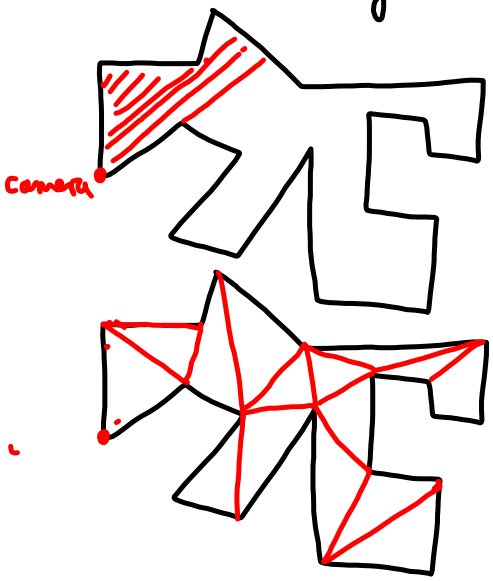
Polygon triangulation

Motivation : Art Gallery problem

Art Gallery ~ simple polygon (no holes)



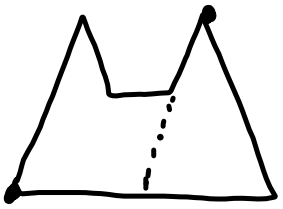
How many cameras does it take to guard an art gallery?

Upper bound : no of triangles required to divide up the polygon (Triangulation)

Theorem : Any simple polygon can be triangulated (if it has n vertices we require n-2 triangles).

- Upper bound for no of cameras is $n-2$.
- In fact, $\lfloor n/3 \rfloor$ cameras suffices
  also uses triangulations + 3-colorings.

$\lfloor n/3 \rfloor$ sometimes required



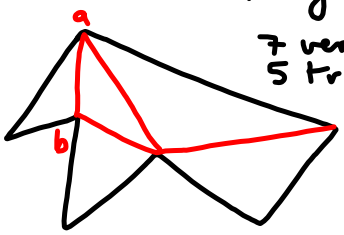
6 vertices

$\lfloor 6/3 \rfloor = 2$

Today : polygon triangulation algorithm

Theorem: Each simple polygon can be triangulated. If it has n vertices, we require precisely n-2 triangulations.

Eg



7 vertices
5 triangles

Proof: $n=3$

obvious

By a diagonal we mean a straight line segment ab, whose endpoints are vertices & which otherwise belongs to interior of polygon.
(Eg ab is a diagonal in example).

- Any diagonal ab splits polygon $P = A \cup_{ab} B$ where $m, k < n$ & $m+k = n+2$

n vertices — m vert — k vert

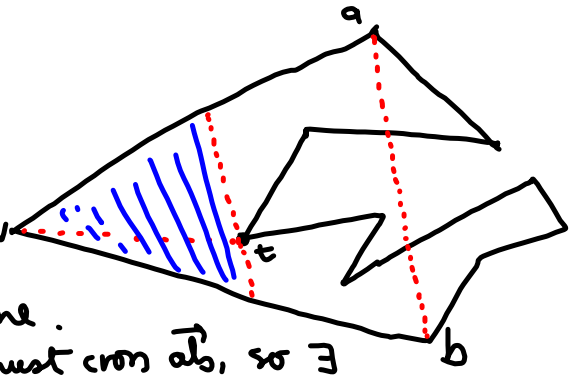
Also formula follows:
By ind, have triang of A, B in m-2, k-2 triangles, so triang. of P in $(m-2)+(k-2) = m+k-4 = n-2$ triangles.

Triangulations of A & B can be combined - so result follows by induction if we can prove existence of diagonal.

Existence of diagonal:

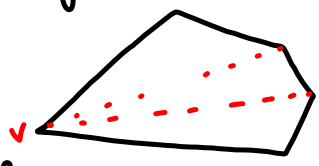
- Let v be smallest vertex on P in lex order (left to right, bottom to top)
- let a, b be the 2 vertices conn to v.
- If $\overrightarrow{ab}$ is a diagonal, we are done.
- If $\overrightarrow{ab}$ not a diagonal, an edge must cross $\overrightarrow{ab}$, so $\exists$ vertices of P inside $\triangle vab$.
- let t be furthest such vertex from $\overrightarrow{ab}$.
- If $\overrightarrow{vt}$ did not lie in interior, an edge of P must cross $\overrightarrow{vt}$, & one of endpoints would lie in blue region, s.t. closer to v than t. But this contradicts defn of t.

Thus $\overrightarrow{vt}$ is a diagonal.



Goal: Find alg. with complexity $O(n \log n)$.

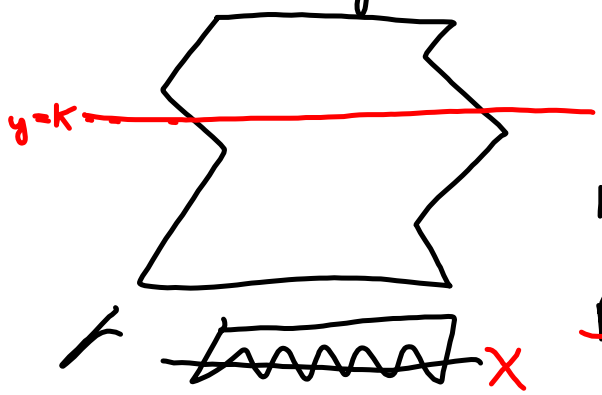
- Convex polygon:



→ easy to triangulate. Draw vertex to each other vertex.

- Weaker notion (still easy to triangulate) called a monotone polygon.

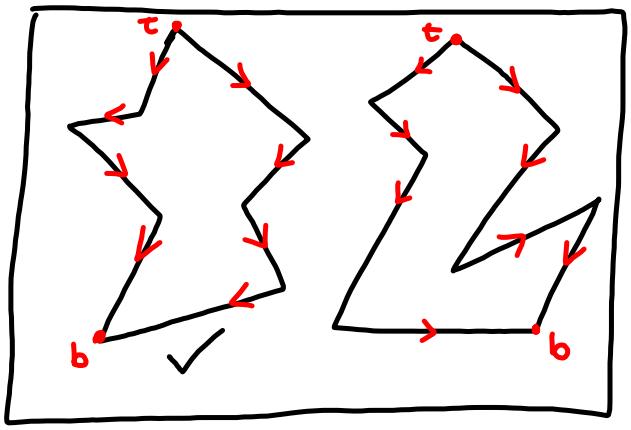
Monotone polygon wrt to axis y:



any horizontal line $y=k$ intersects polygon in line segment, a point or empty set.

However, we work with slightly stronger notion of monotone polygon. More computationally effective.

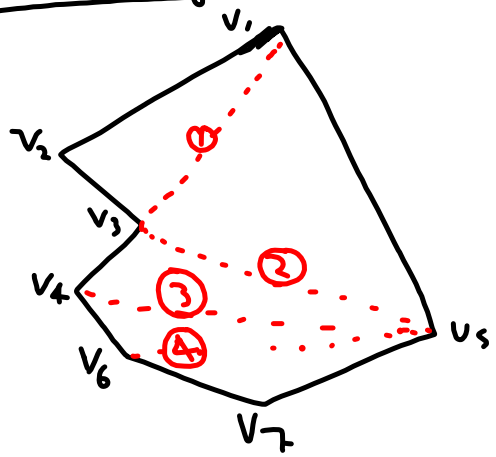
- Monotone polygon:
  Consider lex ordering
  $a > b \Leftrightarrow a_y > b_y$ or
  $(a_y = b_y \ \& \ a_x < b_x)$.
- Determines two paths from t (top) to bottom (b).
- Polygon is monotone if both paths are decreasing (wrt lex. order)



Algorithm: ① divide simple polygon into monotone pieces.
② Triangulate monotone polygon.

This week do ②, next time ①.

② Triangulate monotone polygon



Idea · use lex ord. on vertices.
Draw diagonals to all possible preceding vertices & break off triangles.

- Formally store monotone polygon in DCEL D
- Output: DCEL with diagonals added, capturing triangulated polygon.

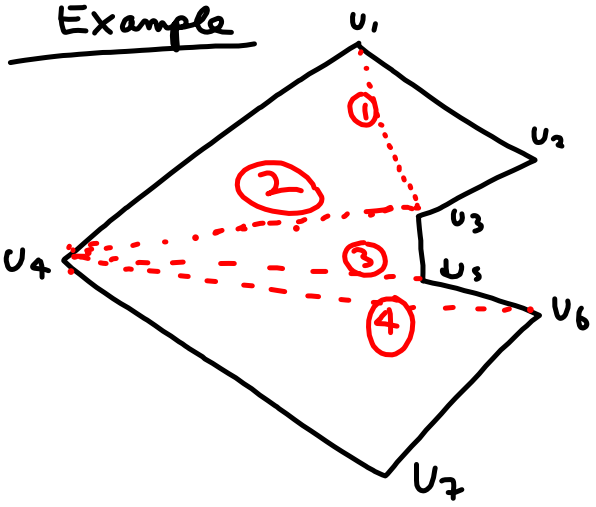
Start of alg:
- calculate left & right paths from top to bottom.
- merge two paths into lex. ordered list $v_1, \ldots, v_n$.

- Initialise empty stack S.
- Push v_1 & v_2 onto S – so $S = (v_2, v_1)$.

For $j = 3$ to $n-1$:

- If v_j & vertex on top of S are on diff. paths:
  - pop all all vertices from S
  - add diagonal in D to each popped except the last one popped.
  - push v_{j-1}, v_j onto S.
- Otherwise, pop top vertex on S
  - pop remaining vertices as long as diagonals to v_j lie inside P.
  - add these diagonals to D.
  - Push last popped vertex onto S.
  - Push v_j onto S.
- At v_n, add diagonals to all vertices in S except first and last ones.

Example



$S = (v_2, v_1)$

At v_3,
pop v_2 so (v_1)
pop v_1 so $()$ &
add v_1v_3 to D
Then $S = (v_3, v_1)$

At v_4, set $S = ()$
$S = (v_4, v_3)$, add v_3v_4 to D

At v_5, add v_5v_4 to D.
$S = (v_5, v_4)$

At v_6, add v_6v_4 to D.
$S = (v_6, v_5)$

At v_7, do nothing.

Complexity

- calc. top, bottom vert $O(n)$
- paths top to bottom $O(n)$
- merge paths $O(n)$
- Loop executes $n-3$ times.
  At each execution push at most 2 vertices.
  So at most $2n-6+2 = 2n-4$ vertices pushed on.
  No of popped $\leq$ no of pushed
  So pops + pushes complexity $O(n)$

$\Rightarrow$ Total complexity $O(n)$.