(simple) polygons can be triangulated (last time)

easy: triangulate convex polygons

connect a vertex with all other vertices

Generalization: monotone polygons

— the intersection with each horizontal line $y = c$ is connected

Related to horizontal levels of vertices

Use ordering of points

$p > q \iff p_y > q_y$ or $(p_y = q_y \;\&\; p_x < q_x)$

More technical def: P polygon

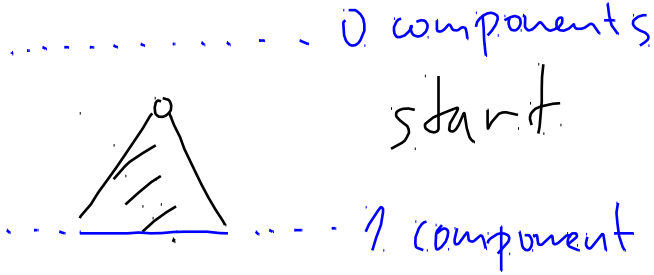
Let u be the minimal vertex in this order (up)
d maximal (down)

Then there are two paths from u to d

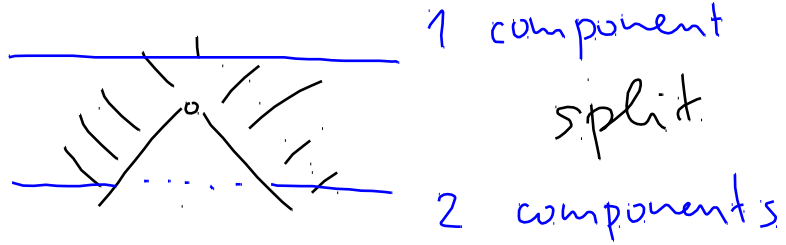
left path, right path

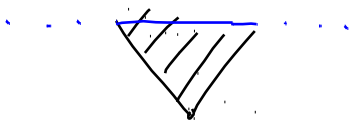
P is monotone if the vertices on the paths form a monotone (decreasing) sequence

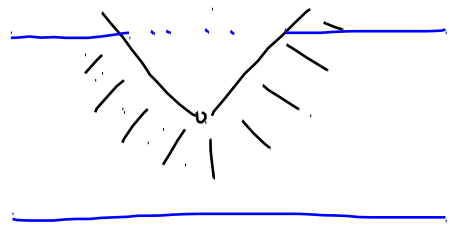
Two tasks:
- divide a polygon into monotone ones
- triangulate monotone polygons



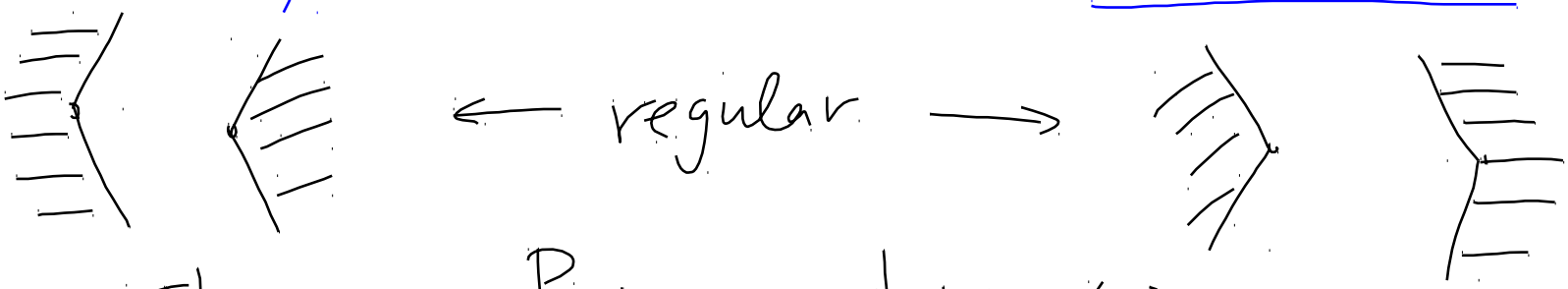
vs







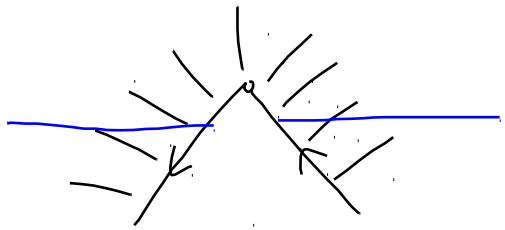
$\emptyset$



Theorem. P is monotone $\Leftrightarrow$ $\Leftrightarrow$ it does not contain vertices of type split & merge.

Proof: $\Rightarrow$ let P be monotone.

& assume it contains a split vertex



right path

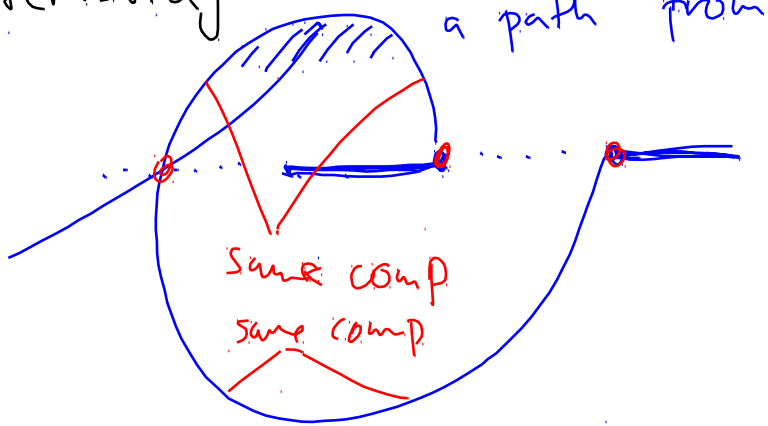
$\Rightarrow$ this level set disconnected

or one of the edges increasing

& other decreasing

$\Leftarrow$ more interesting: assume that P is not monotone

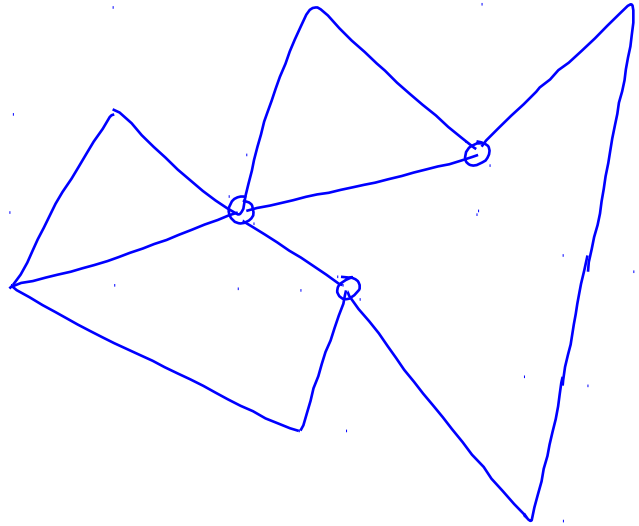
a path from edges



start or split ✓

a level set that is disconnected

Dividing a polygon into monotone ones is the same as removing split & merge vertices.



idea: connect split & merge vertices to other and divide polygon this way into mon. ones.

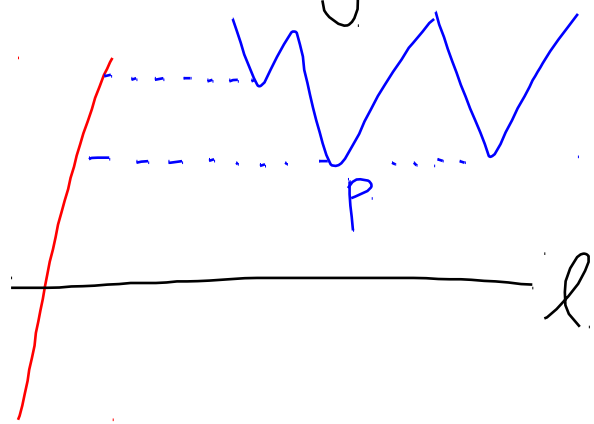
Sweep line method

store: the edges that ← stored in a binary balanced tree

- intersect the line l and
- for which P sits to the right

With each such edge e, we store a vertex $helper(e) = p$ s.t.

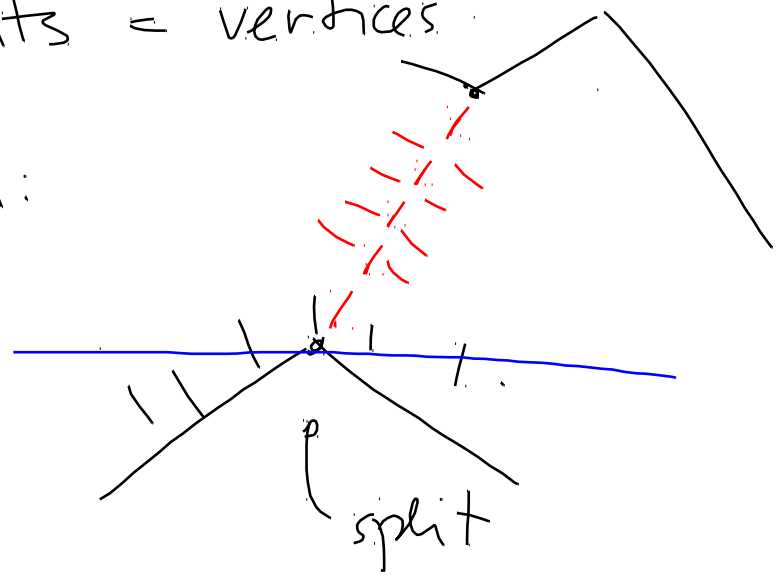
- p lies above ℓ
- the horizontal line from p intersects e with the segment between p and the intersection with e inside P
- p has min y among such vertices



$\rightsquigarrow$ $helper(e)$ unique

Events = vertices

idea:

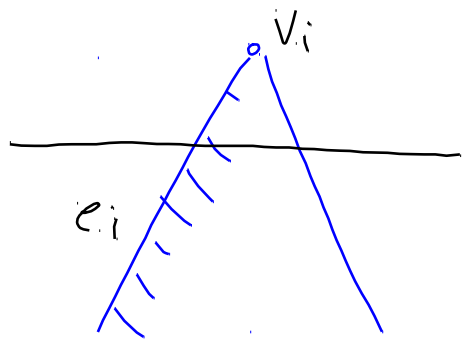


the dual case merge is more complicated

more detail:

depending on the type of the vertex:

start

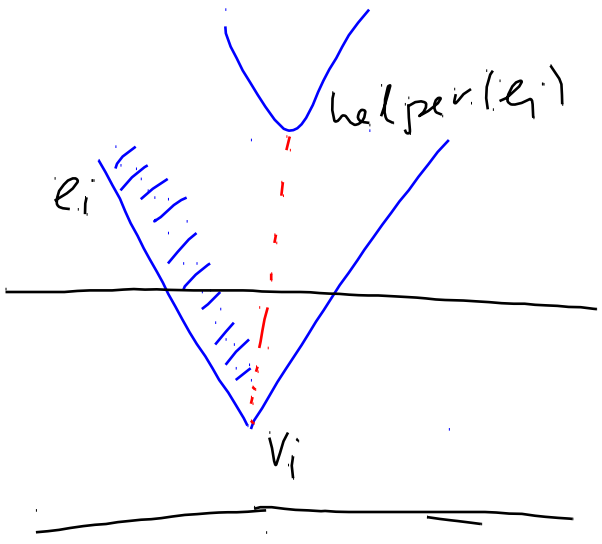


ℓ old

ℓ new

- add e_i to the list of edges
- set helper(e_i) = v_i

end



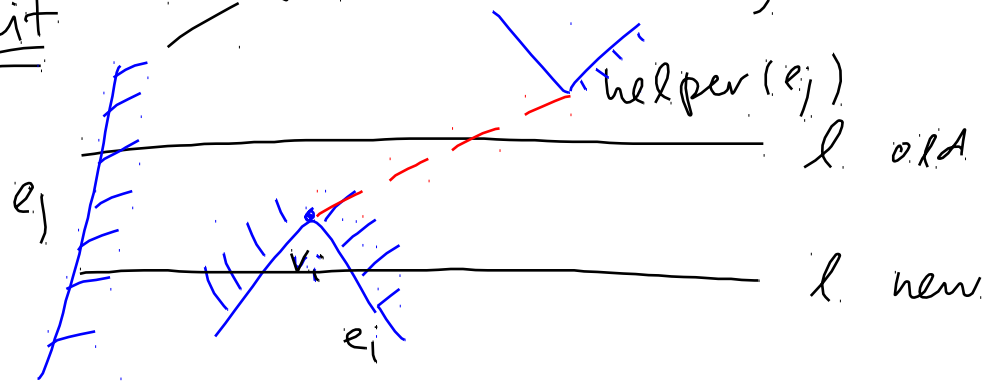
— was not treated yet !

ℓ old

ℓ new

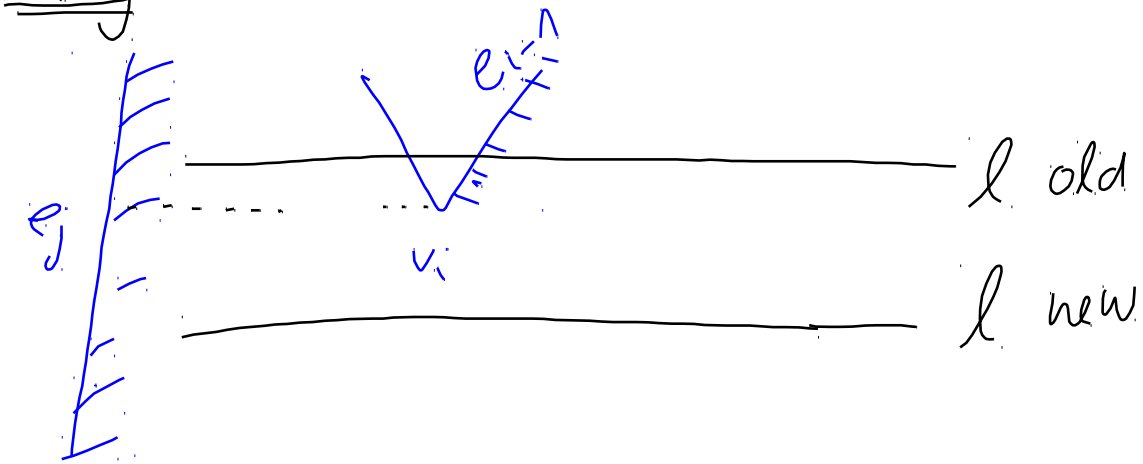
- if helper(e_i) is merge, connect to v_i
- remove e_i from the list

Split — the closest edge to the left of v_i — found from the list



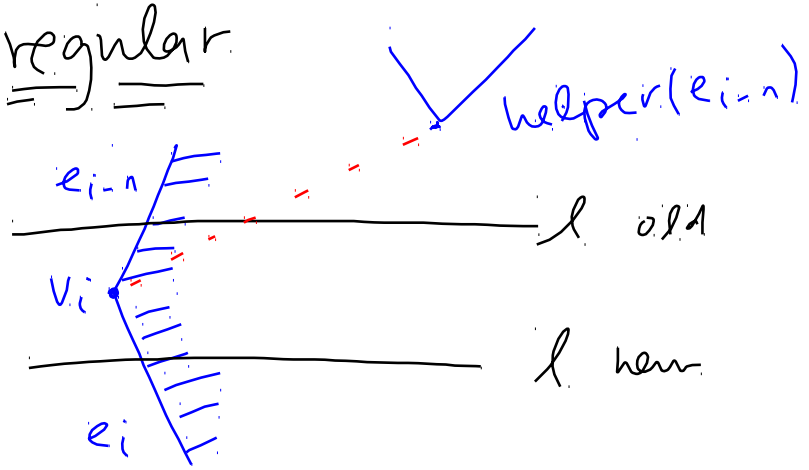
- connect v_i to helper(e_j)
- add e_i to the list
- set helper(e_j) := v_i
- set helper(e_i) := v_i

merge



- for both helper(e_j), helper(e_{i-1})
  - if they are merge vertices, connect them with v_i
- set helper(e_j) := v_i
- remove e_{i-1}

regular

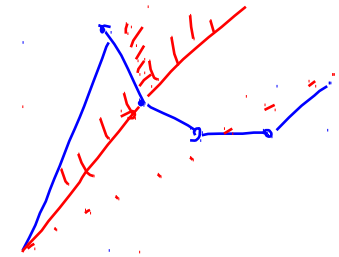


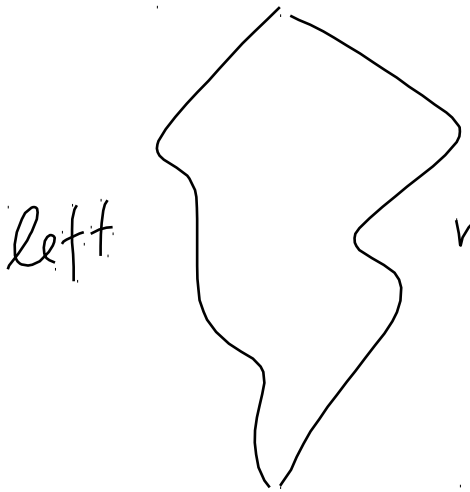
- if helper(e_{i-1}) is merge, connect it to v_i
- remove e_{i-1} from the list
- and add e_i to the list
- helper(e_i) := v_i

and similarly for v_i on the right path

Each step ... $O(\log n)$ time $\Rightarrow$ $O(n \log n)$.

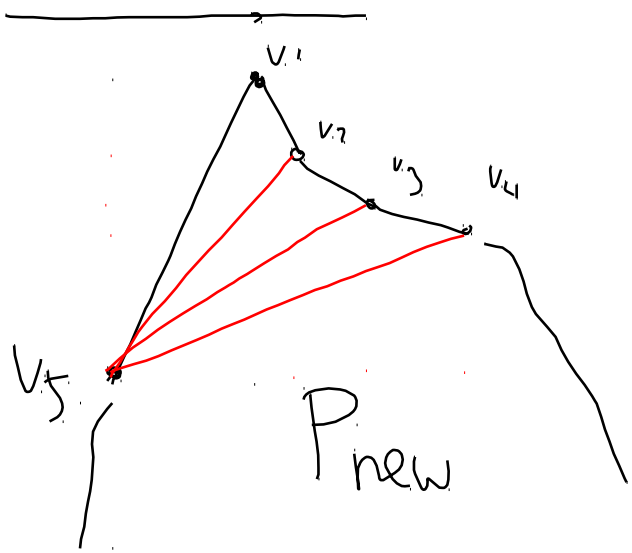
Triangulation of monotone polygons





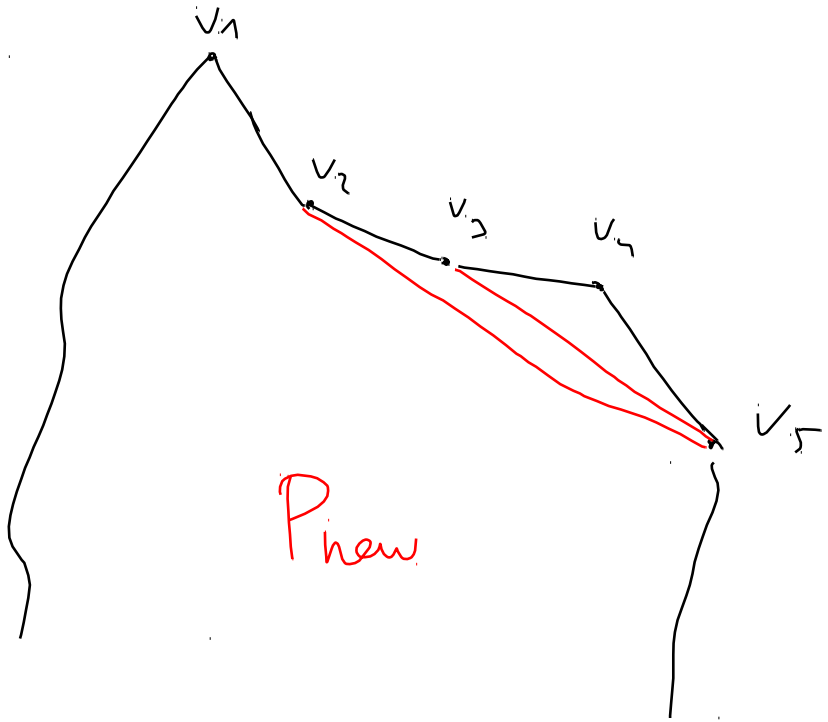
... monotone paths

→ sort vertices according to the lexicogr. ordrn



← new vertex v_5 on the opposite path.

P_{new} has v_4, v_5 on the left path, rest is not processed yet.



← v_5 on the same side

Phew has vertices v_1, v_2, v_5 on the right path, rest not processed yet.

Technically: keep a list of vertices in a heap and, at each point, pop out a number of them and push the new vertex