Rasterisation of Regions

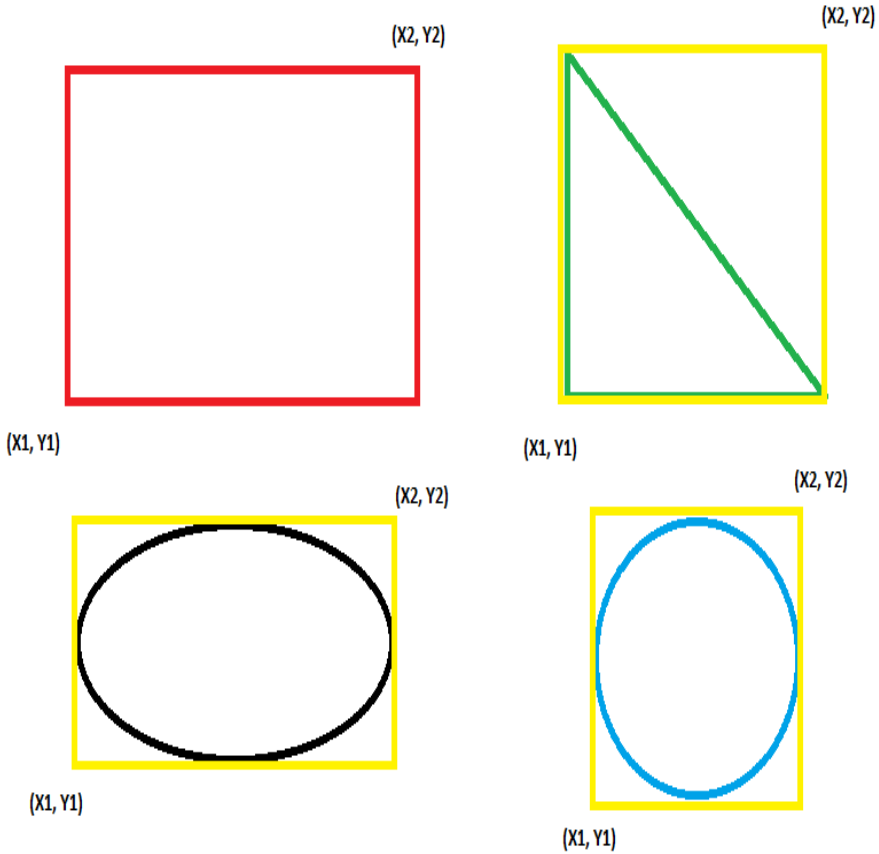
Dr. V Masilamani

masila@iiitdm.ac.in

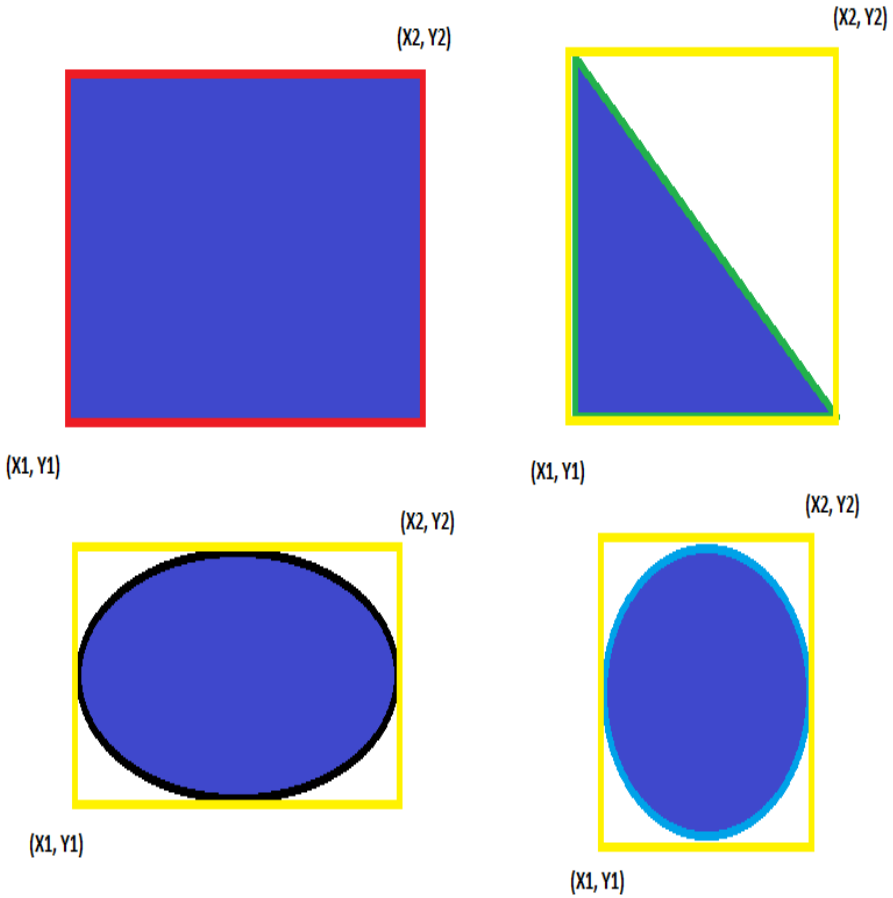
Department of Computer Science and Engineering
IIITDM Kancheepuram
Chennai-127

Overview

Rasterisation of Regular Geometric 2D Regions



Rasterisation of Regular Geometric 2D Regions (cont.)



Rasterisation of Regular Geometric 2D Regions (cont.)

- Rectangular Area
  - i/p: Diagonal vertices(Bottom-Left, Top-Right) $(x_1, y_1), (x_2, y_2)$
  - For each int (x, y), st $x_1 \leq x \leq x_2$ and $y_1 \leq y \leq y_2$ DrawPixel(x,y)
- Triangular Area
  - i/p: Vertices of the triangle to be scan converted
  - Find minimum bounding rectangle with Diagonal vertices(Bottom-Left, Top-Right) $(x_1, y_1), (x_2, y_2)$ for the triangle
  - For each int (x, y), st $x_1 \leq x \leq x_2$ and $y_1 \leq y \leq y_2$ Check if (x, y) is inside the triangle
  - if yes, DrawPixel(x,y)
- Circular Disc
  - i/p: Radius and center of the circular disc to scan converted

Rasterisation of Regular Geometric 2D Regions (cont.)

- Find minimum bounding rectangle with Diagonal vertices(Bottom-Left, Top-Right) $(x_1, y_1), (x_2, y_2)$ for the circle
- For each int (x, y), st $x_1 \leq x \leq x_2$ and $y_1 \leq y \leq y_2$ Check if (x, y) is inside the circle
- if yes, DrawPixel(x,y)

- Area Enclosed by Ellipse
  - i/p: Radius and center of the circular disc to scan converted
  - Find minimum bounding rectangle with Diagonal vertices(Bottom-Left, Top-Right) $(x_1, y_1), (x_2, y_2)$ for the Ellipse
  - For each int (x, y), st $x_1 \leq x \leq x_2$ and $y_1 \leq y \leq y_2$ Check if (x, y) is inside the ellipse
  - if yes, DrawPixel(x,y)

Rasterisation of Arbitrary 2D Regions

How to represent arbitrary region

- Boundary-Defined Region: Represent using boundary points with a fixed colour, called boundary colour
- Interior-Defined Region: Represent the region points with a fixed colour called Region-Colour

What is the expected outcome of Rasterisation

- In case of Boundary-Defined Region, display all pixels in the region bounded by the boundary, with a given colour
- In case of interior-Defined Region , display all pixels with Region-Colour in a new given colour

Challenge: Inside-Outside Test

An Easy Approach to overcome the challenge:
Assume that a seed point p (a point inside the region) is known, Apply the following procedure

Rasterisation of Arbitrary 2D Regions (cont.)

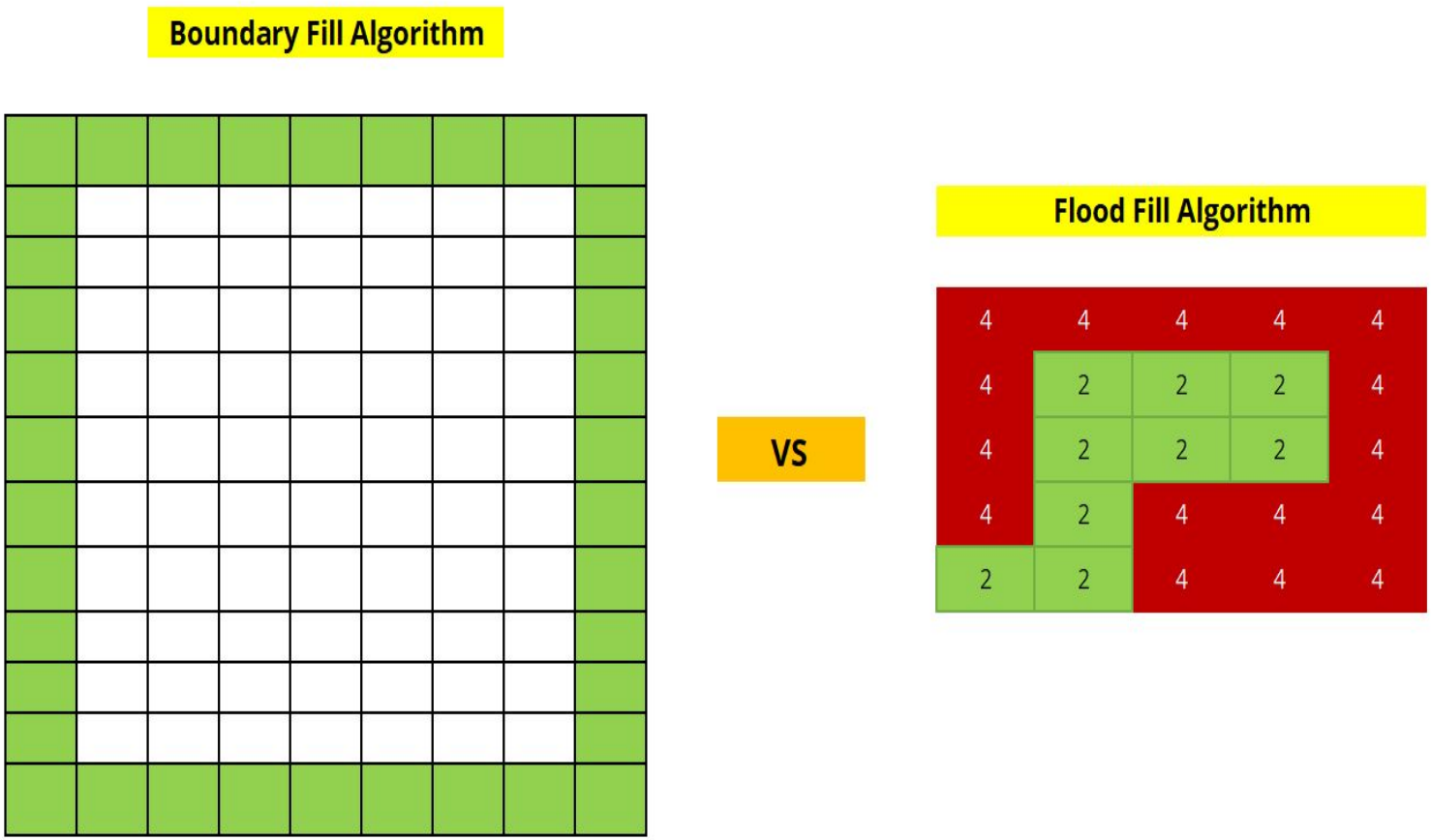
- Display the seed point p with given colour c
- For each neighbour n of p, display n with colour c if n is not a boundary pixel
- For each neighbour n of pixel with colour c, display n with colour c if n is not boundary point and also not already in colour c

If the above procedure is used when Boundary-Defined Region is given, the procedure is called as **Boundary-Fill Algorithm**

If the above procedure is used when Interior-Defined Region is given , the procedure is called as **Flood-Fill Algorithm**

Rasterisation of Arbitrary 2D Regions (cont.)

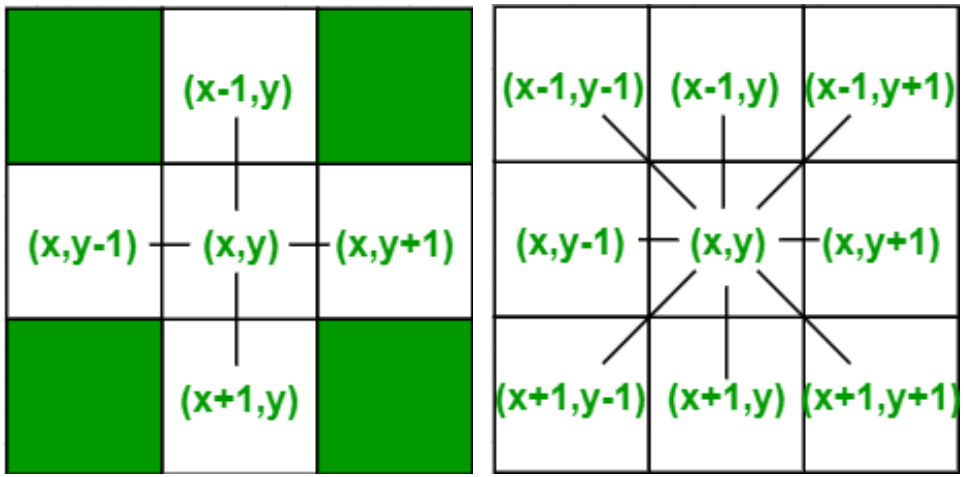
Input of Boundary-Fill Algorithm Vs Input of Flood-Fill Algorithm



Rasterisation of Arbitrary 2D Regions (cont.)

- In Left Image, Green Pixels are boundary pixels, the white pixels need to be coloured with given colour
- In Right Image, Green Pixels are region pixels, the green pixels need to be coloured with given colour

Neighbouring Pixels?



Boundary-Fill Algorithm

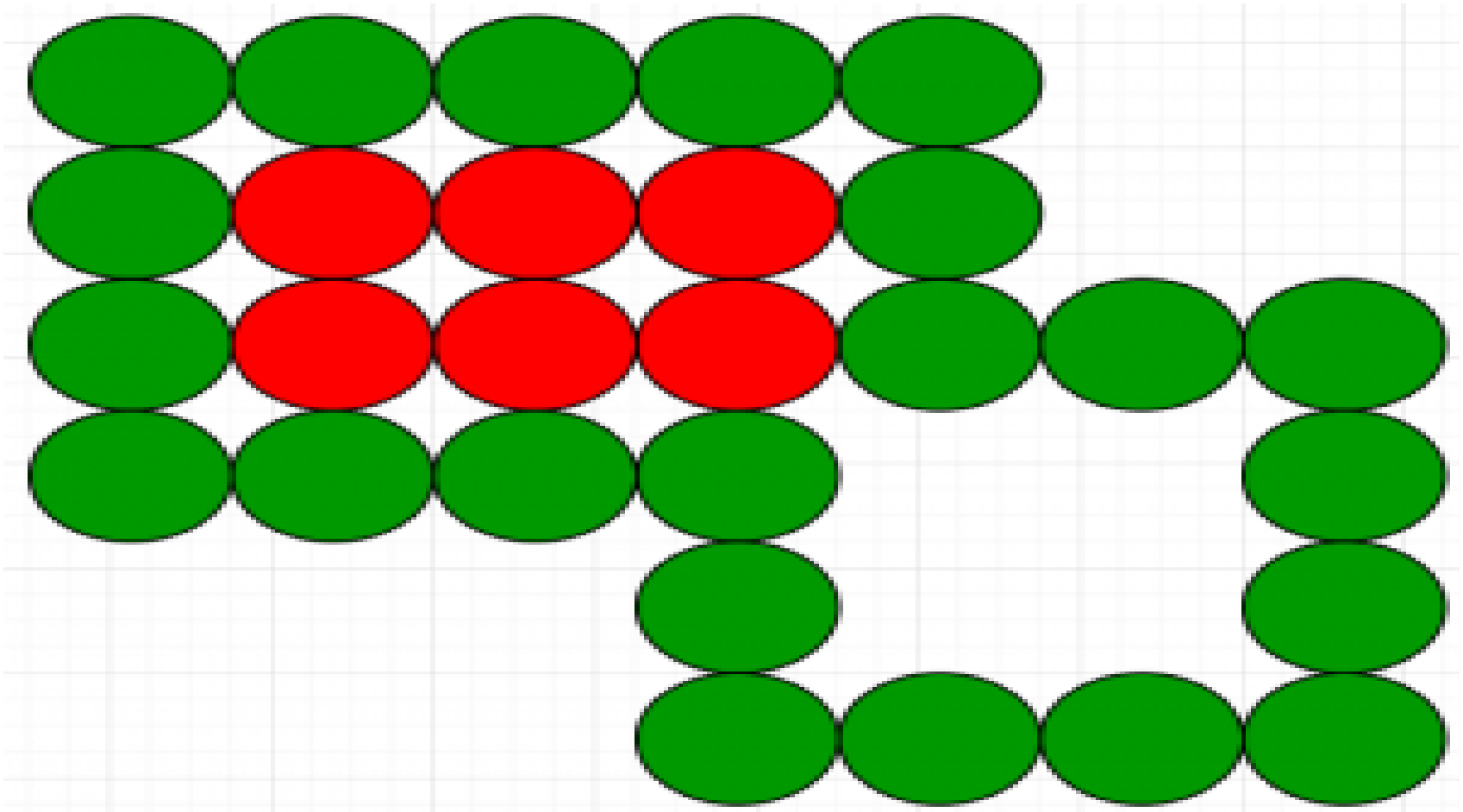
```
1
2  // I[x][y] for 0<=x <m;  0<=y<n
3  // is available in the memory
4
5
6  //  I has all boundary pixels of the  region
7  //  to be filled in "bc",
8  //  and other pixels in background color
9
10 //pixles in the region is to begiven fc
11
12
13 Void boundaryFill8(int x, int y, int fc, int bc)
14 {
15
16 if( (I(x, y) != bc) && (I(x, y) != fc) )
17  {
18          DisplayPixel(x, y, fc);
```

Boundary-Fill Algorithm (cont.)

```
19            boundaryFill8(x + 1, y, fc, bc);
20            boundaryFill8(x, y + 1, fc, bc);
21            boundaryFill8(x - 1, y, fc, bc);
22            boundaryFill8(x, y - 1, fc, bc);
23            boundaryFill8(x - 1, y - 1, fc, bc);
24            boundaryFill8(x - 1, y + 1, fc, bc);
25            boundaryFill8(x + 1, y - 1, fc, bc);
26            boundaryFill8(x + 1, y + 1, fc, bc);
27        }
28    }
```

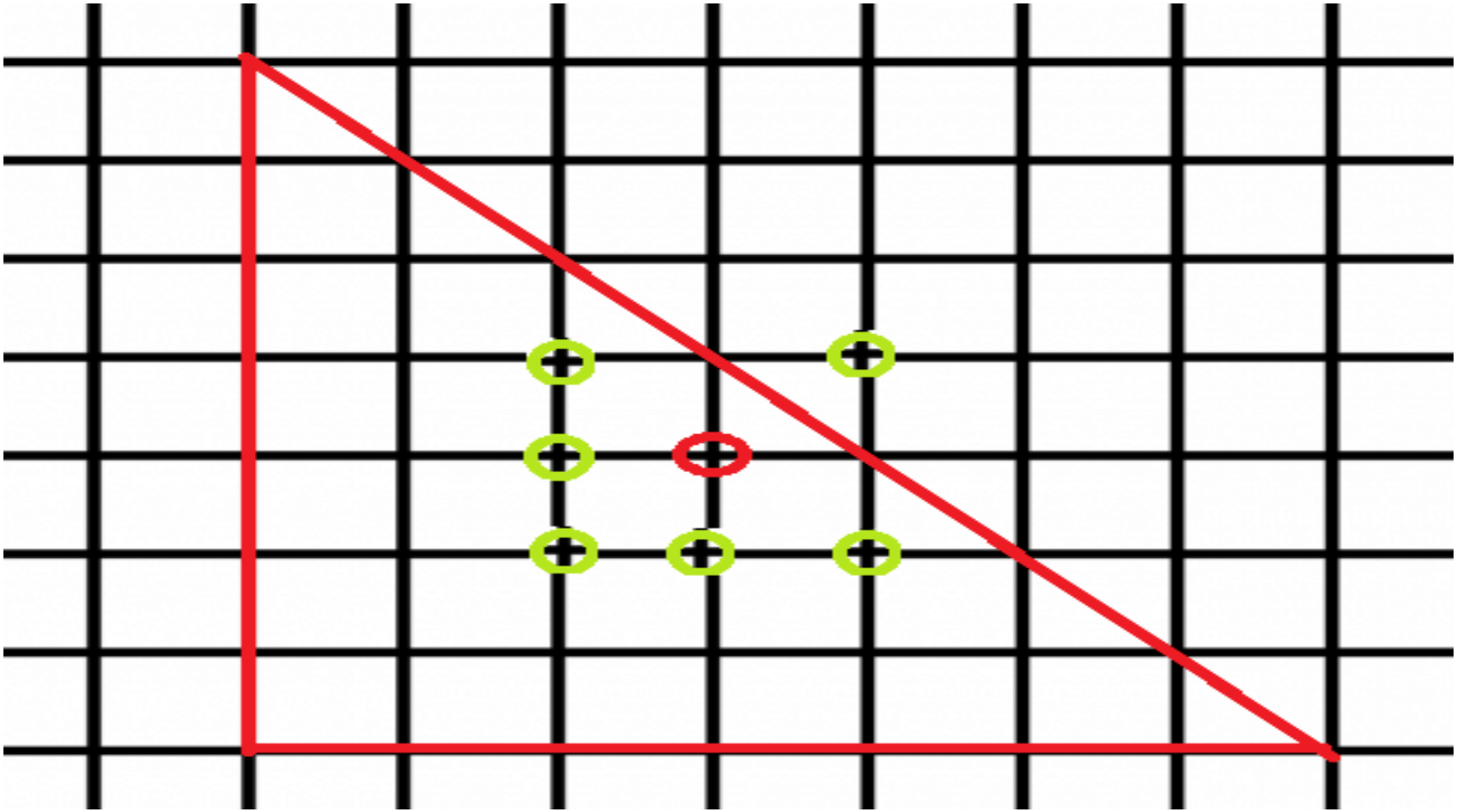
Boundary-Fill Algorithm (cont.)

- Issue with 4 Neighbours: incomplete filling of region
- Soln: Use more seeds, and repeat the algorithm



Boundary-Fill Algorithm (cont.)

- Issue with 8 Neighbours: Leaking outside the boundary
- Soln: Thicken the boundary, and then apply the algorithm



Flood-Fill Algorithm

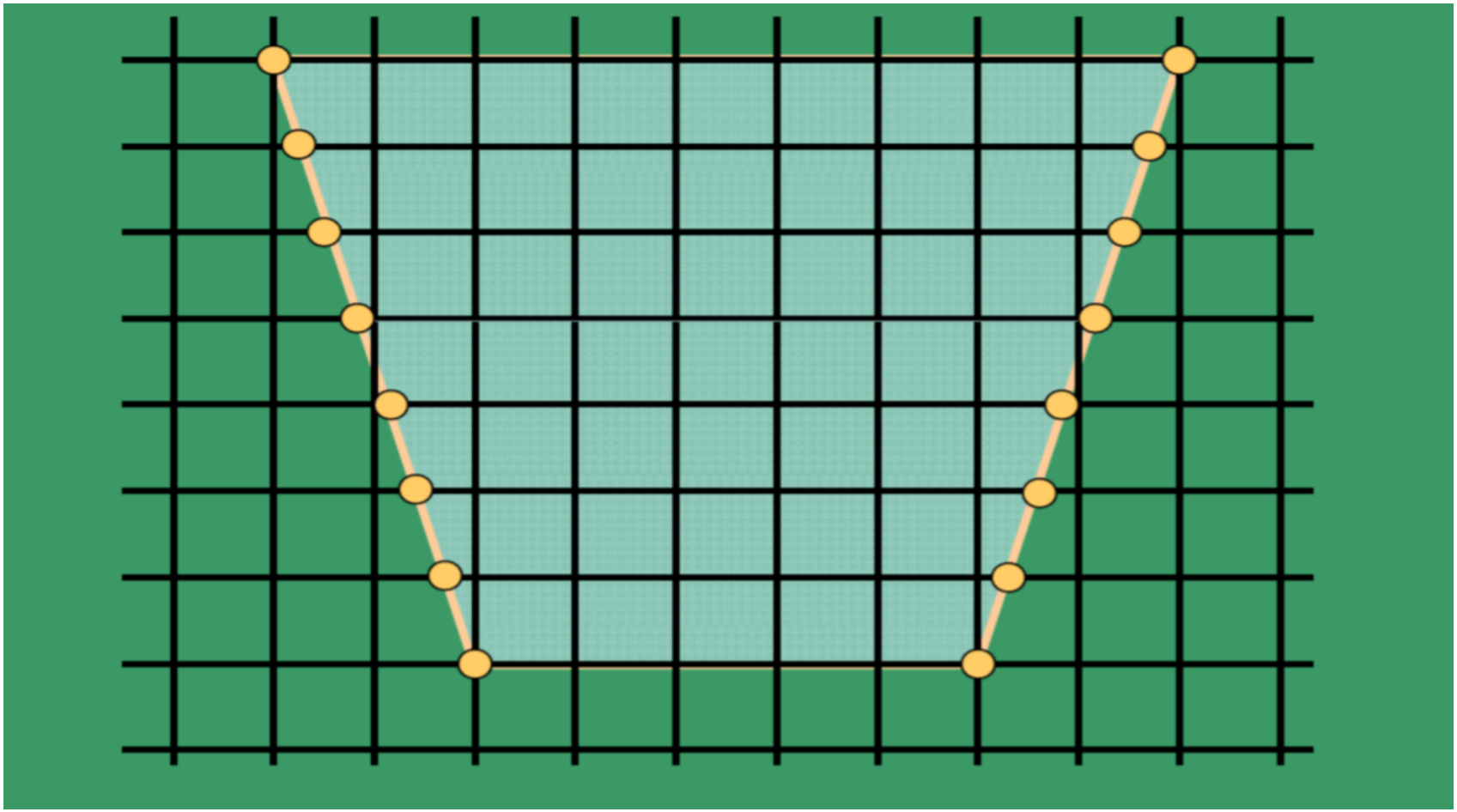
```
1
2  //  A recursive function to replace
3  //  previous color 'prevC' at  '(x, y)'
4  //  and also all surrounding pixels of (x, y)
5  //  with new color 'newC' and
6
7  // I[M][N] has pixel values prevC
8  // in the region to be scan converted
9
10 FloodFill(I[M][N], x, y, prevC, newC)
11 {
12    If x or y is outside the range, then return.
13    If color of I[x][y] is not same as prevC, then return
14
15       FloodFill(I, x+1, y, prevC, newC);
16       FloodFill(I, x-1, y, prevC, newC);
17       FloodFill(I, x, y+1, prevC, newC);
18       FloodFill(I, x, y-1, prevC, newC);
```

Flood-Fill Algorithm (cont.)

```
19        FloodFill(I, x+1, y+1, prevC, newC);
20        FloodFill(I, x+1, y-1, prevC, newC);
21        FloodFill(I, x-1, y-1, prevC, newC);
22        FloodFill(I, x-1, y+1, prevC, newC);
23
24    }
```

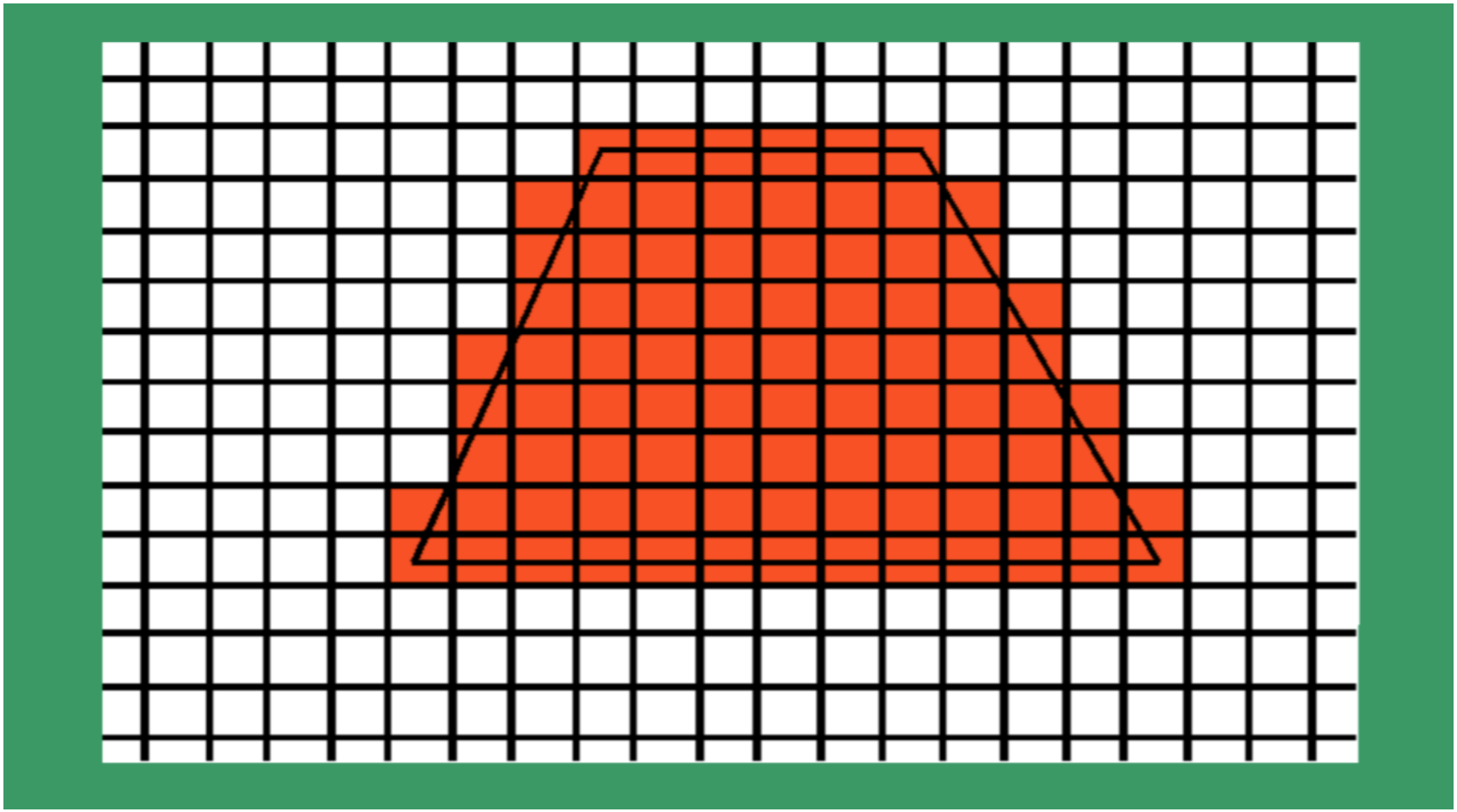
Scan Converting Polygon Region

- Brute Force Algo: 1) Scan Convert each line segment of the polygon boundary 2) Use Boundary Fill algo to scan convert the region



Scan Conversion - POLYFILL

- Issues with Brutre Force Algo:
  1) Not Efficient (More time) 2) Leaking or Partial Filling or Thickening the Boundary 3) Aliasing



Scanline POLYFILL Algorithm

Outline of the Scanline Poly Fill Algorithm:

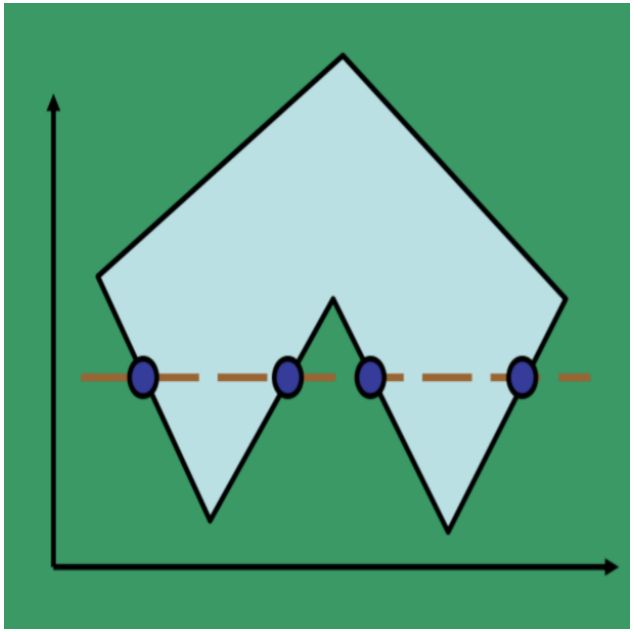
- Find minimum bounding rectangle, say the (X_{min}, Y_{min}) and (X_{max}, Y_{max}) are the BottomLeft and TopRight vertices of the rectangle respectively
- Each row form Y_{min} to Y_{max} is called as scan line
- No of scanlines: $= Y_{max} - Y_{min} + 1$
- For each scanline do
  - Obtain intersection points of scanline with polygon edges.
  - For each scan line, Sort intersection points from left to right (ie based on x coordinates)

Scanline POLYFILL Algorithm (cont.)

- Form pairs of intersections from the list*
- Fill within pairs using rasterization of line segments
- Intersection points are computed for each scan line
- Stop when scan line has reached Y_{max}

* - Intersections at vertices require special handling

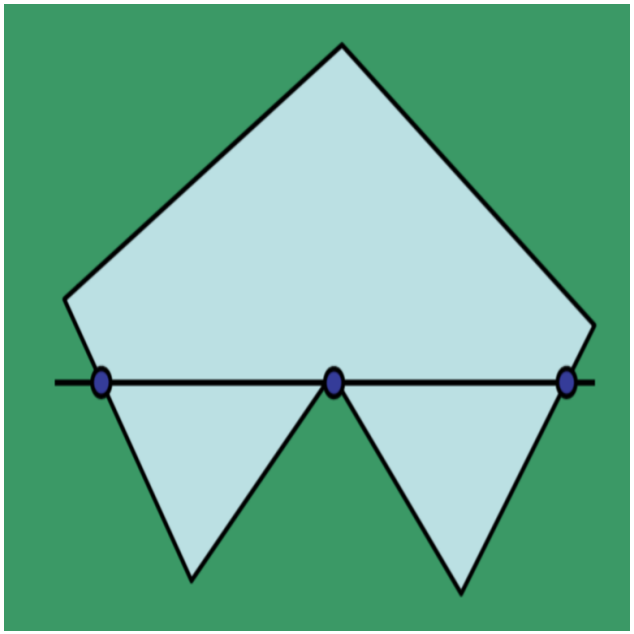
Scanline POLYFILL Algorithm



- Four points are shown on a scan line, call them $p_1,\ p_2,\ p_3,\ p_4$
- **Observation:** All the points in the segments (p_1, p_2) and (p_3, p_4) are inside the polygon, whereas all points inside the segment (p_2, p_3) are outside the polygon
- **Generalization?:** All points in odd segment will be inside the polygon, whereas all the points in the even segments are outside the polygon

Example to contradict the generalization and fixing the issue

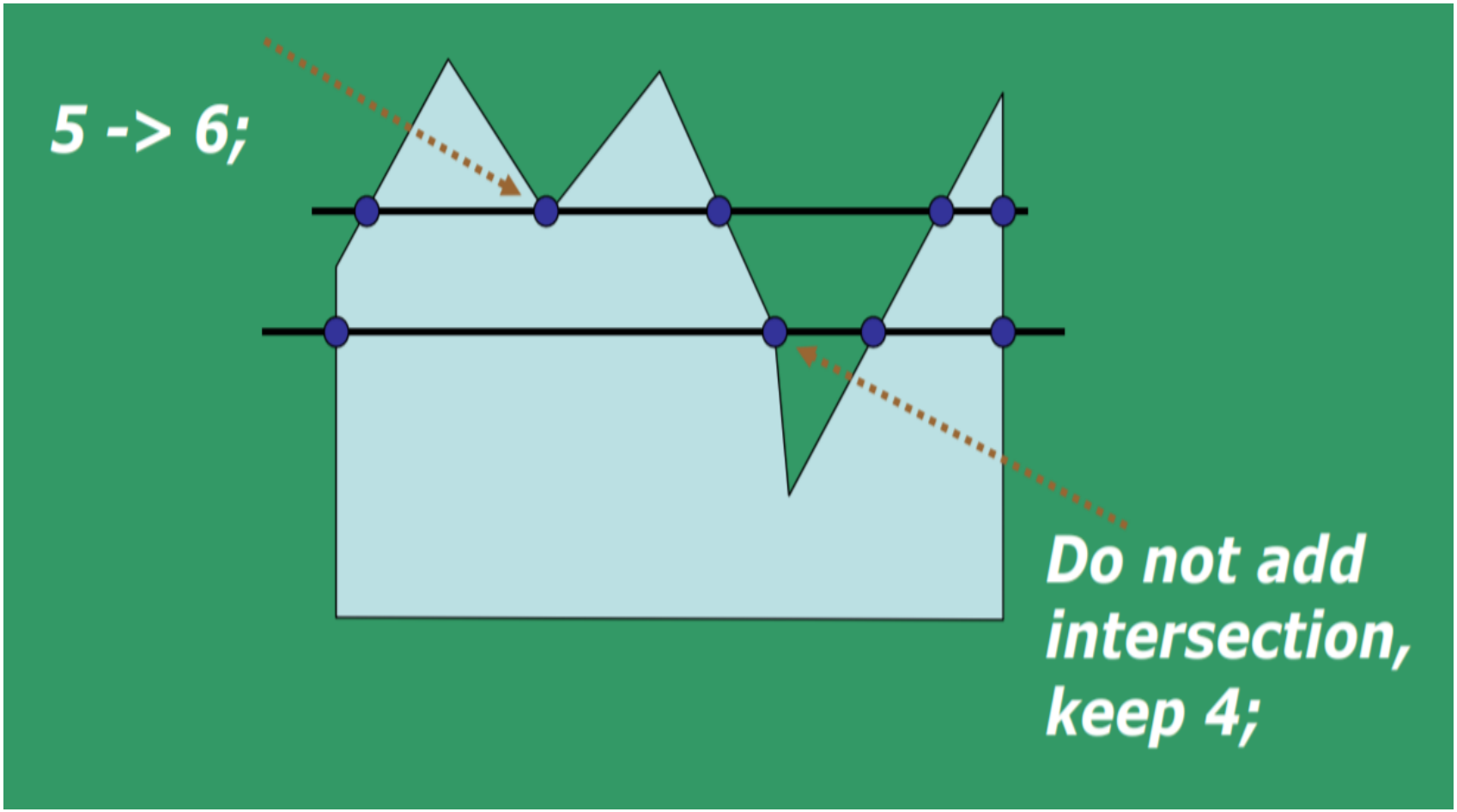
Example to contradict the generalization and fixing the issue (cont.)



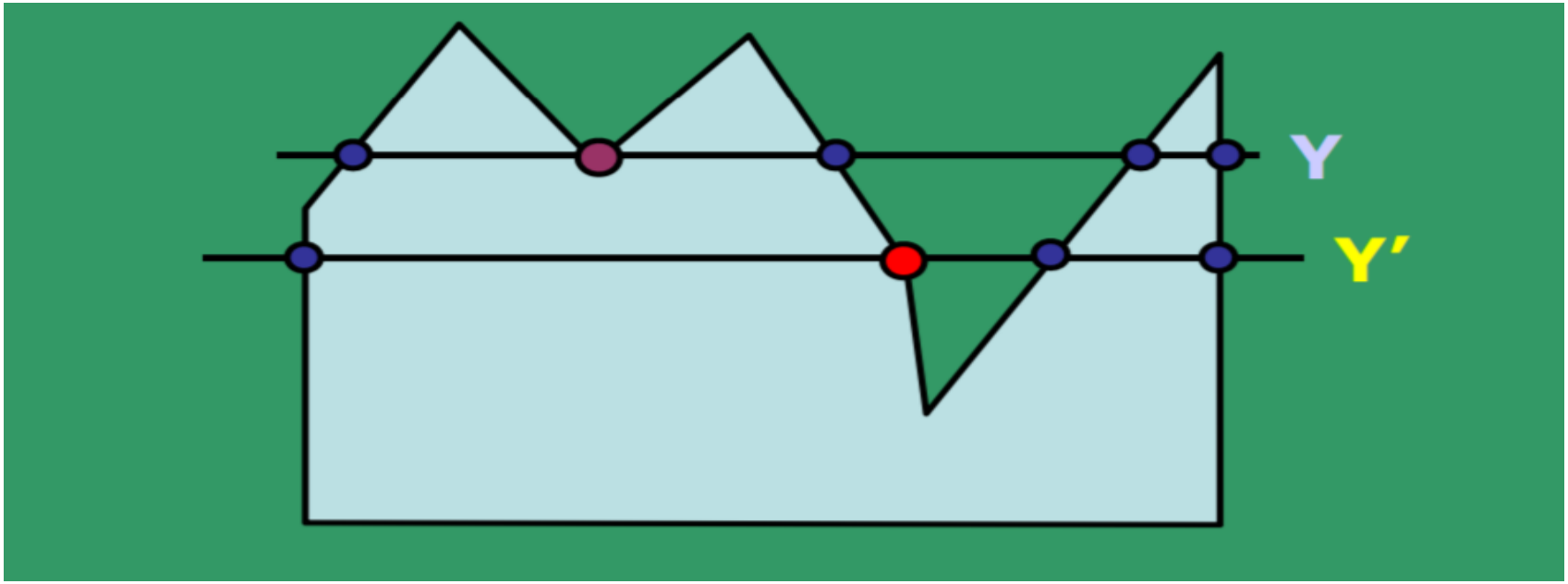
- Even segment is also inside the polygon
- What can be done for not affecting generalization:
- Do the following for not affecting the generalization:
  - When a scanline goes through a vertex, add one more intersection point at vertex
  - Three intersection are there in the scanline, call them p_1, p_2, p_3
  - Add one more p_2, and make the list of intersection points as p_1, p_2, p_2, p_3
  - Now three segments are formed$(p_1, p_2, (p_2, p_2), (p_2, p_3))$
- Now does the generalization stay true

Example to contradict the generalization and fixing the issue (cont.)

Counter Example to the Generalization

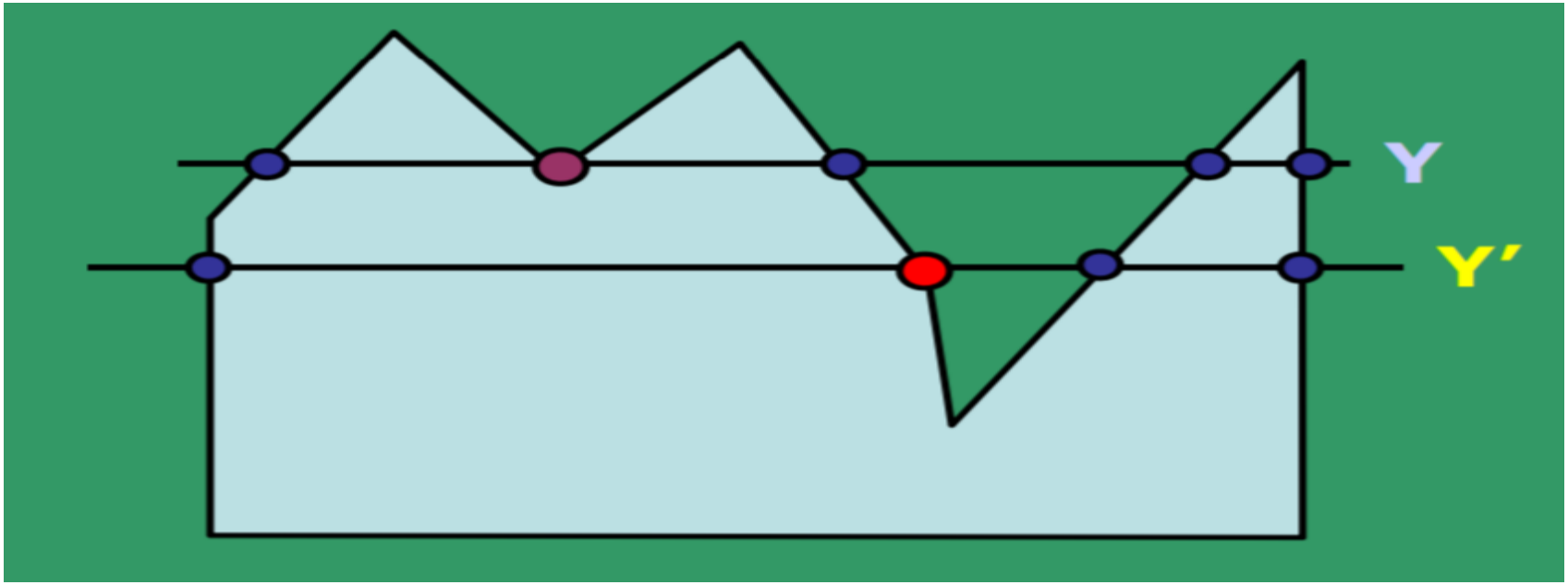


What is the difference between the intersection of the scanlines Y and Y', with the vertices?



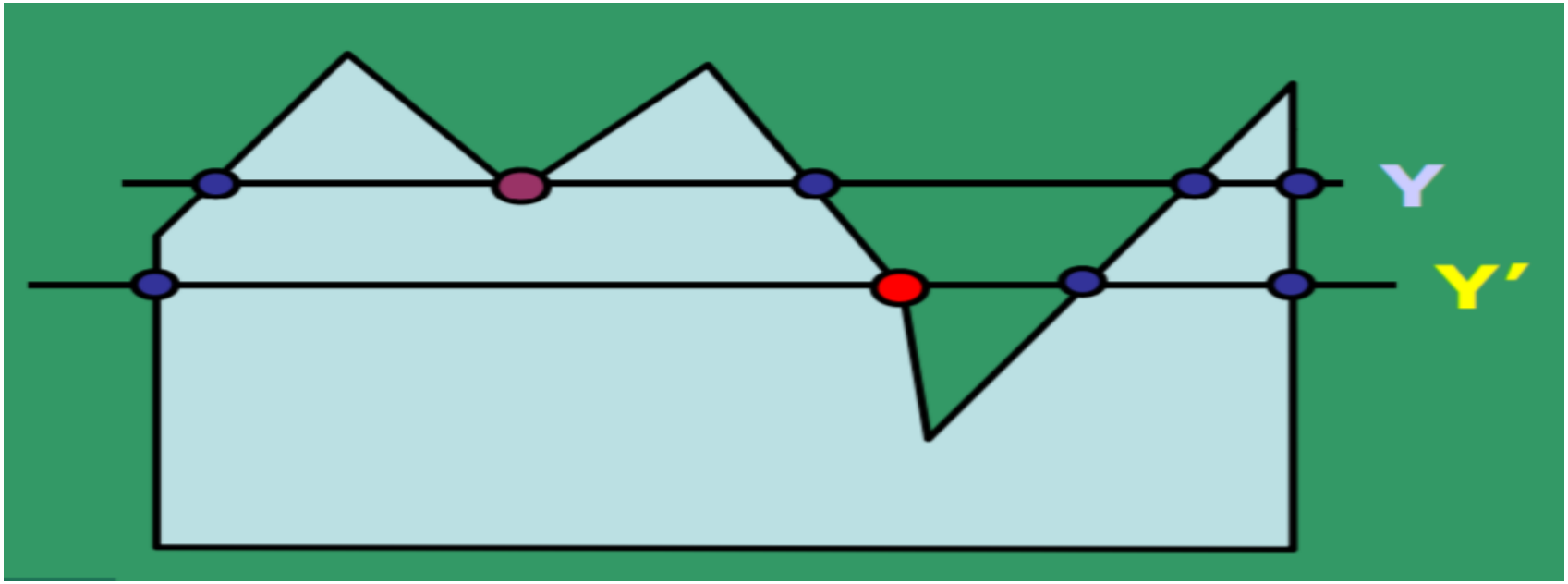
- The vertex (second intersection point), say q, on y needs to be repeated, but the vertex(second intersection point), say p, in line Y' is not to be repeated
- **Observation:** Both edges incident at q are on the same side, but the edges incident at p are on different side of the scan line
- Hence to keep the generalisation true, repeat the vertex for which incident edges are on the same side

How to check if the incident edges are on the same side



- **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).

How to check if the incident edges are on the same side (cont.)



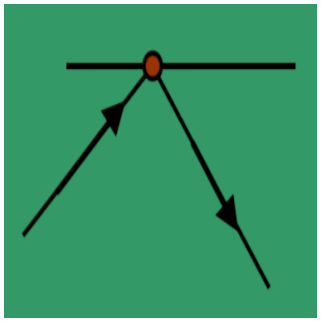
Check the condition:

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

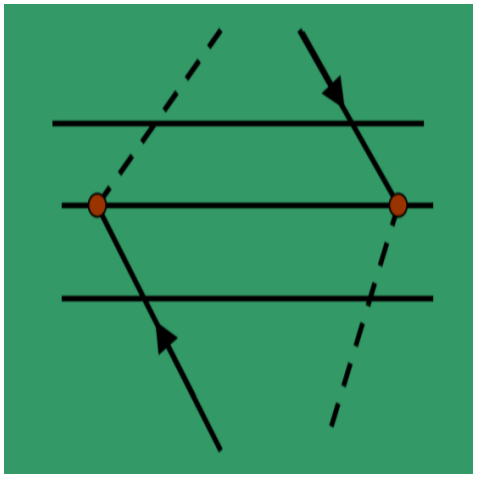
Else the shared vertex represents a *local maximum (or minimum)* on the polygon boundary.*Consider such vertex as two intersection points*

How to check if the incident edges are on the same side (cont.)

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



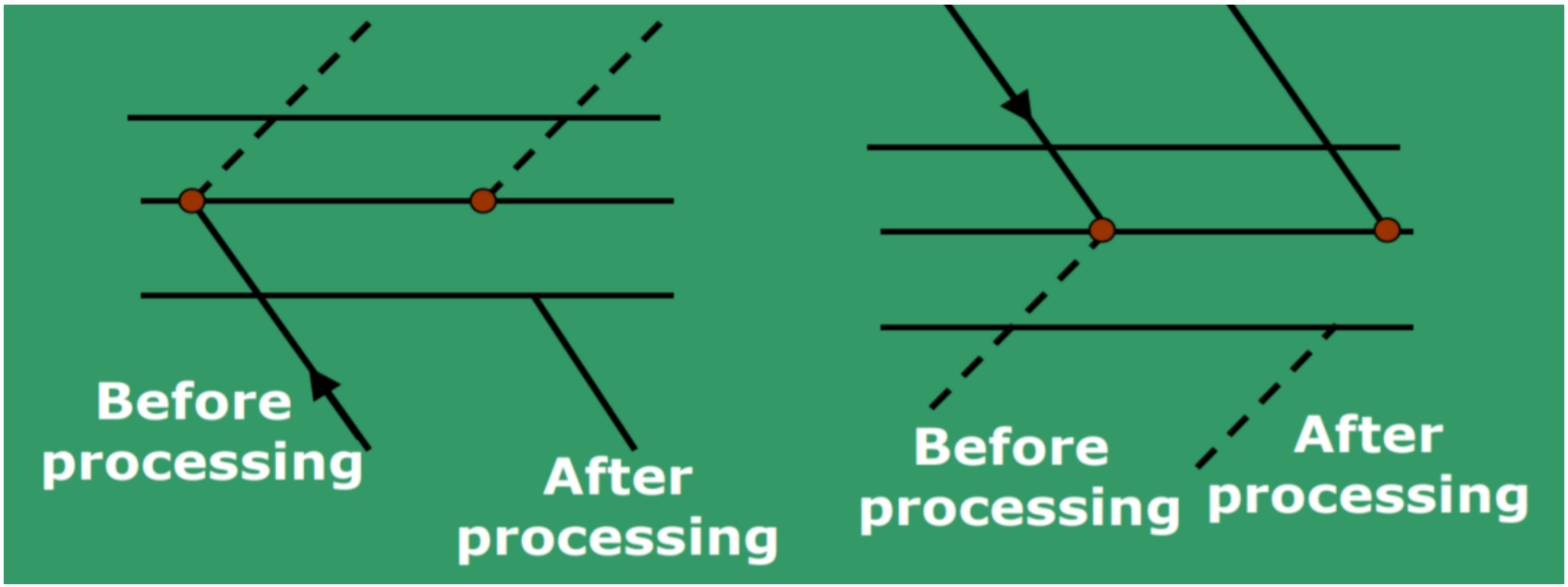
In the following cases, the intersecting vertex is not to be repeated



Implementation

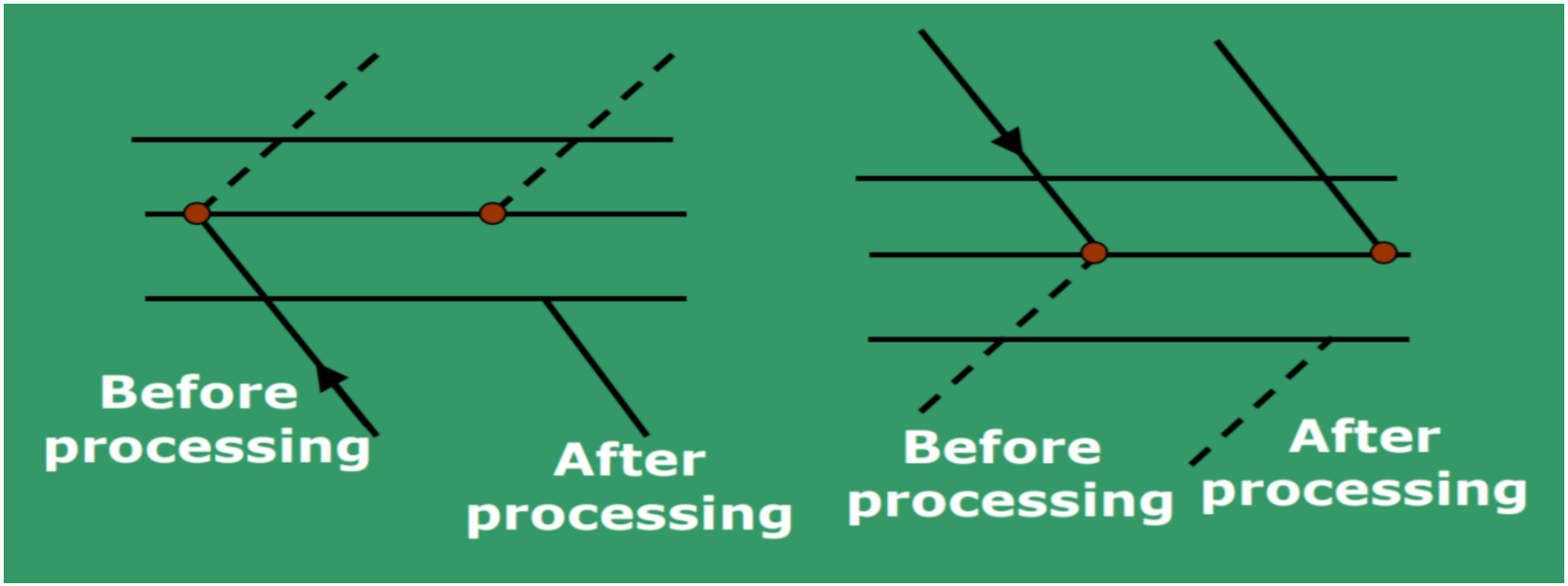
To implement the above:

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:



Implementation (cont.)

- 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}

- intersect scanline with each edge
- sort intersections by increasing X
- fill pairwise ($int0 -> int1, int2 -> int3$, ...)
- Update intersections for next scanline

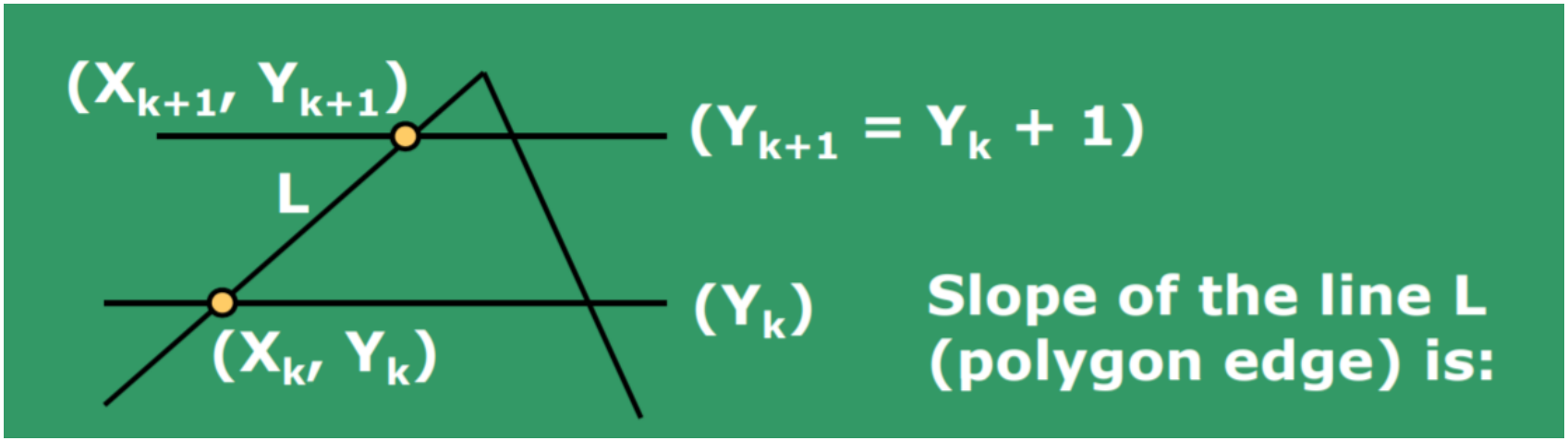
Scanline PolyFill Algorithm(revisited, in brief) (cont.)

How to find intersection points in each scanline efficiently

Consider two important features:

- **scanline coherence:** Two successive scanlines are related, and also two successive points on a scanline are related
  - Mid point line drawing algorithm uses this property and constructs relationship between two successive points on a line segment
- **edge coherence:** Two successive points points on an edge are related.
  ie
  - **Implication:** Any edge intersected by scanline "i" is also intersected by scanline "i+1" very often.
  - How to exploit the edge coherence?
  - Suppose (x_k, y_k) is the intersection of scanline k and an edge e,
  - Then we can find the intersection of next scanline $(k+1)$ with e using (x_k, y_k)

Scanline PolyFill Algorithm(revisited, in brief) (cont.)



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

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:

$Y_{k+1} = Y_k + 1$

Scanline PolyFill Algorithm(revisited, in brief) (cont.)

$X_{k+1} = Floor(X_k + 1/m)$, where $m = dy/dx = (Sign)\Delta Y/\Delta X$, $\Delta X, \Delta Y \geq 0$

How to compute (x_{k+1}, y_{k+1}) from (x_k, y_k) using integer arithmetic?

- After finding (x_k, y_k), to find (x_{k+1}, y_{k+1}), $y_{k+1} = y_k + 1$; $x_{k+1} = x_k + 1/m = x_k + (sign)\Delta X/\Delta Y$
- Assume $\Delta X = min(\Delta X, \Delta Y)$ and also $\Delta X < \Delta Y$
  $x_{k+2} = x_{k+1} + 1/m = x_k + 2/m = x_k + (sign)2\Delta X/\Delta Y$
  ... $x_{k+t} = x_{k+t-1} + 1/m = x_{k-t-2} + 2/m = x_k + t/m =$
  $x_k + (sign)t\Delta X/\Delta Y$ such that $t\Delta X/\Delta Y$ ¿=1, but $(t-1)\Delta X/\Delta Y$ ¡1
  Hence $x_{k+t} = x_k + 1$ whereas $y_{k+t} = y_k + t$
- to calculate the t value, we use a count C (to hold the numerator) which is initialized with ΔC =Min(ΔX, ΔY), and incremented as $C = \Delta C$

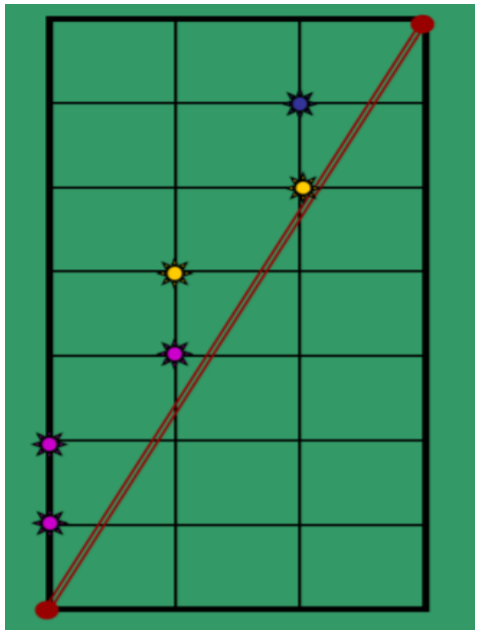
Scanline PolyFill Algorithm(revisited, in brief) (cont.)

- The incrementation of C will be stoped when c becomes equal or greater than Max()
- $x_{(k+i)} = x_k$ for all $1 \leq i \leq t-1$, and $x_{(k+t)} = x_k + 1$

Scanline PolyFill Algorithm(revisited, in brief) (cont.)

How to compute (x_{k+1}, y_{k+1}) from (x_k, y_k) using integer arithmetic?

Take an example: End points of the line segment: $(1, 1)$ and $(3, 8)$
m $= \Delta Y / \Delta X = 7/3$. Set Counter, C $= 0$
and counter-increment, $\Delta C =$ Min(ΔX, ΔY) $= 3$;
For the next three scan lines, successive values of C are : 3,6,9
Thus only at 3rd scanline C >= ΔY.



Then X_k is incremented by 1 only at 3rd, scanline and set as:
C = C – ΔY = 9 – 7 = 2.

Scanline PolyFill Algorithm(revisited, in brief) (cont.)

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

SET (Sorted Edge table): Contains all information necessary to process the scanlines efficiently

- For each non-horizontal edge in the polygon, define EdgNode with (Y_{max}, X_{min}, Sign, Δ X ΔY)
- Using Bucket Sort with m buckets, Sort the EdgeNodes based on Y_{min} of the edge (m is the number of rows(scan lines) of the image to be displayed
- With in each bucket, EdgeNodes are sorted by X coordinate

Data Structure Used (cont.)

Edge structure
(sample record for each scanline):

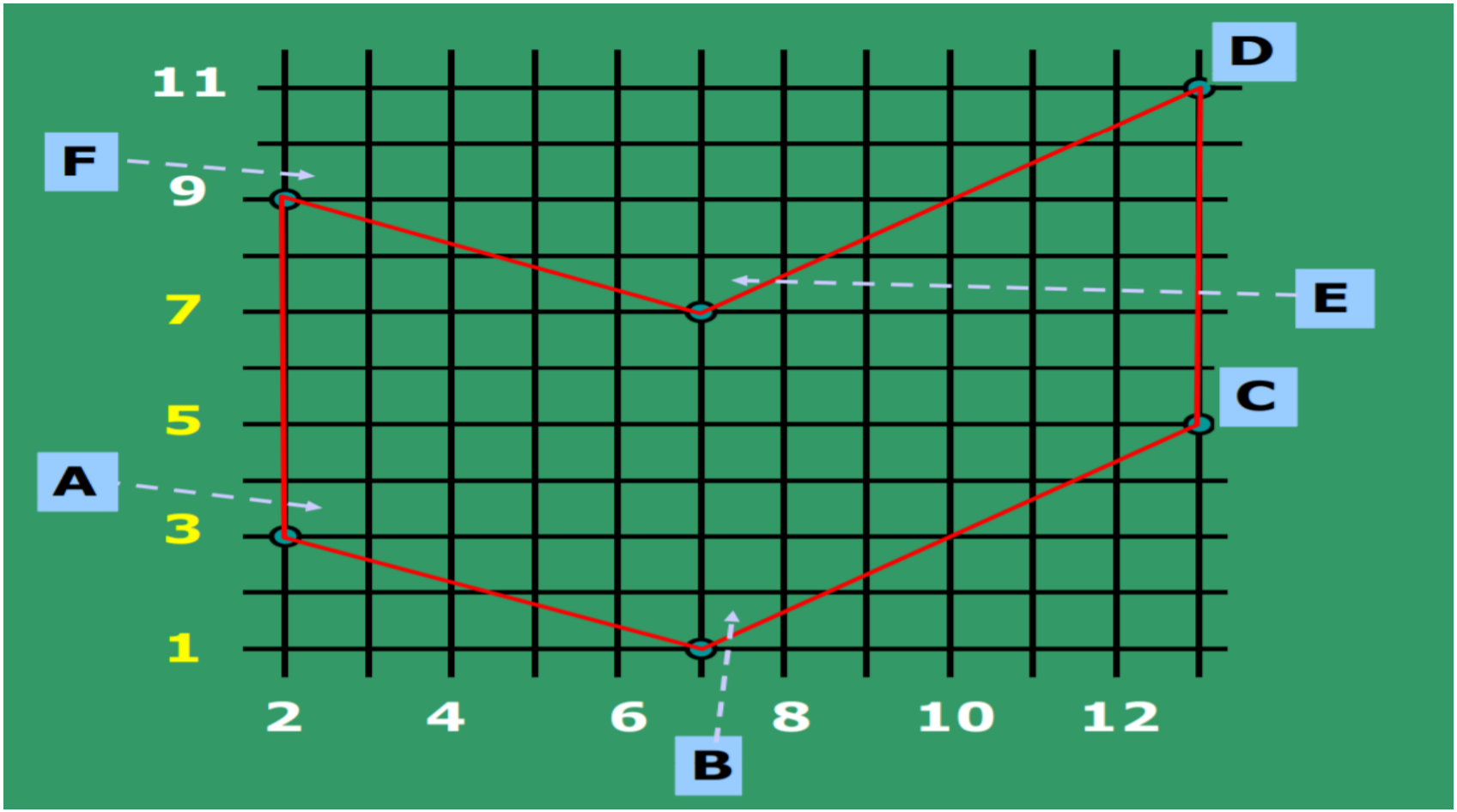
(Y_{max}, X_{min}, Sign, $\Delta X/\Delta Y$, pointer to next edge)

Construction of AEL (Active edge List):

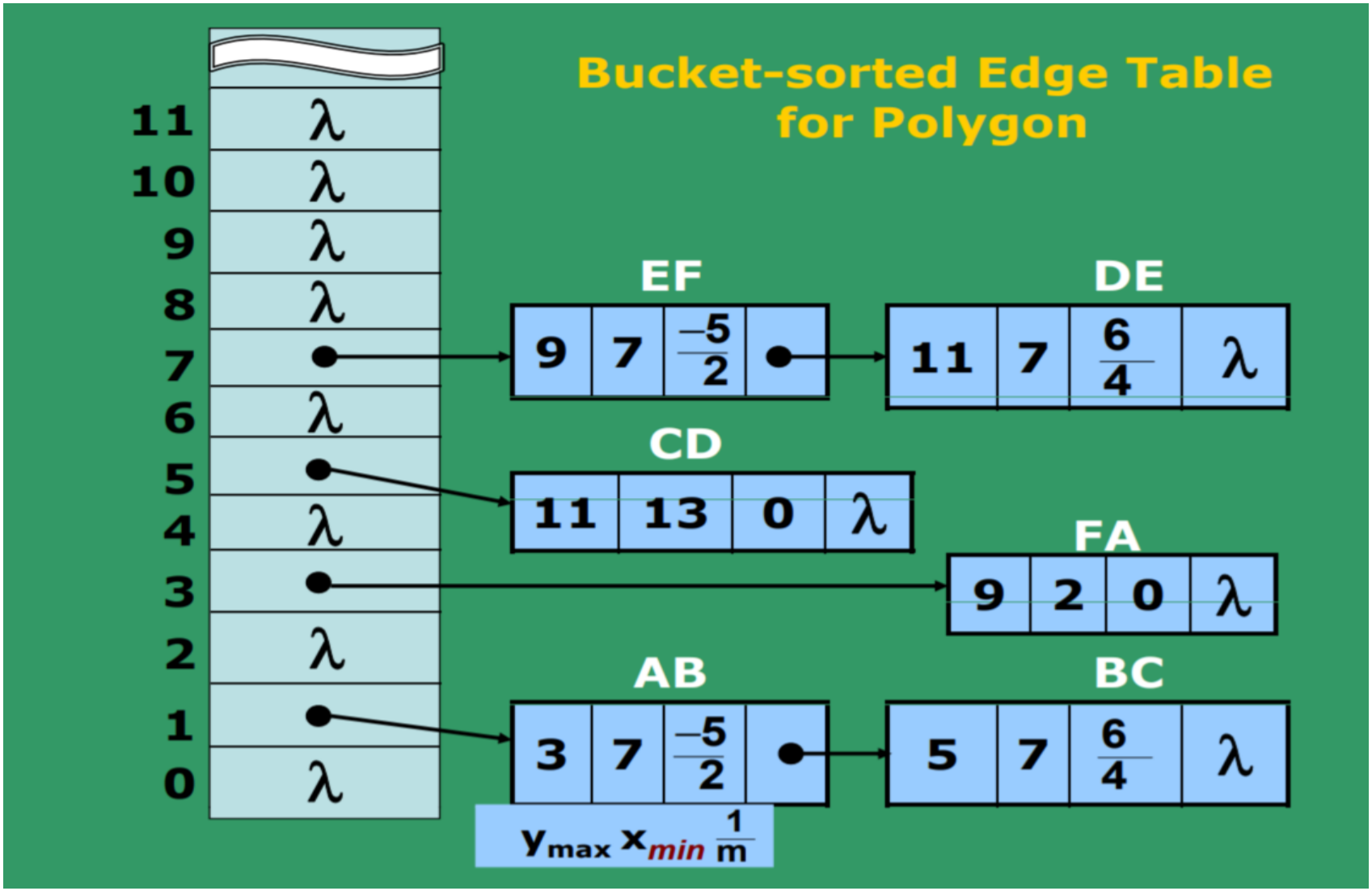
- an edge e is called as active, if the current scan line is between y_{min} and y_{max} of the edge e
- AEL is a List Data Structure which stores all EdgeNodes corresponding to all active edges for the current scan line

AEL is also called: *Active Edge Table (AET).*

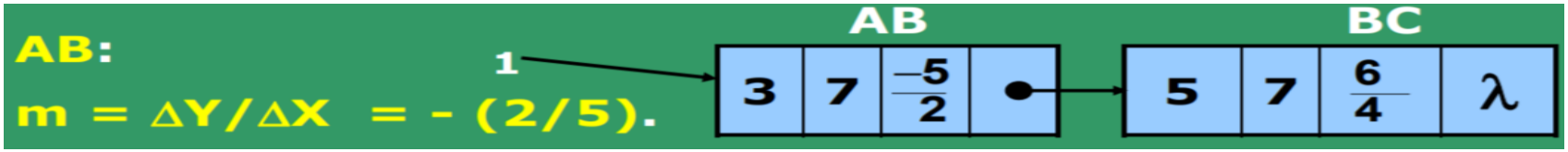
Data Structure Used (cont.)



Bucket-sorted Edge Table



Bucket-sorted Edge Table (cont.)

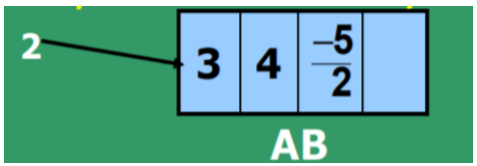


Set Counter C=0 ; and
counter-increment, $\Delta C = \min(\Delta X, \Delta Y) = 2(= \Delta Y)$;
Update for AB (-ve m), when $Y_K = 2$; $Y = 1$:

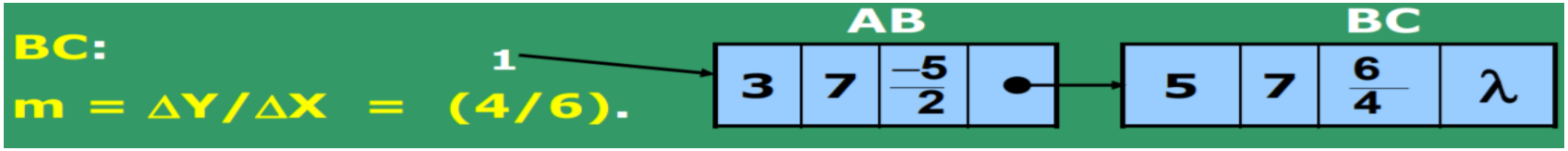
For the next three left (-ve) vertical (Y) scan lines,
successive values of C are : 2, 4, 6; X = 7 - 3 = 4;

Thus only at 3rd iteration: C >= ΔX=5.

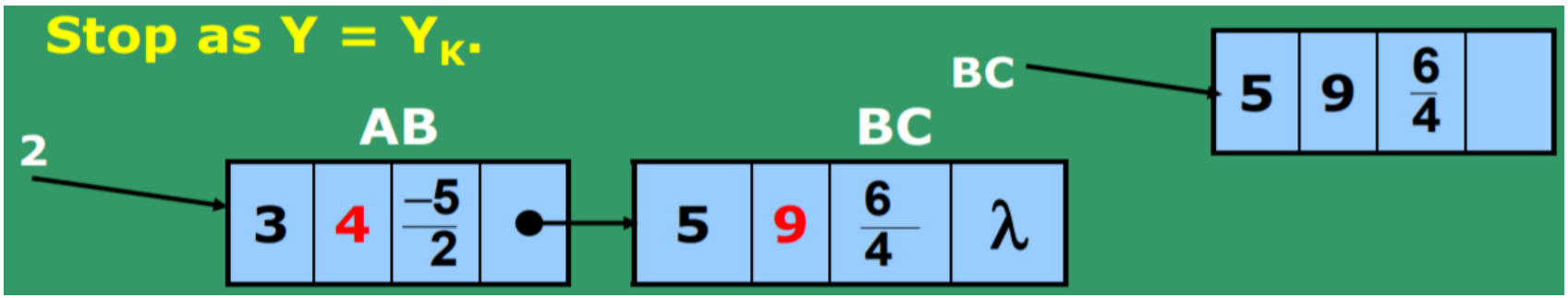
As Y needs to be incremented C >= ΔX, Y is incremented by 1 only at 3rd vertical scanline
and set: $C = C - \Delta X = 6 - 5 = 1$; $Y = 1 + 1 = 2$;
Stop as $Y = Y_k$.



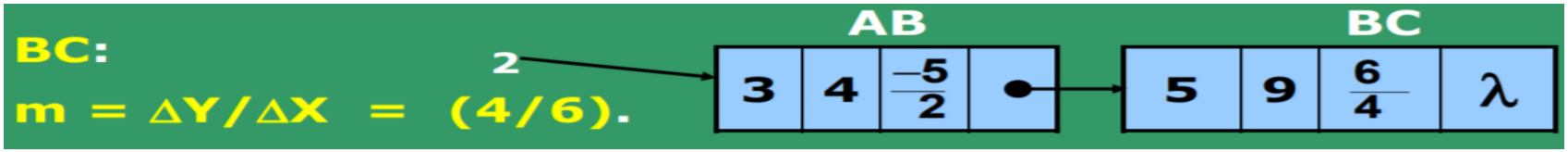
Bucket-sorted Edge Table (cont.)



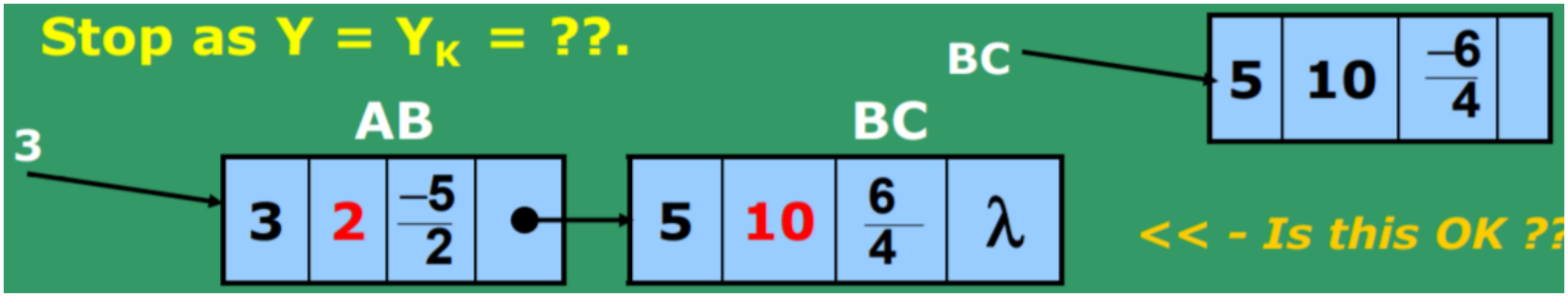
Set Counter C=0 ; and counter-increment, $\Delta C = \min(\Delta X, \Delta Y) = 4(= \Delta Y)$;
Update for BC (+ve m), when $Y_K = 2$; $Y = 1$:
For the next two right vertical (Y) scan lines,
successive values of C are : 4, 8; $X = 7 + 2 = 9$; Thus only at 2nd iteration: $C >= \Delta X$.
Then, Y is incremented by 1 only at 3rd vertical scanline on right and set: $C = C - \Delta X = 8 - 6 = 2$; $Y = 1 + 1 = 2$;



Bucket-sorted Edge Table (cont.)

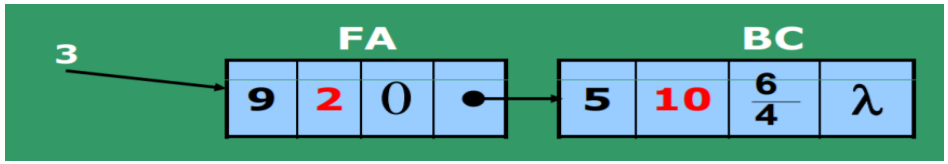


Counter(from earlier iteration) C=2 ; and
counter-increment, $\Delta C = \min(\Delta X, \Delta Y) = 4 (= \Delta Y)$;
Update for BC (+ve m), when $Y_K = 3$; $Y = 2$:
For the next two right vertical (Y) scan line,
successive values of C is : 6; $X = 9 + 1 = 10$; Thus only at 1st iteration:
$C >= \Delta X$.
Then, Y is incremented by 1 only at 1st scanline
and set: $C <- C - \Delta X = 6 - 6 = 0$; $Y = 2 + 1 = 3$;

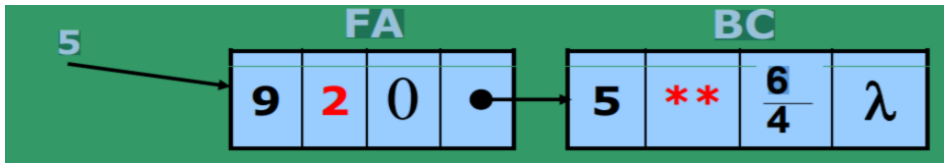


Bucket-sorted Edge Table (cont.)

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

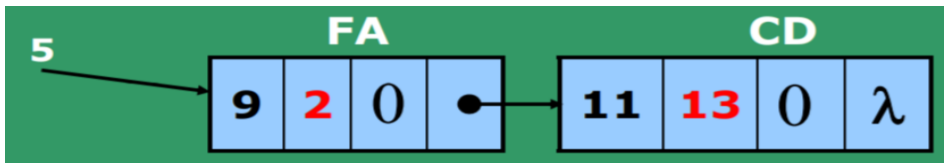


Home task: Complete the next two sets of iterations yourself, till you get :

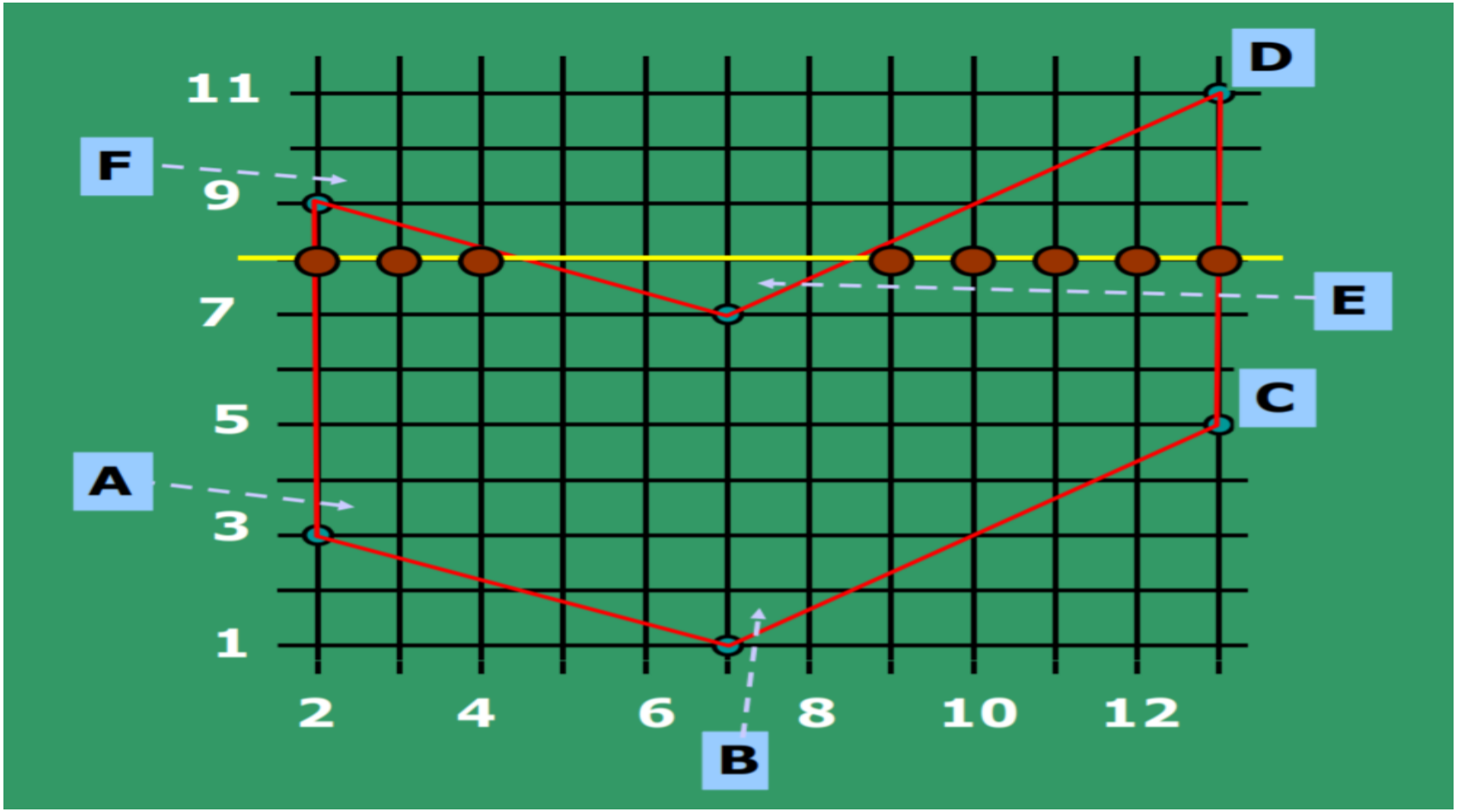


** $= 10 + 3 = 13$, why ??

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

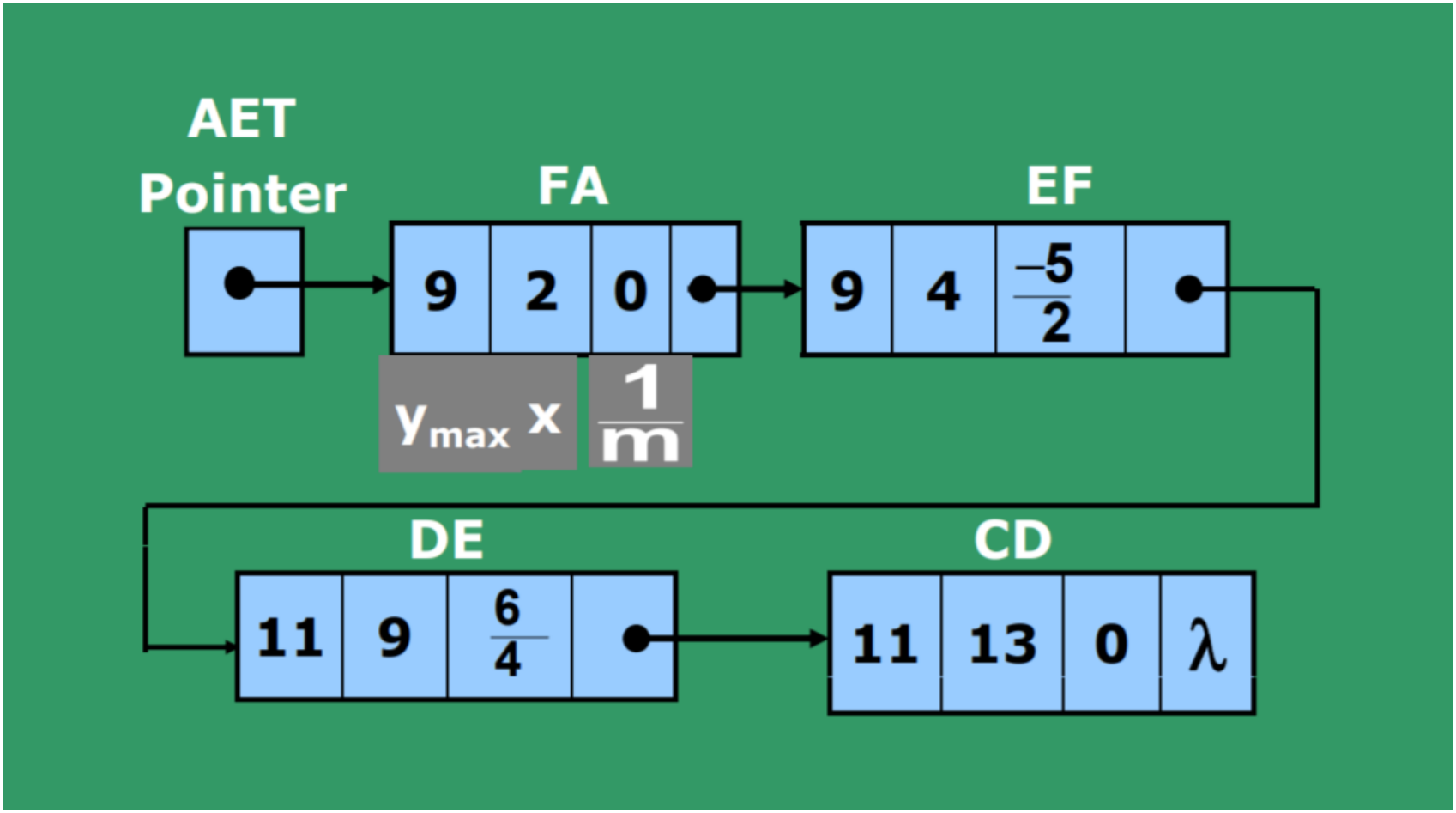


Bucket-sorted Edge Table (cont.)

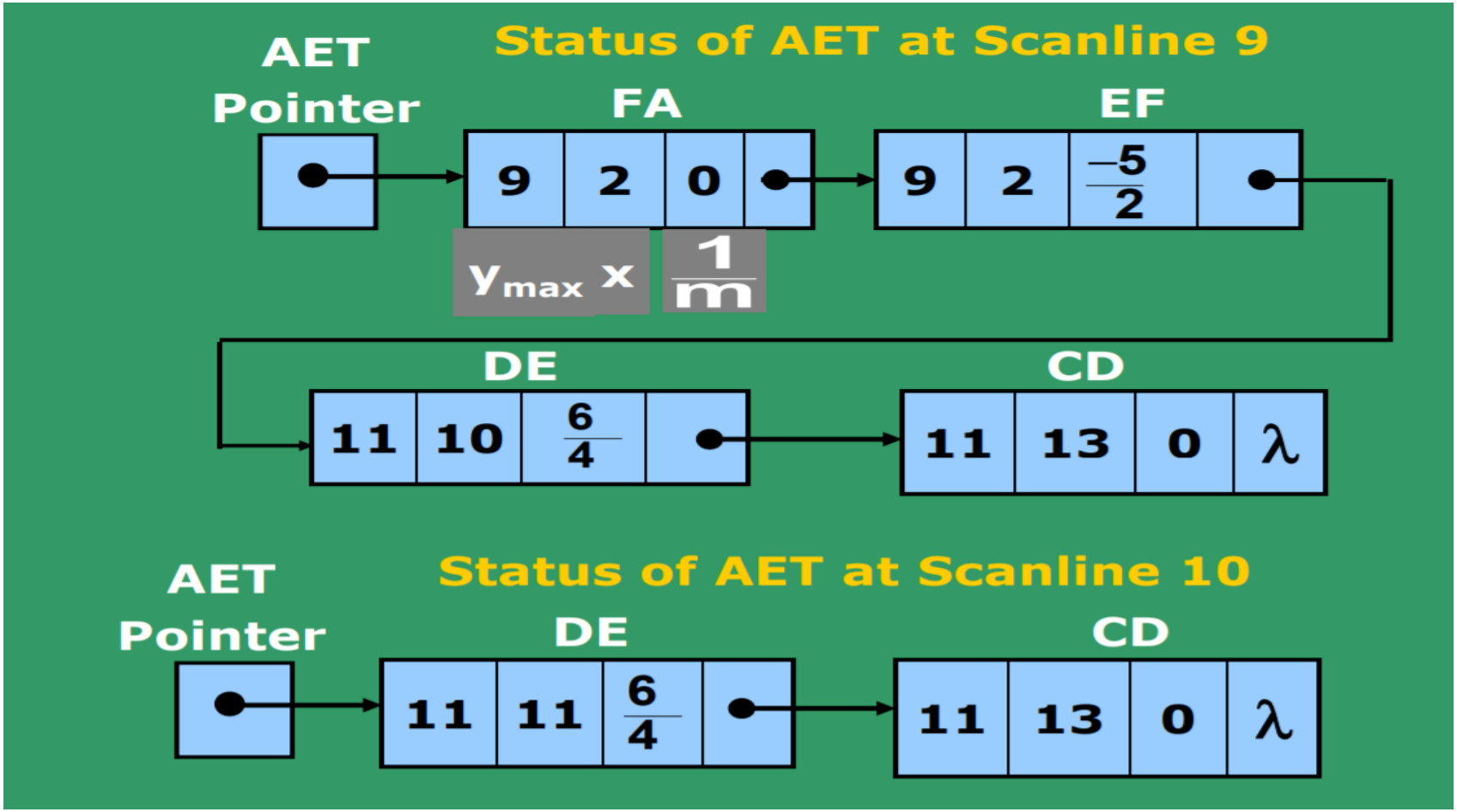


Status of AET

Status of AET at Scanline 8



Status of AET (cont.)



Processing Steps

- Set Y to smallest Y_{min} in SET entry(first non-empty bucket).
- Initialize AET to be empty
- Repeat until both AET and and SET are empty
  - Move SET bucket Y to to AET (the linked list at Y).
  - Sort AET on x coordinate (By merging previous AET with Bucket Y).
  - Fill pixels in scanline Y using pairs of Xcoordns. from AET
  - Increment scanline Y by 1.
  - Remove from AET those entries for which Y = Y_{max}
  - For each non-vertical edge in AET, update X for new Y .

END-LOOP

The algorithm: scan-fill(polygon)

- Construct the Edge table (ET) with edge nodes(only non horizontal edges) involving Y_{max}, sign, X-coordinate of Y_{min}, ΔX and ΔY using bucket sort on Y_{min} and X-coordinate of Y_{min}
- Y_{min} = min (all Y in the SET) ; Y_{max}=max(all y in the SET)
- AET = null
- for Y=Y_{min} to Y_{max}
  - *Merge* ET[Y] into AET by X value
  - Scan convert line segment with endpoints (x, y) and (x', y) that are forming odd pairs in AET. Here y is the current scan line.
  - for each edge in AET
    - if Y = Y_{max} then
      remove the edge from AET
    - else
      edge.X = edge.X + dx/dy
  - sort AET by X value using merge sort
- end scan-fill

C code for x coordinate in the next scan line on the giv edge

```
#include<stdio.h>
#include<math.h>
int c=0, Delc=0;
int main()
{
   int x1,x2,y1,y2, sign, Delx, Dely;
   int  Delc, x,y, ymin, ymax;

   printf("Enter endpoints of a line");
   scanf("%d%d", &x1, &y1);
   scanf("%d%d", &x2, &y2);

   int dx=x2-x1;
   int dy=y2-y1;

 //Find sign of slope
   if(((dx>0) && (dy>0))||(dx<0)&&(dy<0))
    sign=1;
```

C code for x coordinate in the next scan line on the giv edge (cont.)

```
19    else
20      sign=-1;
21
22  Delx=abs(dx);
23  Dely=abs(dy);
24
25
26  // Find ymin and  x cordinate of ymin
27
28  if(y1<y2)
29  {
30     ymin = y1;
31     ymax=y2;
32     x=x1;
33  }
34  else
35  {
36     ymin=y2  ;
```

C code for x coordinate in the next scan line on the giv edge (cont.)

```
37      ymax=y1;
38      x=x2  ;
39    }
40
41
42  //Printing first point
43  printf("x=%d;y=%d",   x,  ymin  );
44
45  y=ymin;
46  while(y<ymax)
47  {
48  y=y+1; // next scan line
49  // find x coordinate of next scanline (y+1)
50  x= Find_Next_edge_point(x,Delx,Dely,sign);
51  printf("x=%d;y=%d",   x,  y  );
52  }
53     return(0);
54  }
```

C code for x coordinate in the next scan line on the giv edge (cont.)

```
55
56
57 //Finds the value of x in the scan line y
58 //on the edge with
59 // slope = sign(Del y/ Del x )
60
61 int Find_Next_edge_point(int x,int Delx,
62                          int Dely,int sign)
63 {
64
65
66  if(Delx < Dely) //x will grow slowly
67   {
68       Delc=Delx;
69       c=c+Delc;
70
71     if (c >= Dely)
72     {
```

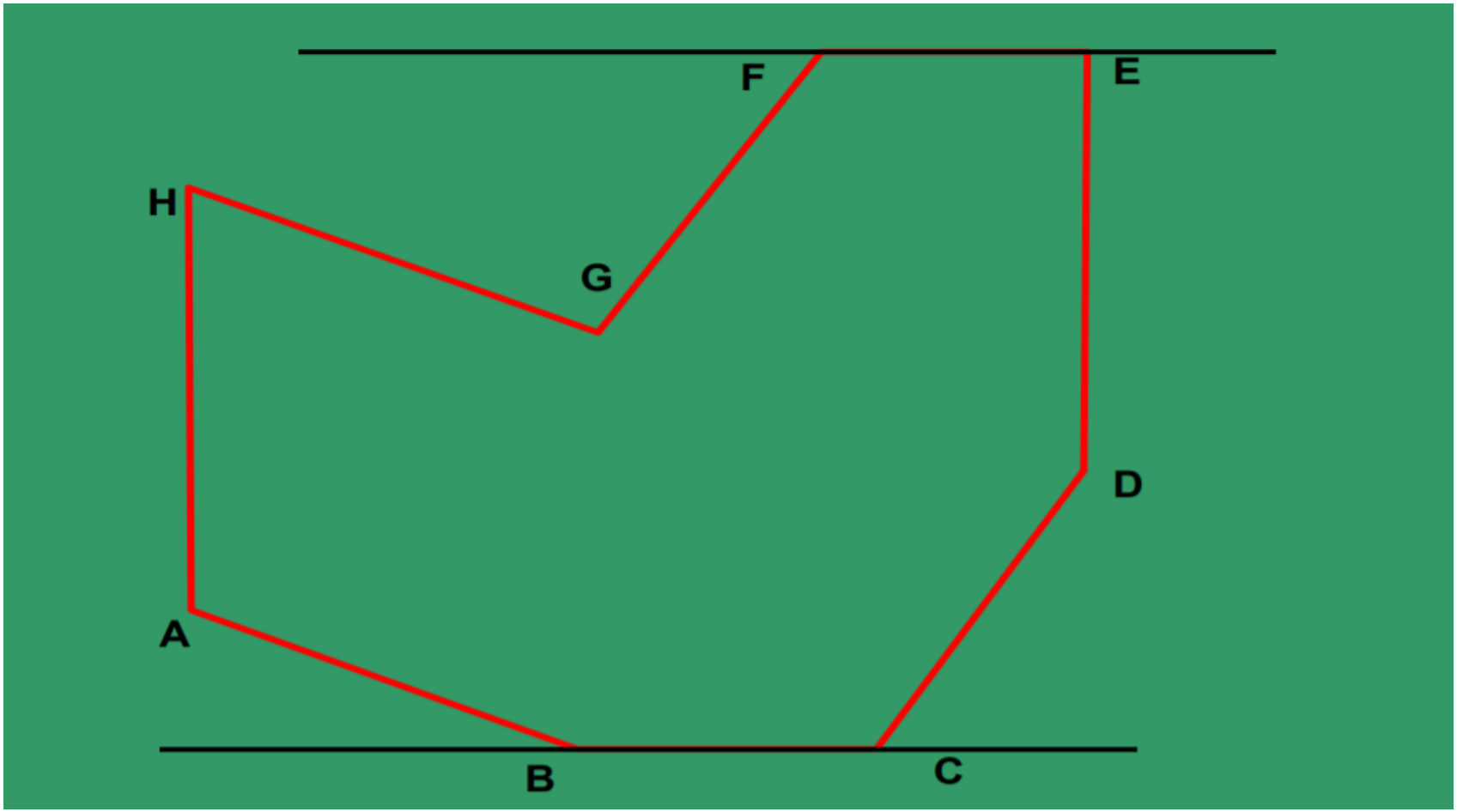
C code for x coordinate in the next scan line on the giv edge (cont.)

```
73          c=c-Dely;
74          return (x+ sign);
75       }
76       else
77         return   (x);
78     }
79
80   else   //x will grow fast
81     {
82          Delc = Dely;
83       while(c < Delx)
84       {
85          c=c+Delc;
86          x= x+ sign;
87
88       }
89         c=c-Delx;
90         return(x);
```

C code for x coordinate in the next scan line on the giv edge (cont.)

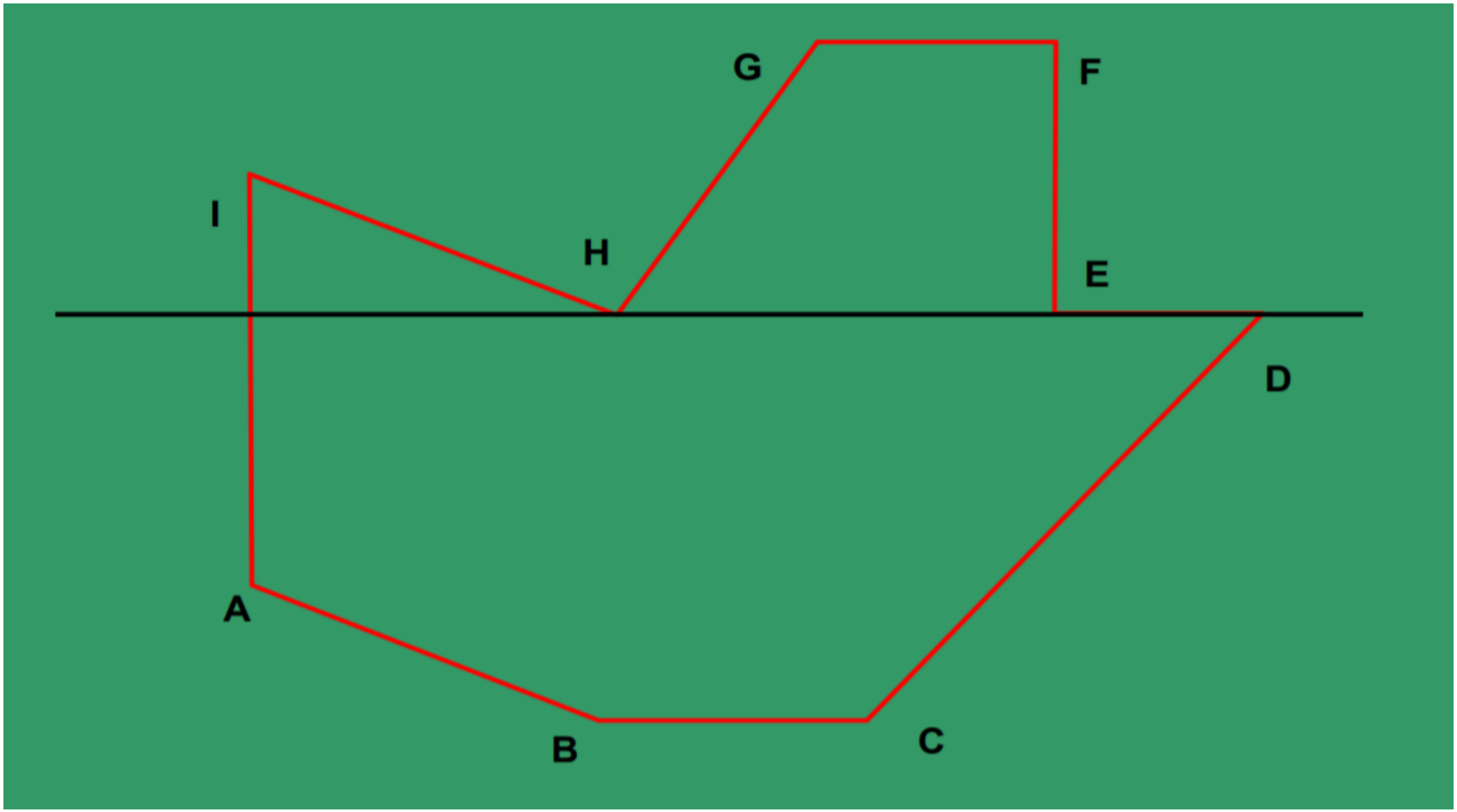
```
91      }
92
93  }//end of function
```

Horizontal Edges



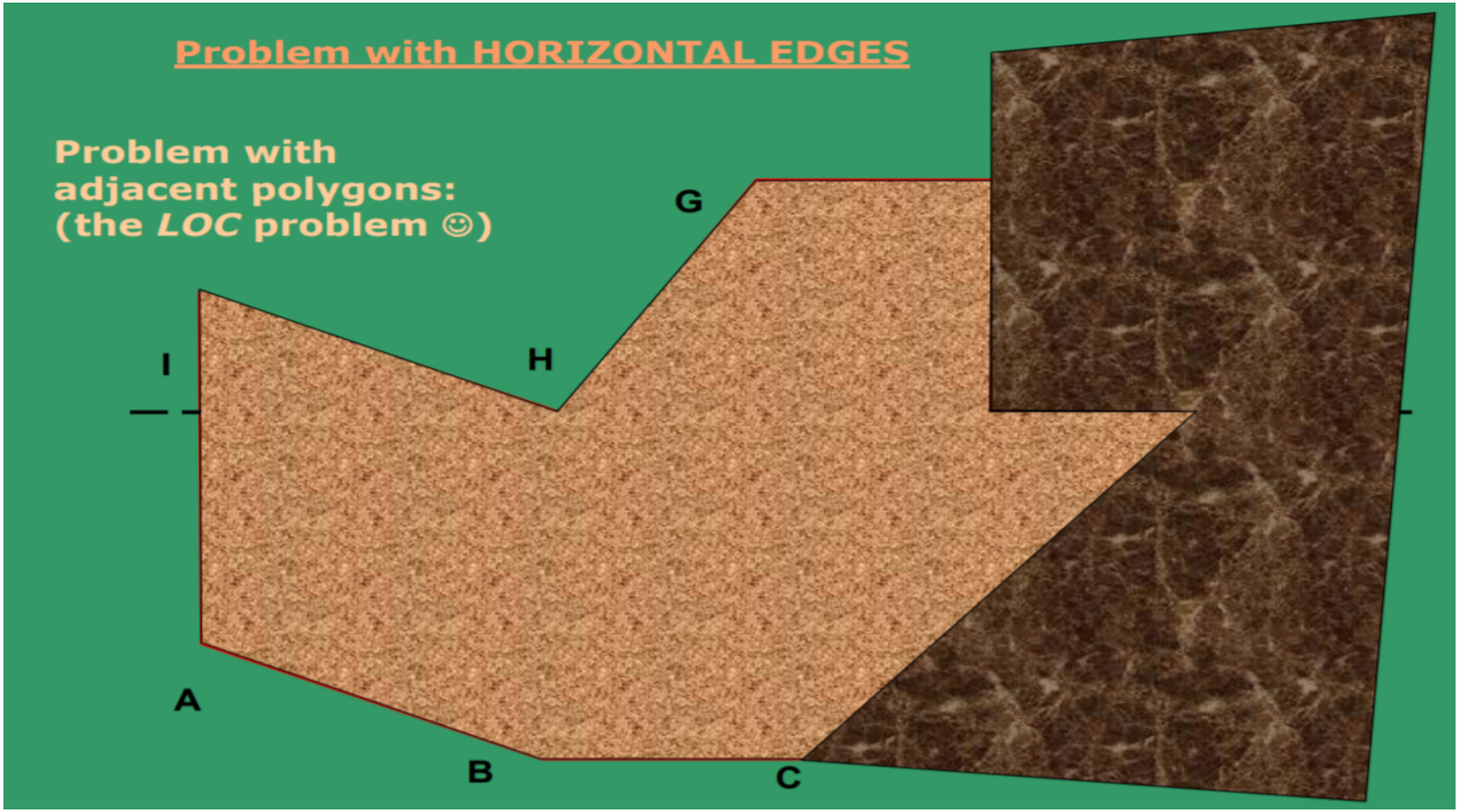
What about vertex processing on both sides of a horizontal edge?

Problem with Horizontal Edges



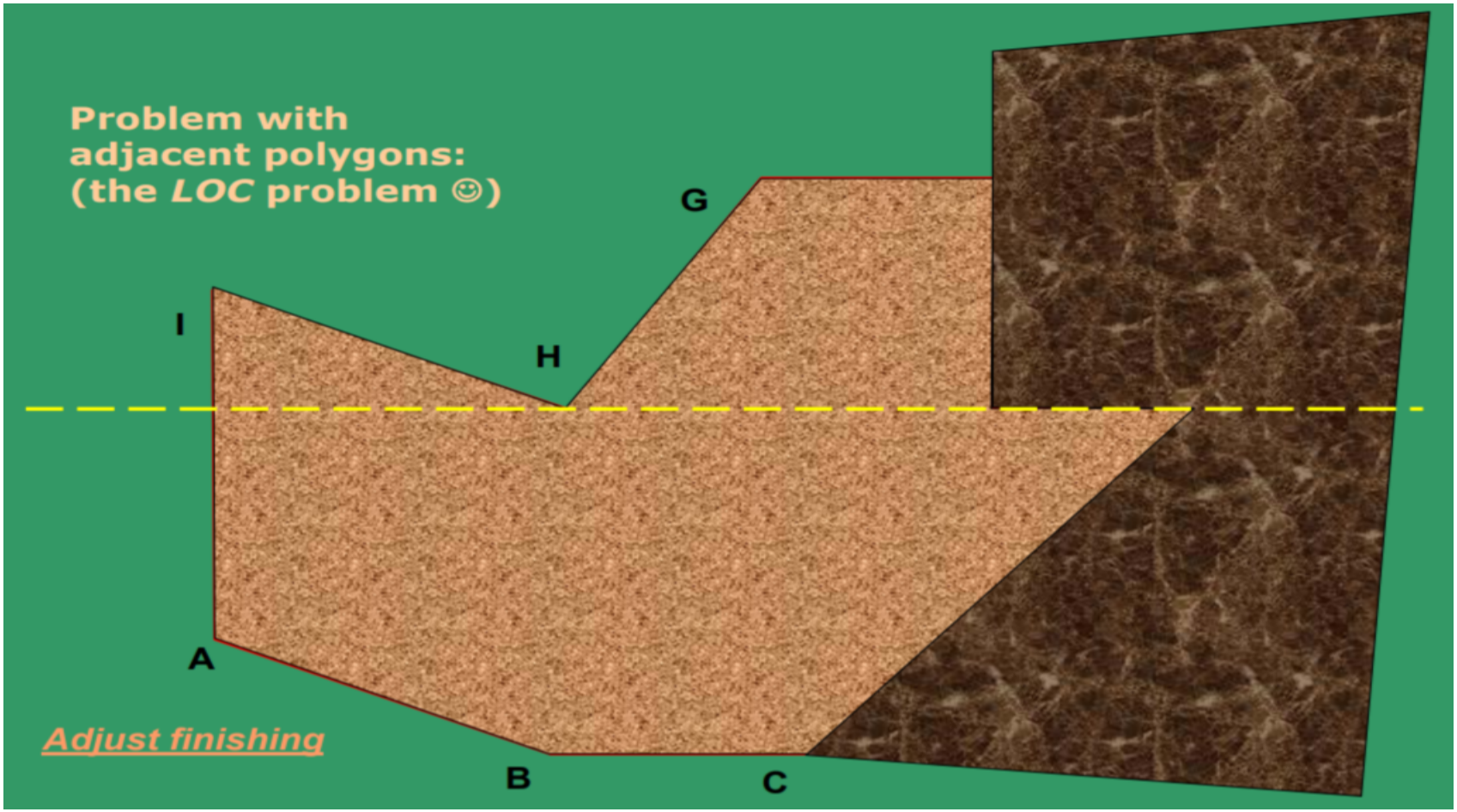
what do you do to fill till vertex D-odd number of intersections?

Problem with Horizontal Edges (cont.)

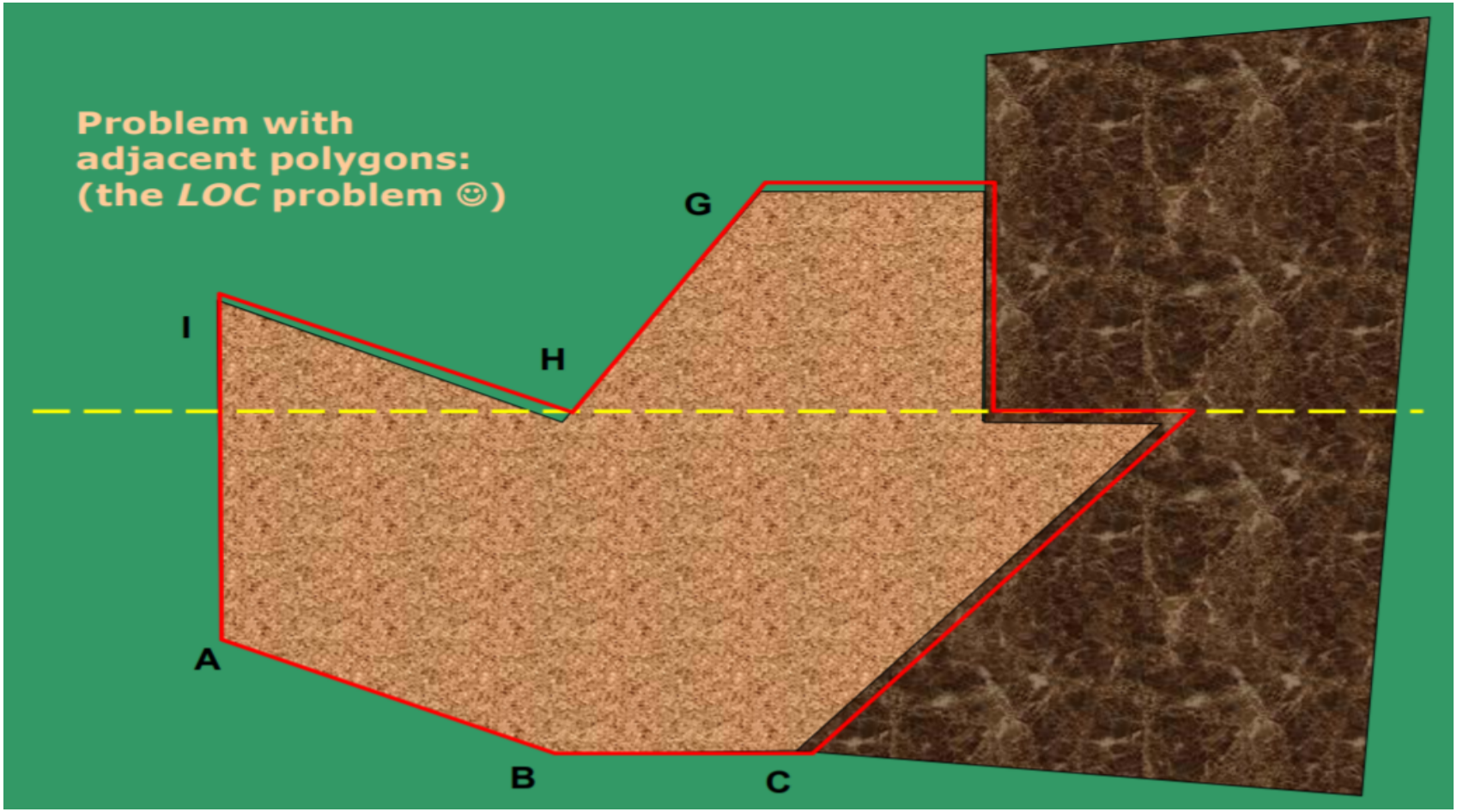


Think of the background surrounding polygon, producing the same problem at the edges.

Problem with Horizontal Edges (cont.)



Problem with Horizontal Edges (cont.)



Acknowledgements

- Some of the slides have been adopted from NPTEL and other internet sources. The due credits are acknowledged.

Thank You! :)