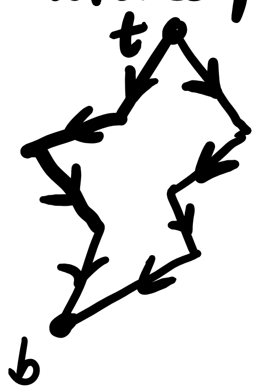
lecture 5

Last time,
monotone polygons :



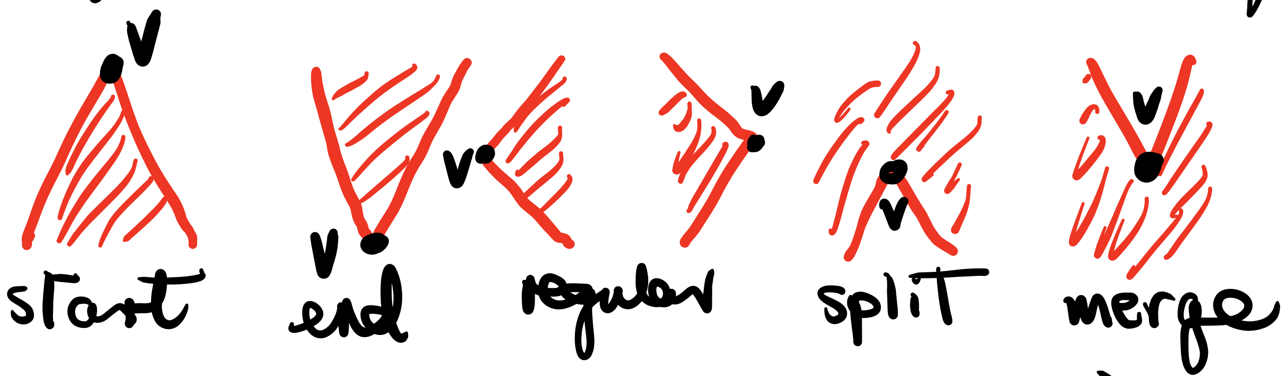
both paths from top to bottom are decreasing wrt. lex. ordering
$a > b \iff a_y > b_y$ or $(a_y = b_y \,\&\, a_x < b_x.)$

Algorithm : Triangulate simple polygon :

① Divide it into y-monotone parts
② Triangulate monotone polygon.

- Last time, did ② time $O(\log n)$.
- This week, we do ① in time $O(n \log n)$.

Types of vertices vs monotonicity



Start : v > p, q (adjacent vertices) & has polygon below.

End : v < p, q & has polygon above.

Reg : p < v < q or q < v < p.

Split : v > p, q & polygon above

Merge : v < p, q & polygon below.

- Recall P is y-monotone (monotone wrt y axis) if each horizontal line intersects P in 1 connected component - ϕ, a pt or a segment.

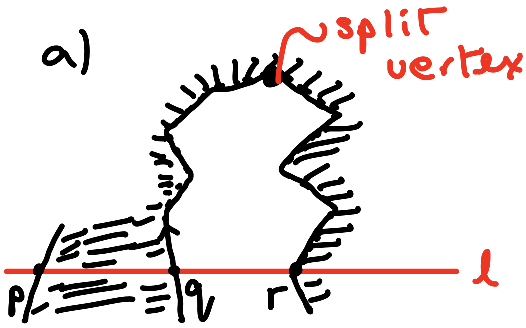
Theorem

A simple polygon is y-monotone $\Leftrightarrow$ it contains no split or merge vertices.

Proof

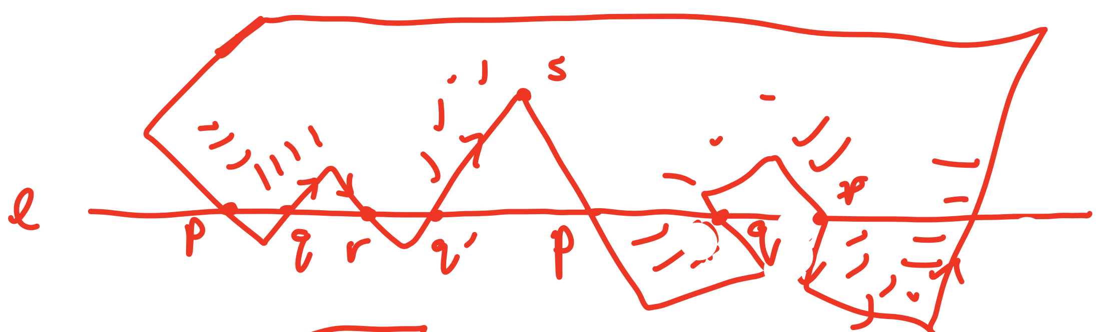
- If P contains a split vertex  ℓ the line ℓ splits it into 2 components $\Rightarrow$ not y-monotone. The merge vertex case is similar.
- Conversely, suppose P is not y-monotone, so there is horizontal line ℓ which intersects P in more than one connected component.
- Can assume leftmost component of $P \cap \ell$ is a segment, not a point (else, move ℓ slightly vertically).
- Let p be left pt & q right point of segment
- Starting at q, follow boundary of P so P lies to left of boundary.
- Then at a point r, boundary of P intersects ℓ again.
- Two cases: a) $p \neq r$ & b) $p = r$.

a)

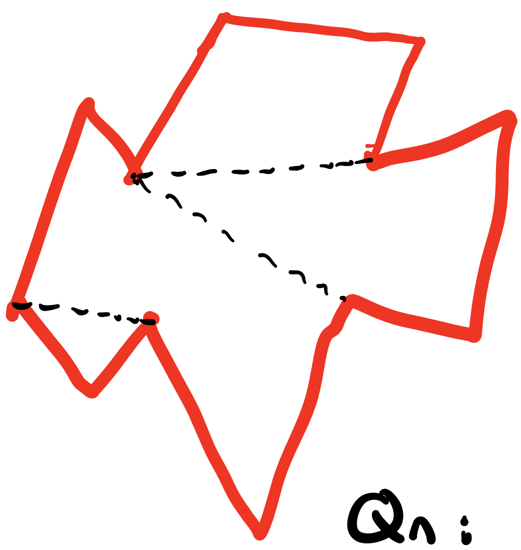


b)

- In case a) highest vertex between q,r is split.
- In case b), follow boundary from q in opposite direction & let r' be intersection point.
- The lowest vertex between q & r' is then a merge vertex. □



Given the above, we can break simple polygons into y-monotone ones by removing split & merge vertices.



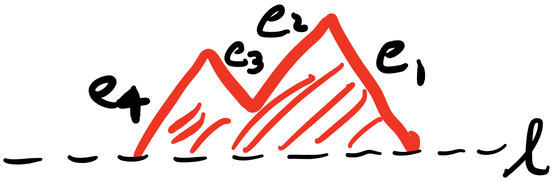
Idea: at merge vertex, draw a line downwards to a vertex

- At split vertex, draw a line upwards to a vertex.

Qn: To which vertices, do we draw lines?

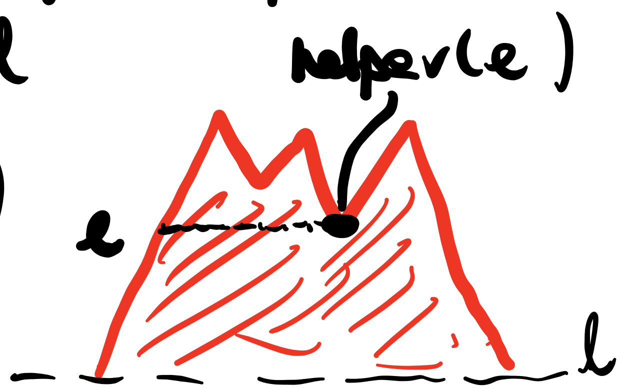
Naturally, we use a sweep-line algorithm from top to bottom.

- Polygon stored in a DCEL.
- Event queue Q (actually a bal. bin. Tree) stores vertices of polygon in lex order.
- Also bal. bin. tree T which stores edges intersecting sweep-line & having the polygon to their right.



At ℓ, $T = \{e_4, e_2\}$

- Also, with each edge e in T we store a vertex $p = \text{helper}(e)$:



- helper(e) lies above ℓ
- horizontal segment between e & helper(e) belongs to P.
- helper(e) is lex. least vertex with these properties.

- It may be the case that helper(e) is its upper endpoint.

Overview of algorithm

When sweepline passes a vertex, we do some of:

- connect a vertex with helper of an edge in DCEL;
- add edges & their helpers into T;
- remove edges & their helpers from T;
- change helpers of some edges in T.

Also, we use anticlockwise enumeration of vertices & edges

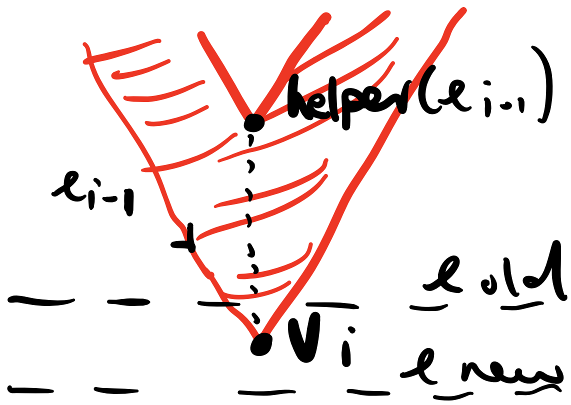
beginning from the top (calculate using DCEL)

Cases:

Start   Passing start vertex v_i.

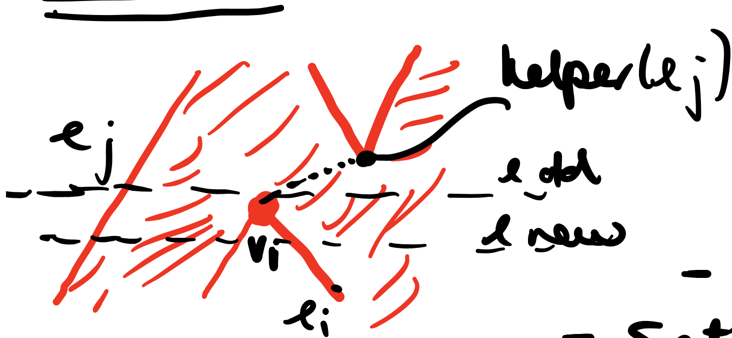
- Add e_i to T
- Set helper(e_i) = v_i

End



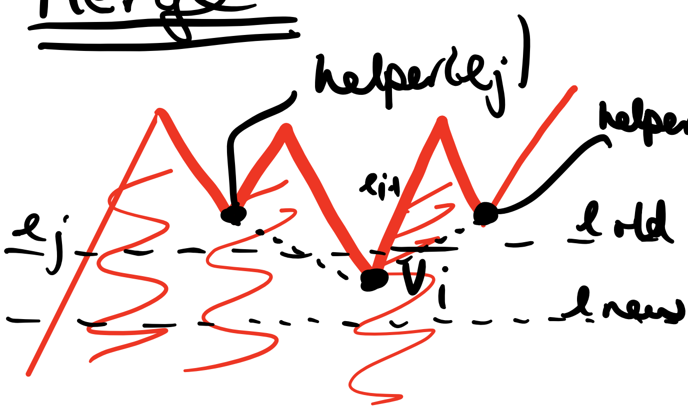
- If helper(e_{i-1}) is merge, add edge from v_i to it in DCEL $\mathcal{D}$.
- Remove e_{i-1} from $\mathcal{T}$.

Split



- Search $\mathcal{T}$ for closest edge e_j to left of v_i
- Add edge from v_i to helper(e_j).
- Add e_i to $\mathcal{T}$.
- Set helper(e_i) = v_i & helper(e_j) = v_i.

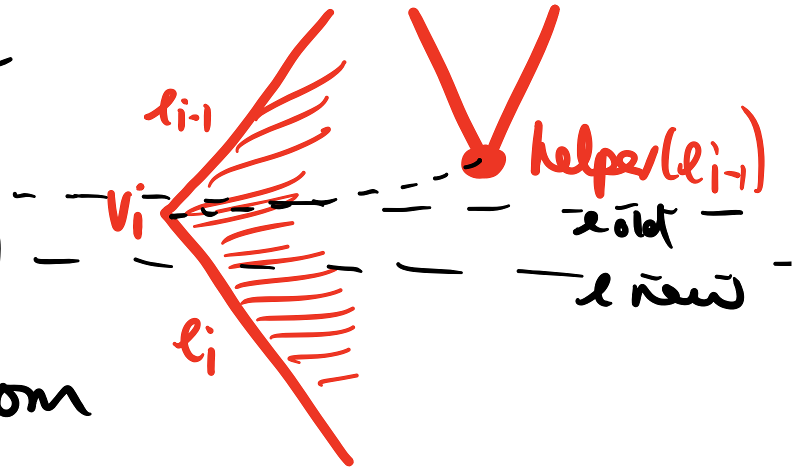
Merge



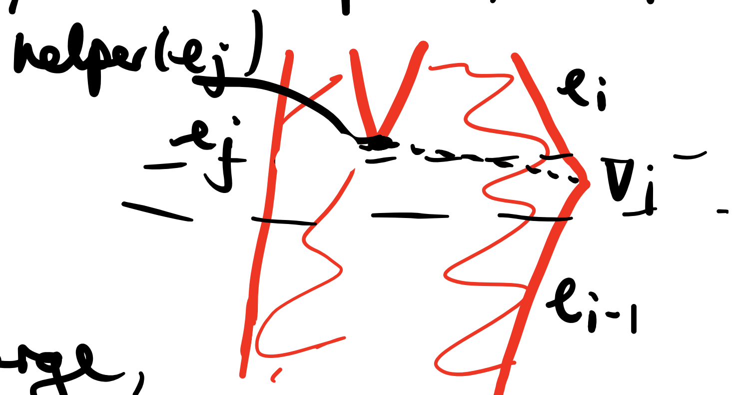
- If helper(e_{i-1}) is merge, add edge to v_i in $\mathcal{D}$
- If helper(e_j) is merge, add edge to v_i in $\mathcal{D}$.
- Set helper(e_j) = v_i)

Regular vertex

- If P lies to right of v_i,
then if helper(e_{i-1}) is a merge vertex, we draw a line from it to v_i.



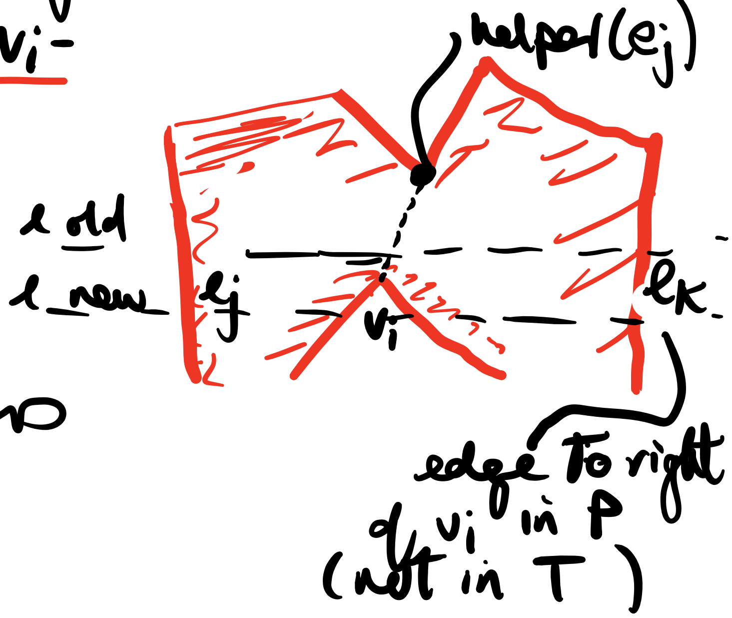
- Delete e_{i-1} from T.
- Insert e_i into T, with helper(e_i) = v_i)



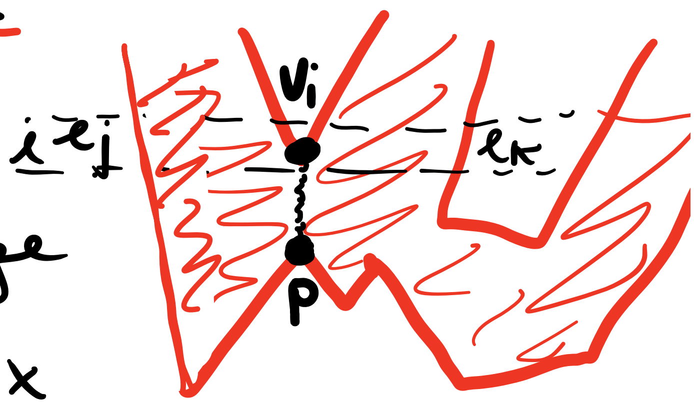
- Otherwise, P lies to left of v_i.
- Search T for edge e_j to left of v_i.
- If helper(e_j) is merge, draw line from it to v_i.
- Set helper(e_j) = v_i.

Why does the algorithm work? (Sketch)

- Why are split & merge vertices removed?
- Consider split vertex v_i - is connected to helper(e_j), the lowest vertex between its left & right neighbours e_j & e_k



- Consider merge vertex v_i, left right neighbours e_j & e_k.
- At vertex v_i, change helper(e_j) to v_i.
- Then at max vertex p between e_j & e_k & below the sweep-line, we add an edge to v_i.



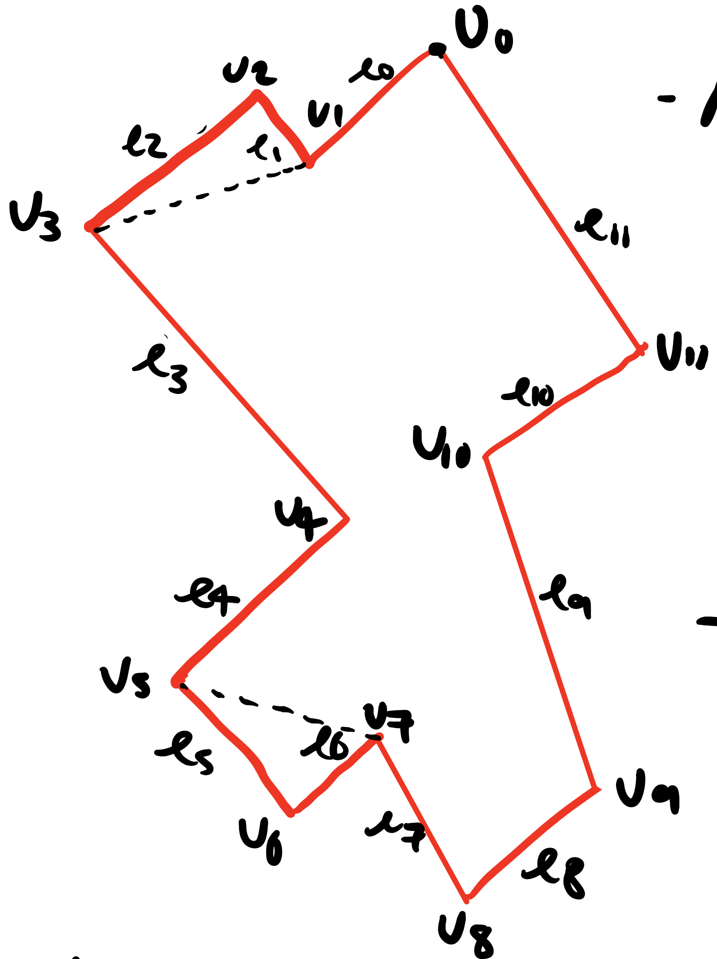
- In this way both split & merge vertices are removed.

Complexity:

- $O(n\log n)$ order vertices into Q.
- $O(n)$ to calculate anticlockwise order.
- Each event involves searching, rebalancing tree - time $O(\log n)$ - plus constant time operations: updating helpers, adding edges to DCEL.
- So time $O(n\log n)$ to handle these n events.
- Therefore complexity is

$$O(n\log n) + O(n) + O(n\log n) = O(n\log n).$$

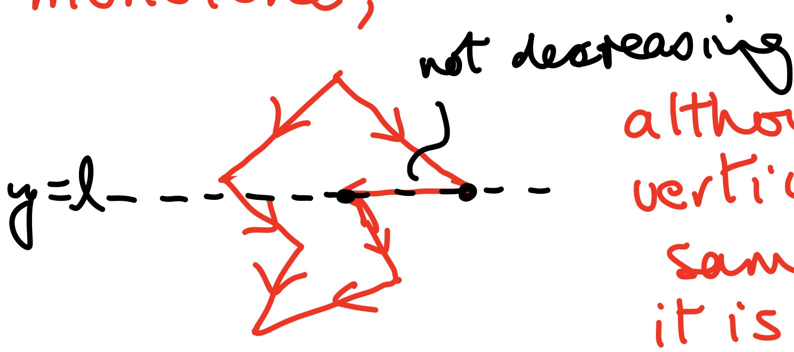
Example



- At v_0, add e_0 to T & $h(e_0) = v_0$.
- At v_2, also start, add e_2 to T, set $h(e_2) = v_2$.
- At v_1, $h(e_0) = v_0$ not merge, $h(e_2) = v_2$ not merge. Do nothing. Change $h(e_2) = v_1$.
- At v_3, reg. vertex, $h(e_2) = v_1$ merge. Add line v_1 to v_3. Remove e_2. Add e_3.
- At v_{11}, $h(e_3) = v_3$ so do nothing.
- At v_{10}, change $h(e_3) = v_{10}$.
- At v_4, remove e_3 & add e_4.
- At v_5, rem. e_4 & add e_5.
- At v_7 split, $h(e_5) = v_5$ so add line v_5 to v_7. Ch $h(e_5) = h(v_7)$. Add e_7.
- At v_9, do nothing.
- At v_6, remove e_5. At v_8, remove e_7. $\square$

Note: here we have described an alg. for dividing a simple polygon into y-monotone parts; last time, we described alg. for triangulating monotone polygon.

Not every y-monotone polygon is monotone,



although if no two vertices have same x-coordinate it is monotone.

So the algorithm for triangulating a monotone polygon will work if no two points have same x-coord.

To handle the degenerate case, one can make a small rotation to the polygon - we will not treat this here.