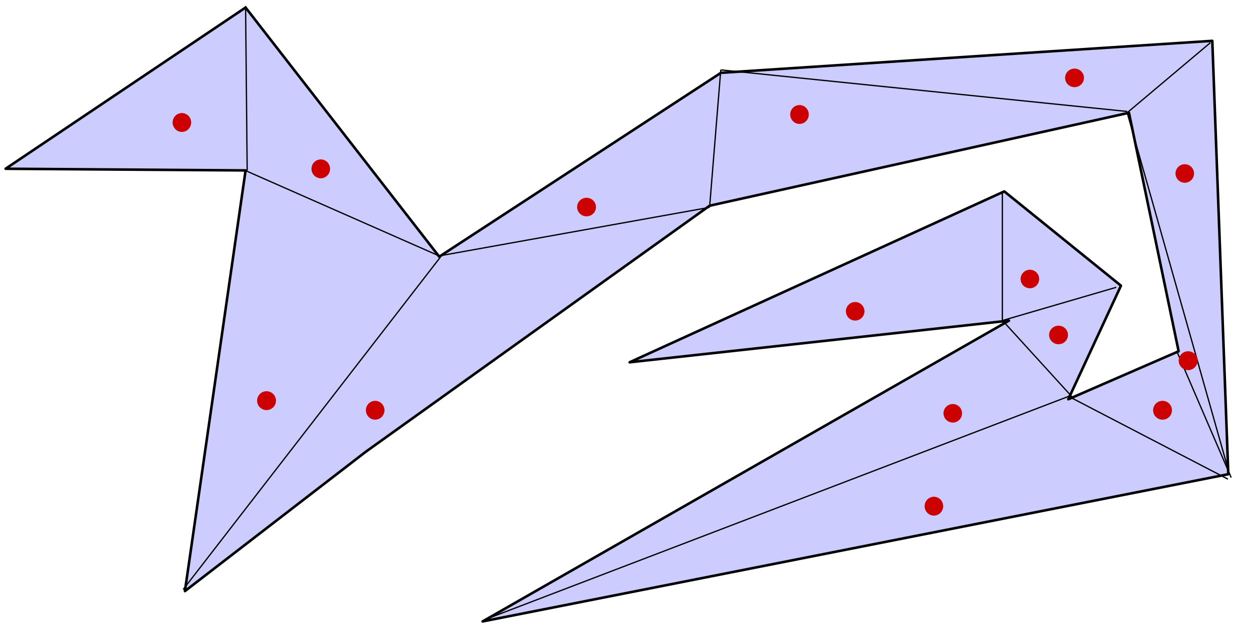
Computational Geometry



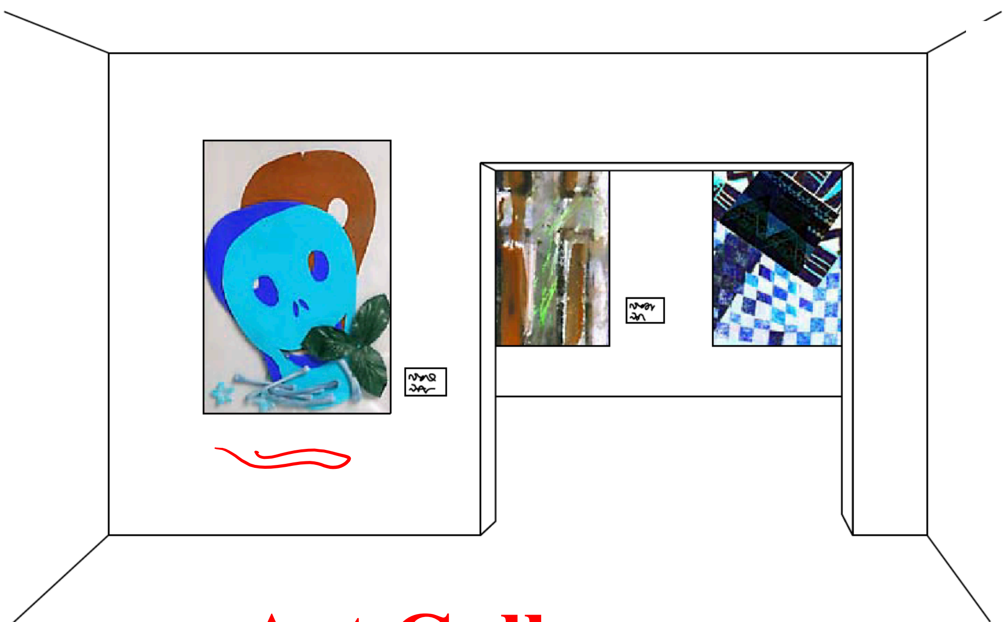
Triangulations and Guarding Art Galleries

Michael T. Goodrich

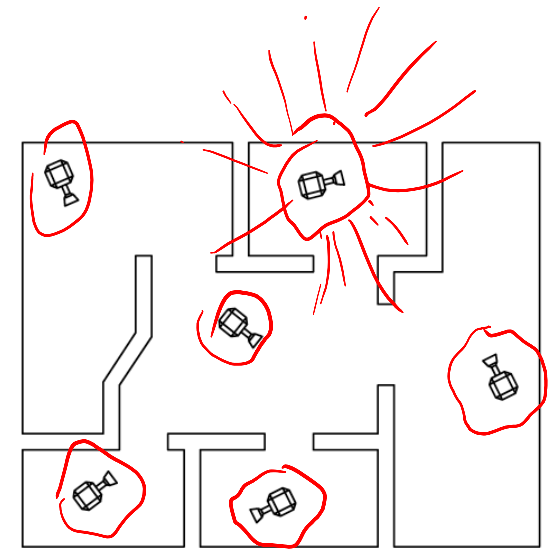
with slides by Carola Wenk, Tulane Univ., and Subhash Suri, UCSB

Guarding an Art Gallery

- **Problem:** Given the floor plan of an art gallery, place (a small number of) cameras/guards such that every point in the art gallery can be seen by some camera.



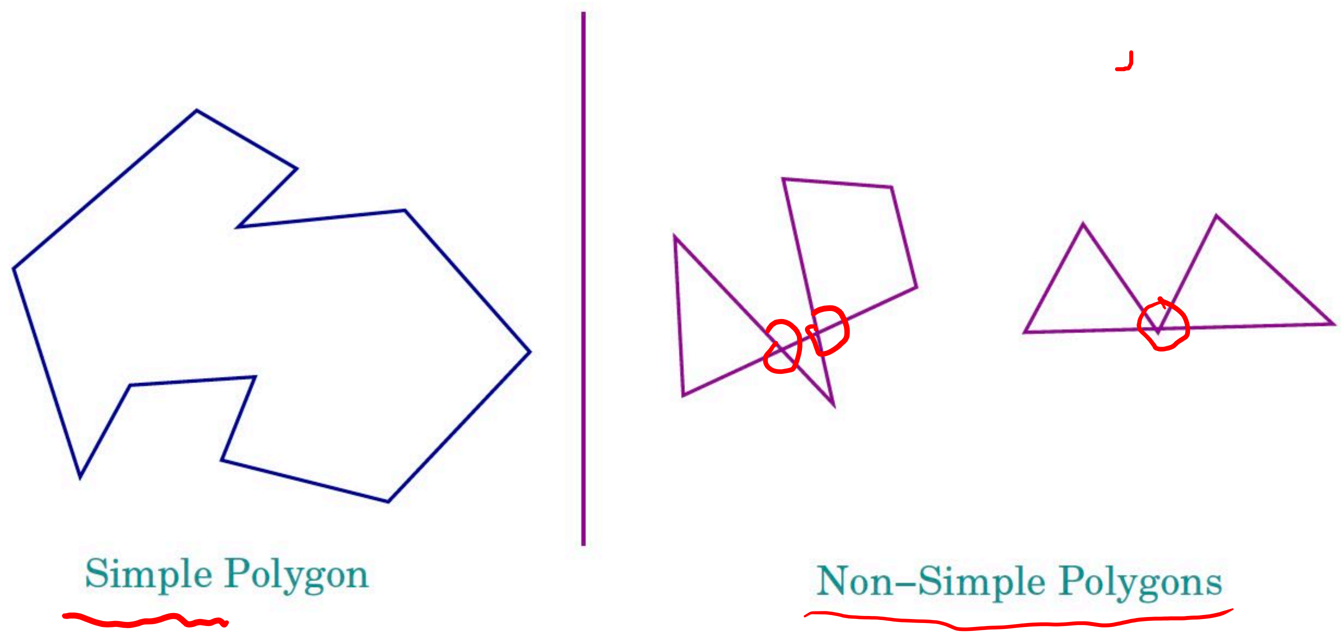
Art Gallery



Floor plan

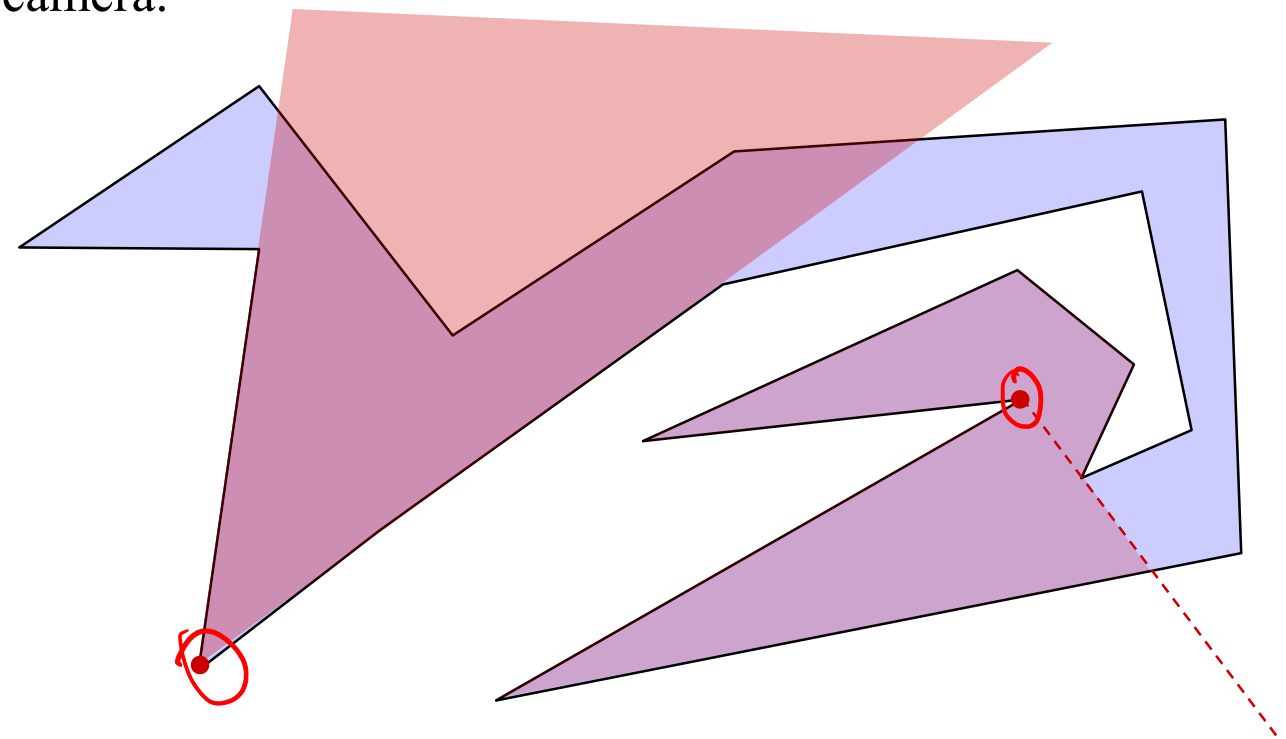
Polygons – same start & end

- A **polygonal curve** is a finite chain of line segments.
- Line segments called **edges**, their endpoints called **vertices**.
- A **simple polygon** is a closed polygonal curve without self-intersection.



Guarding an Art Gallery: Computational Geometry version

- **Problem:** Given the floor plan of an art gallery as a simple polygon *P* in the plane with *n* vertices. Place (a small number of) cameras/guards on vertices of *P* such that every point in *P* can be seen by some camera.

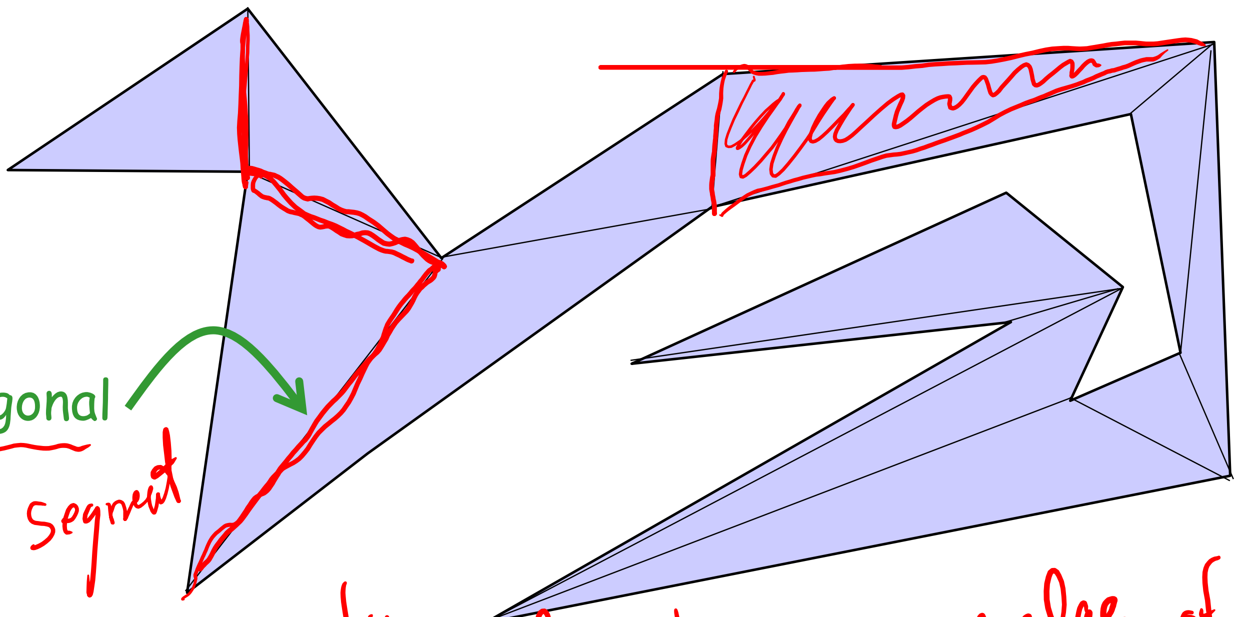


Guarding an Art Gallery

- There are many different variations:
  - Guards on vertices only, or in the interior as well
  - Guard the interior or only the walls
  - Stationary versus moving or rotating guards
- Finding the minimum number of guards is NP-hard (Aggarwal ’84)
- First subtask: Bound the number of guards that are necessary to guard a polygon in the worst case.

Guard Using Triangulations

- Decompose the polygon into shapes that are easier to handle: triangles
- A **triangulation** of a polygon *P* is a decomposition of *P* into triangles whose vertices are vertices of *P*. In other words, a triangulation is a maximal set of non-crossing diagonals.

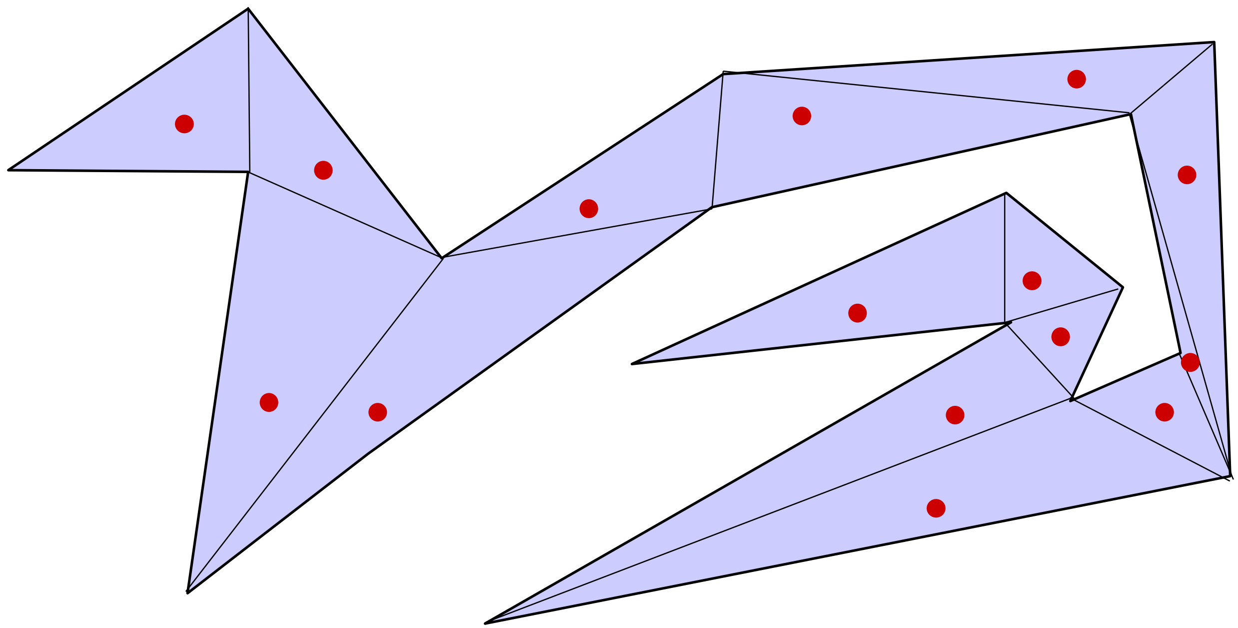


diagonal

~~edge~~ segment between 2 vertices, doesn't cross an edge of P

Guard Using Triangulations

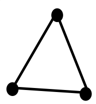
- A polygon can be triangulated in many different ways.
- Guard polygon by putting one camera in each triangle: Since the triangle is convex, its guard will guard the whole triangle.

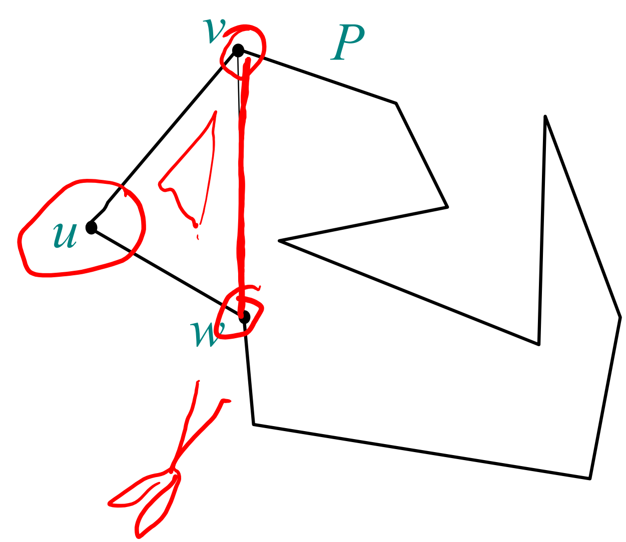


Triangulations of Simple Polygons

Theorem 1: Every simple polygon admits a triangulation, and any triangulation of a simple polygon with n vertices consists of exactly n-2 triangles.

Proof: By induction.

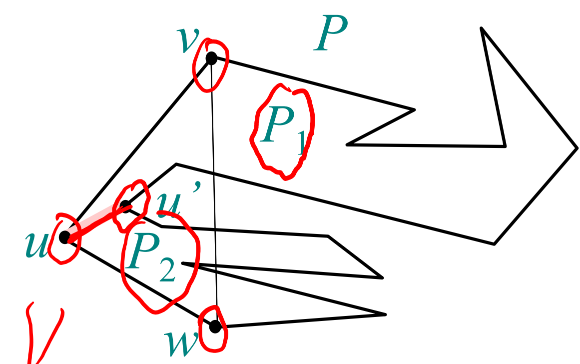
- n=3:  n-2 triangles
- n>3: Let u be leftmost vertex, and v and w adjacent to v. If $\overline{vw}$ does not intersect boundary of P: #triangles = 1 for new triangle + (n-1)-2 for remaining polygon = n-2



Triangulations of Simple Polygons

Theorem 1: Every simple polygon admits a triangulation, and any triangulation of a simple polygon with n vertices consists of exactly n-2 triangles.

If $\overline{vw}$ intersects boundary of P: Let $u' \neq u$ be the the vertex furthest to the left of $\overline{vw}$. Take $\overline{uu'}$ as diagonal, which splits P into P_1 and P_2.



#triangles in P

= #triangles in P_1 + #triangles in P_2

= #vertices in P_1 – 2 + #vertices in P_2 - 2

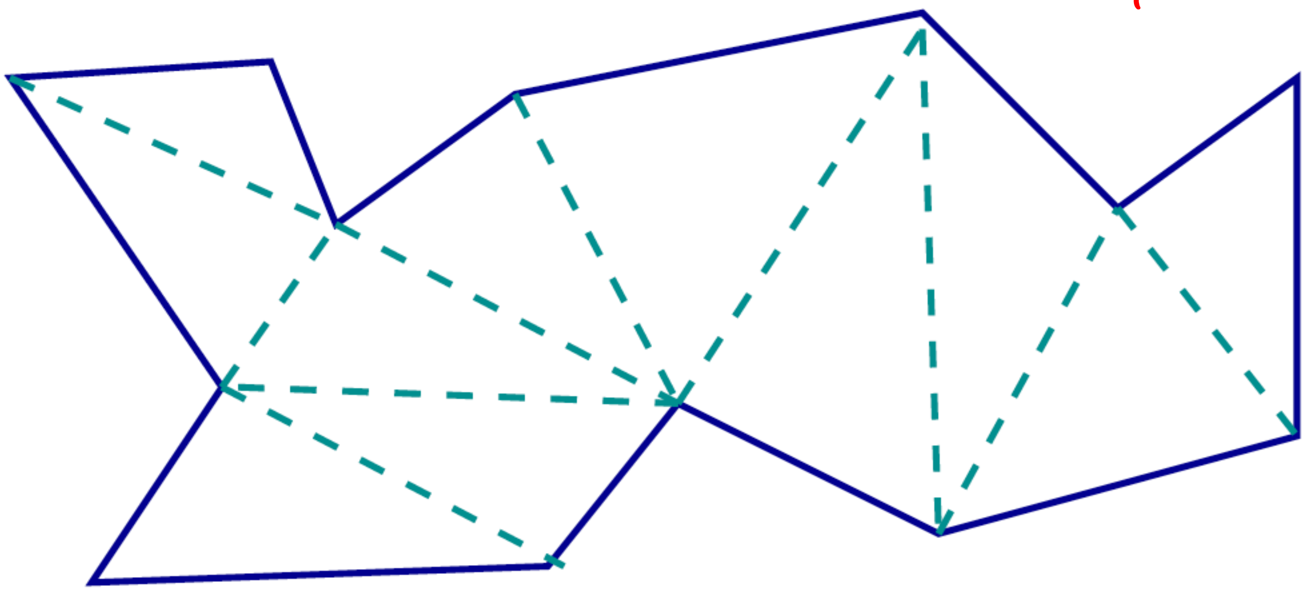
= n + 2 - 4 = n-2 ◻

Example

n vertices
n-2 guards

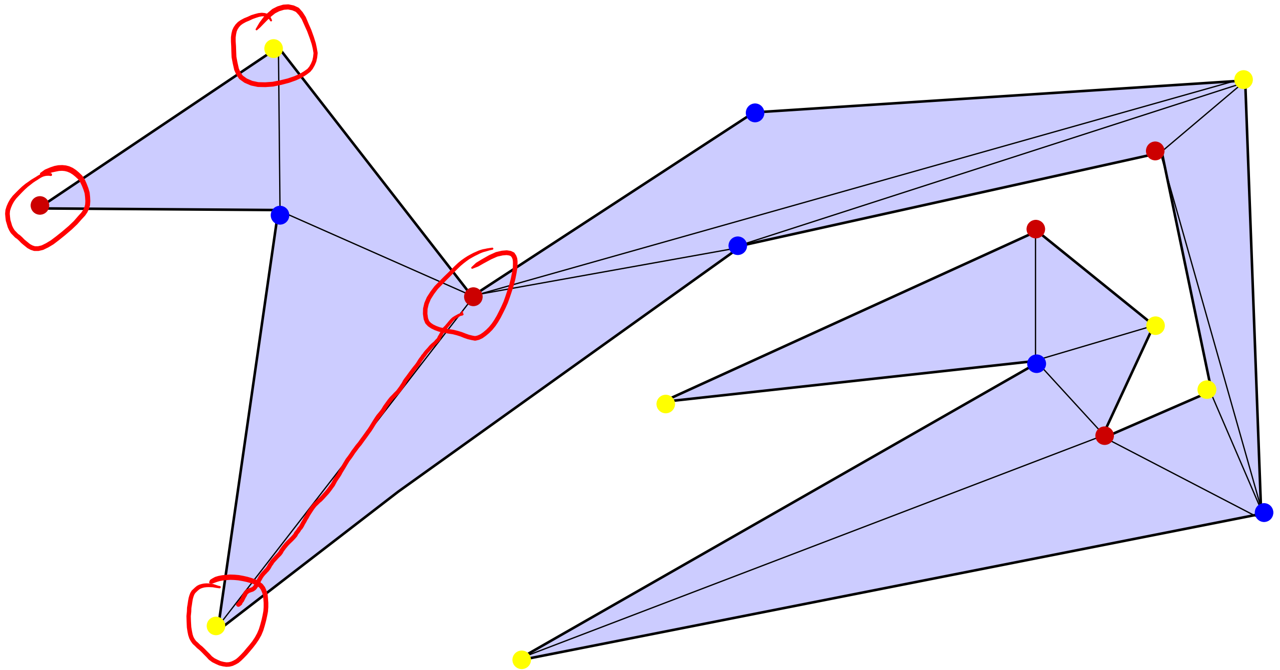
- Polygon below has n = 13, and 11 triangles.

11 cameras/guard



3-Coloring

- A 3-coloring of a graph is an assignment of one out of three colors to each vertex such that adjacent vertices have different colors.

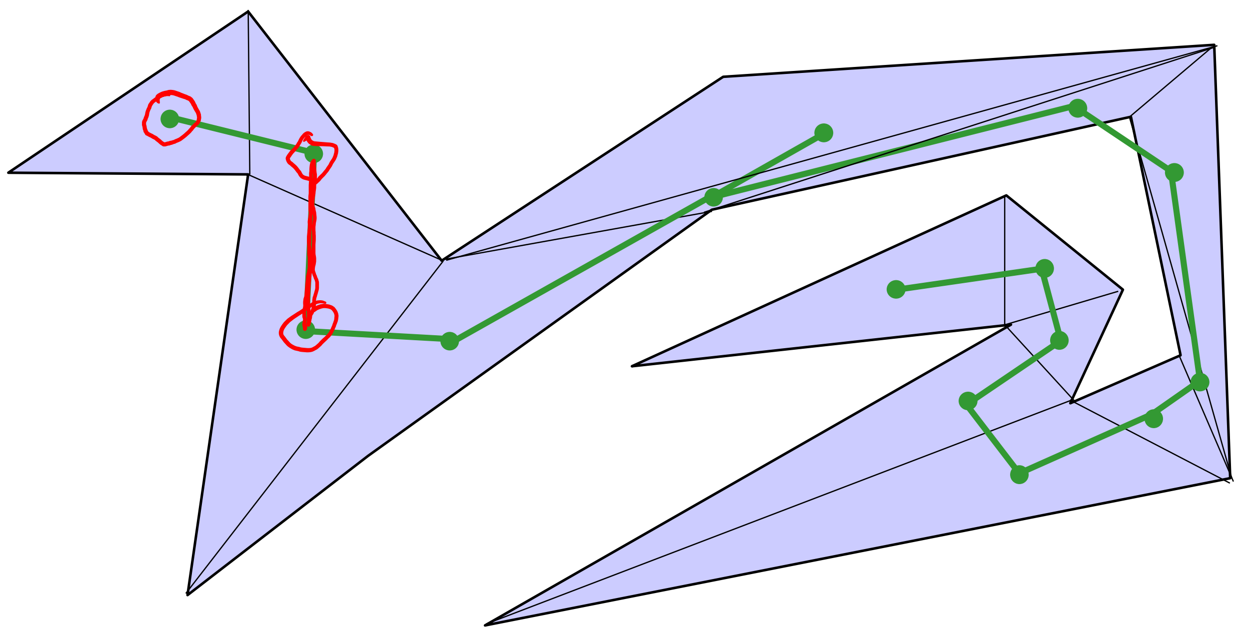


3-Coloring Lemma

Lemma: For every triangulated polgon there is a 3-coloring.

Proof: Consider the **dual graph** of the triangulation:

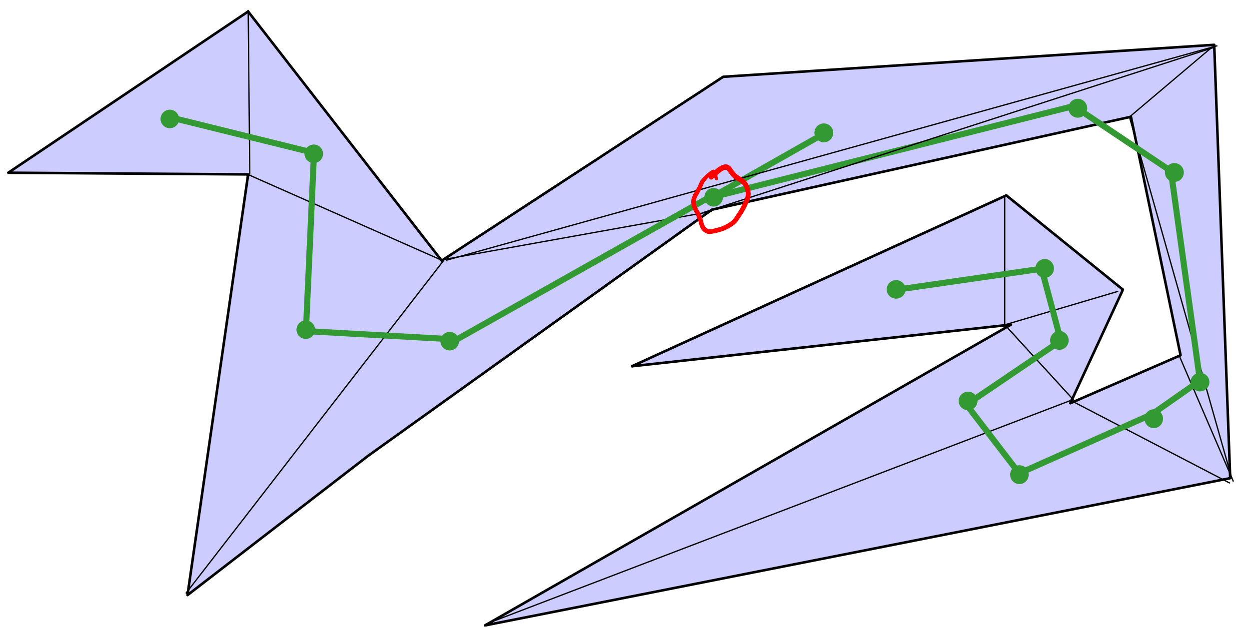
- vertex for each triangle
- edge for each edge between triangles



3-Coloring Lemma

Lemma: For every triangulated polgon there is a 3-coloring.

The dual graph is a tree (connected acyclic graph): Removing an edge corresponds to removing a diagonal in the polygon which disconnects the polygon and with that the graph.

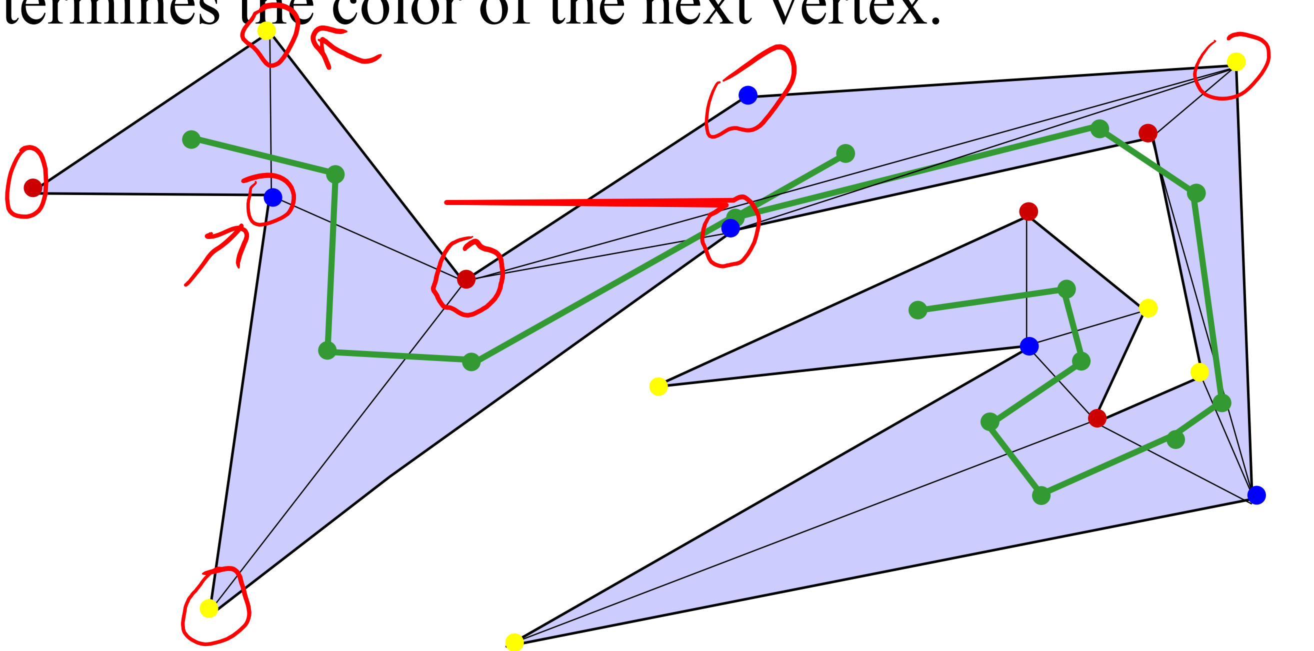


degree is ≤ 3

3-Coloring Lemma

Lemma: For every triangulated polgon there is a 3-coloring.

Traverse the tree (DFS). Start with a triangle and give different colors to vertices. When proceeding from one triangle to the next, two vertices have known colors, which determines the color of the next vertex.



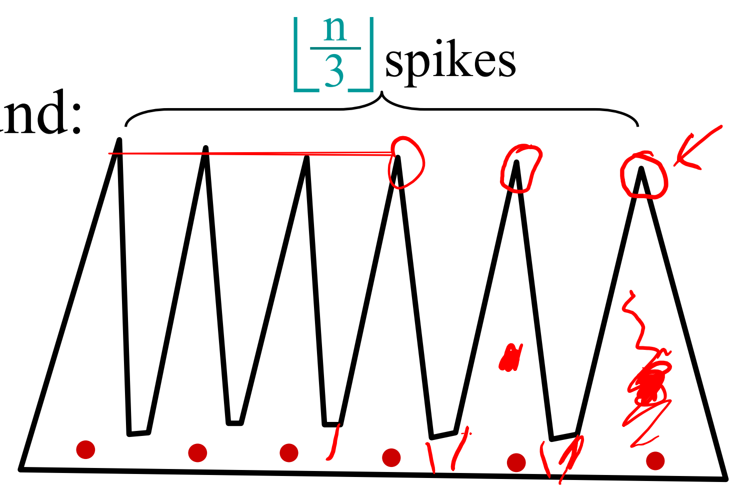
fewest color $\leq \lfloor \frac{n}{3} \rfloor$ vertices

Art Gallery Theorem

Theorem 2: For any simple polygon with n vertices $\lfloor \frac{n}{3} \rfloor$ guards are sufficient to guard the whole polygon. There are polygons for which $\lfloor \frac{n}{3} \rfloor$ guards are necessary.

Proof: For the upper bound, 3-color any triangulation of the polygon and take the color with the minimum number of guards.

Lower bound:



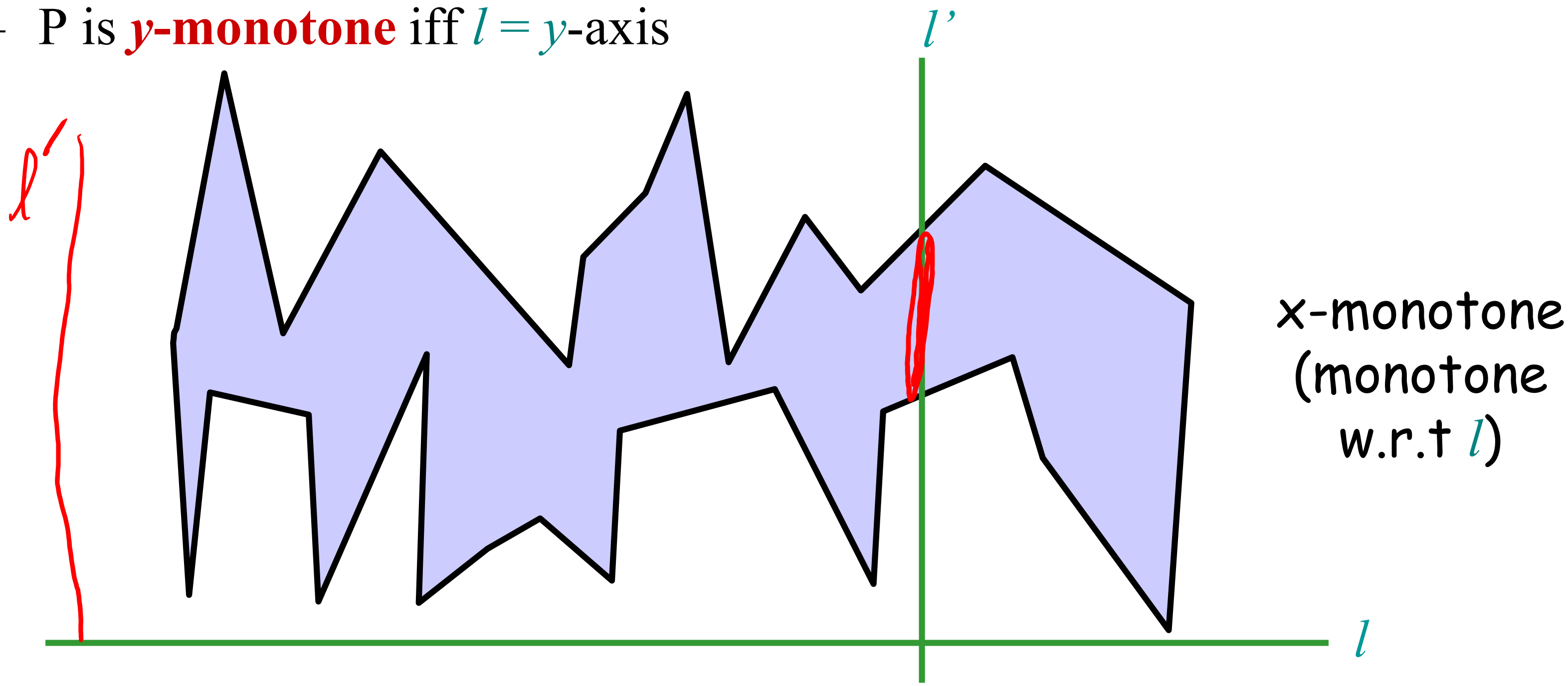
Need one guard per spike.

Triangulating a Polygon

- There is a simple $O(n^2)$ time algorithm based on the proof of Theorem 1.
- There is a very complicated $O(n)$ time algorithm (Chazelle '91) which is impractical to implement.
- We will discuss a practical $O(n \log n)$ time algorithm:
  1. Split polygon into **monotone polygons** ($O(n \log n)$ time)
  2. Triangulate each monotone polygon ($O(n)$ time)

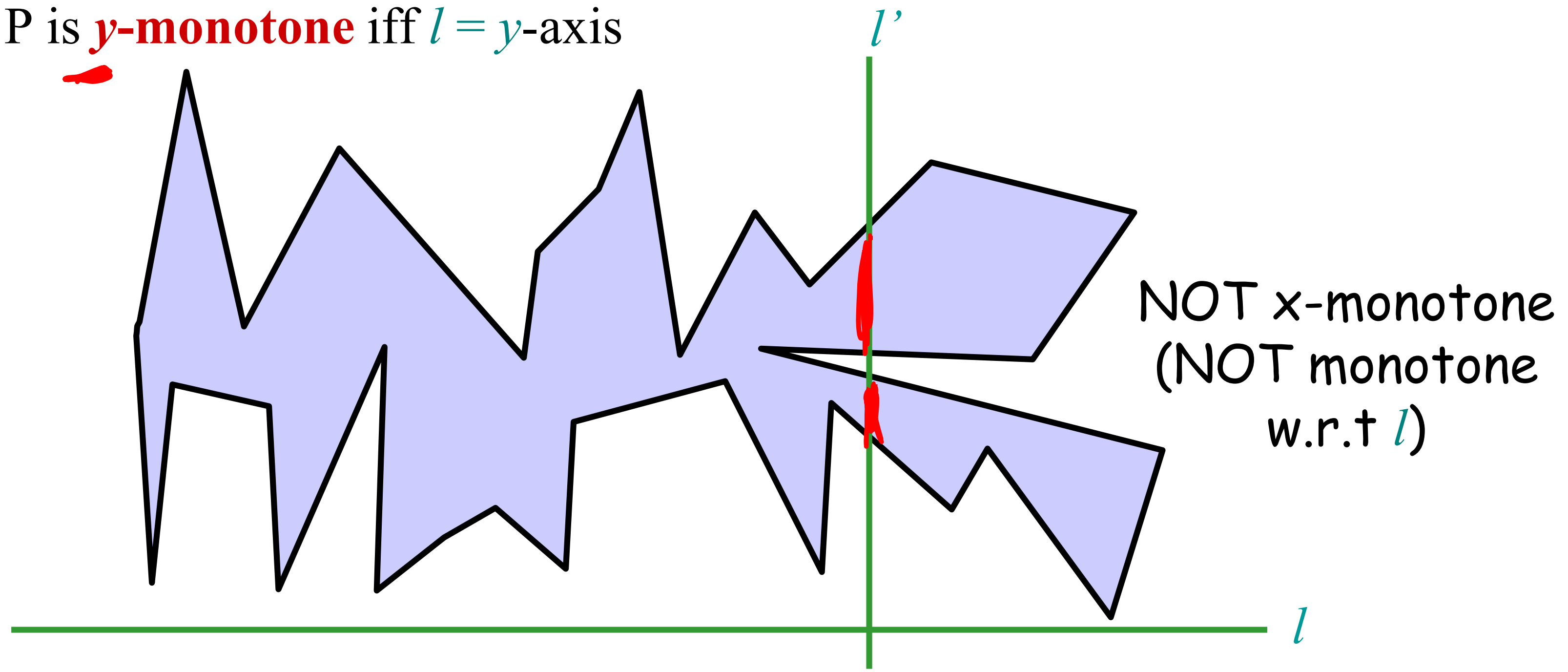
Monotone Polygons

- A simple polygon *P* is called **monotone with respect to a line** *l* iff for every line *l'* perpendicular to *l* the intersection of *P* with *l'* is connected.
  - P is ***x*-monotone** iff *l* = *x*-axis
  - P is ***y*-monotone** iff *l* = *y*-axis



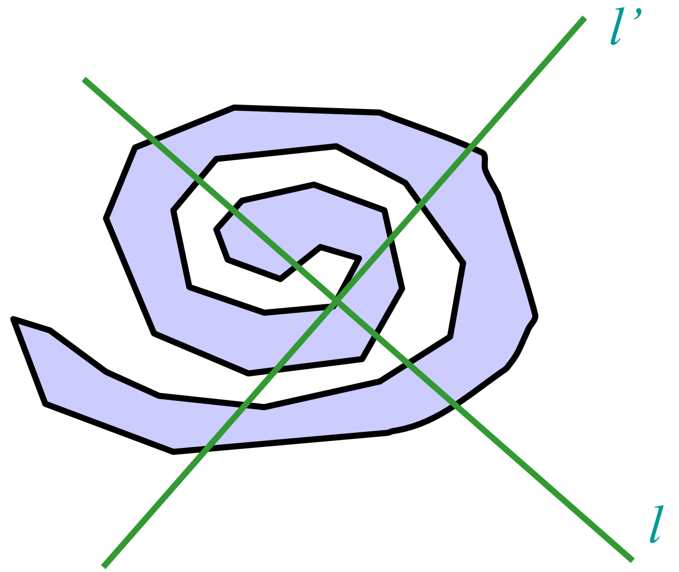
Monotone Polygons

- A simple polygon *P* is called **monotone with respect to a line** *l* iff for every line *l'* perpendicular to *l* the intersection of *P* with *l'* is connected.
  - P is ***x*-monotone** iff *l* = *x*-axis
  - P is ***y*-monotone** iff *l* = *y*-axis



Monotone Polygons

- A simple polygon P is called **monotone with respect to a line** l iff for every line l' perpendicular to l the intersection of P with l' is connected.
  - P is ***x*-monotone** iff $l = x$-axis
  - P is ***y*-monotone** iff $l = y$-axis



NOT monotone w.r.t any line l

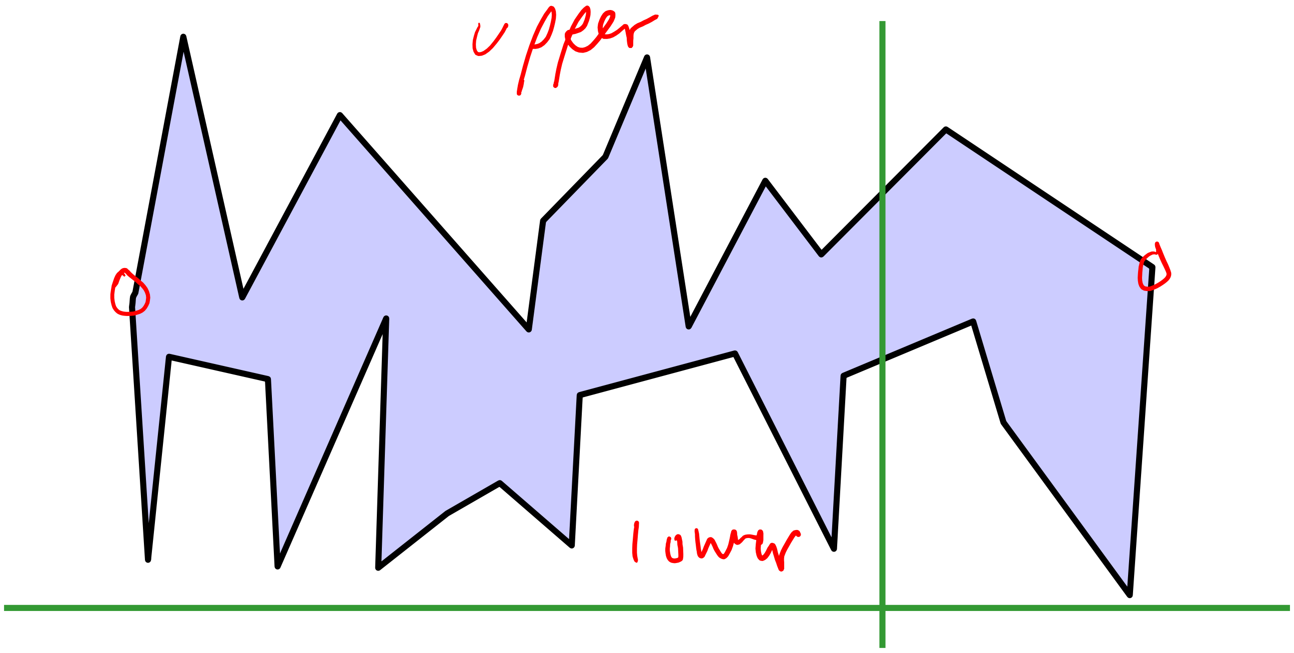
Test Monotonicity

How to test if a polygon is *x*-monotone?

– Find leftmost and rightmost vertices, O(*n*) time

→ Splits polygon boundary in upper chain and lower chain

– Walk from left to right along each chain, checking that x-coordinates are non-decreasing. O(*n*) time.

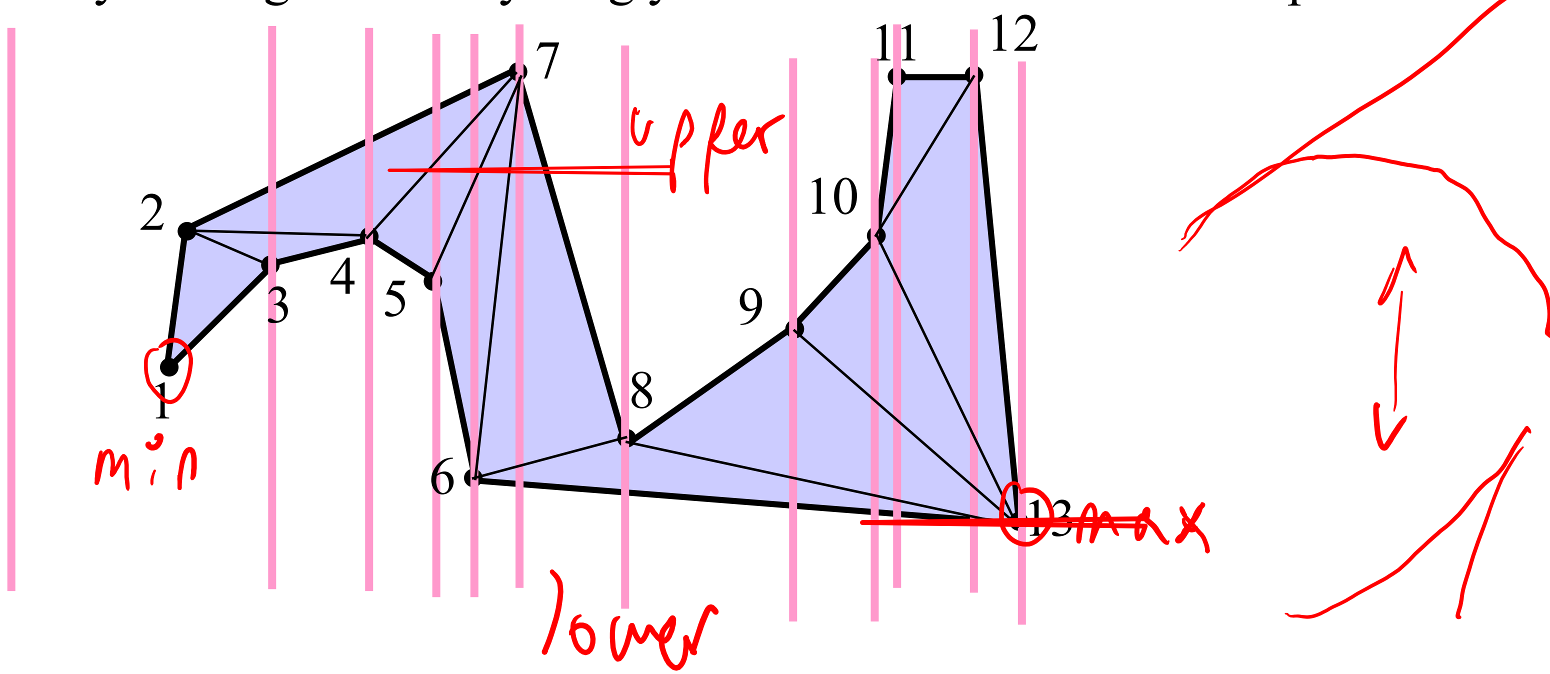


Triangulating a Polygon

- There is a simple $O(n^2)$ time algorithm based on the proof of Theorem 1.
- There is a very complicated $O(n)$ time algorithm (Chazelle '91) which is impractical to implement.
- We will discuss a practical $O(n \log n)$ time algorithm:
  1. Split polygon into **monotone polygons** ($O(n \log n)$ time)
  2. Triangulate each monotone polygon ($O(n)$ time)

Triangulate an *l*-Monotone Polygon

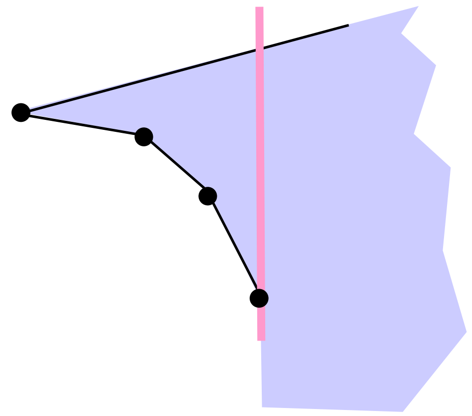
- Using a greedy plane sweep in direction *l*
- Sort vertices by increasing *x*-coordinate (merging the upper and lower chains in O(*n*) time)
- Greedy: Triangulate everything you can to the left of the sweep line.



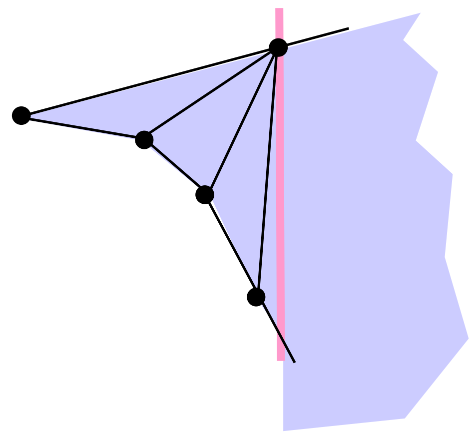
l

Triangulate an *l*-Monotone Polygon

- Store stack (sweep line status) that contains vertices that have been encountered but may need more diagonals.
- **Maintain invariant:** Un-triangulated region has a **funnel shape**. The funnel consists of an upper and a lower chain. One chain is one line segment. The other is a **reflex chain** (interior angles >180°) which is stored on the stack.

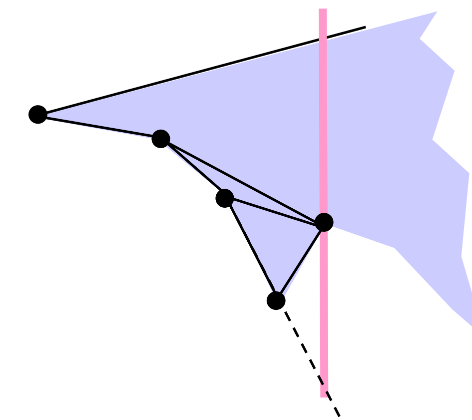


- Update, case 1: new vertex lies on chain opposite of reflex chain. Triangulate.

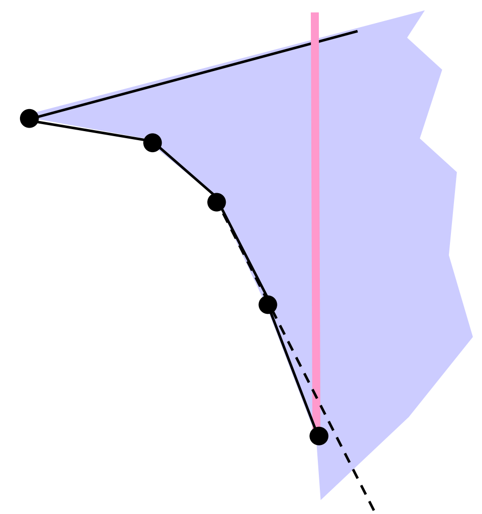


Triangulate an *l*-Monotone Polygon

- Update, case 2: new vertex lies on reflex chain
  - Case a: The new vertex lies above line through previous two vertices: Triangulate.



  - Case b: The new vertex lies below line through previous two vertices: Add to reflex chain (stack).



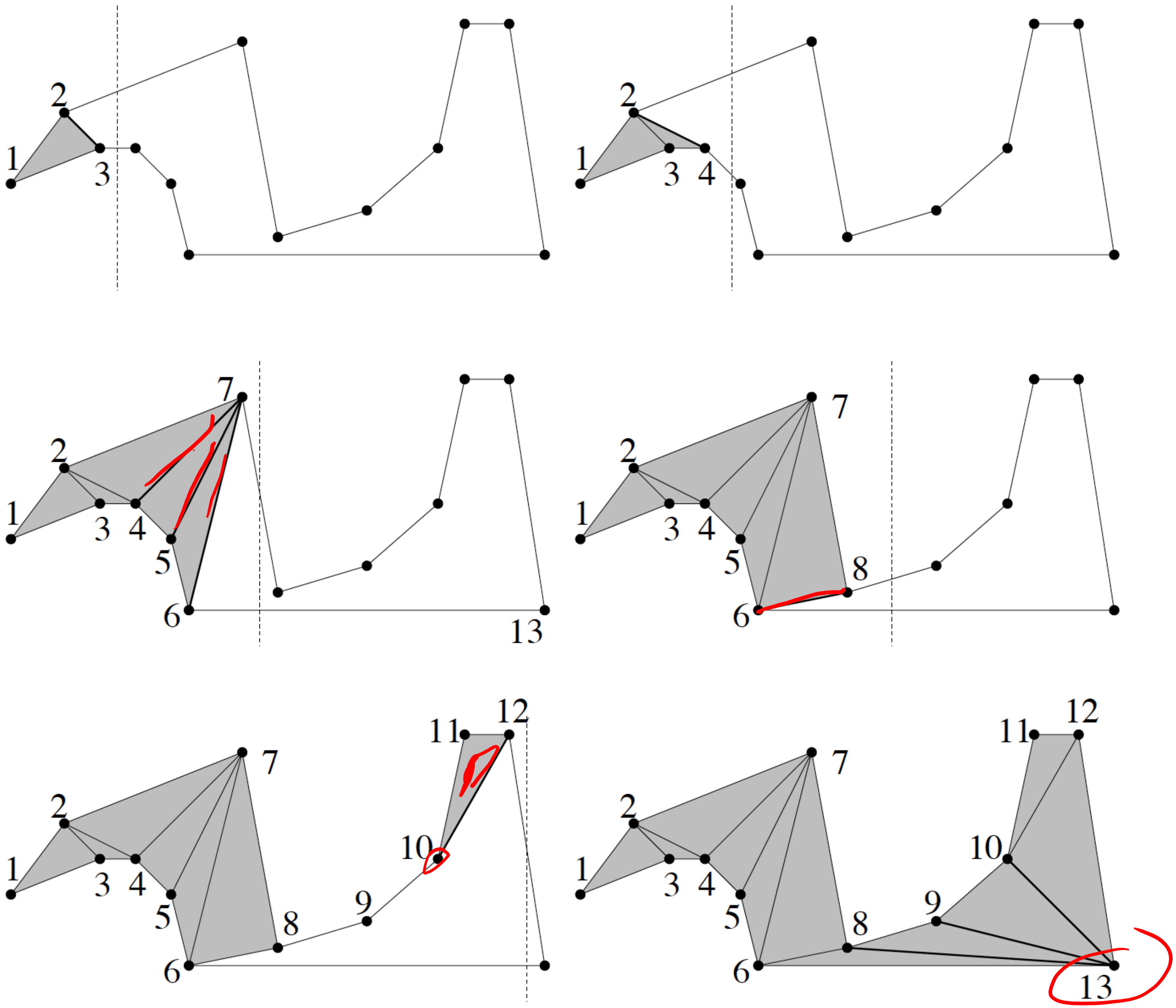
Triangulate an *l*-Monotone Polygon

- Distinguish cases in constant time using half-plane tests
- Sweep line hits every vertex once, therefore each vertex is pushed on the stack at most once.
- Every vertex can be popped from the stack (in order to form a new triangle) at most once.

$\Rightarrow$ Constant time per vertex

$\Rightarrow$ $O(n)$ total runtime

Example



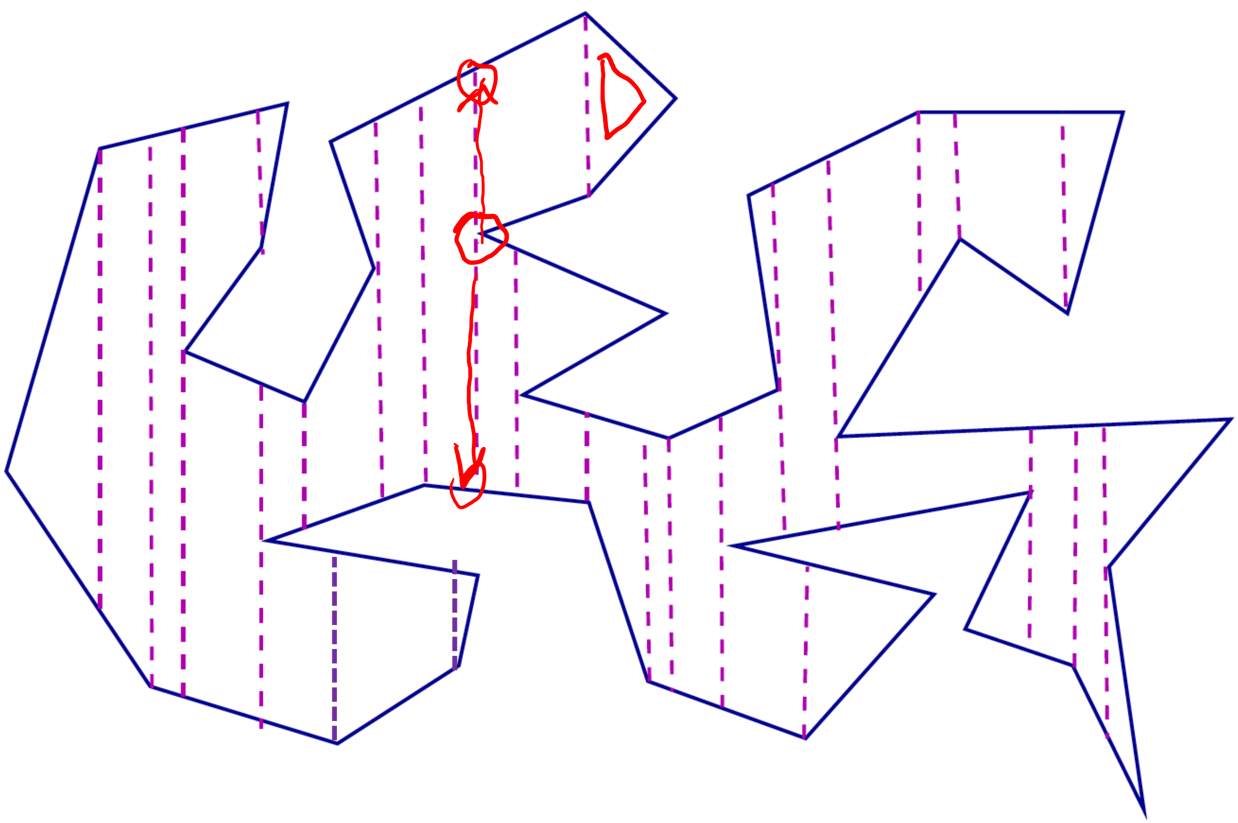
1
2
3
1
2
3
4
1
2
3
4
5
6
7
13
1
2
3
4
5
6
7
8
1
2
3
4
5
6
7
8
9
10
11
12
1
2
3
4
5
6
7
8
9
10
11
12
13

Triangulating a Polygon

- There is a simple $O(n^2)$ time algorithm based on the proof of Theorem 1.
- There is a very complicated $O(n)$ time algorithm (Chazelle '91) which is impractical to implement.
- We will discuss a practical $O(n \log n)$ time algorithm:
  1. Split polygon into **monotone polygons** ($O(n \log n)$ time)
  2. Triangulate each monotone polygon ($O(n)$ time)

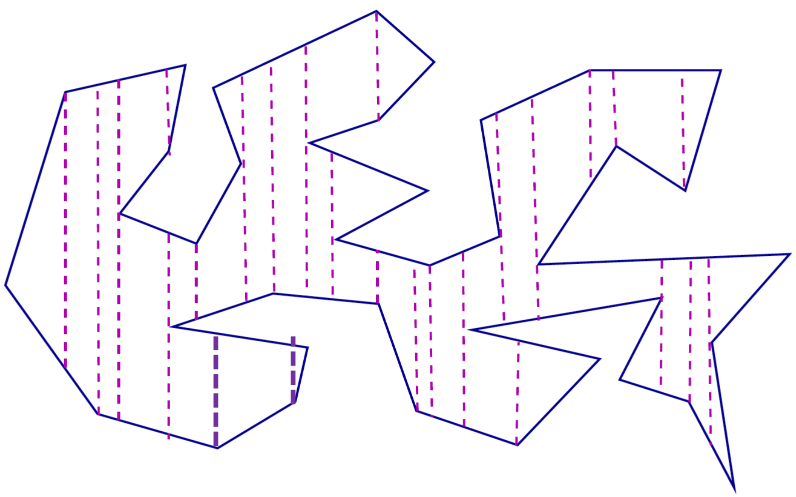
Trapezoidal Decomposition

- Extend a vertical ray up and/or down into the interior of the polygon from each vertex until it hits the boundary



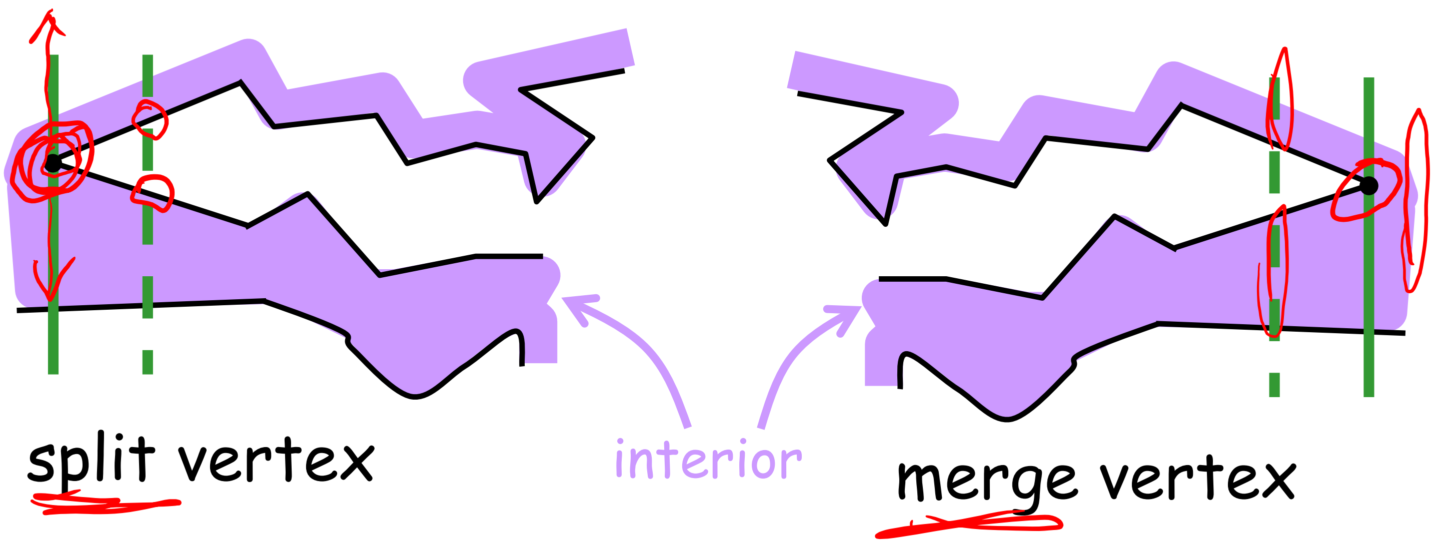
Trapezoidal Decomposition

- Use plane sweep algorithm.
- At each vertex, extend vertical line until it hits a polygon edge.
- Each face of this decomposition is a trapezoid; which may degenerate into a triangle.
- Time complexity is O(n log n).



Computing a Monotone Subdivision

- **Monotone subdivision:** subdivision of the simple polygon *P* into monotone pieces

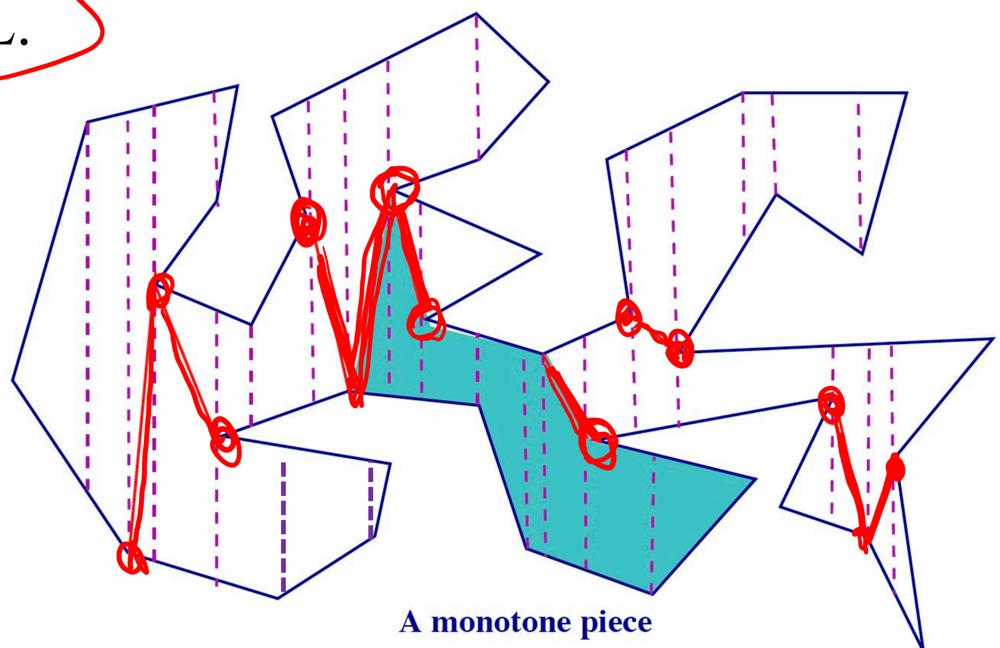


Monotone Subdivision

of simple polygon

- Call a reflex vertex with both rightward (leftward) edges a **split** (**merge**) vertex.
  - Non-monotonicity comes from split or merge vertices.
- Add a diagonal to each to remove the non-monotonicity.
- To each split (merge) vertex, add a diagonal joining it to the polygon vertex of its left (right) trapezoid. Output as a DCEL.

$O(n \log n)$



A monotone piece