Discrete and Computational Geometry
Cesar Ceballos

Last time: Art Gallery Theorem

Today: Algorithmic Solution.

Recall

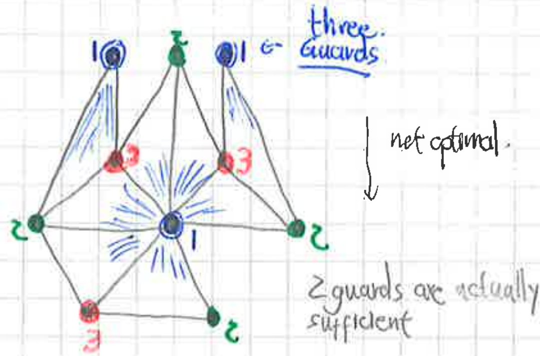
Theorem (Art Gallery Theorem)

For a polygon with n vertices, $\lfloor \frac{n}{3} \rfloor$ guards are occasionally necessary and always sufficient to observe the entire polygon.

Idea:
- triangulate
- 3-coloring of the vertices

$\overset{\bullet}{c}$—$\overset{\bullet}{c'}$ $\rightarrow$ $c \neq c'$

- place guards at less frequently used color.



Question How do we decide to place $\lfloor \frac{n}{3} \rfloor$ guards efficiently?

Today's result:

Theorem Let P be a polygon with n vertices. A set of positions of $\lfloor \frac{n}{3} \rfloor$ guards to observe the entire polygon can be computed in $O(n \log n)$ time.

Remark Note that $\lfloor \frac{n}{3} \rfloor$ is not necessarily an optimal solution.

The problem of finding the minimum number of guards for a given polygon was shown to be NP-hard by Aggarwal '84, Lee & Lin '86

Book by O'Rourke: Art Gallery Theorems and Algorithms. '87

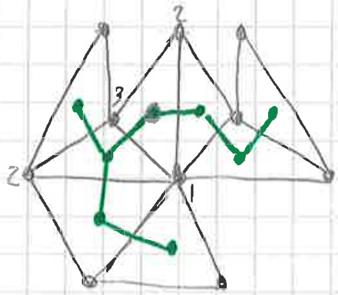
Towards the proof

Step (1): Assume we are already given a triangulation T_P of P.

Question: How fast can we find a 3-coloring of the vertices so that adjacent vertices have different colors?

Answer: linear time $O(n)$.

To argue this, let $G(T_P)$ be the dual graph of the triangulation



$G(T_P)$ has one node for each triangle, and two nodes are connected by an edge when the corresponding triangles share a diagonal.

Note that $G(T_P)$ is a tree:
since any diagonal cuts P into two, removing an edge of $G(T_P)$ splits the graph into two.

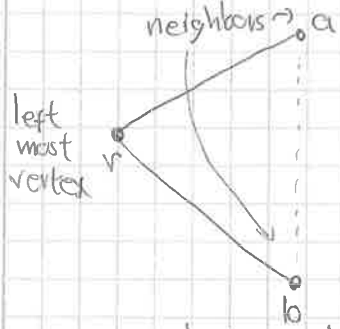
So, we can find a 3-coloring of the vertices of the triangulation using a simple graph traversal (i.e. depth first)

- Start at any vertex of $G(T_P)$ and color the vertices of the corresponding triangles with 3 colors.
- Move to a neighbor and color the new vertex of the corresponding triangle with the color that is still "available"!

This is performed in $O(n)$ time.

Step (2): If we are given a polygon P, how fast can we triangulate it?

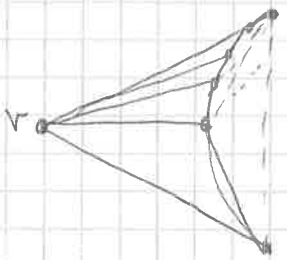
Option 1: Find a diagonal decomposing P into two polygons P_1 and P_2, and repeat the process.



If ab is a diagonal ✓

If not: find the farthest point x from ab inside the triangle vab
⇒ vx is a diagonal
↳ Takes linear time.

If vbx is a triangle then P_1 is a triangle and P_2 has $n-1$ vertices. So, repeating the process takes quadratic time in the worst case.



$O(n^2)$.

Question: Can we do better?

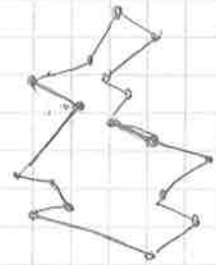
Today: - Consider a simpler case: y-monotone polygons (linear time $O(n)$)
- Sketch the proof of the theorem in the general case ($O(n\log n)$)

- <u>Triangulating a monotone triangle</u>.

We say that a polygon P is <u>y-monotone</u> if the intersection of any horizontal line with P is a connected line segment (or empty)

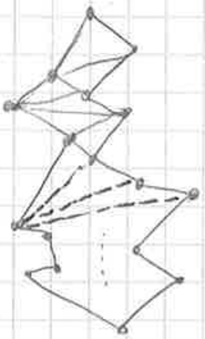
Equivalenty, if we walk from a topmost to a bottommost vertex along the left (or the right) boundary chain, then we always move downwards or horizontaly, never upwards

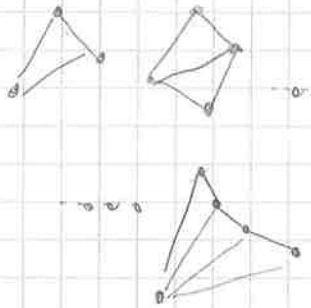
For simplicity, we assume that P is <u>strictly y-monotone</u>, that is, it is y-monotone and does not contain horizontal edges.



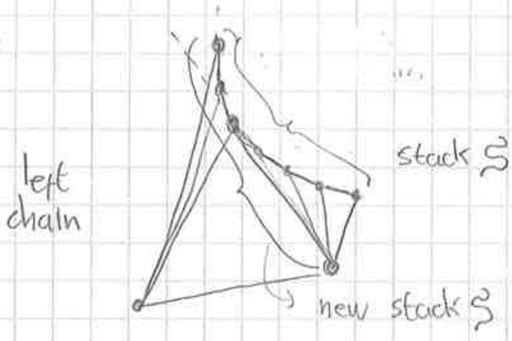
Triangulating a strictly y-monotone polygon is relatively simple:

Idea: triangulate from top to bottom ↓





Another interesting case:



Next time:

<u>Algorithm</u>: Input: strictly y-monotone polygon P stored in a doubly-connected edge list.
Output: A triangulation of P stored in a doubly-connected edge list $\mathcal{D}$.