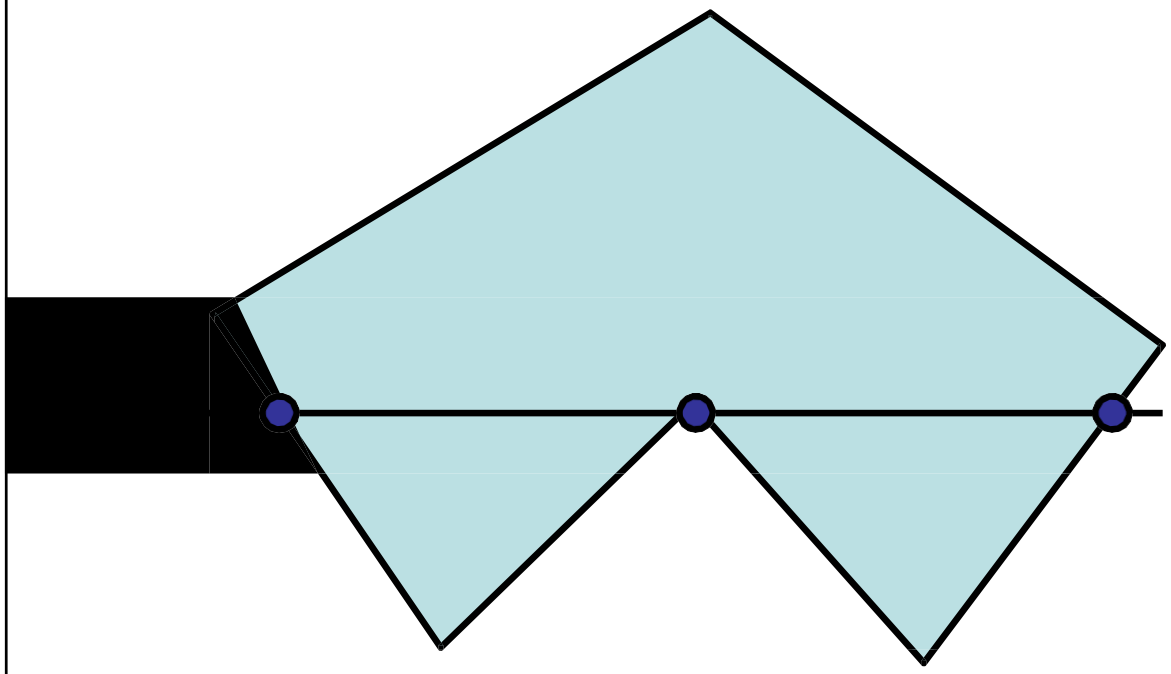


**Check if a point is within a Polygon?**

**Look left or right, and count the number of intersection points of the scanline with the edges of the Polygon.**

**If the number is ODD, point is *Inside***
**else *Outside***

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



**Case - I**

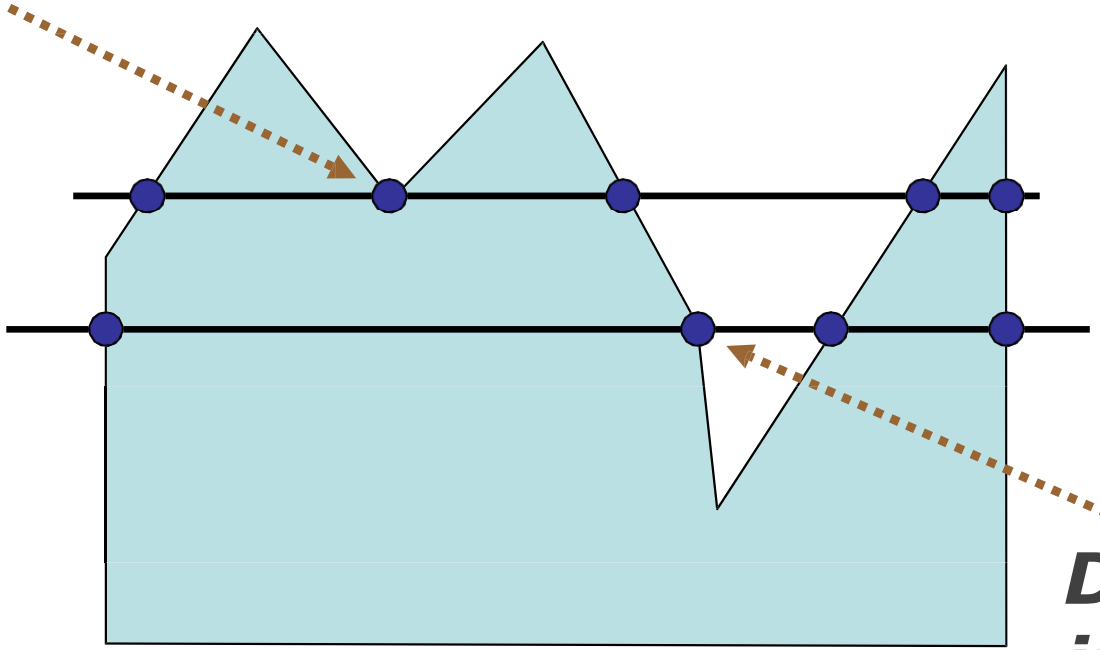
← ***Add one more intersection:***

***3 -> 4***

Case - II

***Add one more intersection:***

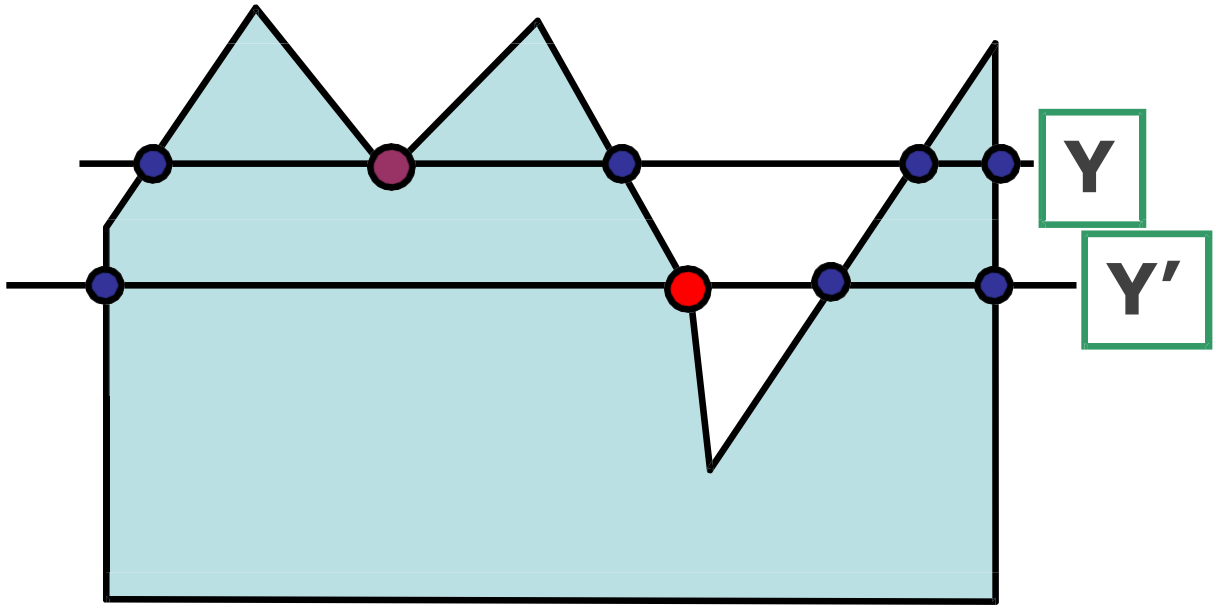
***5 -> 6;***



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

***HOW ??***

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

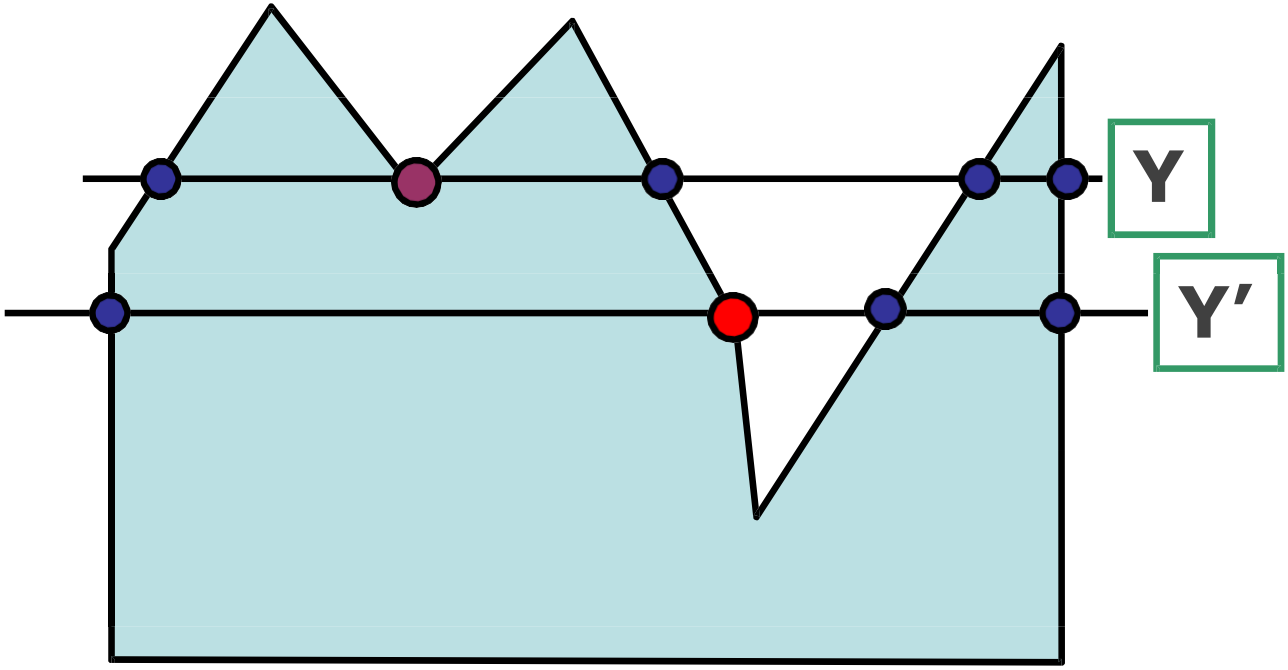



**For Y, the edges at the vertex are on the same side of the scanline.**

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

**In this case, we require additional processing.**

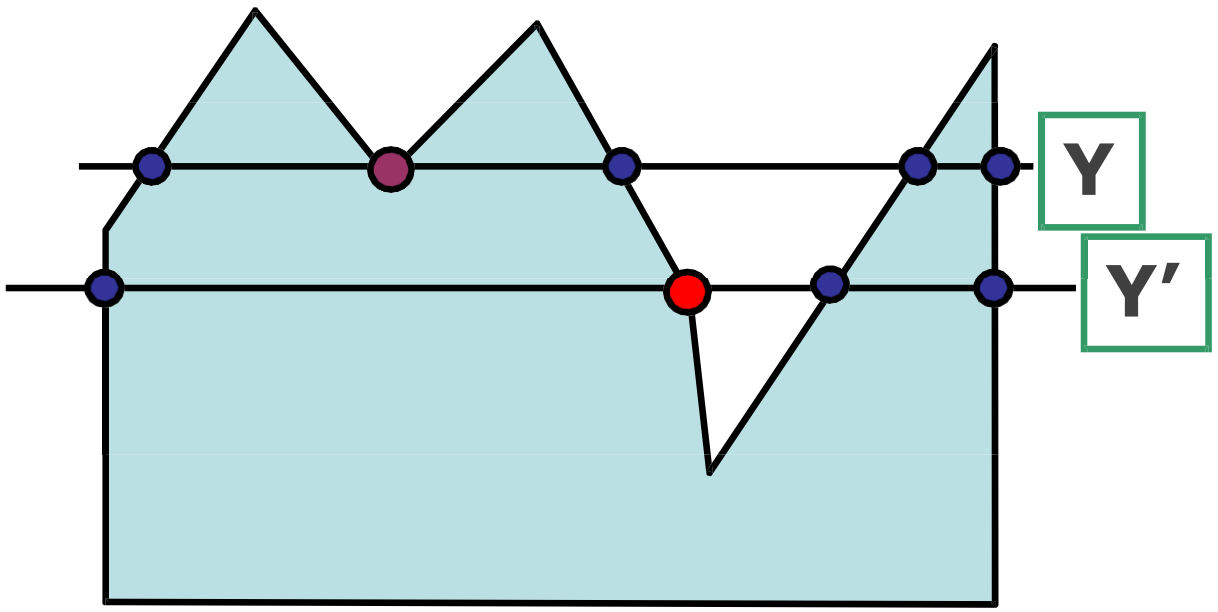
## Vertex counting in a scanline




**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

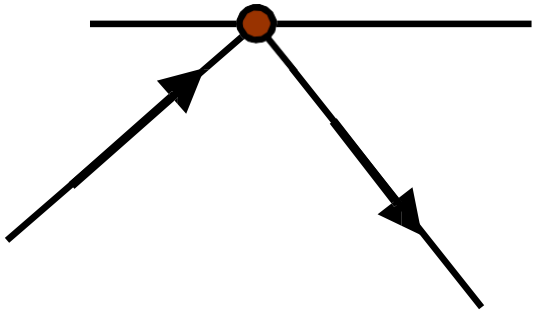



**Check the condition:**

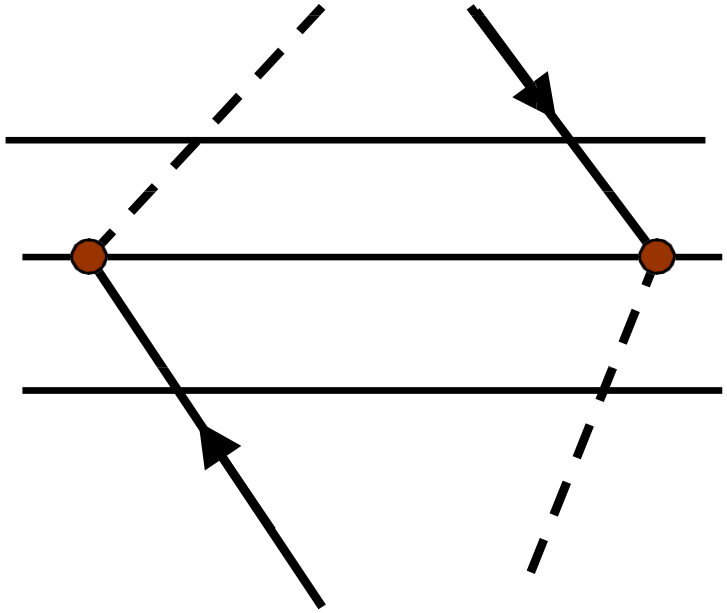
**If *end-point Y values* of two consecutive edges *monotonically increase or decrease*, count the middle vertex as a single intersection point for the scanline passing through it.**

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

If the vertex is a *local extrema*, consider (or add) two intersections for the scan line corresponding to such a shared vertex.



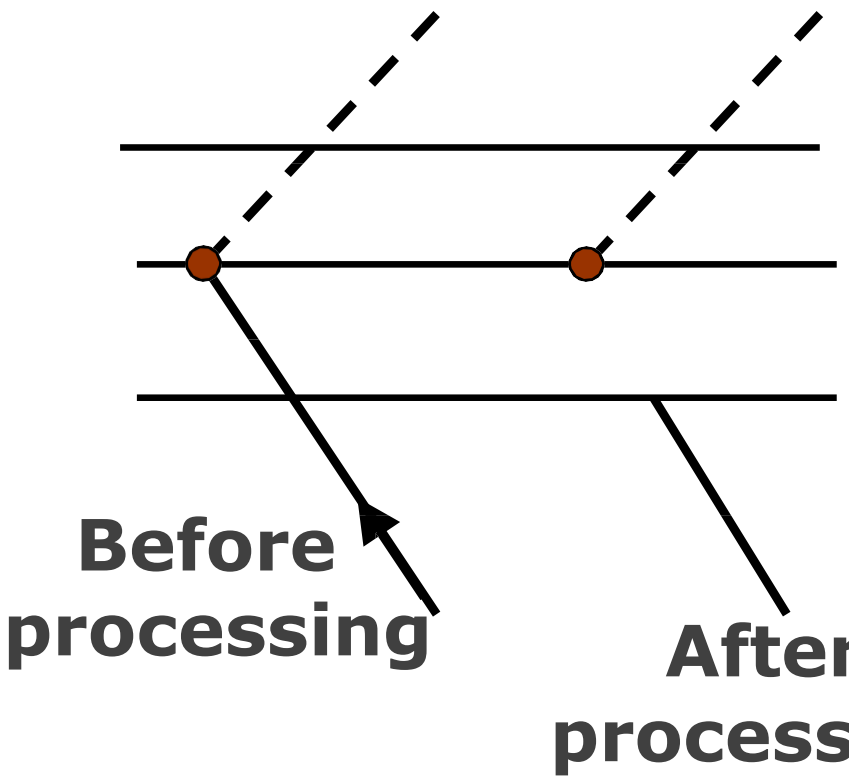
**Must avoid this to happen in cases, such as:**

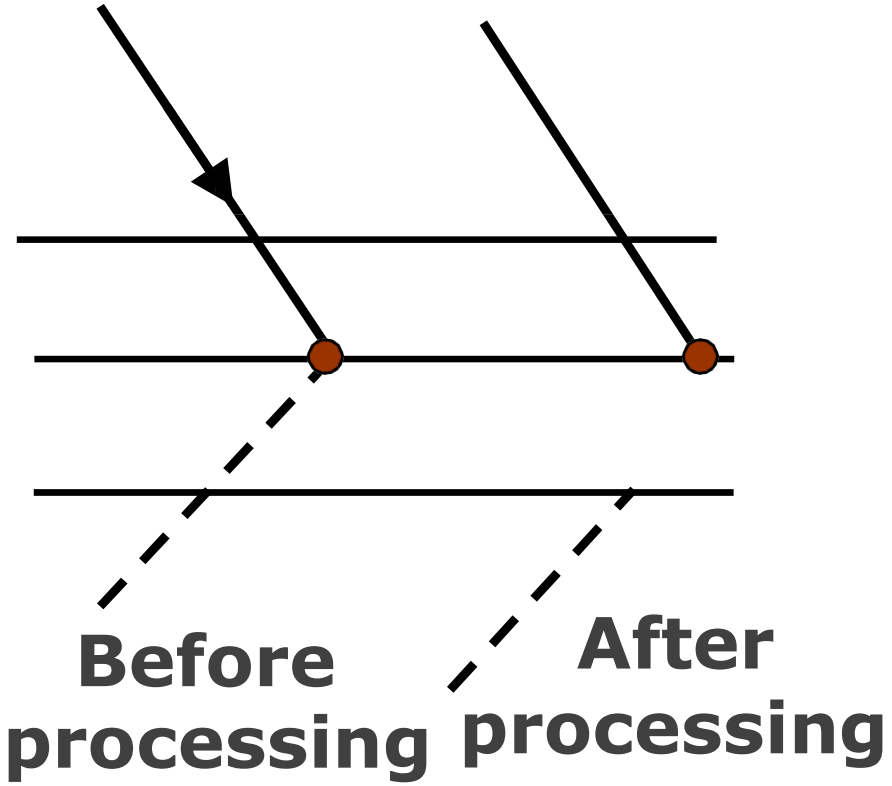


**To implement the above:**

**While processing non-horizontal edges (generally) along a polygon boundary in any order, check to determine the condition of monotonically changing (increasing or decreasing) endpoint Y values.**

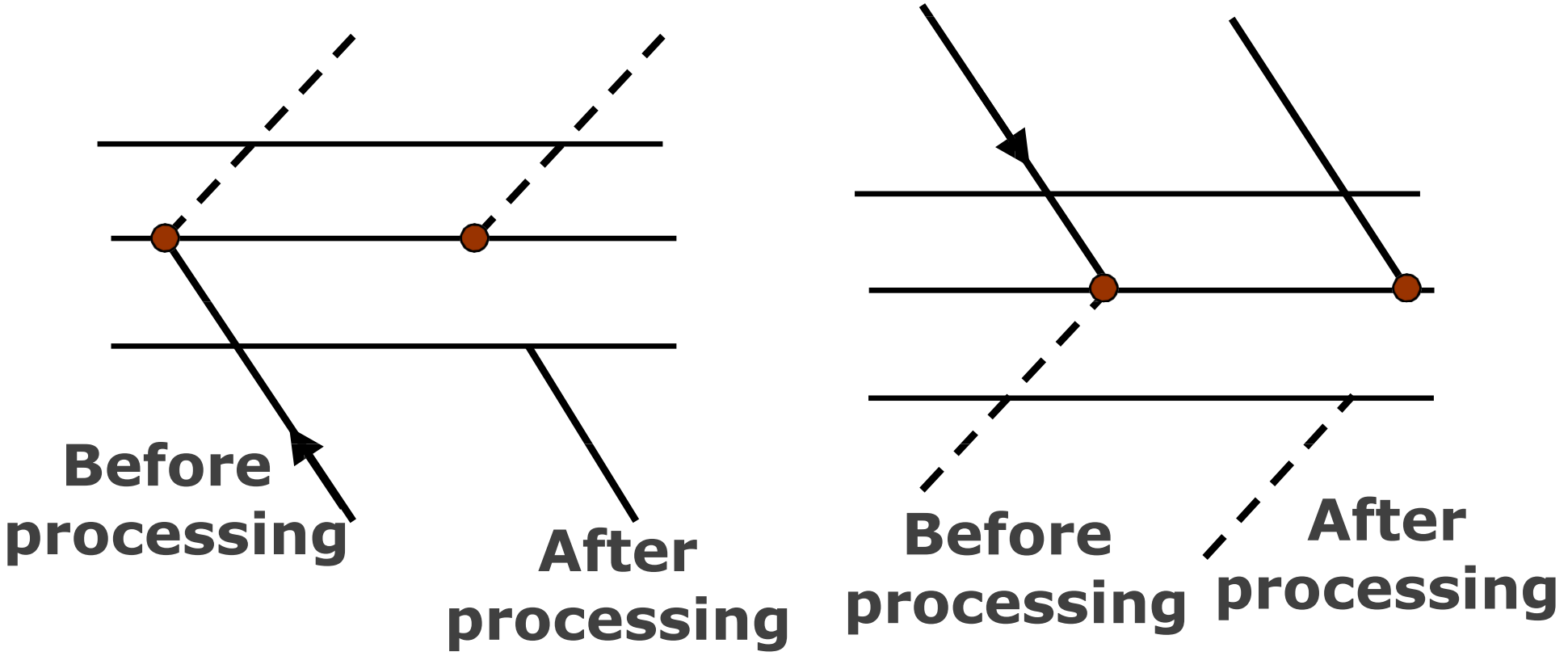
**If so:**



**To implement the above:**

**Shorten the lower edge to ensure only one intersection point at the vertex.**



# Scanline PolyFill Algorithm (revisited, in brief)

Intersect scanline with polygon edges.

Fill between pairs of intersections

Basic Structure:

For y = $Y_{min}$ to $Y_{max}$

1) intersect scanline with each edge

2) sort intersections by increasing X

3) fill pairwise (int0 -> int1, int2 -> int3, ...)

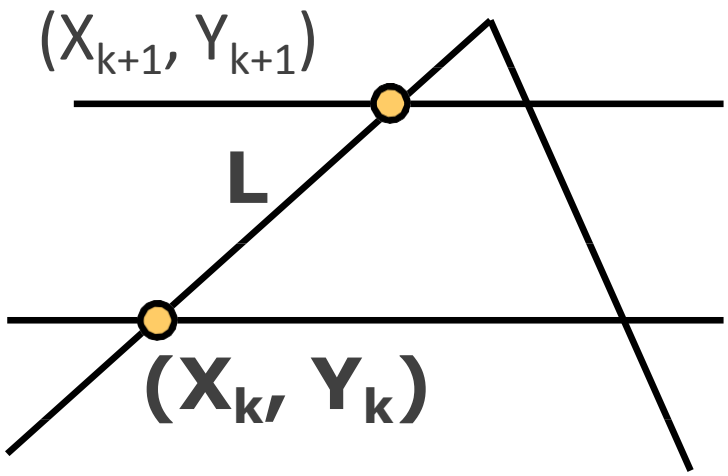
4) Update intersections for next scanline

**This is the basic structure, but we are going to handle some special cases to make sure it works correctly and fast.**

**Two important features of scanline-based polygon filling are:**

- ***scanline coherence*** **- values don't change much from one scanline to the next - the coverage (or visibility) of a face on one scanline typically differs little from the previous one.**

- ***edge coherence*** **- edges intersected by scanline "i" are typically intersected by scanline "i+1".**




**Slope of the line L (polygon edge) is:**

$$m = \frac{Y_{k+1} - Y_k}{X_{k+1} - X_k}$$

**If, $Y_{k+1} = Y_k + 1$;**

**Then, $X_{k+1} = X_k + 1/m$**

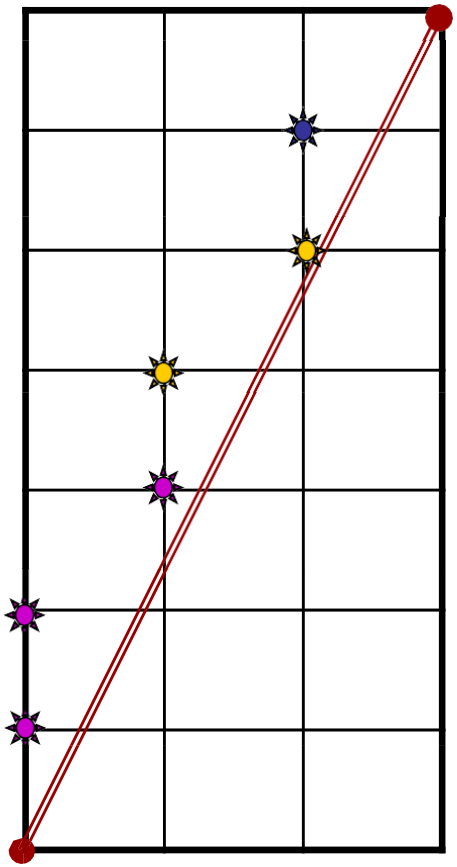
**Thus the intersection for the next scanline is obtained as:**

**$X_{k+1}$ = round $(X_k + 1/m)$, where $m = \Delta Y/\Delta X$.**

Take an example: $m = \Delta Y / \Delta X = 7/3$.

Set Counter, $C = 0$

and counter-increment, $\Delta C = \Delta X = 3$;



For the next three scan lines,
successive values of C are : 3, 6, 9.

Thus only at 3rd scanline C >= Y.

Then, $X_k$ is incremented by 1 only at 3rd scanline and set as: $C \leftarrow C - \Delta Y = 9 - 7 = 2$

Repeat the above step(s) till $Y_k$ reaches $Y_{max}$.

After 2 more scanlines: 2 + 3 + 3 = 8; 8 – 7 = 1;

After 2 more scanlines: 1 + 3 + 3 = 7;

**Data Structure Used (typical example):**

**SET (Sorted Edge table):**

**Contains all information necessary to process the scanlines efficiently.**

**SET is typically built using a bucket sort, with a many buckets as there are scan lines.**

**All edges are sorted by their $Y_{min}$ coordinate, with a separate Y bucket for each scanline.**

**Within each bucket, edges sorted by increasing X of $Y_{min}$ point.**

**Only non-horizontal edges are stored. Store these edges at the scanline position in the SET.**

**Edge structure**
**(sample record for each scanline):**

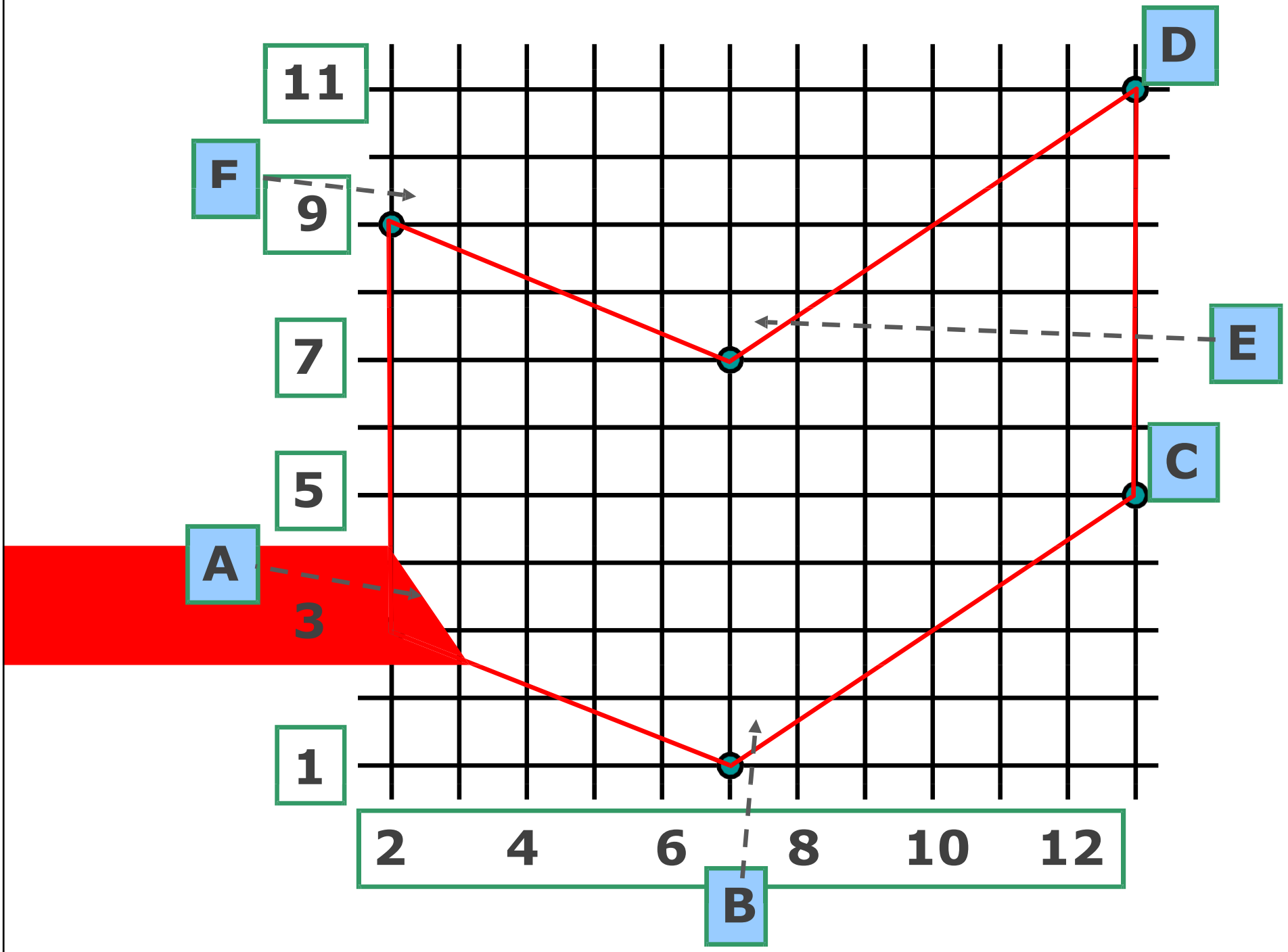
**($Y_{max}$, $X_{min}$, $\Delta X/\Delta Y$, pointer to next edge)**

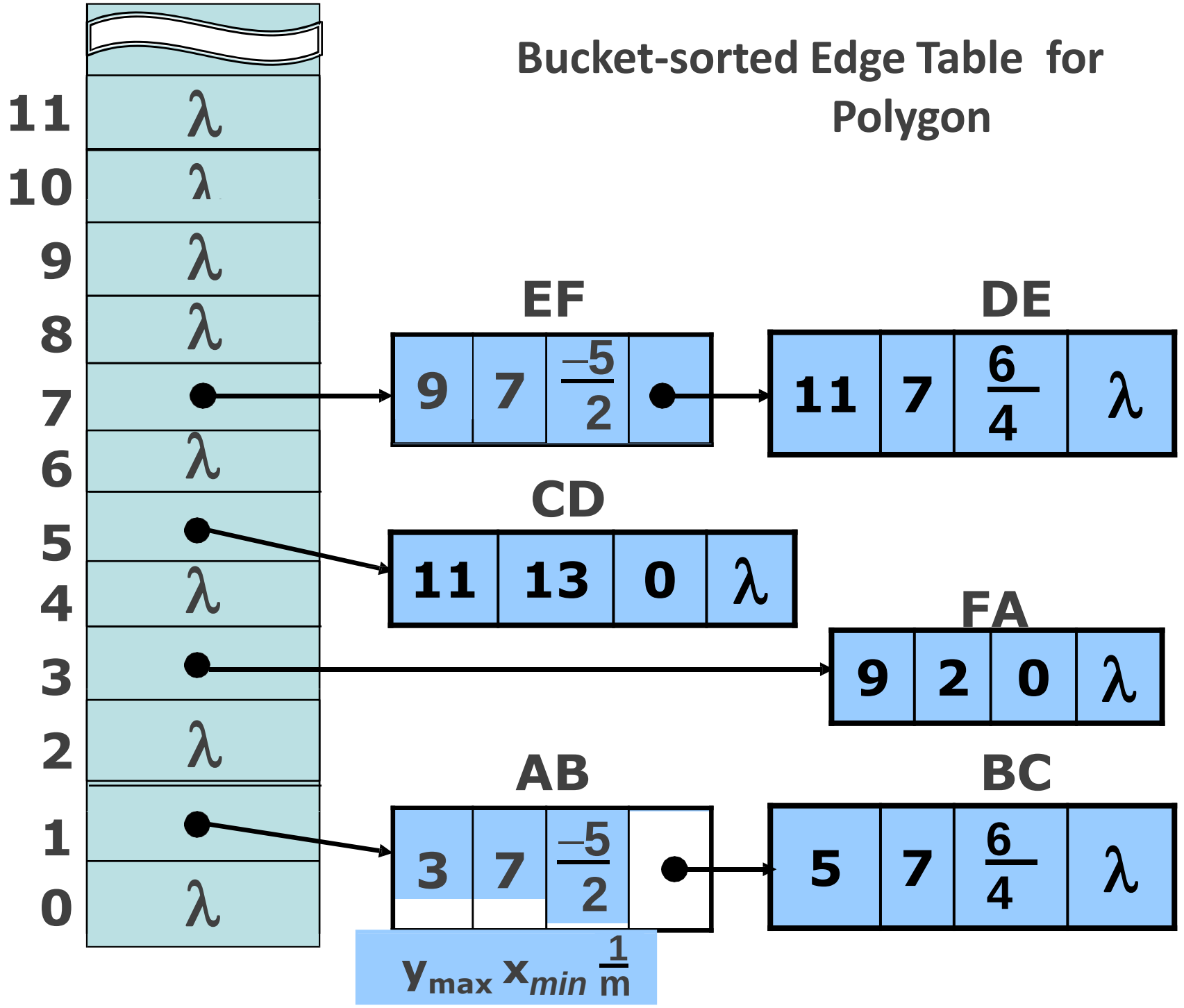
**AEL (Active edge List):**

**Contains all edges crossed by a scanline at the current stage of iteration.**

**This is a list of edges that are active for this scanline, sorted by increasing X intersections.**

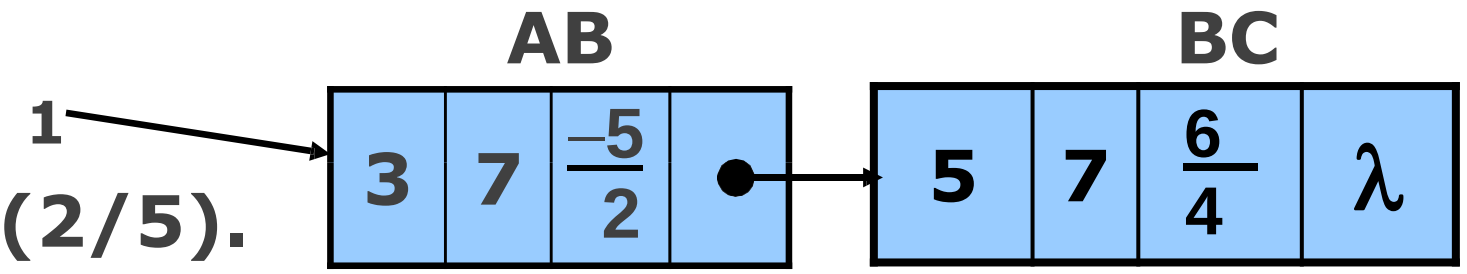
**Also called: Active Edge Table (AET).**


11
9
7
5
3
1
2
4
6
8
10
12
A
B
C
D
E
F


Bucket-sorted Edge Table for Polygon
11
10
9
8
7
6
5
4
3
2
1
0
λ
EF
9 7 −5/2
DE
11 7 6/4 λ
CD
11 13 0 λ
FA
9 2 0 λ
AB
3 7 −5/2
BC
5 7 6/4 λ
y_max x_min 1/m

AB:

m = $\Delta Y / \Delta X$ = - (2/5).




Set Counter, C = 0 ; and

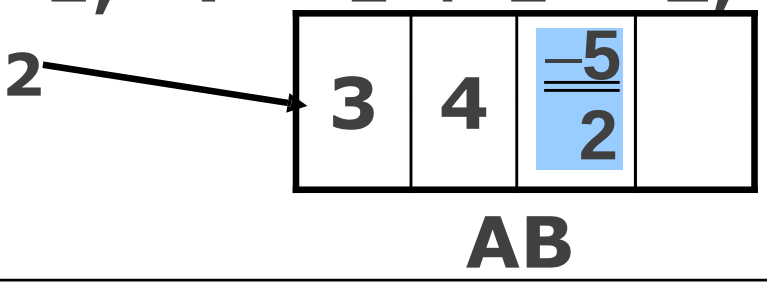
counter-increment, $\Delta C = \min(\Delta X, \Delta Y) = 2$ (= $\Delta Y$)

Update for AB (-ve m), when $Y_K = 2$; Y = 1:

For the next three left (-ve) vertical (Y) scan lines, successive values of C are : 2, 4, 6; X = 7 - 3 = 4;

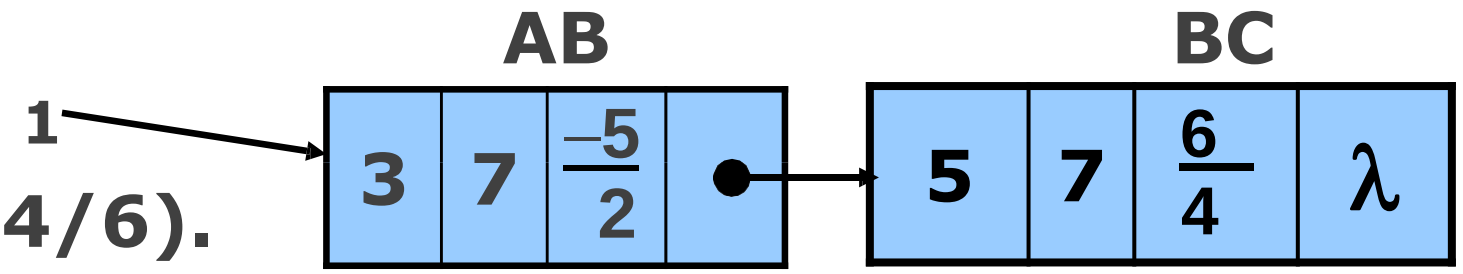
Thus only at 3rd iteration: C >= $\Delta X$.

Then, Y is incremented by 1 only at 3rd scanline and set: C ← C – $\Delta X$ = 6 – 5 = 1; Y = 1 + 1 = 2;
Stop as $Y = Y_K$.



**BC:**

**m = ΔY/ΔX = (4/6).**

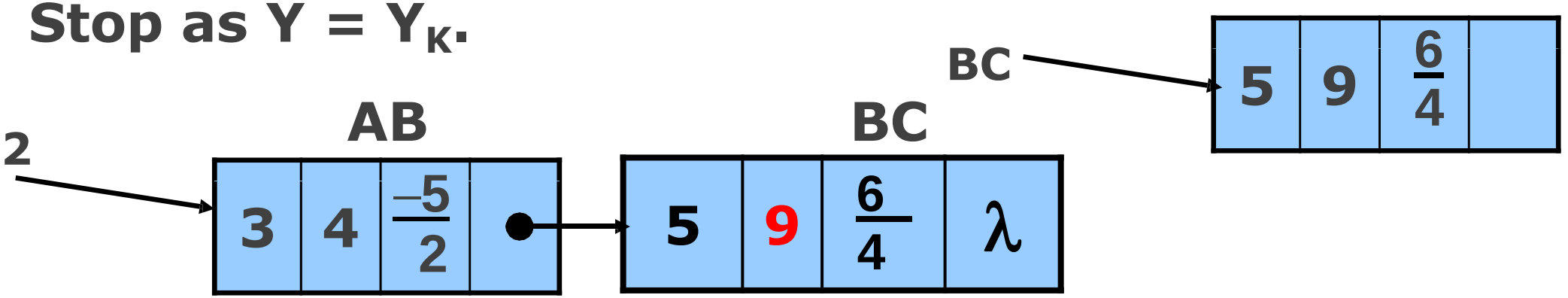


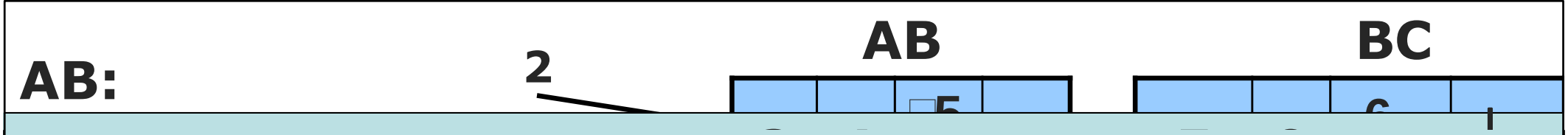

**Set Counter , C = 0 ; and**

**counter-increment, ΔC = min (ΔX, ΔY) = 4 (= ΔY)**

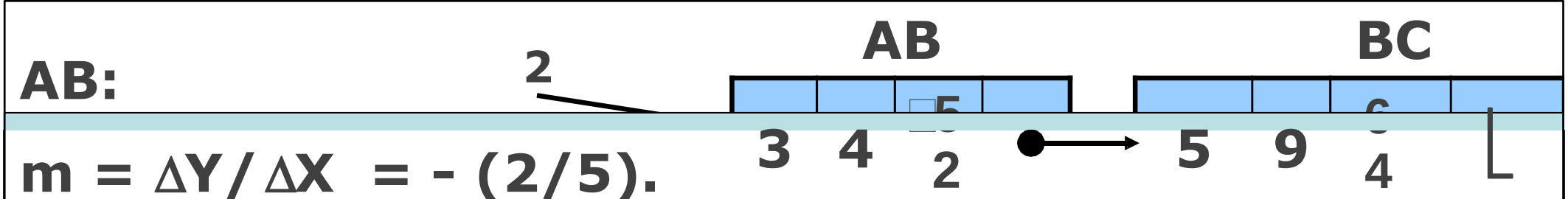
**Update for BC (+ve m), when $Y_K$ = 2; Y = 1:**
**For the next two right vertical (Y) scan lines, successive values of C are : 4, 8; X = 7 + 2 = 9;**

**Thus only at 2nd iteration: C >= ΔX.**
**Then, Y is incremented by 1 only at 2nd scanline and set : C ← C – ΔX = 8 – 6 = 2; Y = 1 + 1 = 2;**
**Stop as Y = $Y_K$.**






**Do you need all this ??**




m = $\Delta Y / \Delta X$ = - (2/5).

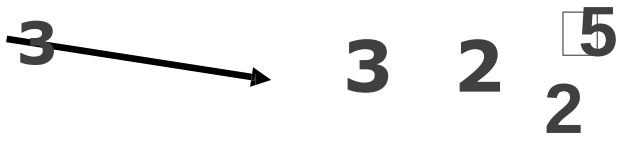
Counter (from earlier iteration), C = 1 ; and

counter-increment, $\Delta C = \min(\Delta X, \Delta Y) = 2$ (= $\Delta Y$)

Update for AB (-ve m), when $Y_K = 3$; Y = 2:
For the next two left (-ve) vertical (Y) scan lines, successive values of C are : 3, 5; X = 4 - 2 = 2;

Thus only at 2nd iteration: C >= $\Delta X$.

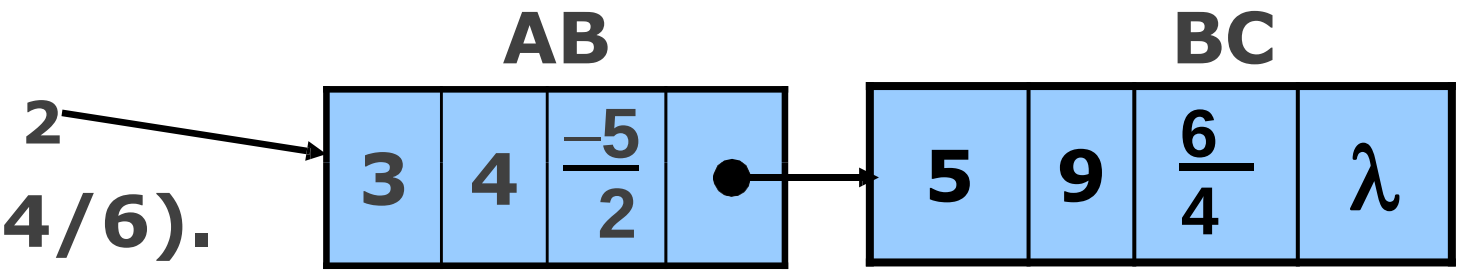
Then, Y is incremented by 1 only at 2nd scanline and set: C ← C – $\Delta X$ = 5 – 5 = *0*; Y = 2 + 1 = 3;
Stop as Y = $Y_K$.




AB: To go out of AEL, after use, as $Y_{max} = Y_K$.

**BC:**

**m = ΔY/ΔX = (4/6).**




**Counter (from earlier iteration) , C = 2 ; and**

**counter-increment, ΔC = min (ΔX, ΔY) = 4 (= ΔY)**

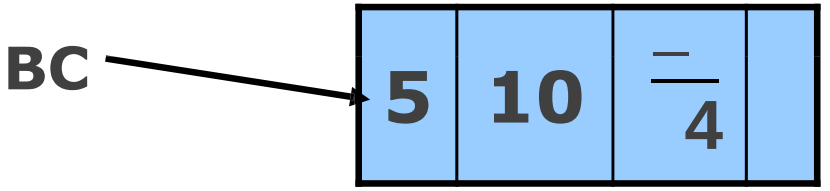
**Update for BC (+ve m), when $Y_K$ = 3; Y = 2:**

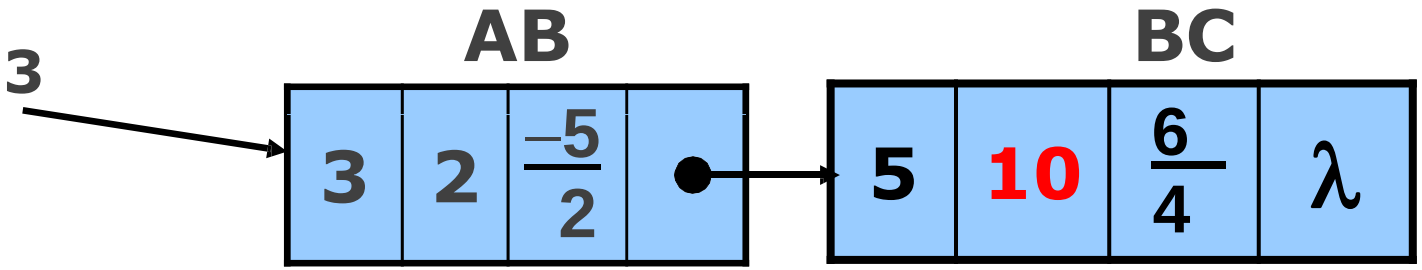
**For the next right vertical (Y) scan line, the successive value of C is : 6; X = 9 + 1 = 10;**

**Thus only at 1st iteration: C >= ΔX.**

**Then, Y is incremented by 1 only at 1st scanline and set : C ← C – ΔX = 6 – 6 = *0*; Y = 2 + 1 = 3;**

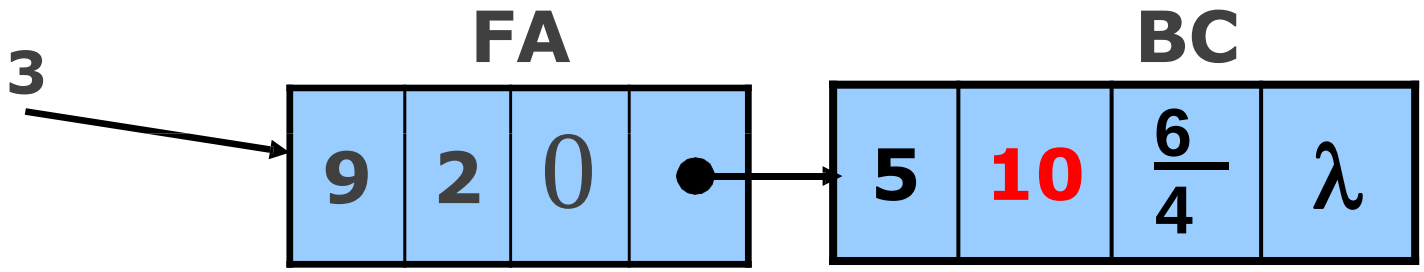
**Stop as Y = $Y_K$ = ??.**



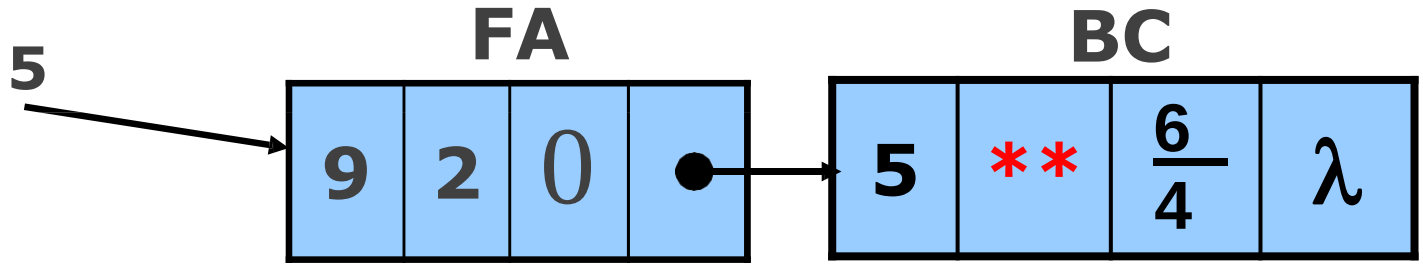

***<< - Is this OK ??***

**After post-processing (update from SET) at 3rd scanline:**



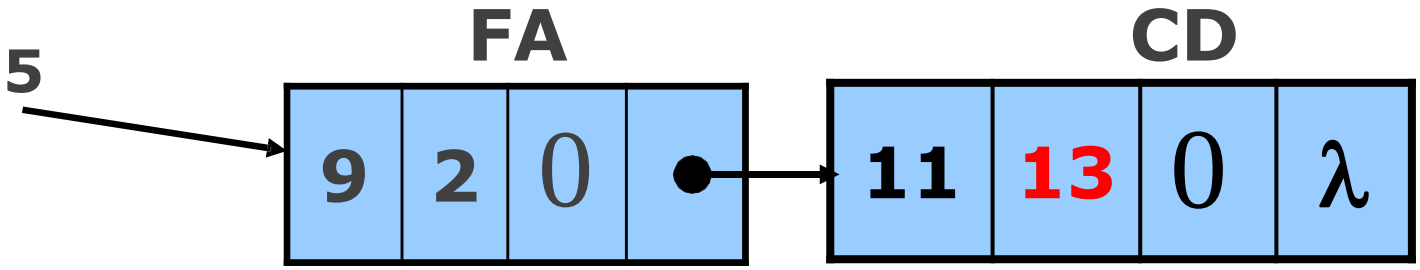

**Home task:**

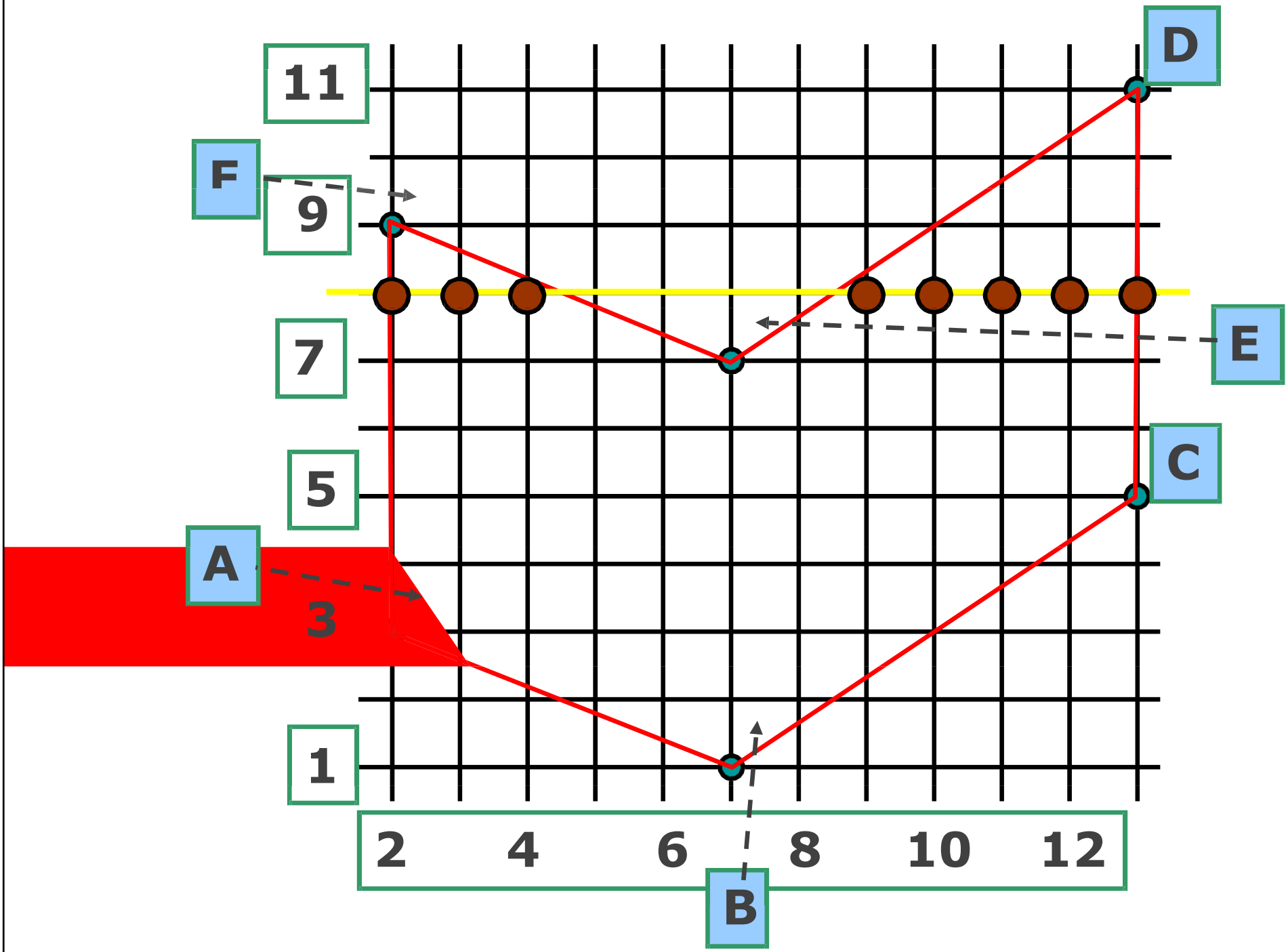
**Complete the next two sets of iterations yourself, till you get :**



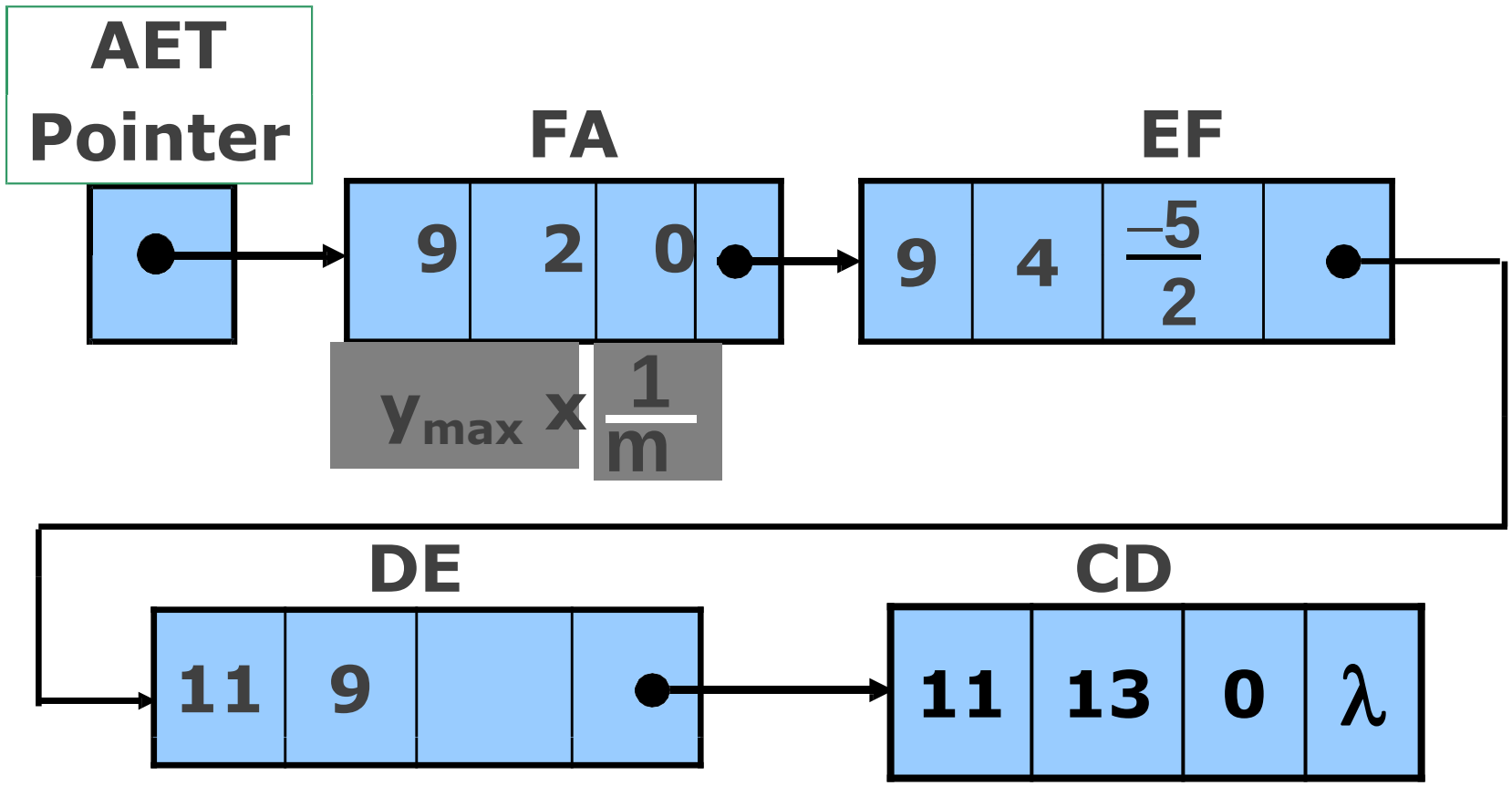

**** = 10 + 3 = 13, why ??**

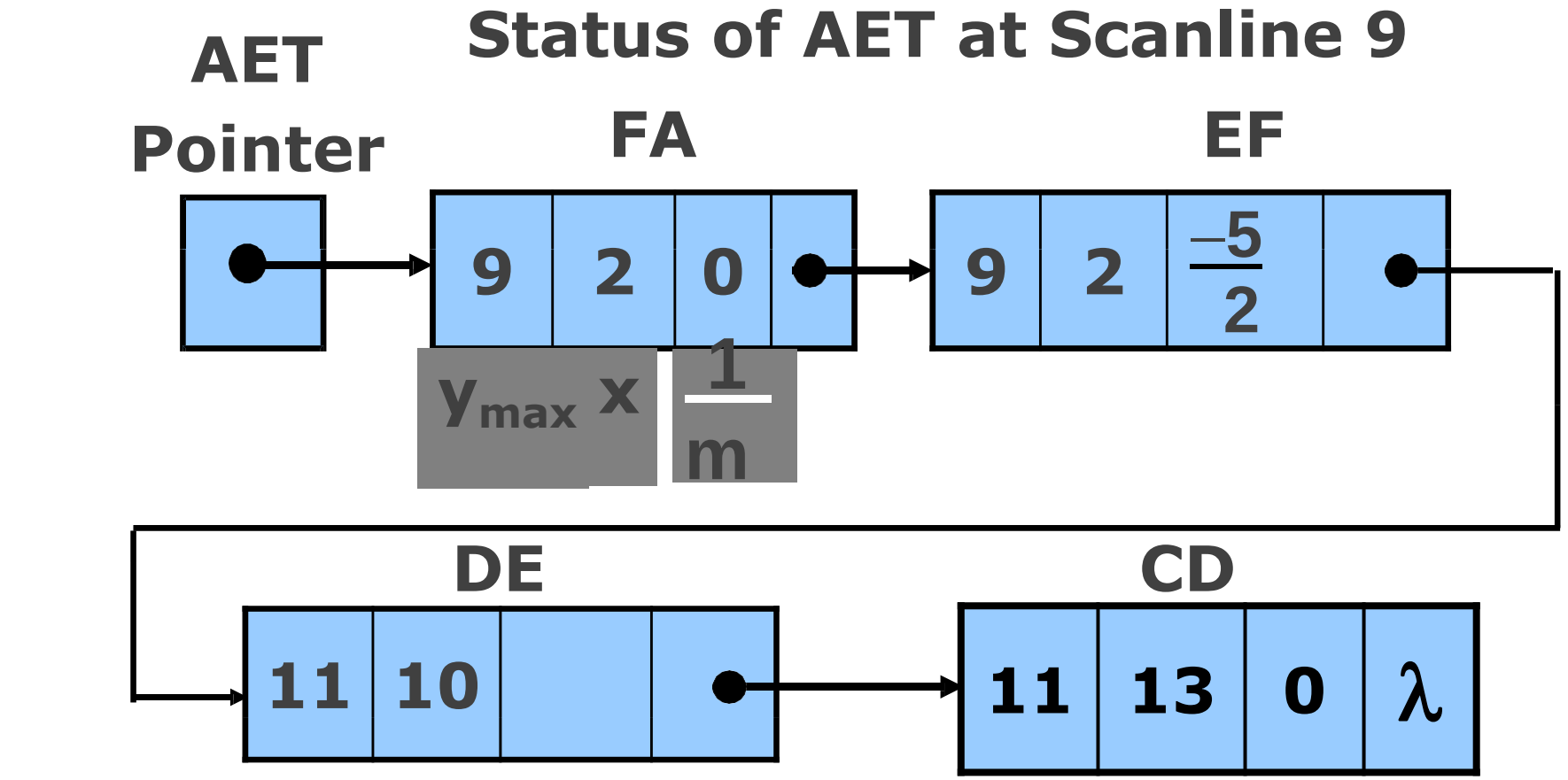
**After post-processing (update from SET) at 5th scanline:**

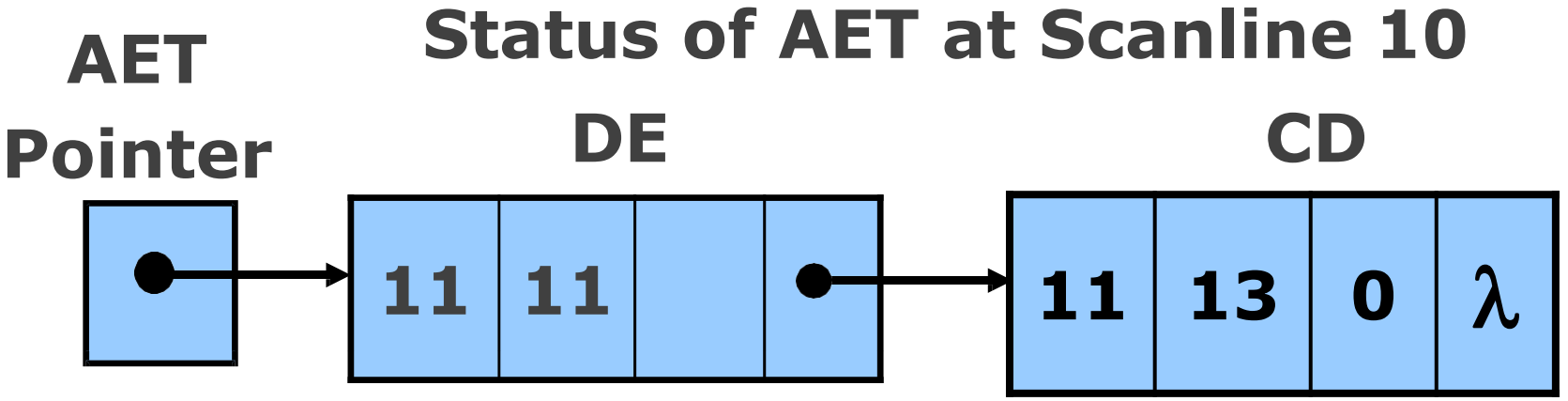



11
F
9
7
5
A
3
1
2
4
6
8
10
12
B
D
E
C

# Status of AET at Scanline 8




Status of AET at Scanline 9
AET Pointer
FA
EF
9
2
0
9
2
−5
2
$y_{max}$
x
1
m
DE
CD
11
10
11
13
0
λ


Status of AET at Scanline 10
AET Pointer
DE
CD
11
11
11
13
0
λ

**Precautions:**

**Intersection has an integer Y coordinate**

**If this point is the $Y_{min}$ of the edge's endpoints, count it.**

***If the edge is horizontal and on the scanline, don't count it.***

**During iteration process with each scanline, the *AET is updated*.**

**For each scanline the AET keeps track of the set of edges it has to intersect and stores the intersection points in it. The sorting of the entries is w.r.t the X-intersection values.**

**Have a re-look at it.**

## Processing Steps:

**Set Y to smallest Y in SET entry (first non-empty bucket)**

- **Initialize AET to be empty**

- **Repeat until both AET and SET are empty**

  **(i) Move from SET bucket Y to AET, those edges whose $Y_{min} = Y$.**

  **(ii) Sort AET on X (simple, as SET is pre-sorted anyway).**

(iii) Fill pixels in scanline Y using pairs of X-coordns. from AET.

(iv) Increment scanline by 1.

(v) Remove from AET those entries for which $Y = Y_{max}$ (edges not involved).

(vi) For each non-vertical edge in AET, update X for new Y.

END-LOOP

**The algorithm:**

*scan-fill(polygon)*

*Construct the Edge table (ET)*

$Y_{min}$ = *min (all Y in the ET)*

*AET = null*

*for Y =* $Y_{min}$ *to* $Y_{max}$

*Merge-sort ET[Y] into AET by X value*

*Fill between pairs of X in AET.*

```
    for each edge in AET
        if edge.Ymax = Y
            remove edge from AET
        else
            edge.X = edge.X + dx/dy
        endif
    sort AET by X value
end scan_fill
```

# End of Lectures on

# POLYFILL -

# SCAN CONVERSION

# of a POLYGON