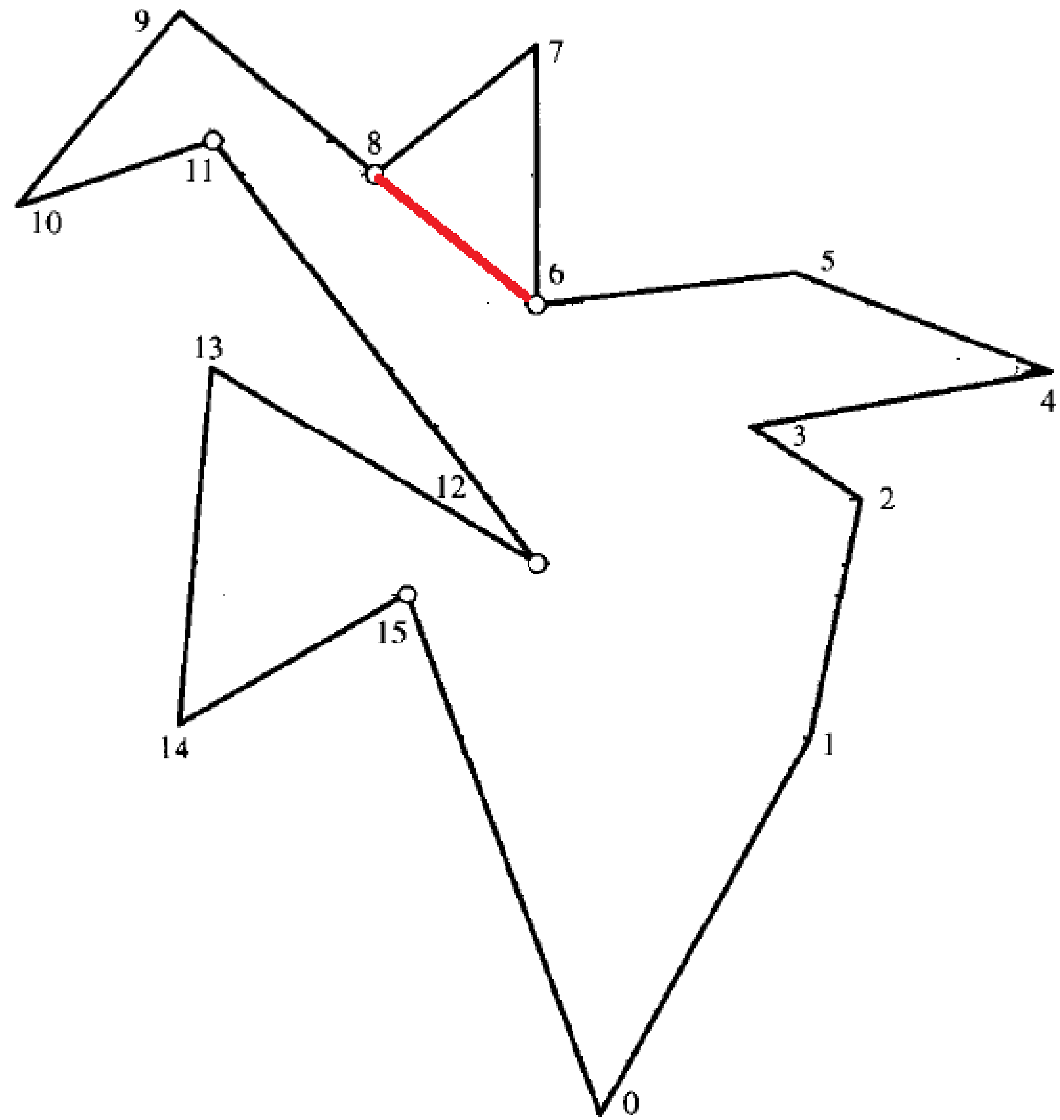
Algorithm to triangulate a non-monotone (normal) polygon

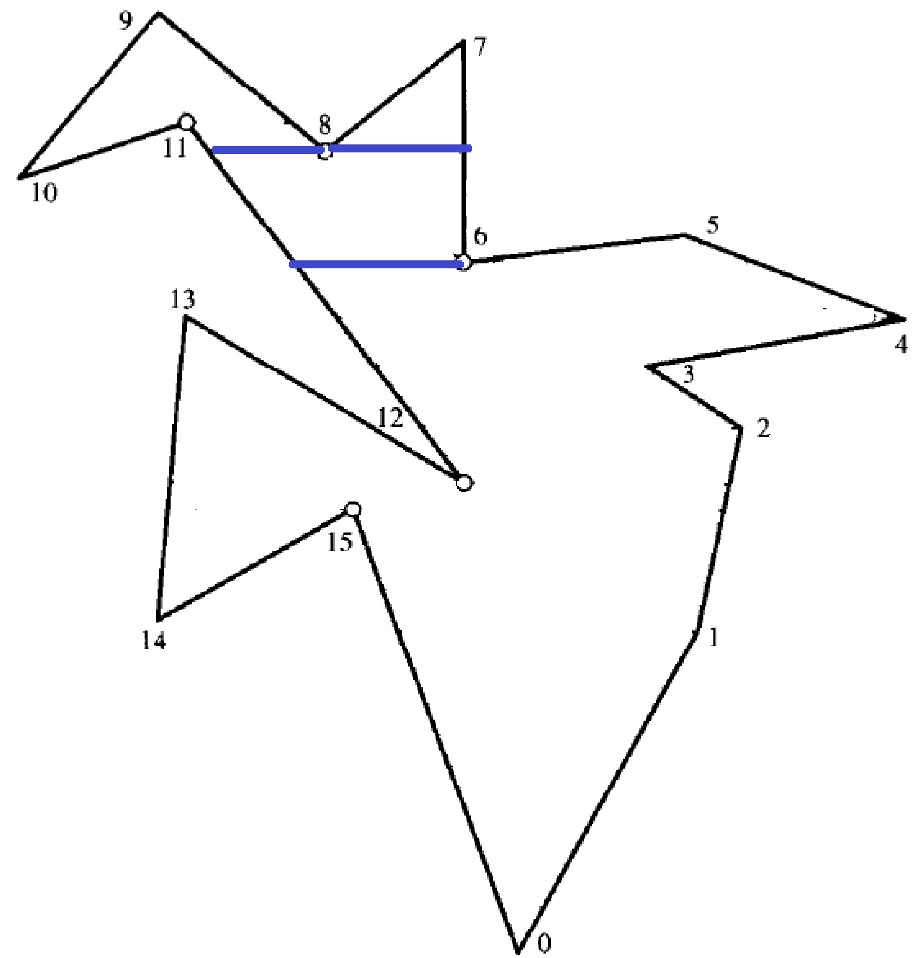
- Step 1: Partition a Polygon to monotone pieces
- Step 2 :Triangulate each monotone piece (can be done in linear time)
- If step 1 can be done efficiently (less than $O(n^2)$), then we can develop an efficient algorithm than the current $O(n^2)$ algorithm for triangulating a polygon
- We proceed focusing on a normal polygon



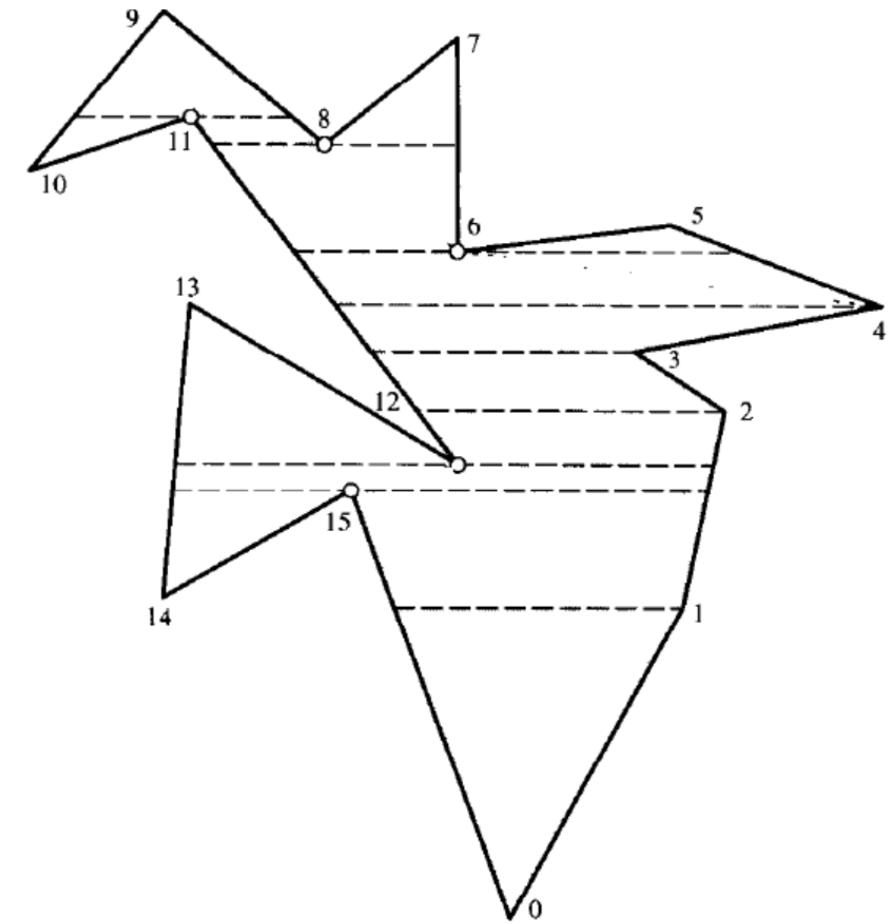
9
7
8
11
10
6
5
13
4
3
12
2
15
14
1
0

Restrict our choice of a vertex to connect to :

- If we can restrict the choice of a vertex



Horizontal trapezoidalization



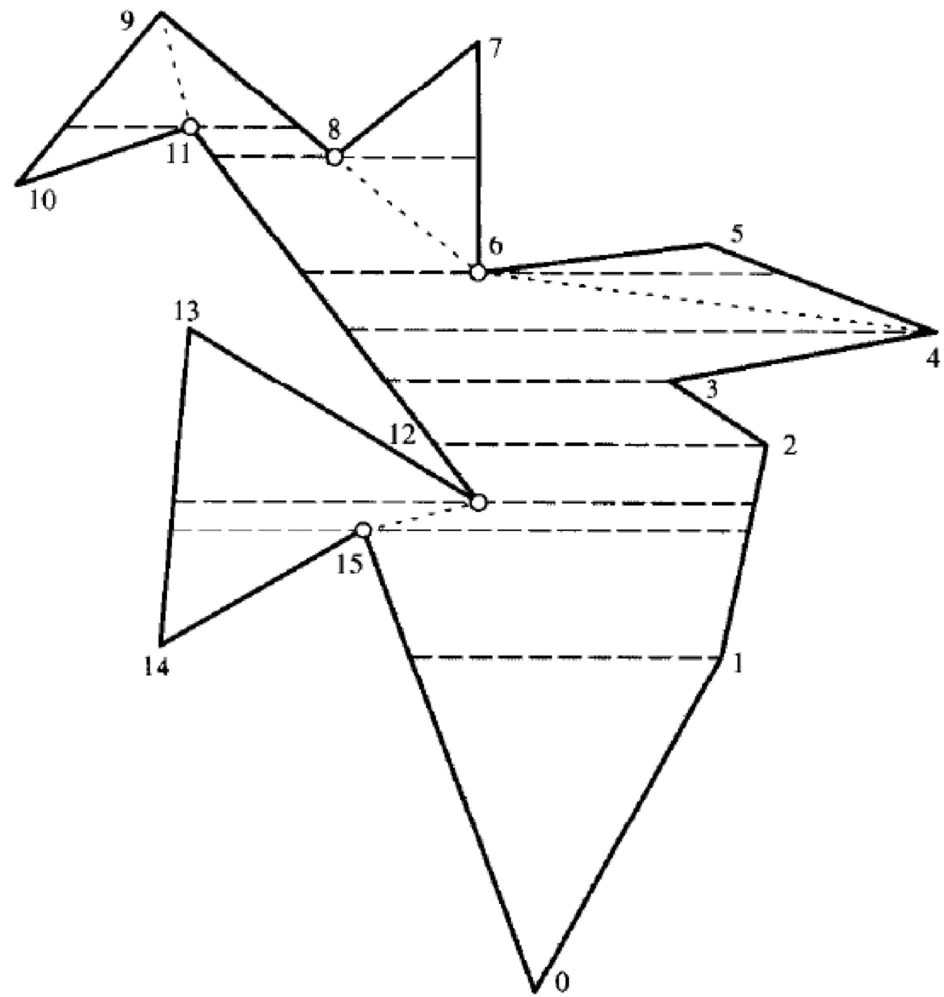
Removing the interior cusp

- Downward pointing cusps

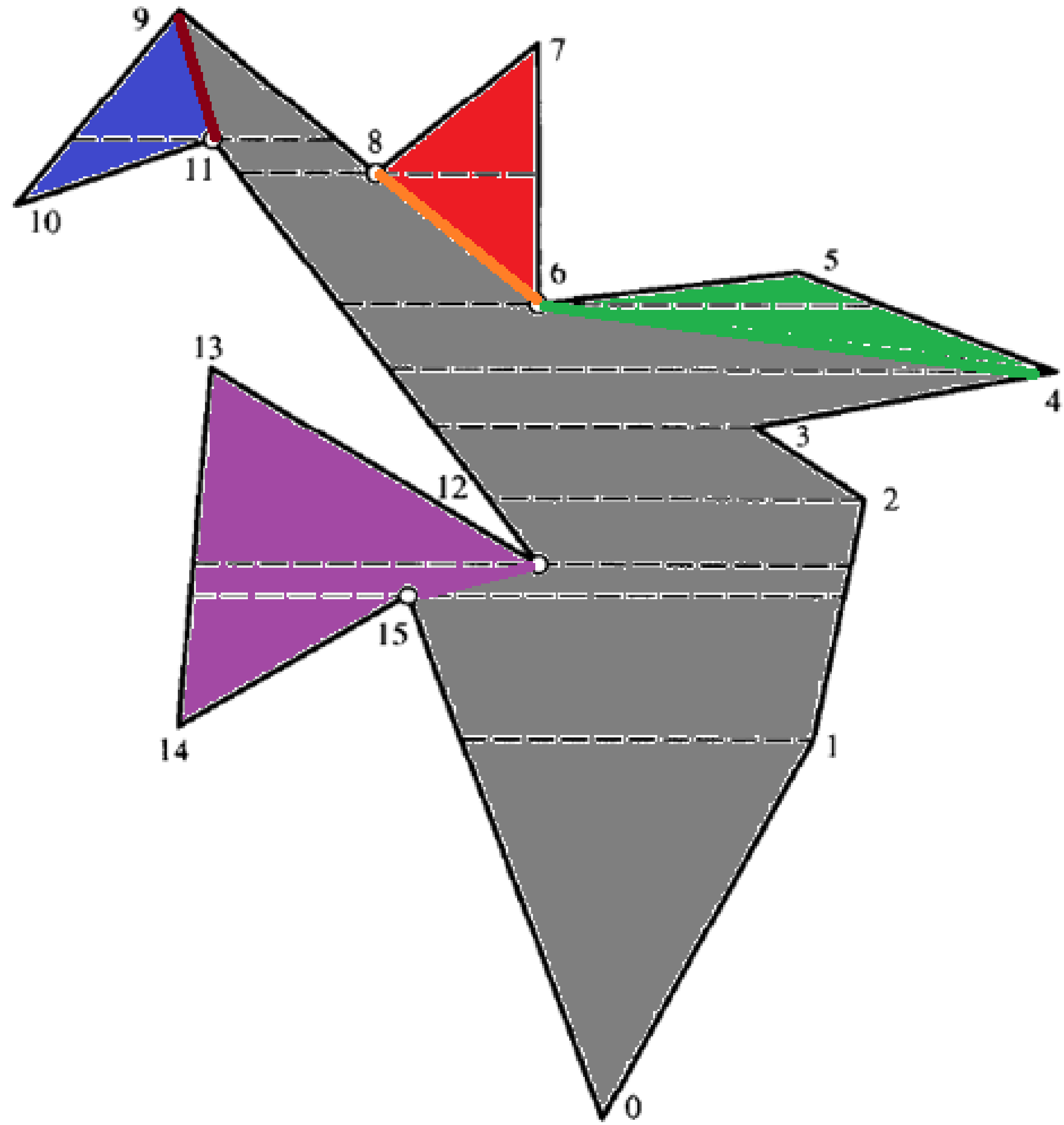
  Eg: 8,6,12

- Upward pointing cusp

  Eg: 11, 15

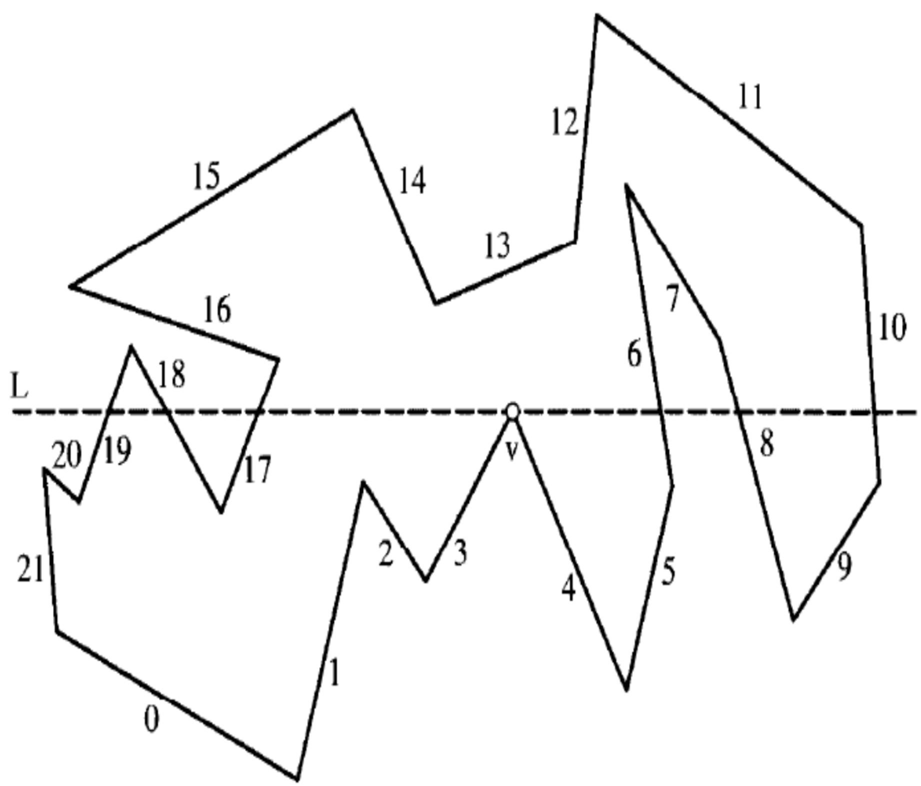


Monotone sub polygons



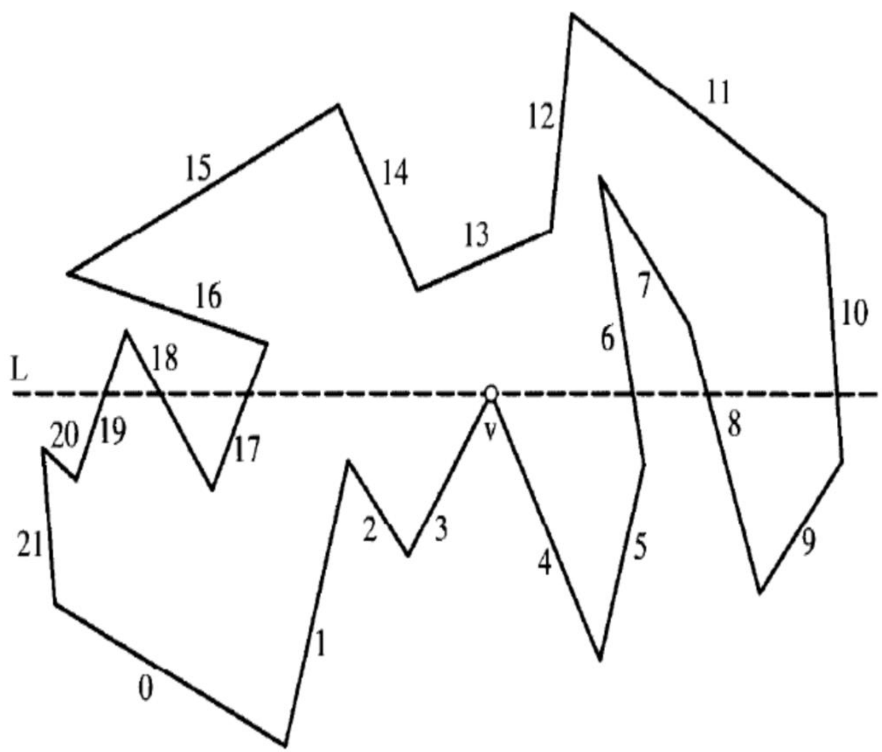
Idea of the Algorithm for trapezoidalization

- Uses a technique : Line Sweep (Plane Sweep)
- Line sweep (Nievergelt & Preparata 1982)
- Sweep a horizontal line over the plane maintaining a data structure along the line



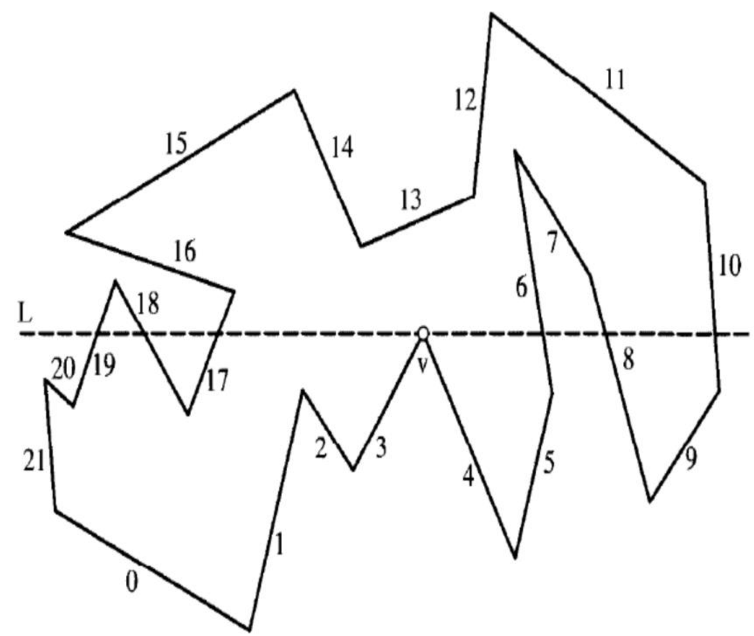
Line Sweep

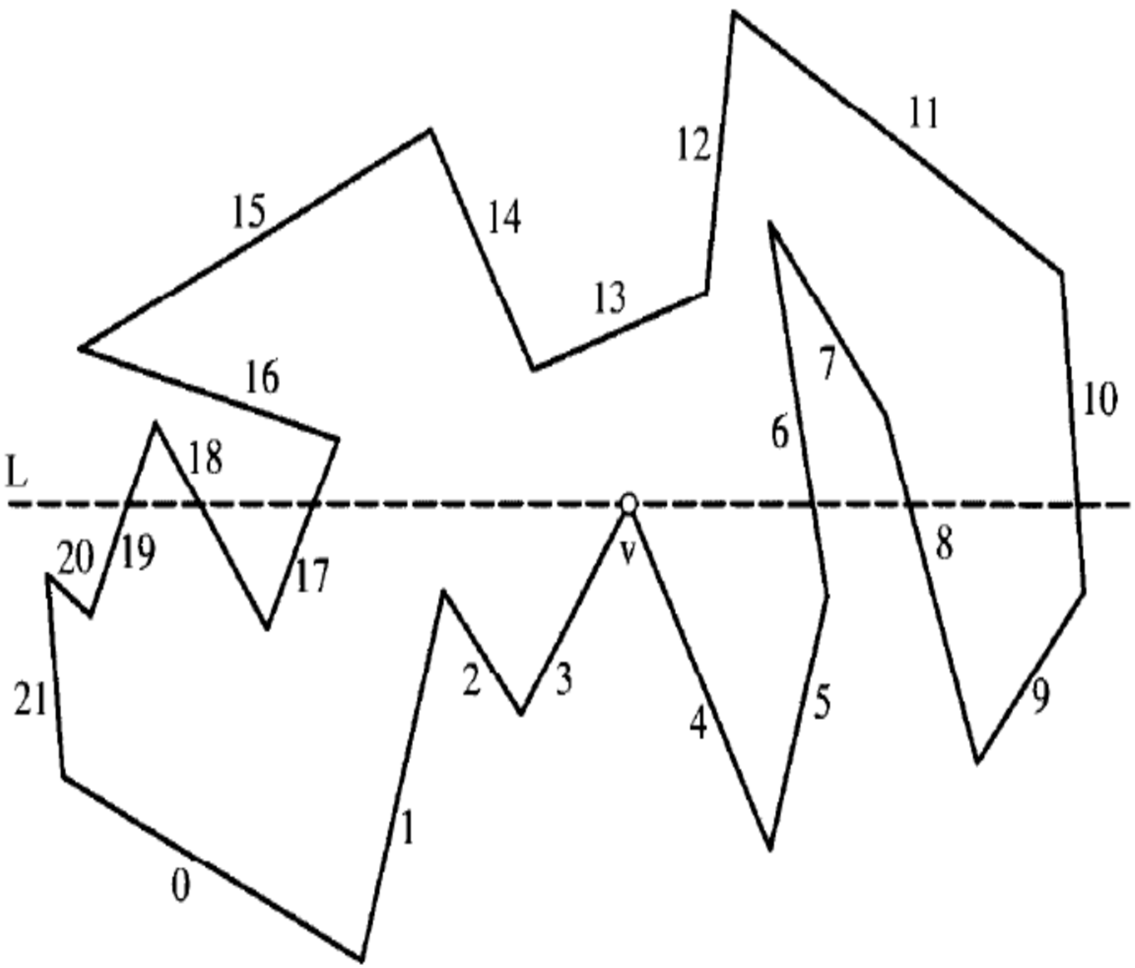
- The horizontal line L sweeps downward stopping at each vertex
- The sweep stops at discrete events and the data structure is updated



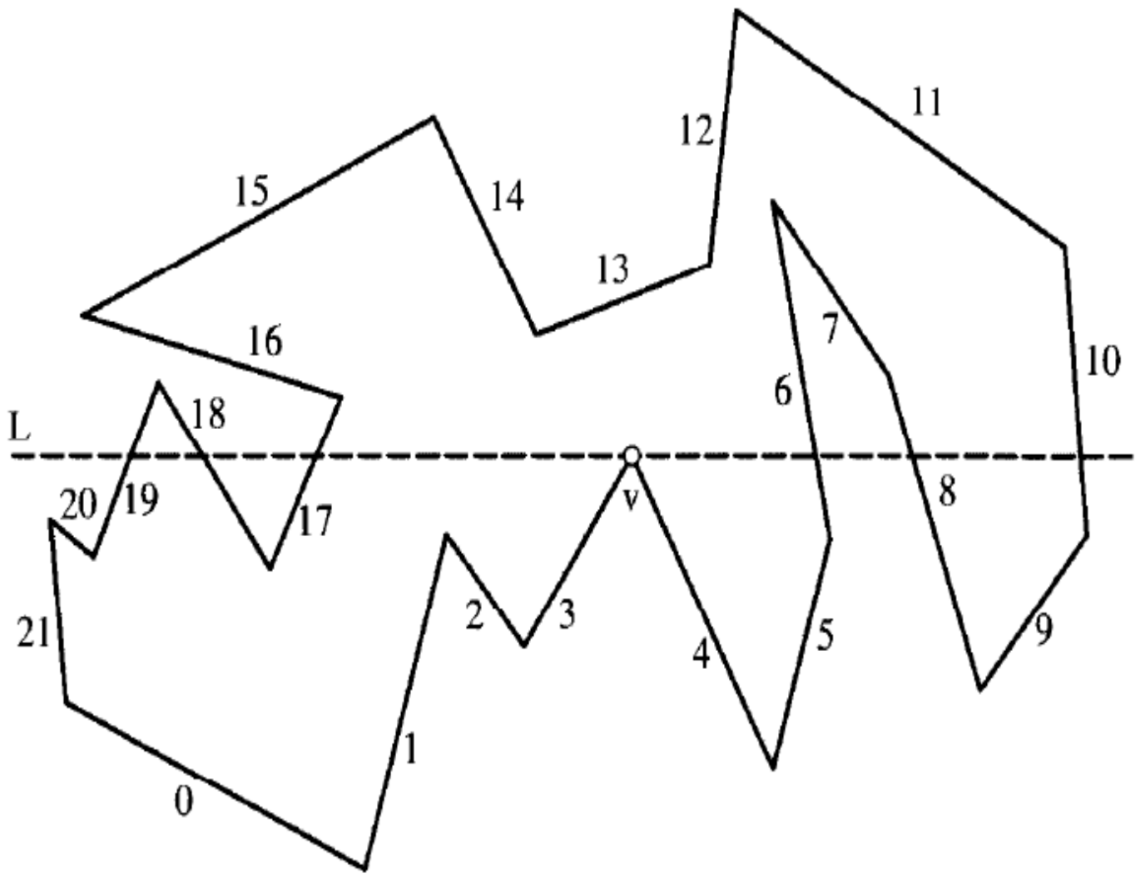
Updating the data structure

- The processing required at each vertex is finding the edge immediately to the left and immediately to the right of v along L
- To do this efficiently, a sorted list (LIST) of polygon edges pierced by L is maintained all times
- Hence, the vertices should be sorted with respect to x coordinate (For sorting : O(n logn))





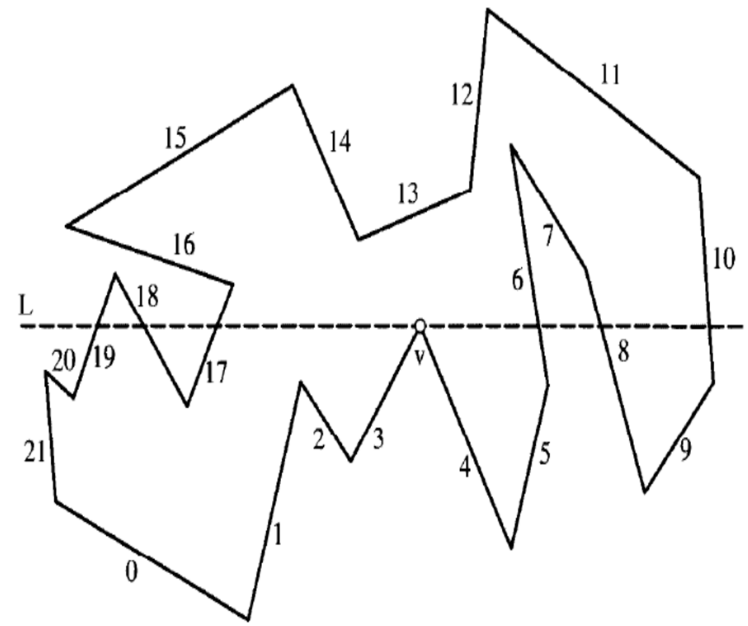
- For example in the figure,
  LIST = (e_{19}, e_{18}, e_{17}, e_6, e_8, e_{10})



- How to find out v lies between which two lines (efficiently)?

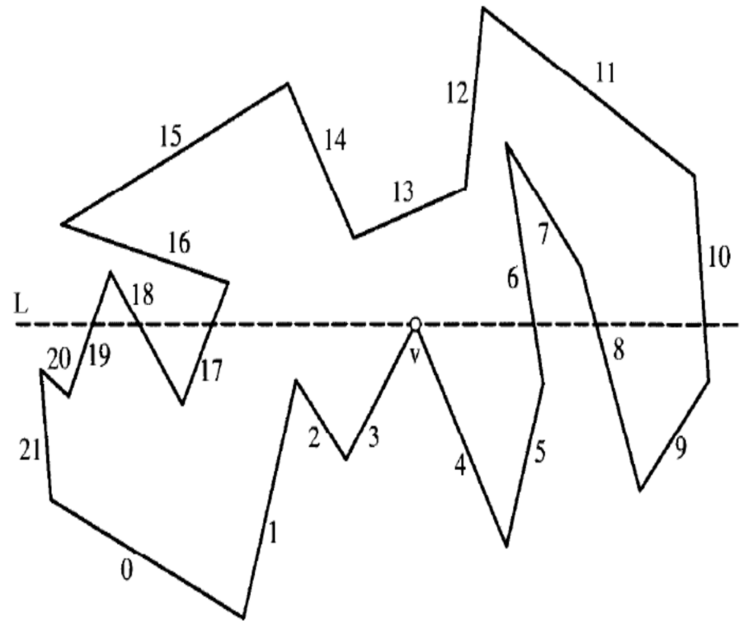
How to find out v lies between which all lines (efficiently)?

- Suppose LIST = (e_{19}, e_{18}, e_{17}, e_6, e_8, e_{10}) is available
- Suppose e_i is a pointer to an edge from which the coordinates of its endpoints can be found out
- Suppose the vertical coordinate of v (and L) is y which is known
- We have to find out the x coordinate of the intersection between L and e_i



Eg: How to find out v lies between which all lines (efficiently)?

- We have pointers for e_{19}, e_{18}, e_{17}, e_6, e_8, e_{10}
- We know y coordinate of L
- We know the coordinates of v
- We can compute :
  - x coordinate of the intersection bw L and all e_i



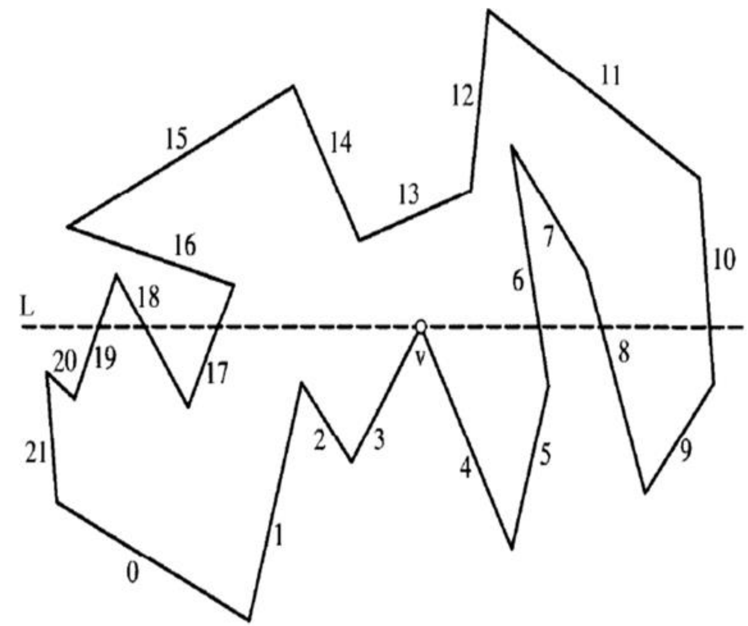
- Thus we know v lies bw e_{17} and e_6

Complexity of maintaining LIST

- If we search the whole LIST to search v lies between which all edges, this will take O(n)
- Suppose the list is maintained as a height balanced tree, the search can be done in O(log n)
- It is enough to show that all the operations on the data structure take place in O(log n) time
- What all are the operations on this data structure?
- The operations depends on events.
- What all are the events possible?

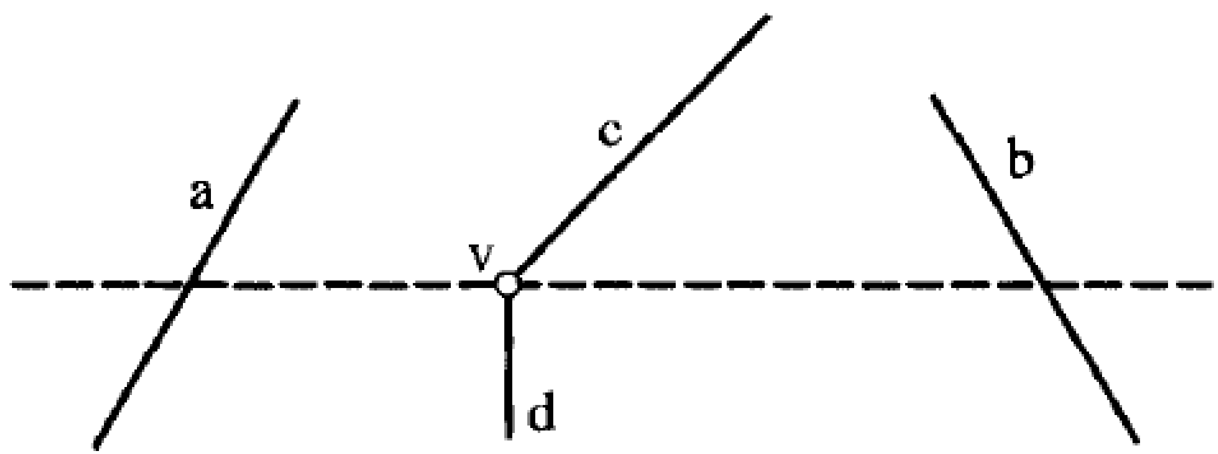
Events : Example

- Assume:
  - v falls between edges a and b on L
  - v be shared by edges c and d on L

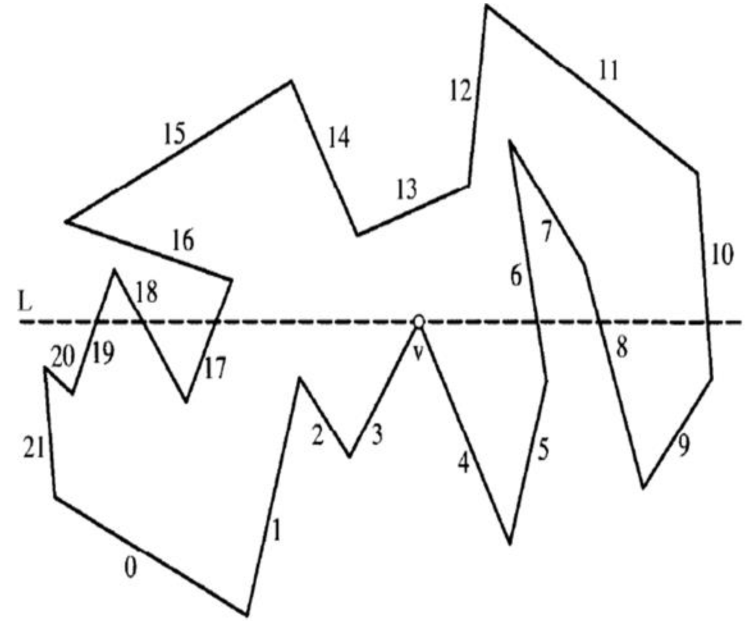


- Eg:

(1)

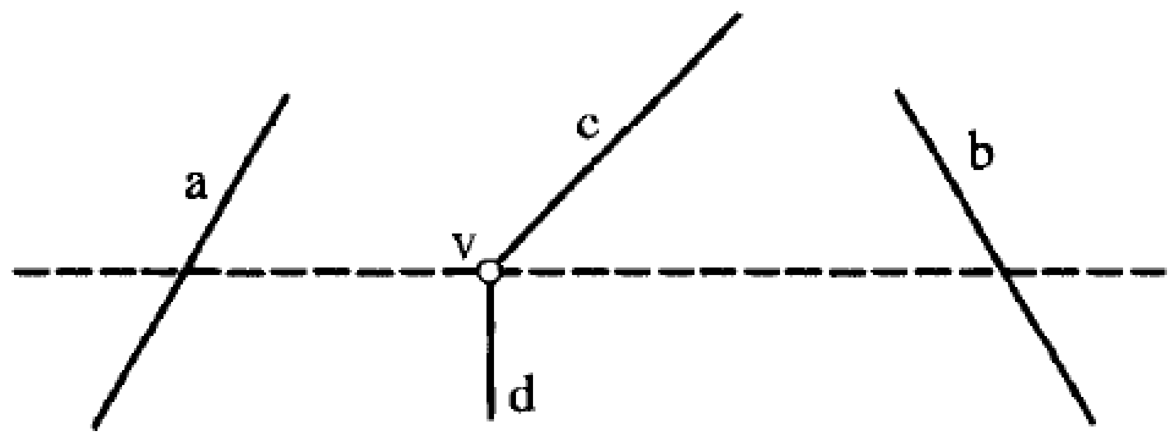


Three events : Event 1

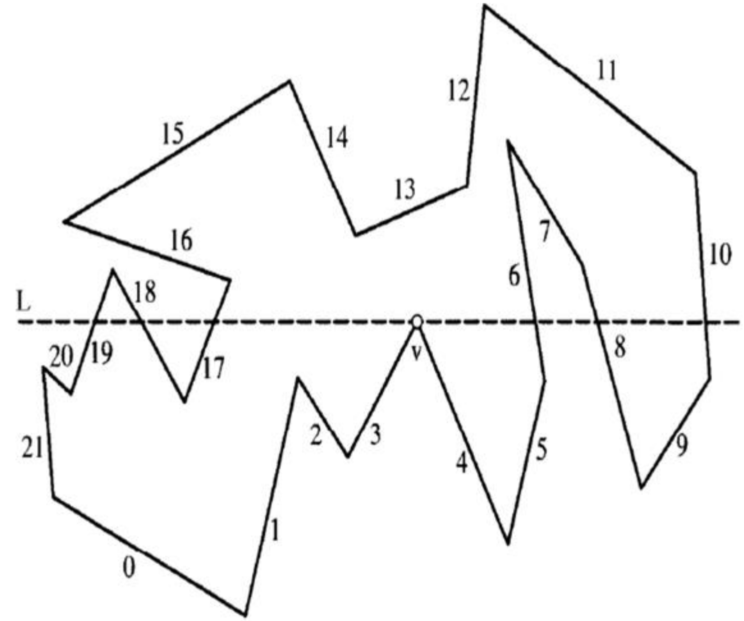


1. c is above L and d below. Then delete c from $\mathcal{L}$ and insert d:

$$(\ldots, a, c, b, \ldots) \Rightarrow (\ldots, a, d, b, \ldots).$$

(1) 

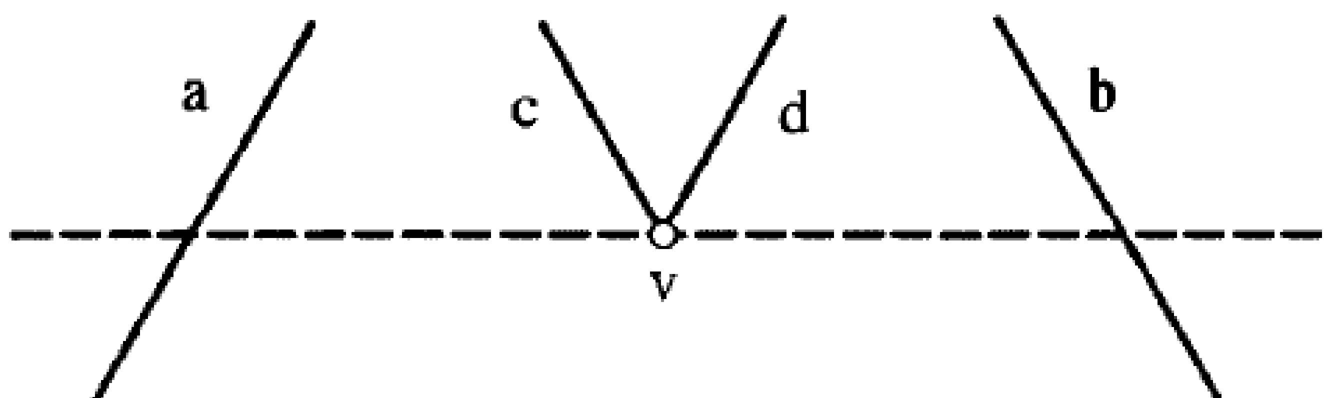
Event 2



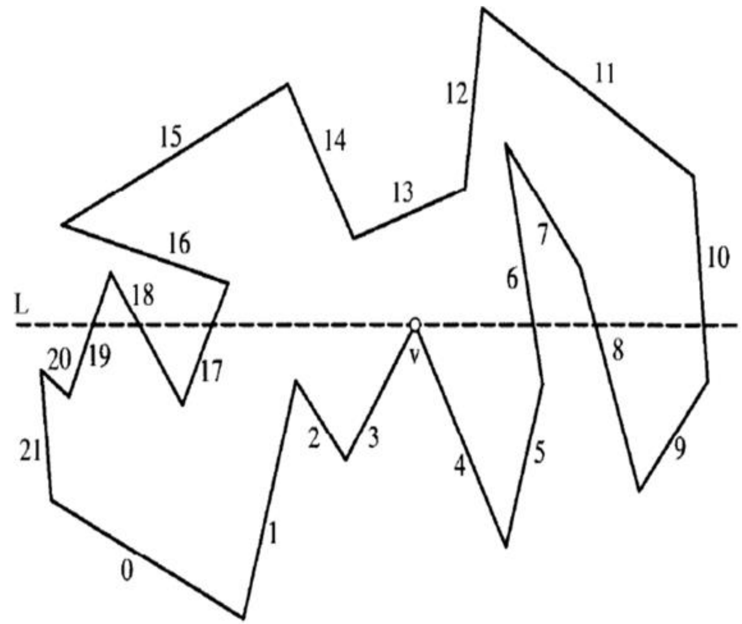
2. Both c and d are above L. Then delete both c and d from $\mathcal{L}$:

$$(\ldots, a, c, d, b, \ldots) \Rightarrow (\ldots, a, b, \ldots).$$

(2)



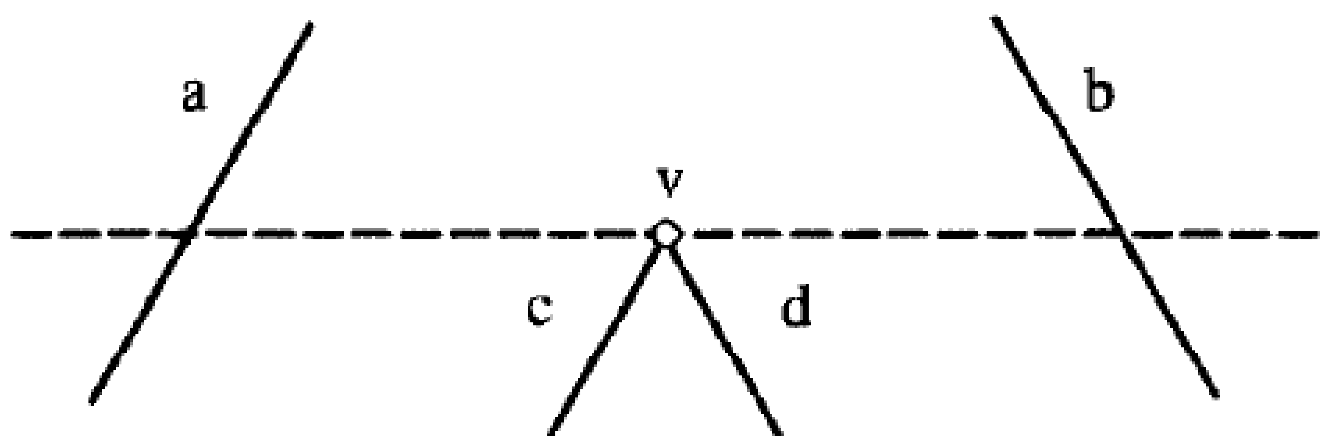
Event 3



3. Both c and d are below L. Then insert both c and d into $\mathcal{L}$:

$$(\ldots, a, b, \ldots) \Rightarrow (\ldots, a, c, d, b, \ldots).$$

(3)



Complexity of maintaining LIST

- Complexity is O(log n)
- Why?
- Only additions and deletions to a List
- Both can be done in O(log n) in a height balanced tree

$$(e_{12}, e_{11})$$

$$(e_{15}, e_{14}, e_{12}, e_{11})$$

$$(e_{15}, e_{14}, e_{12}, e_6, e_7, e_{11})$$

$$(e_{15}, e_{14}, e_{13}, e_6, e_7, e_{10})$$

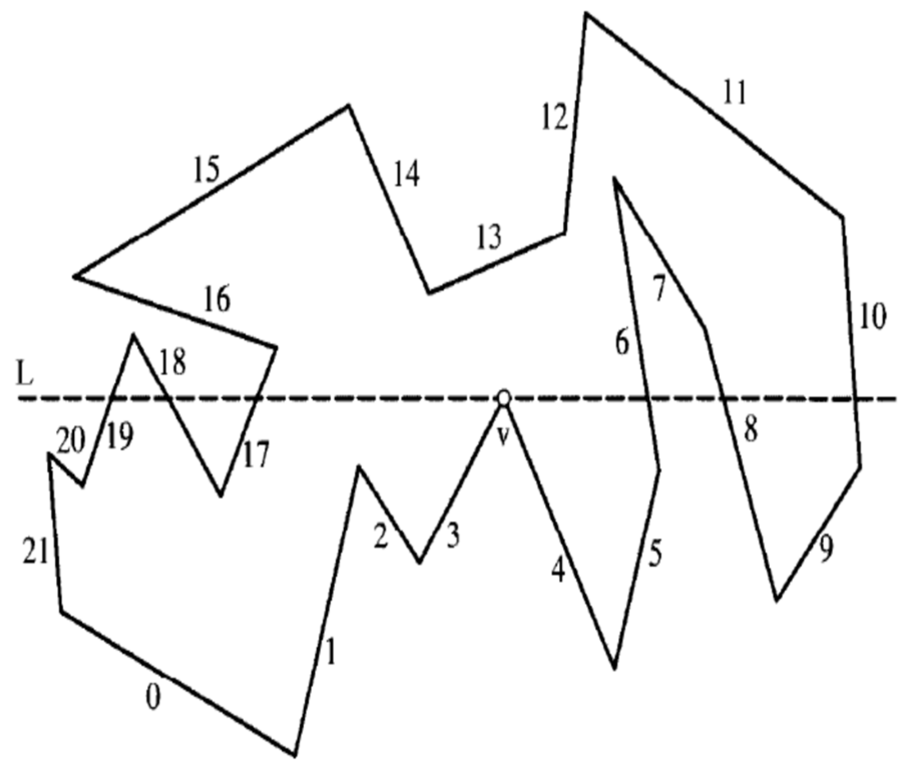
$$(e_{16}, e_{14}, e_{13}, e_6, e_7, e_{10})$$

$$(e_{16}, e_6, e_7, e_{10})$$

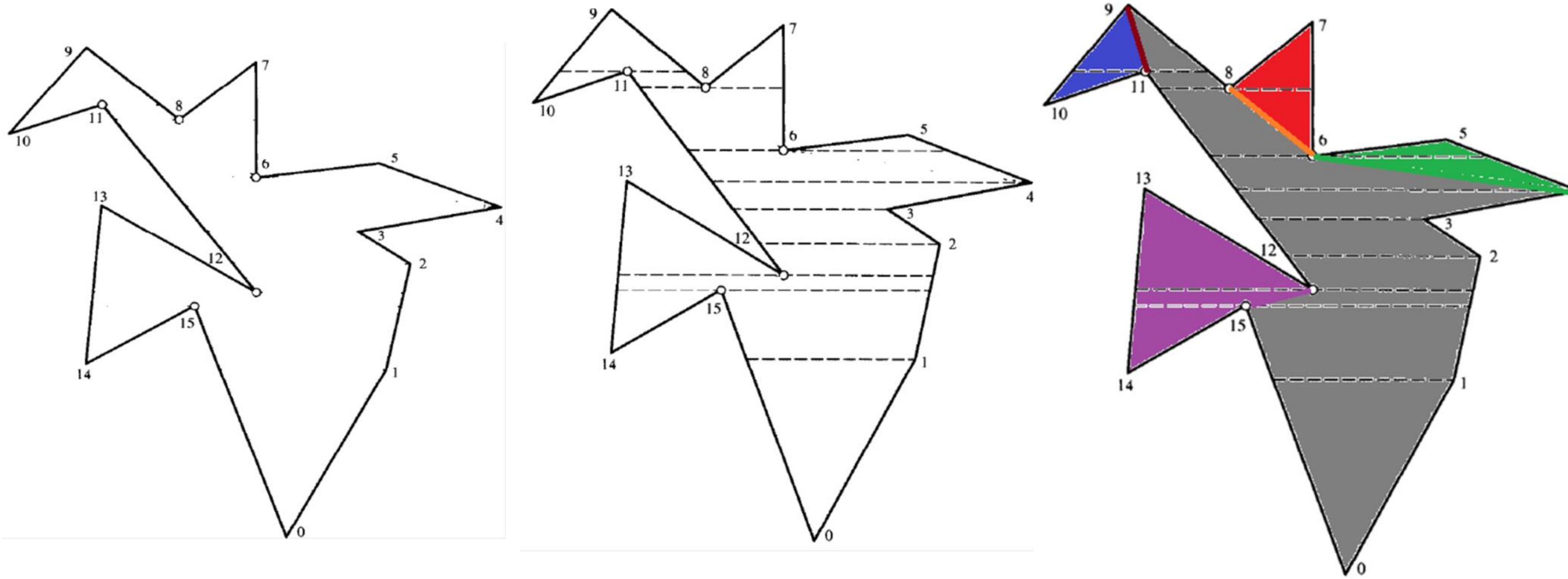
$$(e_{16}, e_6, e_8, e_{10})$$

$$(e_{19}, e_{18}, e_{16}, e_6, e_8, e_{10})$$

$$(e_{19}, e_{18}, e_{17}, e_6, e_8, e_{10}) \ .$$



Triangulation



Algorithm: POLYGON TRIANGULATION: MONOTONE PARTITION

Sort vertices by y coordinate.

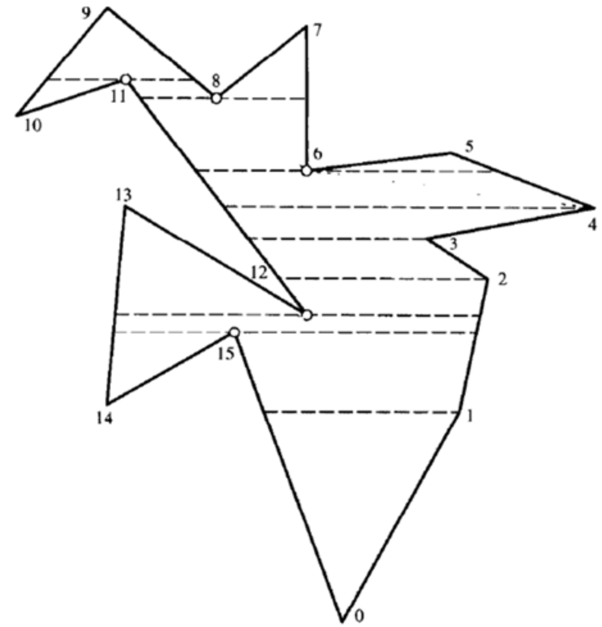
Perform plane sweep to construct trapezoidalization.

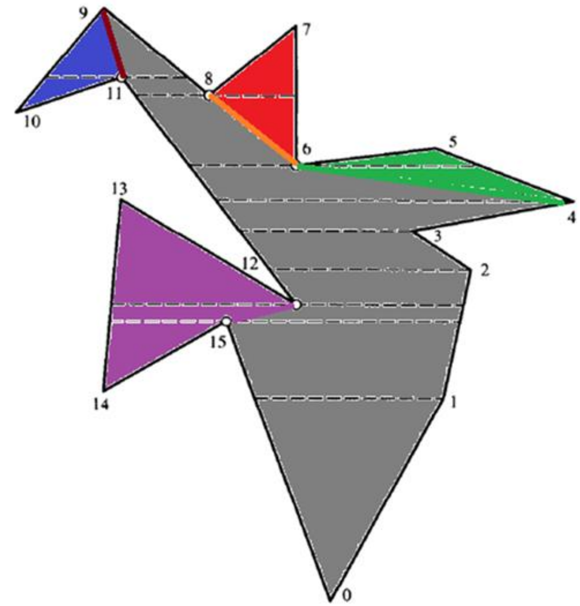
Partition into monotone polygons by connecting from interior cusps.

Triangulate each monotone polygon in linear time.

Partition into monotone polygons by connecting from interior cusps.

Monotone Partitioning Algorithm





- (1) Sort vertices by decreasing height: $v_0, \ldots, v_n$, with v_0 being the highest
- (2) {Descending pass}
- **for** i = 1 **to** n **do**
- Remove upward-pointing interior cusps
- (3) {Ascending pass}
- **for** $i = n - 1$ **downto** 0 **do**
- Remove downward-pointing interior cusps

Reading exercise

- Partitioning in to monotone mountains
- Triangulation of monotone mountains
- Refer J. O Rourke, *Computational Geometry in C*, 2/e, Cambridge University Press, 1998.
- Reading exercise will be considered for Midterm test

References

- J. O'Rourke: Art Gallery Theorems and Algorithms
- J. O Rourke, *Computational Geometry in C,* 2/e, Cambridge University Press, 1998 *)*
- *https://www.cs.jhu.edu/~misha/Spring16/05.pdf*
  - *From John Hopkins University*

Thank you