

### COMP5045 Computational Geometry

#### COMMONWEALTH OF AUSTRALIA Copyright Regulations 1969 WARNING

This material has been reproduced and communicated to you by or on behalf of the University of Sydney pursuant to Part VB of the Copyright Act 1968 (**the Act**).

The material in this communication may be subject to copyright under the Act. Any further copying or communication of this material by you may be the subject of copyright protection under the Act.

**Do not remove this notice.**

---

---

---

---

---

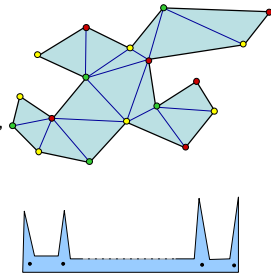
---

---

---

### Recall

- Every simple polygon with  $n$  vertices can be decomposed into  $n-2$  triangles.
- Every triangulated simple polygon can be 3-coloured.
- Every simple polygon can be "guarded" by  $n/3$  guards, and  $n/3$  guards is sometimes necessary.
- To find a guard set our algorithm requires a triangulation. Previously:  $O(n^2)$  time



---

---

---

---

---

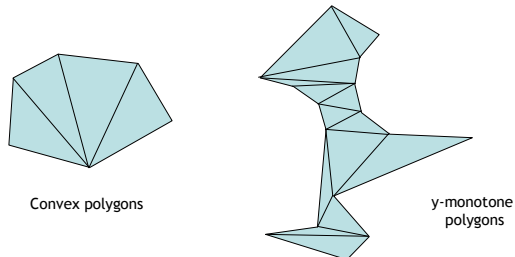
---

---

---

### Triangulating a simple polygon

**Observation:** Some polygons are very easy to triangulate.



---

---

---

---

---

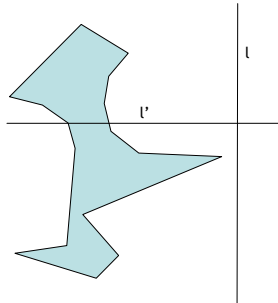
---

---

---

### l-monotone polygon

A simple polygon is called monotone w.r.t. a line  $l$  if for any line  $l'$  perpendicular to  $l$  the intersection of the polygon with  $l'$  is connected (y-monotone, if  $l = y$ -axis).



**Observation:** if  $P$  is  $l$ -monotone then  $P$  consists of two  $l$ -monotone chains.

---

---

---

---

---

---

---

---

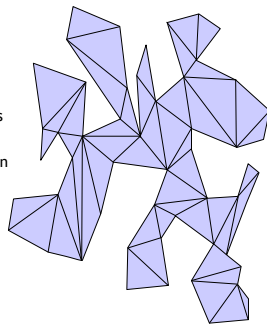
### Triangulate simple polygon

**Main idea:**

1. Partition  $P$  into  $y$ -monotone polygons  
Time  $O(n \log n)$
2. Triangulate each  $y$ -monotone polygon  
Time  $O(n_i)$

**Prove:**

A simple polygon can be triangulated in  $O(n \log n)$  time.



---

---

---

---

---

---

---

---

### Triangulate monotone polygon

**Idea:**

Use a plane sweep algorithm.

Try to triangulate everything you can below the sweep line by adding diagonals, and then remove the triangulated region from further consideration.

---

---

---

---

---

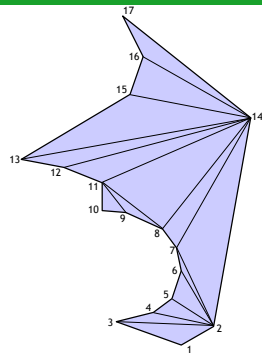
---

---

---

### Triangulate monotone polygon

- Advance along y-sorted vertex-list from bottom to top.
- For each vertex  $v$  in y-sorted order, add downward directed diagonals from  $v$  to visible vertices, starting from most recent  $u$  backwards. (Use a stack. See next slide.)



7

---

---

---

---

---

---

---

---

---

---

### Triangulate monotone polygon

- merge the vertices of the left and right chains of  $P$  into y-sorted order, say,  $u_1, u_2, \dots, u_n$ .
- push  $u_1$  and  $u_n$  onto an initially empty stack  $S$ .
- for  $j \leftarrow 3 \dots n-1$  do
  - if  $u_j$  and  $v_{\text{top}} \leftarrow \text{top}(S)$  are on different chains
  - then pop all vertices from  $S$ 
    - add a diagonal between  $u_j$  and each popped vertex except the last one.
    - push  $u_{j-1}$  and  $u_j$  onto  $S$ .
  - else pop one vertex from  $S$ 
    - pop all vertices from  $S$  that are visible from  $u_j$  and add a diagonal between  $u_j$  and each popped vertex.
    - push last popped vertex back onto  $S$ .
    - push  $u_j$  onto  $S$ .
- add diagonals from the last vertex  $u_n$  to all stack vertices except first and last.

8

---

---

---

---

---

---

---

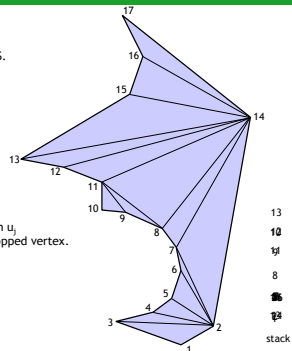
---

---

---

### Triangulate monotone polygon

- push  $u_1$  and  $u_n$  onto an initially empty stack  $S$ .
- for  $j \leftarrow 3 \dots n-1$  do
- if  $u_j$  and  $v_{\text{top}} \leftarrow \text{top}(S)$  are on different chains
- then pop all vertices from  $S$ 
  - add a diagonal between  $u_j$  and each popped vertex except the last one.
  - push  $u_{j-1}$  and  $u_j$  onto  $S$ .
- else pop one vertex from  $S$ 
  - pop all vertices from  $S$  that are visible from  $u_j$  and add a diagonal between  $u_j$  and each popped vertex.
  - push last popped vertex back onto  $S$ .
  - push  $u_j$  onto  $S$ .



9

---

---

---

---

---

---

---

---

---

---

### Running time?

1. merge the vertices of the left and right chains of P into y-sorted order, say,  $u_1, u_2, \dots, u_n$ .
2. push  $u_1$  and  $u_n$  into an initially empty stack S.
3. for  $j \leftarrow 3 \dots n-1$  do
  - a. if  $u_j$  and  $v_{\text{top}(S)}$  are on different chains
  - b. then pop all vertices from S
    - add a diagonal between  $u_j$  and each popped vertex except the last one.
    - push  $u_{j-1}$  and  $u_j$  onto S.
  - c. else pop one vertex from S
    - pop all vertices from S that are visible from  $u_j$
    - add a diagonal between  $u_j$  and each popped vertex.
    - push last popped vertex back onto S.
    - push  $u_j$  onto S.
4. add diagonals from the last vertex  $u_n$  to all stack vertices except first and last.

Step 3: n times - each iteration may take  $O(n)$  time  $\Rightarrow O(n^2)$  time

Can it be improved?

How many vertices are pushed onto the stack in each iteration? At most 2

How many vertices are popped in total?  $O(n) \Rightarrow O(n)$  time

---

---

---

---

---

---

---

---

---

---

---

---

### Result

#### Theorem:

A y-monotone polygon can be triangulated in  $O(n)$  time!

---

---

---

---

---

---

---

---

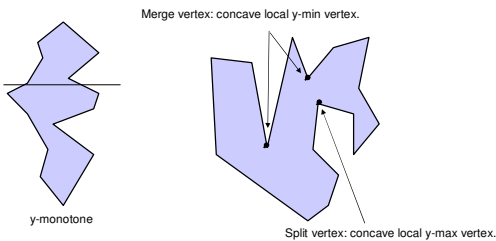
---

---

---

---

### Partition into monotone polygons



FACT: P is y-monotone if and only if it does not have any turn vertices.

Subdivide the simple polygon into monotone sub-polygons by adding diagonals to split and merge vertices.

---

---

---

---

---

---

---

---

---

---

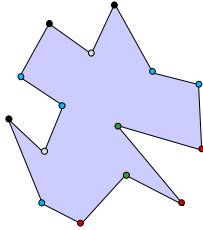
---

---

## Monotonic partitioning

**Idea:**

- sweep the polygon from top to bottom
- keep track of all regions that the sweep line intersects.
- When two regions merge, or one is split, add edges to separate the regions into monotonic parts.
- Events? vertices
- Sweep line data structure?  
binary tree that keeps tracks of the order of the edges intersecting the sweep line



13

---

---

---

---

---

---

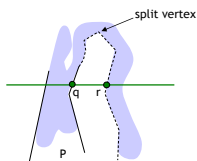
---

---

## Monotonic partitioning

**Lemma:** A polygon is y-monotone if it has no split vertices or merge vertices.

**Proof:** Assume P is not y-monotone. Prove that P has a split or merge vertex.



On the walk from q to r there must be some highest (or lowest) point. This point must be a split (or merge) vertex.

14

---

---

---

---

---

---

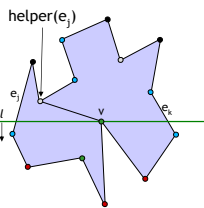
---

---

## Monotonic partitioning: Algorithm

**Goal:** add diagonals from each split vertex to a vertex above it. Which one?

A vertex close to it?



Consider a split vertex v. Let  $e_j$  ( $e_k$ ) be the edge immediately to the left (right) of v along the sweep line.

We can always connect v to the lowest vertex above v and in between  $e_j$  and  $e_k$ .

If there is no such vertex then we can connect v to either the top end vertex of  $e_j$  or  $e_k$ .

Formally:  $helper(e_j)$  is lowest vertex above l, s.t. horizontal segment from that vertex to  $e_j$  is in P

15

---

---

---

---

---

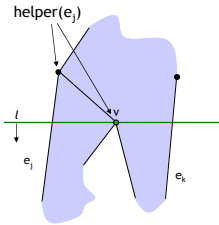
---

---

---

### Removal of split and merge vertices

At a split vertex  $v$ : add diagonal from  $v$  to the helper of the edge to the left



- 1 Find edge  $e_j$  directly left of  $v$
- 2 Insert diagonal between helper( $e_j$ ) and  $v$
- 3 helper( $e_j$ ) :=  $v$
- 4 Insert new edges

---

---

---

---

---

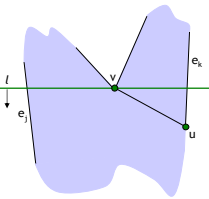
---

---

---

### Removal of split and merge vertices

For a merge vertex  $v$ : wait for the next update of the helper of the edge to the left



- 1 Delete edges
- 2 Search edge  $e_j$  directly left of  $v$
- 3 helper( $e_j$ ) =  $v$

"Merge vertices are split vertices in reverse"

Aim is to connect  $v$  to the highest vertex below the sweep line in between  $e_j$  and  $e_k$ .

Whenever we update the helper of  $e_j$  again, We might connect the old and new helpers.

---

---

---

---

---

---

---

---

### Sweep Line Algorithm

**Events:** Vertices of polygon, sorted in decreasing order by y-coordinate. (No new events will be added)

**Sweep line status:** List of edges intersecting sweep line, sorted by x-coordinate. Stored in a balanced binary search tree.

**Event processing of vertex  $v$ :** ?

---

---

---

---

---

---

---

---

### Sweep Line Algorithm

Event processing of vertex  $v$ :

1. **Split vertex**
2. **Merge vertex**
3. **Start vertex:**
  - Insert the two edges in the binary search tree.
  - Set helper of left edge to  $v$ .
4. **End vertex:**
  - Delete both edges from binary search tree.
5. **Left chain vertex:**
  - Replace upper edge with lower edge in BST.
  - Make  $v$  helper of new edge.
6. **Right chain vertex:**
  - Search edge  $e$  directly left of  $v$
  - Make  $v$  new helper of  $e$ .
  - Replace upper edge with lower edge in BST.



At "every" vertex  $v$ :

Whenever we delete an edge  $e$  or update helper( $e$ ), we check if old helper( $e$ ) is a merge vertex, and if so, we add diagonal between the old helper and  $v$ .

---

---

---

---

---

---

---

---

---

---

### Monotone partition: Analysis

**Correctness?** There are no split or merge vertices remaining.

**Time complexity?**

Sort the vertices into an event queue  $Q$ .

We have  $n$  events and in each event we perform:

- One query on  $Q$  (which event to process),
- at most one query in  $T$  (the binary search tree),
- at most one deletion on  $T$ , and
- at most one insertion on  $T$ .

Each update operation can be performed in  $O(\log n)$  time

**Total time:**  $O(n \log n)$

---

---

---

---

---

---

---

---

---

---

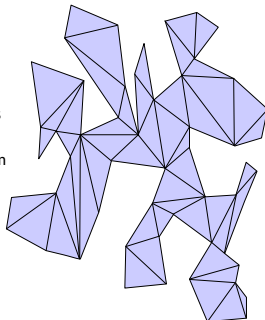
### Triangulate simple polygon

**Main idea:**

1. Partition  $P$  into  $y$ -monotone polygons  
Time  $O(n \log n)$
2. Triangulate each  $y$ -monotone polygon  
Time  $O(n_i)$

**Result:**

A simple polygon can be triangulated in  $O(n \log n)$  time.




---

---

---

---

---

---

---

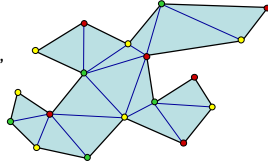
---

---

---

### Previously

- Every simple polygon with  $n$  vertices can be decomposed into  $n-2$  triangles.
- Every triangulated simple polygon can be 3-coloured.
- Every simple polygon can be “guarded” by  $n/3$  guards, and  $n/3$  guards is sometimes necessary.
- To find a guard set our algorithm requires a triangulation.  
Earlier:  $O(n^2)$  time  
Today:  $O(n \log n)$



---

---

---

---

---

---

---

---