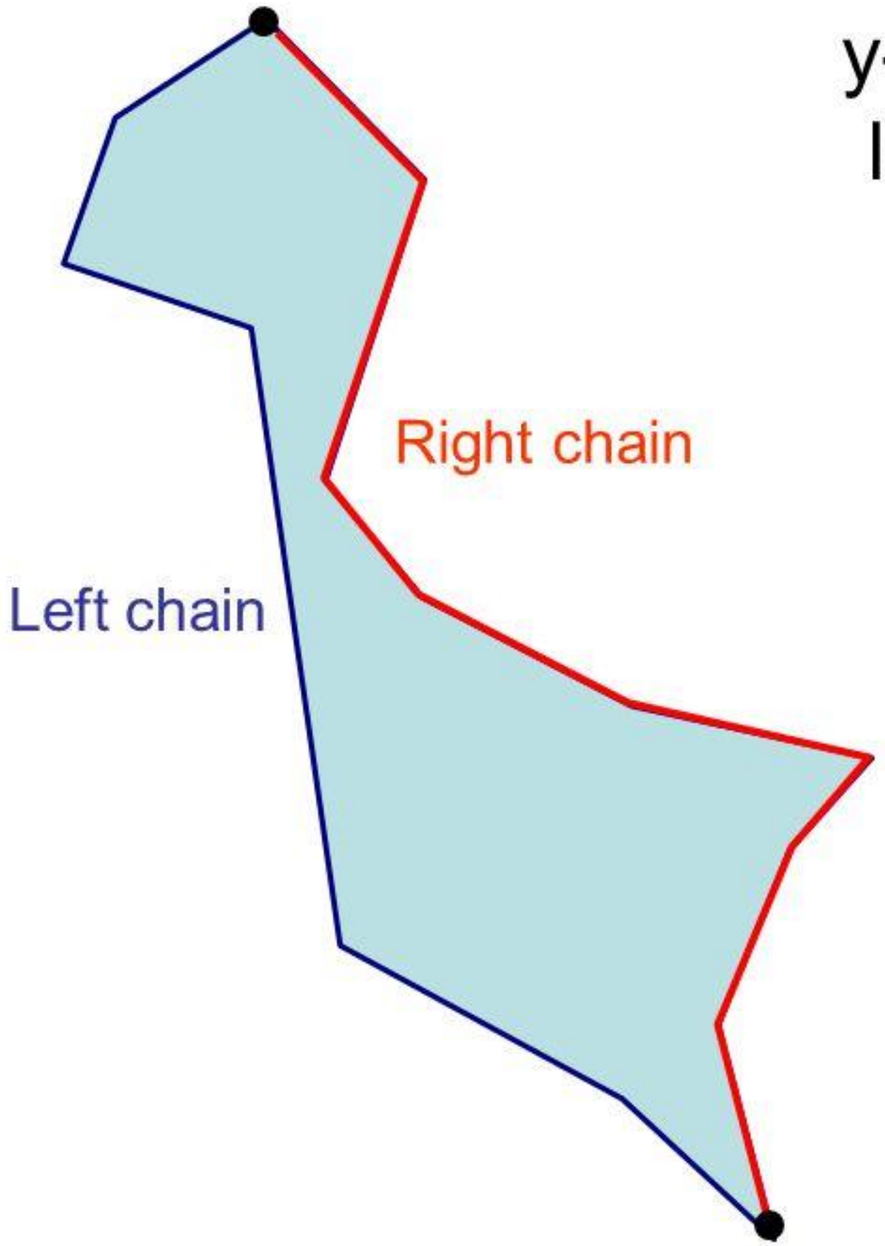
Polygon Partitioning

Partitioning the Polygon to Monotone polygons

A monotone polygon

- A polygon P is said to be monotone with respect to line L if ∂P can be split in to two polygonal chains such that each chain is monotone with respect to L
- The two chains share a vertex at either end

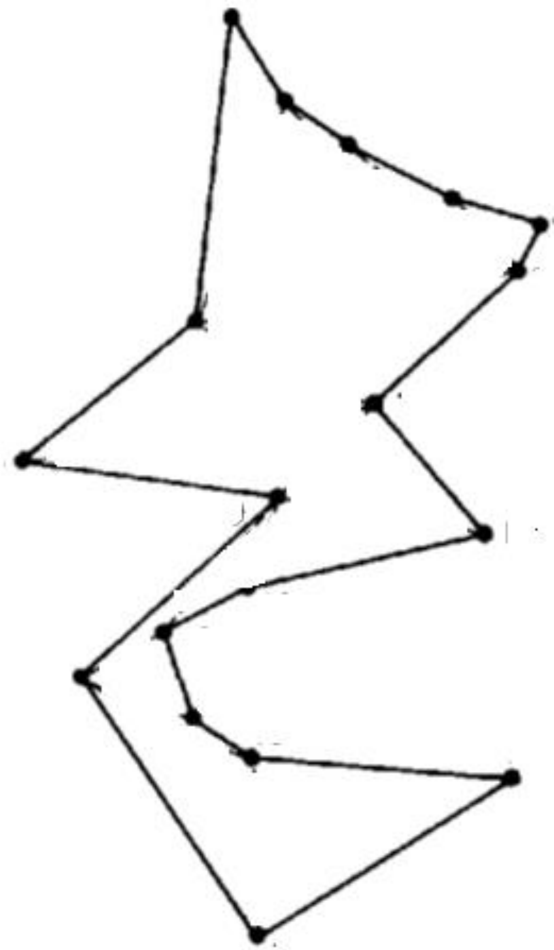
y-monotone polygon: left and right chains

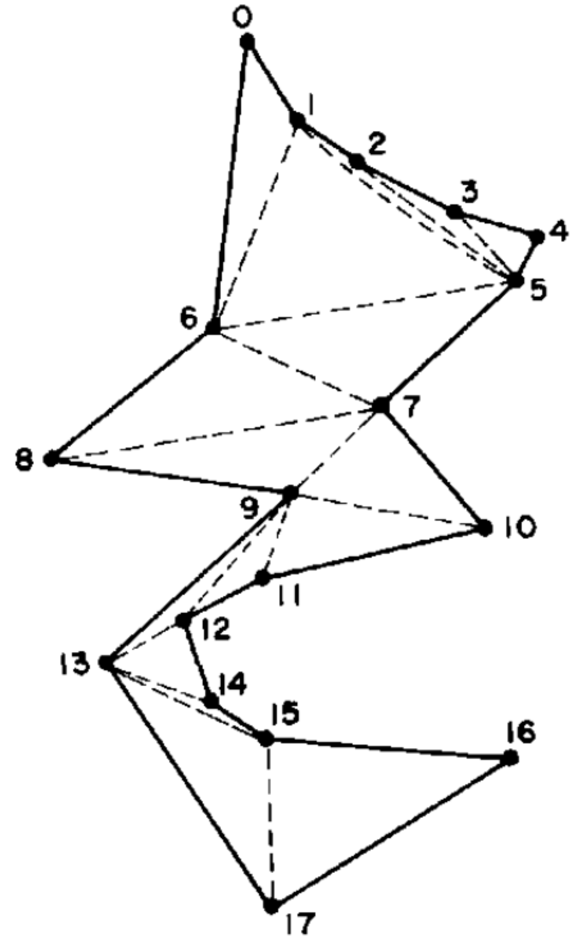


We will also assume that the polygon is strictly y-monotone, i.e. it is y-monotone and has no horizontal edges. Additionally, you may assume that no two vertices have the same y-coordinate

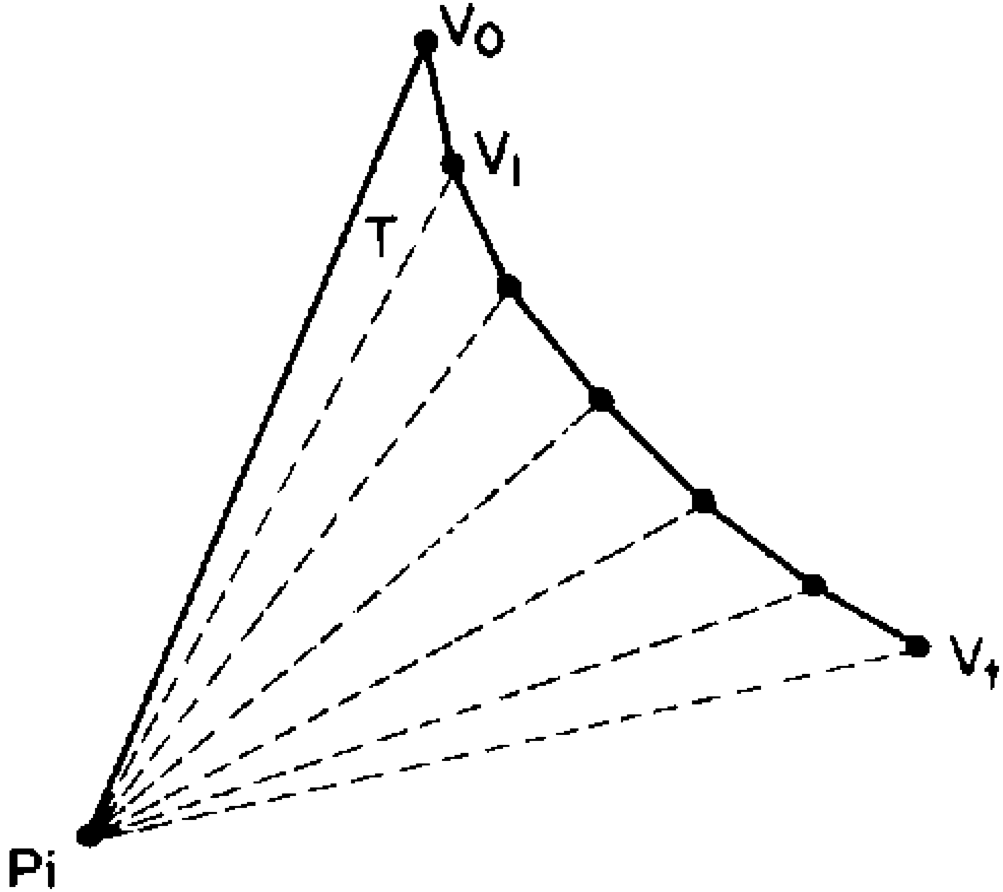
Triangulation of a monotone polygon

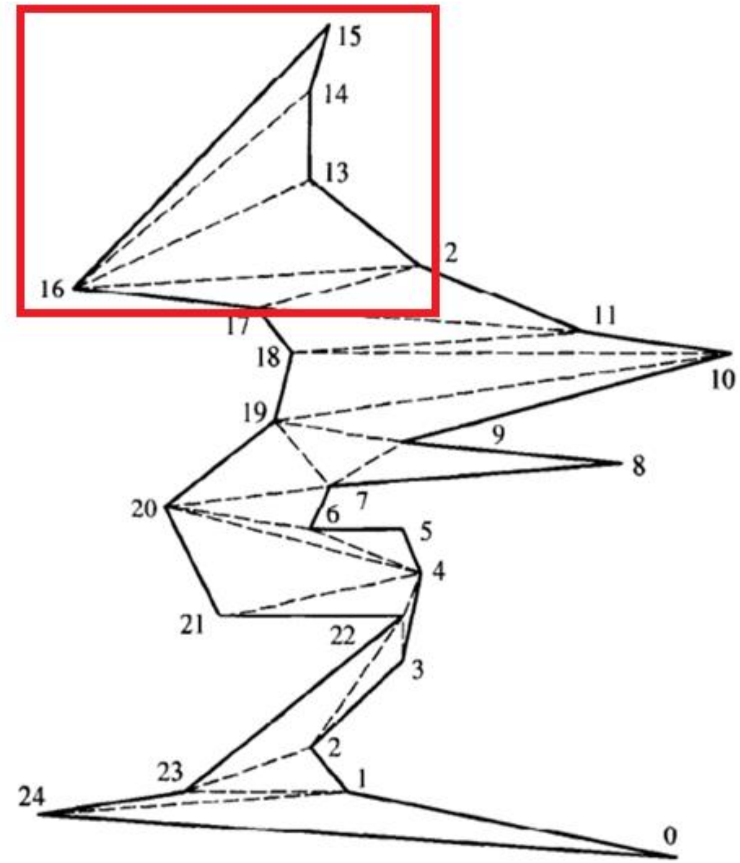
Input and Output





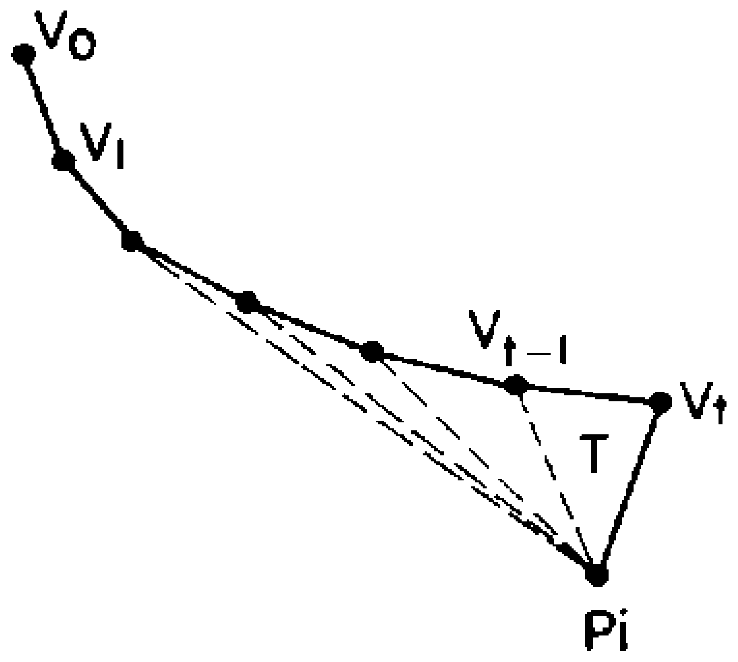
To join the vertices to form triangles: Case-1

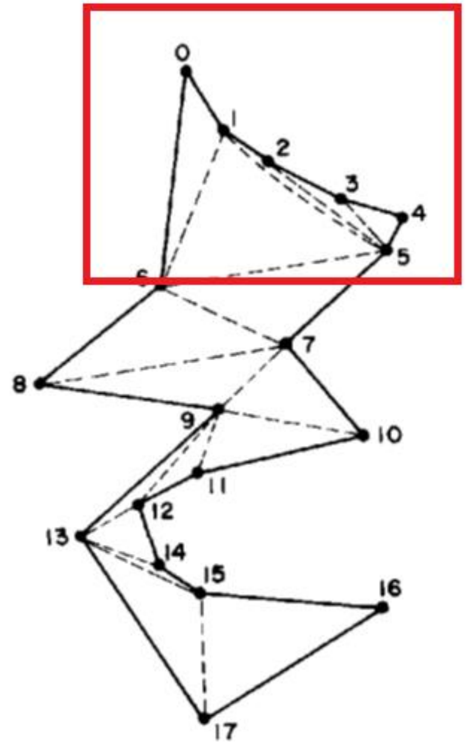




To join the vertices to form triangles

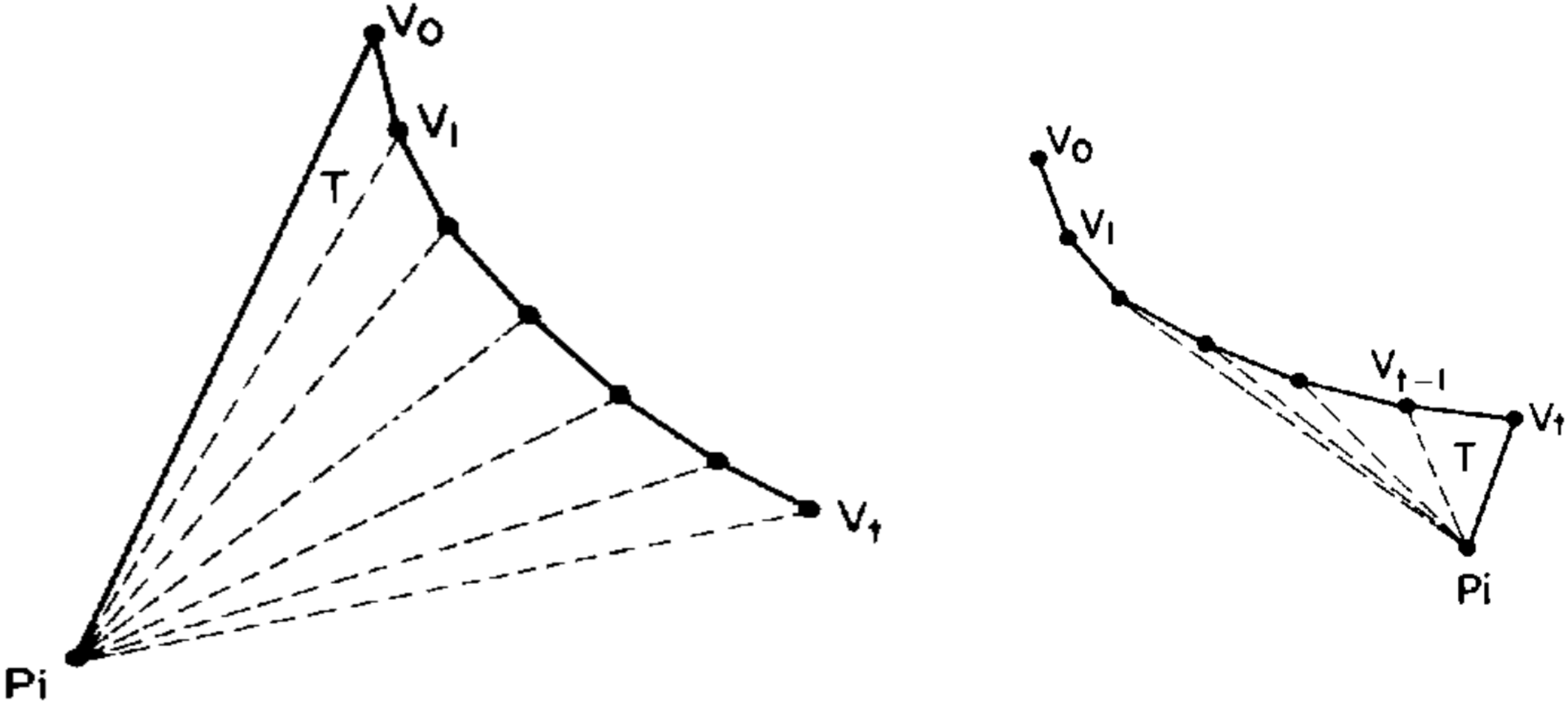
- Case-2 :





Data Structure

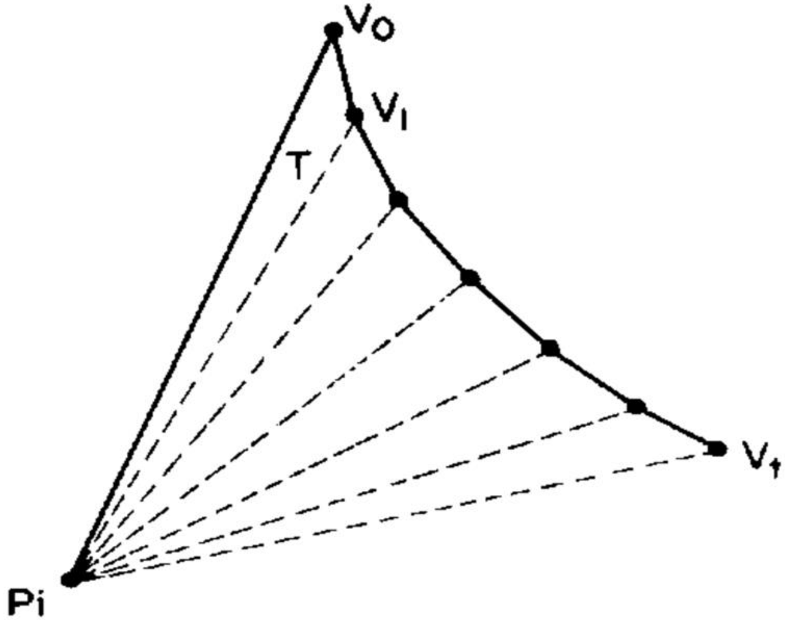
- Stack
- $v_0, v_1, \ldots.., v_t$ - vertices in the stack with v_t as the stack top
- p_i is the vertex to be processed

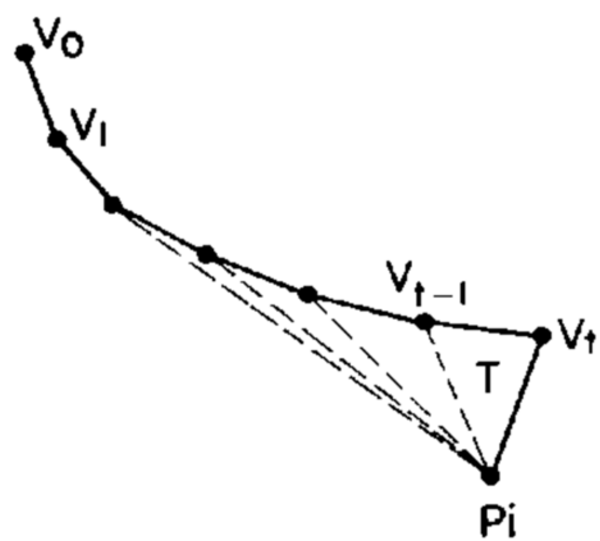


Stack properties & Complexity Analysis : Triangulation of a monotone polygon

Recall: Stack properties

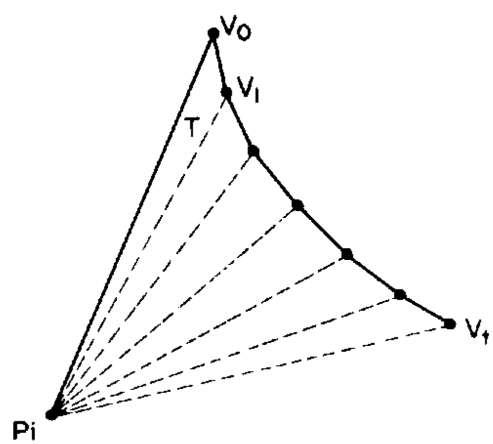
- **Maintained throughout the processing:**
- *v0, . . . , vt* decrease by height, *vt* lowest.
- *v0 . . . , vt* form a chain of consecutive vertices on the boundary of the polygon *Pi*
- *v1 . . . , vt-1* are reflex vertices.
- The next vertex *pi* to be processed is adjacent via a polygon edge of *Pi* to either *v0* or *vt* (or to both).

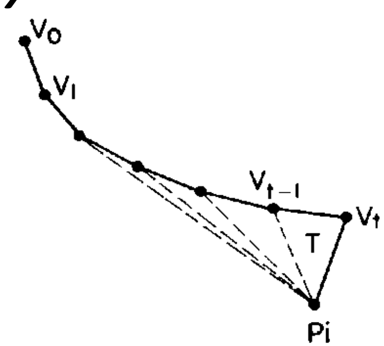




Argument : the maintenance of the stack properties

- ***Stack Propety 1: v0, . . . , vt* decrease by height, *vt* lowest**
- Only *pi*, and *vi's* are pushed onto the stack, and when both are pushed they are pushed in the correct vertical order.
- Thus the vertices are in decreasing order by *y*-coordinate





for i = 2 **to** *n* - 1 **do**
if *pi* is adjacent to *v0* **then**
Begin
  while *t* > 0 **do**
  begin
  Draw diagonal *pi —>vt.*
  Pop
  end
  Pop
  Push *vt*
  Push *pi*
end

else if *pi* is adjacent to *vt* **then**
begin
  while *t* > 0 and *vt* is not reflex **do**
  begin
  Draw diagonal *pi —> vt-1*
  Pop
  end
  Push *pi*
end

Argument : maintenance of the stack properties

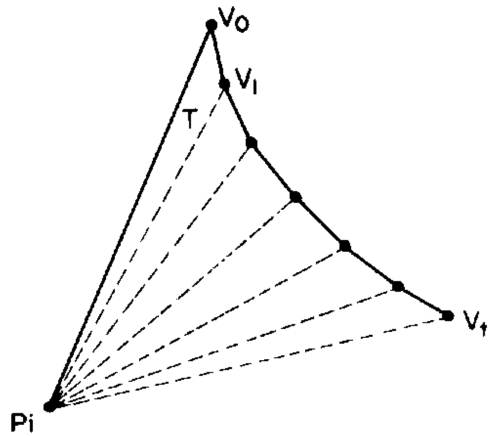
- ***Stack Propety 2: $v_0, \ldots, v_t$* form a chain of consecutive vertices on the boundary of the polygon P_i**
- The vertices form a chain, because of adjacency in both cases

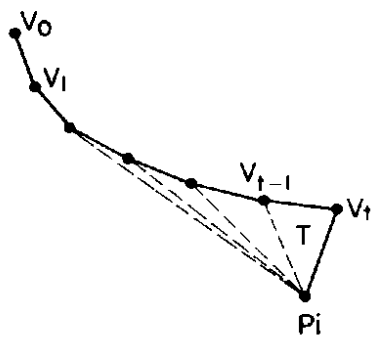
```
for i = 2 to n - 1 do
if  pi is adjacent to v0 then
Begin
        while t > 0 do
        begin
        Draw diagonal pi —>vt.
        Pop
        end
        Pop
        Push vt
        Push pi
end
```

```
else if pi is adjacent to vt then
begin
        while t > 0 and vt is not reflex do
        begin
        Draw diagonal pi —> vt-1
        Pop
        end
        Push pi
end
```

Argument : the maintenance of the stack properties

- ***Stack Propety 3: v_1 . . . , v_{t-1}* are reflex vertices**
- The internal angles are reflex because p_i is only pushed when v_t is reflex in the second **while.**



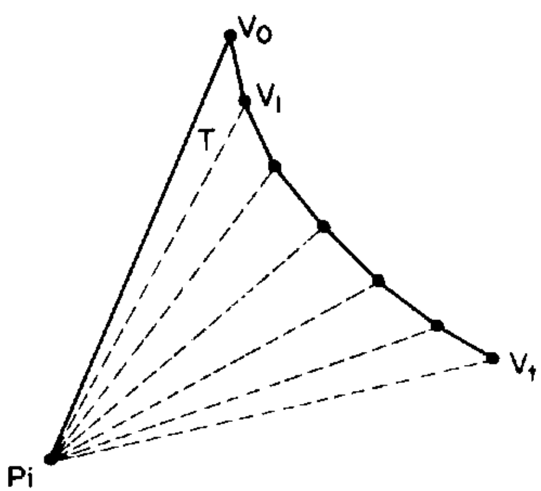


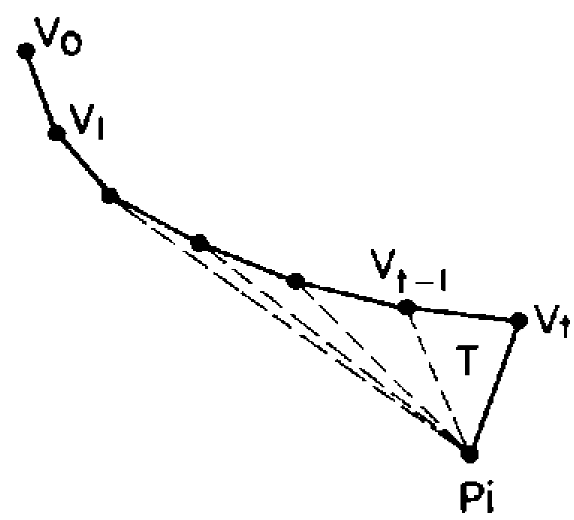
for i = 2 **to** *n* - 1 **do**
if *pi* is adjacent to *v0* **then**
Begin
    while *t* > 0 **do**
    begin
    Draw diagonal *pi —>vt.*
    Pop
    end
    Pop
    Push *vt*
    Push *pi*
end

else if *pi* is adjacent to *vt* **then**
begin
    while *t* > 0 and *vt* is not reflex **do**
    begin
    Draw diagonal *pi —> vt-1*
    Pop
    end
    Push *pi*
end

Argument : maintenance of the stack properties

- ***Stack Propety 4:* The next vertex *pi* to be processed is adjacent via a polygon edge of *Pi* to either *v0* or *vt* (or to both)**
- pi, is either adjacent to *v0* or *vt* because the monotonicity of *Pi* guarantees that *pi*, has a (unique) neighbor above it, and in the chain *v0. . . , vt,* only *v0* and *vt* do not have all their neighbors accounted for.





Time complexity

- Identify the major steps contributing to the time complexity

- ***Algorithm* : Triangulation of a Monotone Polygon**
- Sort vertices by decreasing y-coordinate, resulting in $p_0, \ldots, p_n$.
- Push p_0.
- Push p_1.
- **for** i = 2 **to** n - 1 **do**
- **if** p_i is adjacent to v_0 **then**
- **begin**
  - **while** $t > 0$ **do**
  - **begin**
  - Draw diagonal $p_i \longrightarrow v_t$.
  - Pop
  - **end**
  - Pop
  - Push v_t
  - Push p_i
- **end**
- **else if** p_i is adjacent to v_t **then**
- **begin**
  - **while** $t > 0$ and v_t is not reflex **do**
  - **begin**
  - Draw diagonal $p_i \longrightarrow v_{t-1}$
  - Pop
  - **end**
  - Push p_i
- **end**

Major steps :Time complexity

- Sorting in linear time
- Each vertex is pushed at most twice on the stack, once as *pi* and once as *vi.*
- Examination of the code shows that for each Push there is a corresponding Pop, and thus the algorithm requires *O(n)* time.

i	stack	condn	while	diag
2	0,1	else	No (1 refl)	
3	0,1,2	else	No(2 refl)	
4	0,1,2,3	else	No(3 refl)	
5	0,1,2,3, 4	else	Yes(4 notref)	5,3
5	0,1,2,3	Yes(angle 532 or angle 3 *not reflex*)		5,2
5	0,1,2	Yes(angle 521 or angle 2 *not reflex*)		5,1
5	0,1		No(1 ref)	
5	0,1,5 as pi			
6	0,1,5 as vt	if	Yes	6,5
6	0,1		Yes	6,1
6	0		No	
6	5,6			

Triangulation of a monotone polygon

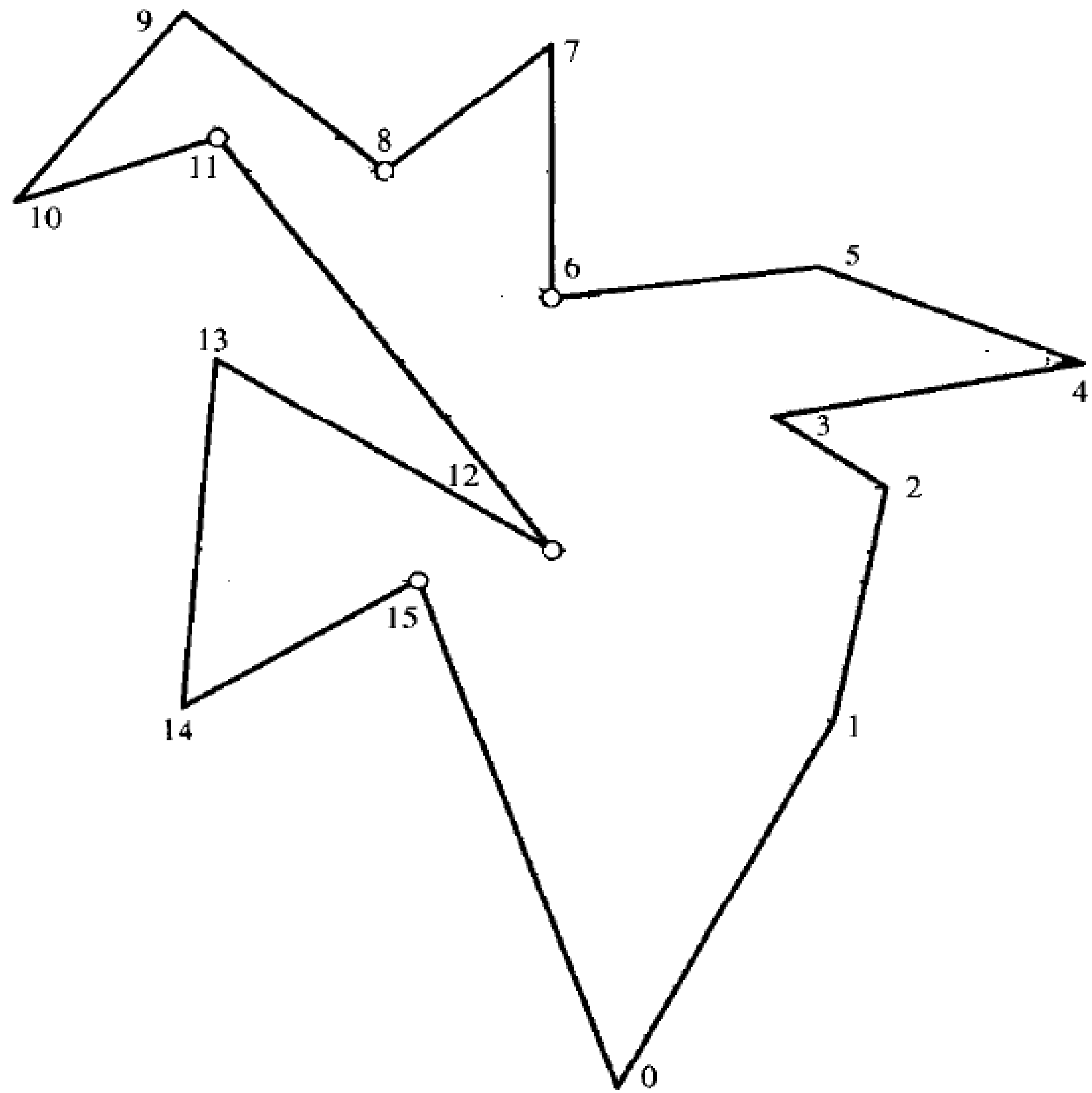
- **Triangulation of a monotone polygon can be done in linear time**

- Is it possible to use this algorithm (triangulation of a monotone polygon) to triangulate a non-monotone polygon (normal polygon) efficiently?
- The time complexity of the algorithm for triangulation of a normal polygon - $O(n^2)$

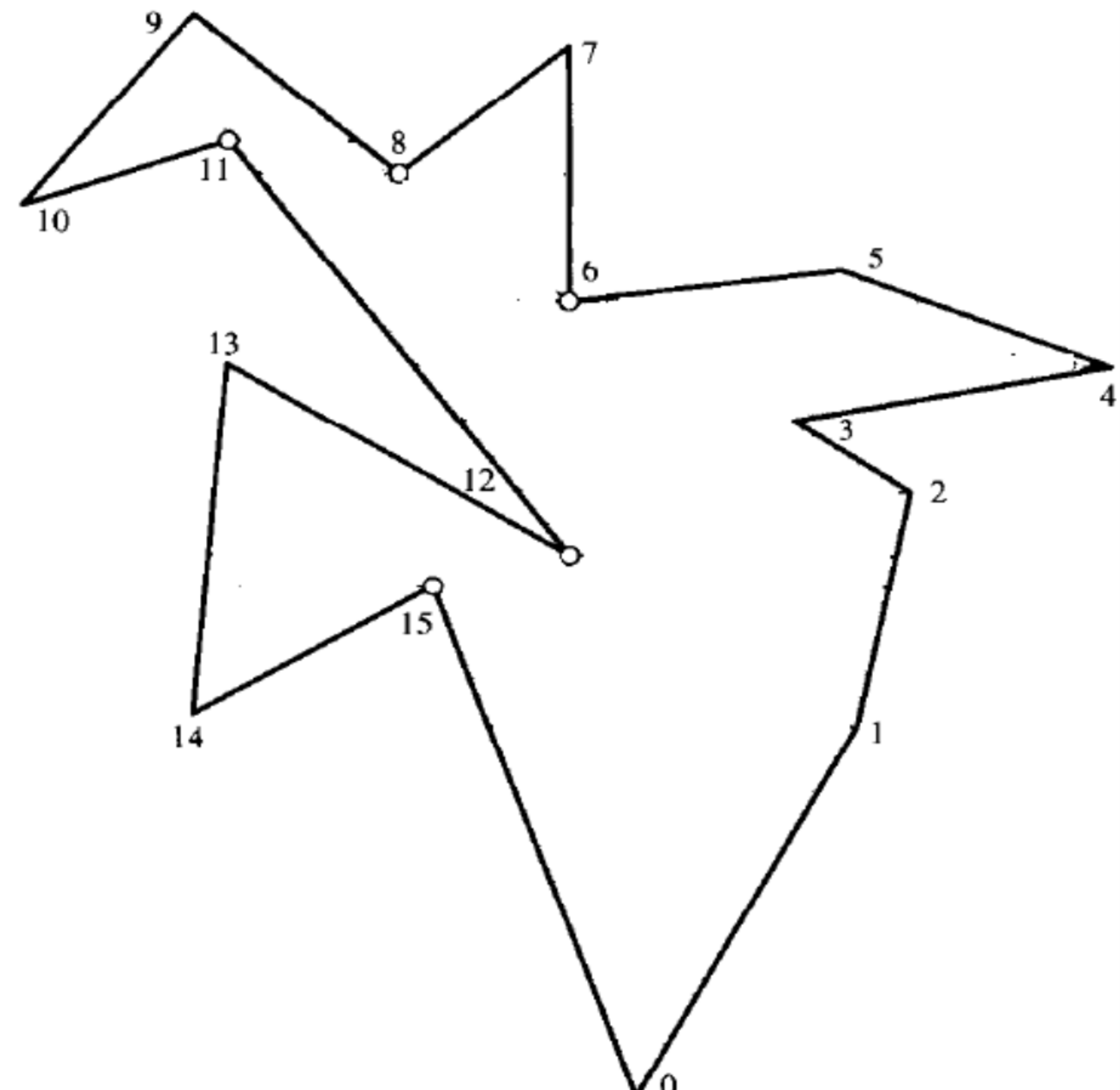
Algorithm to triangulate a non-monotone (normal) polygon

- 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

Is P monotone?

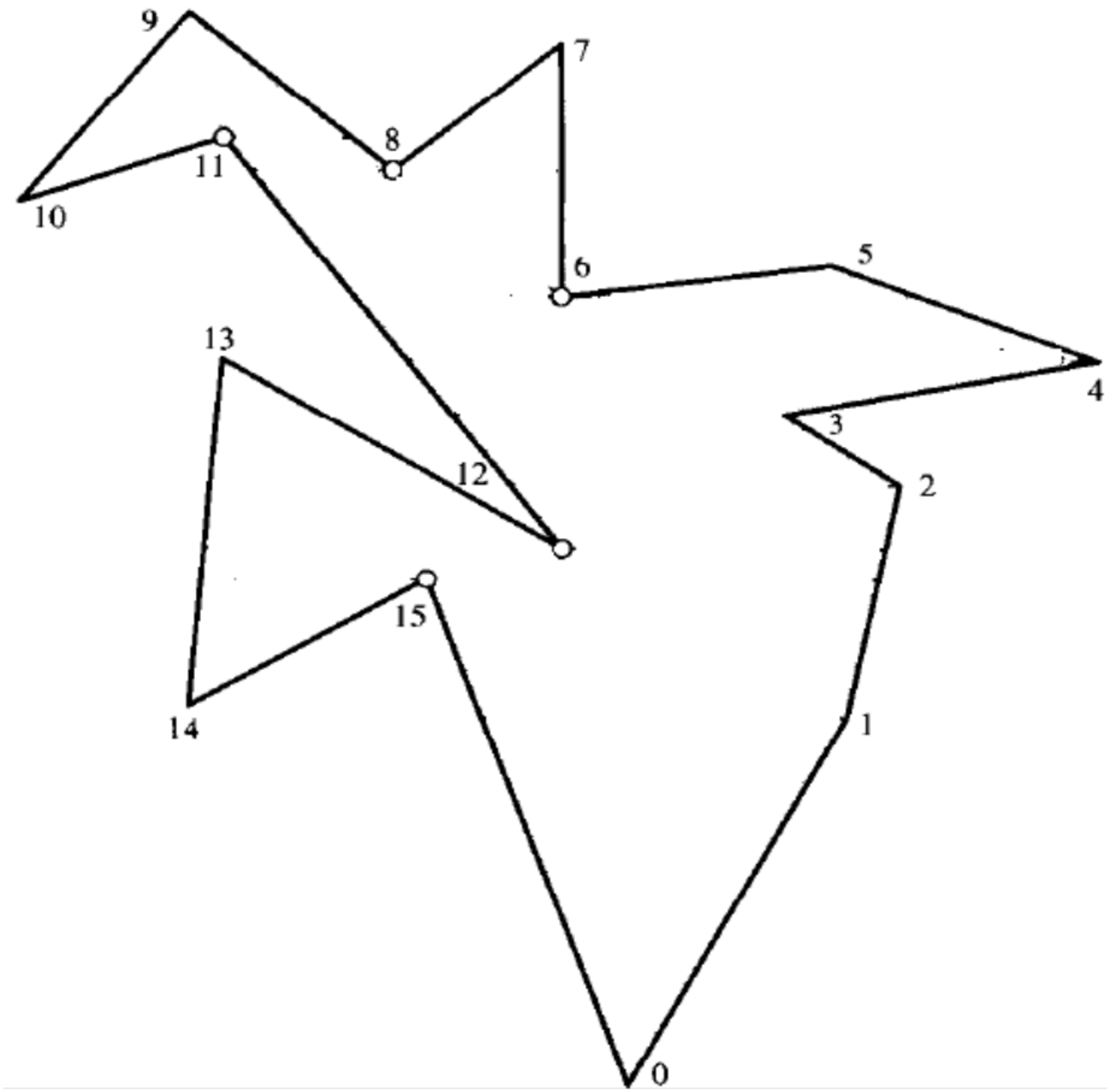


What characteristic makes P non-monotone?



- Interior cusps

To make P monotone:



- Remove/ Break interior cusps
- Consider vertex 8

References

- J. O’Rourke, *Computational Geometry in C*, 2/e, Cambridge University Press, 1998
- J. O'Rourke: Art Gallery Theorems and Algorithms

Thank you