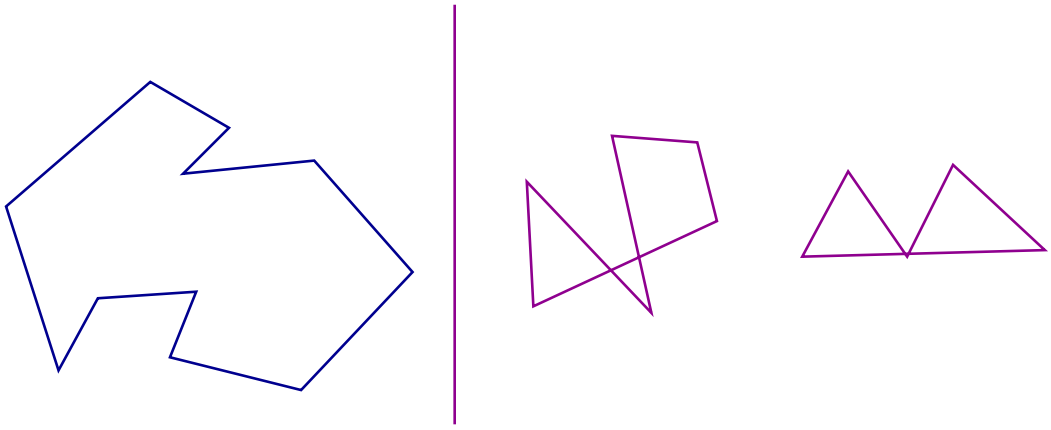
Polygon Triangulation

- 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.



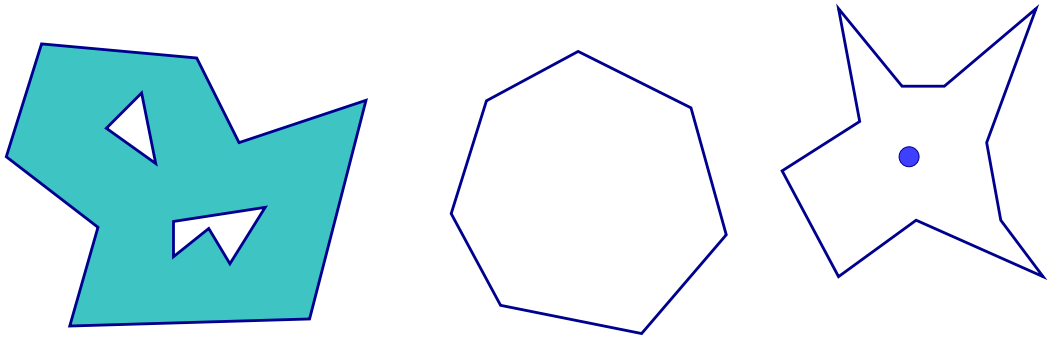
Simple Polygon

Non–Simple Polygons

- By Jordan Theorem, a polygon divides the plane into interior, exterior, and boundary.
- We use polygon both for boundary and its interior; the context will make the usage clear.

Polygons

- **Polygons with holes** are topologically different; **two paths may not be homeomorphic.**

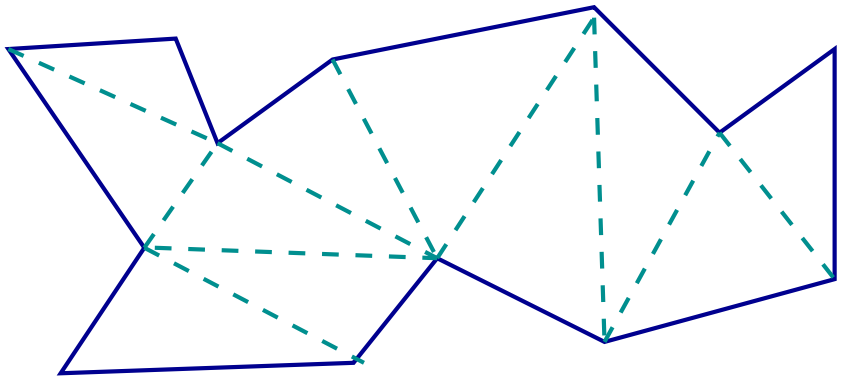


Polygon with Holes | Convex Polygon | Star–Shaped

- Other common classes of polygons are **convex, star-shaped, monotone**.
- **Polygons are basic buliding blocks in most geometric applications.**
  - **Flexible:** model arbitrarily complex shapes.
  - **Efficient:** admit simple algorithms and algebraic representation/manipulation.
  - Thus, **significantly more powerful**, say, than rectangles as building blocks.

Triangulation

- **Partition** polygon P into non-overlapping triangles using diagonals only.
- Is this always possible for any simple polygon? If not, which polygons are triangulable.
- Does the number of triangles depend on the choice of triangulation? How many triangles?

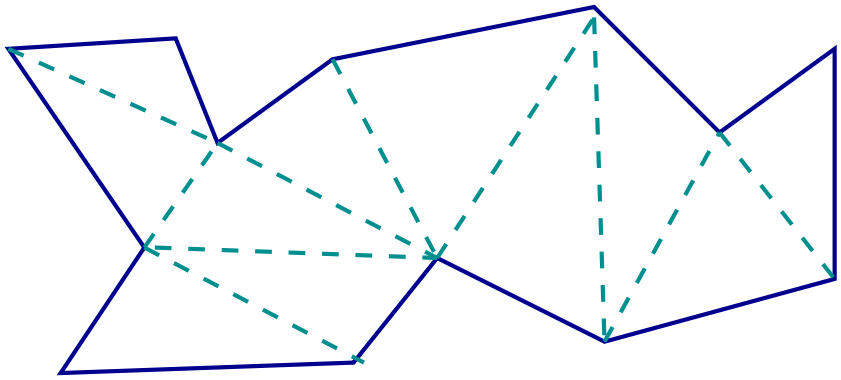


- **Triangulation** reduces complex shapes to collection of simpler shapes. First step of many advanced algorithms.
- Many applications: visibility, robotics, mesh generation, point location etc.

Triangulation Theorem

1. **Every simple polygon admits a triangulation.**

2. **Every triangulation of an n-gon has exactly $n-2$ triangles.**

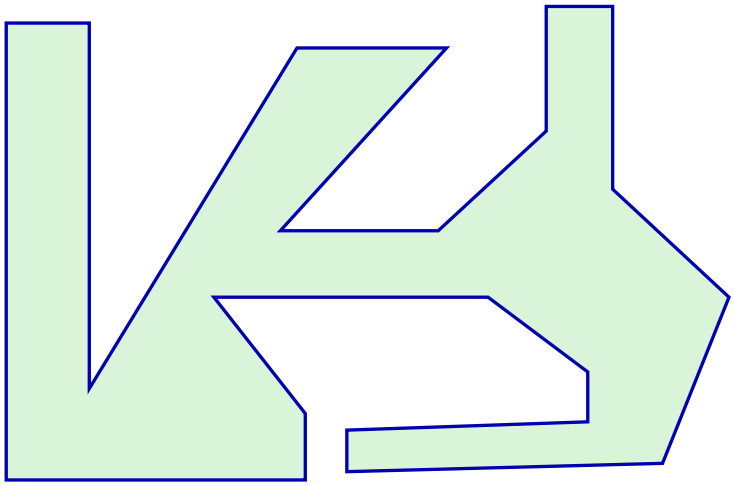


3. **Polygon in picture has $n = 13$, and 11 triangles.**

4. **Before proving the theorem and developing algorithms, consider a cute puzzle that uses triangulation: Art Gallery Theorem.**

Art Gallery Theorem

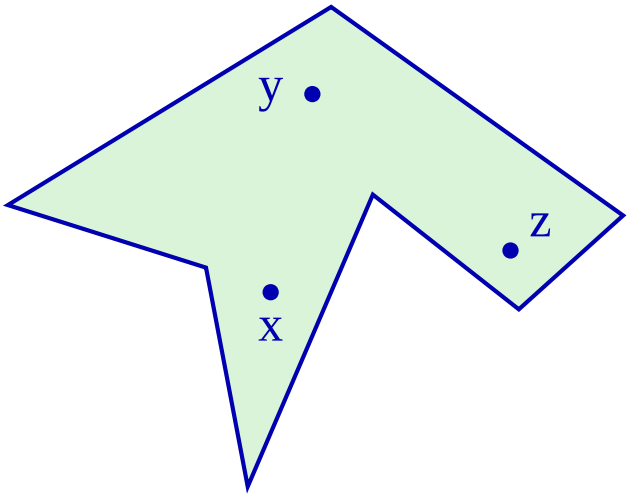
- **The floor plan of an art gallery modeled as a simple polygon with n vertices.**
- **How many guards needed to see the whole room?**
- **Each guard is stationed at a fixed point, has 360^o vision, and cannot see through the walls.**



- **Story: Problem posed to Vasek Chvatal by Victor Klee at a math conference in 1973. Chvatal solved it quickly with a complicated proof, which has since been simplified significantly using triangulation.**

Formulation

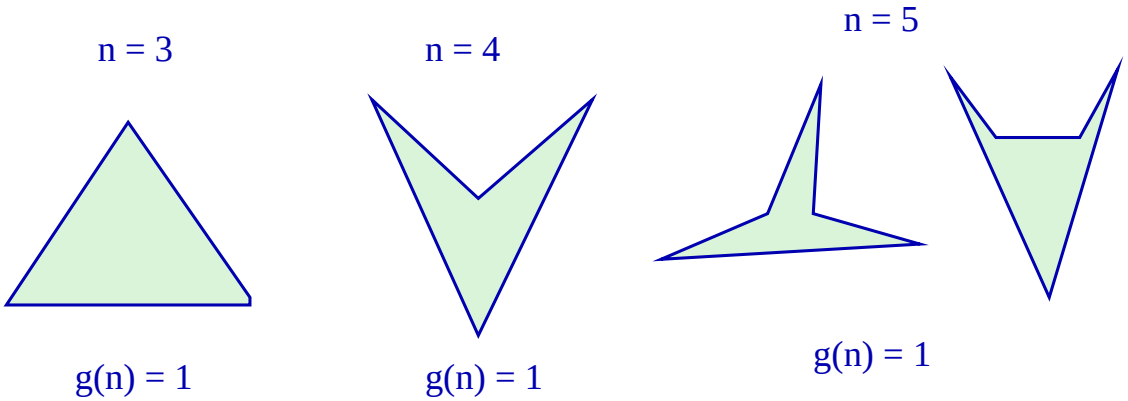
- **Visibility:** p, q **visible if** $pq \in P$**.**
- y **is visible from** x **and** z**. But** x **and** z **not visible to each other.**



- $g(P)$ = **min. number of guards to see** P
- $g(n) = \max_{|P|=n} g(P)$
- **Art Gallery Theorem asks for bounds on function** $g(n)$**: what is the smallest** $g(n)$ **that always works for any** n**-gon?**

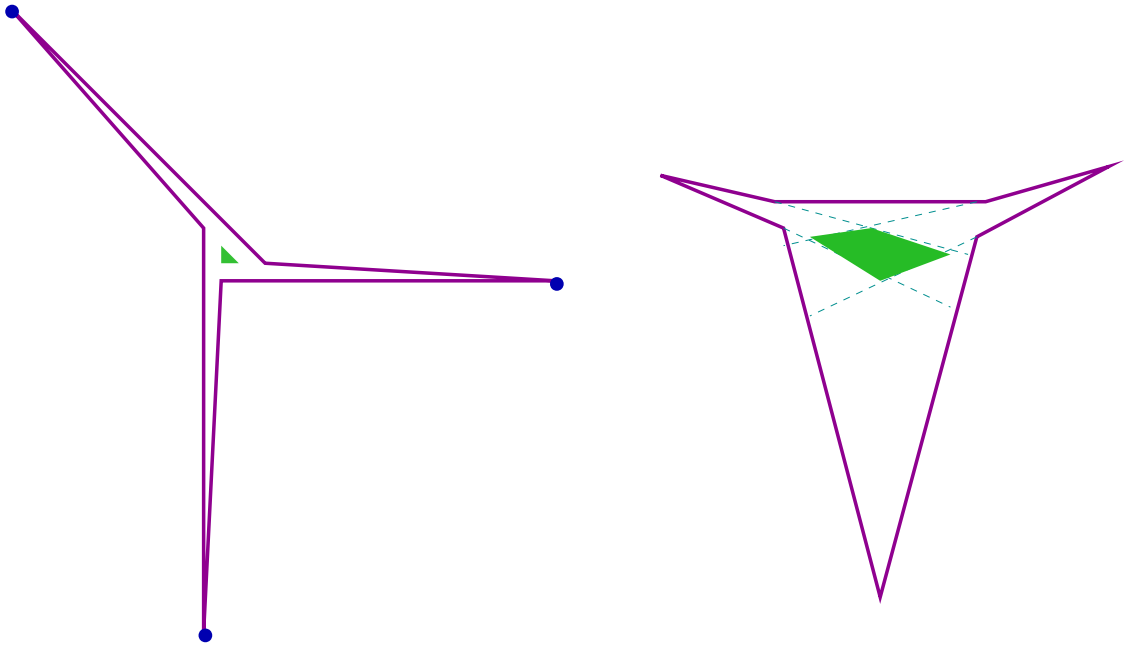
Trying it Out

1. For $n = 3, 4, 5$, we can check that $g(n) = 1$.



2. Is there a general formula in terms of n?

Pathological Cases

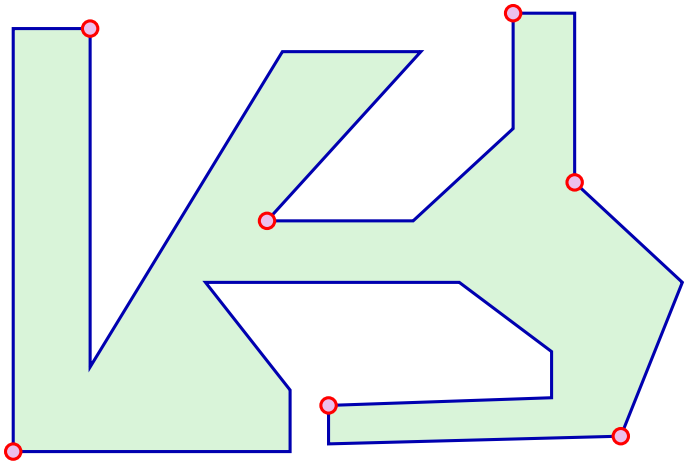


1. **Fig. on left shows that seeing the boundary $\neq$ seeing the whole interior!**

2. **Even putting guards at every other vertex is not sufficient.**

3. **Fig. on right shows that putting guards on vertices alone might not give the best solution.**

Art Gallery Theorem

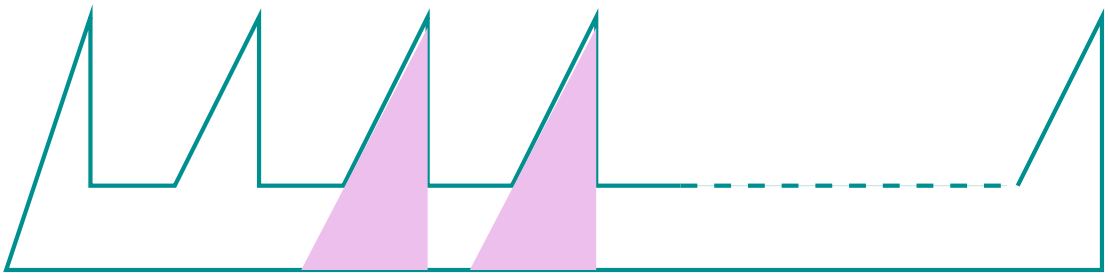


22–gon.

7 Guards

Theorem: $g(n) = \lfloor n/3 \rfloor$

1. **Every n-gon can be guarded with $\lfloor n/3 \rfloor$ vertex guards.**
2. **Some n-gons require at least $\lfloor n/3 \rfloor$ (arbitrary) guards.**

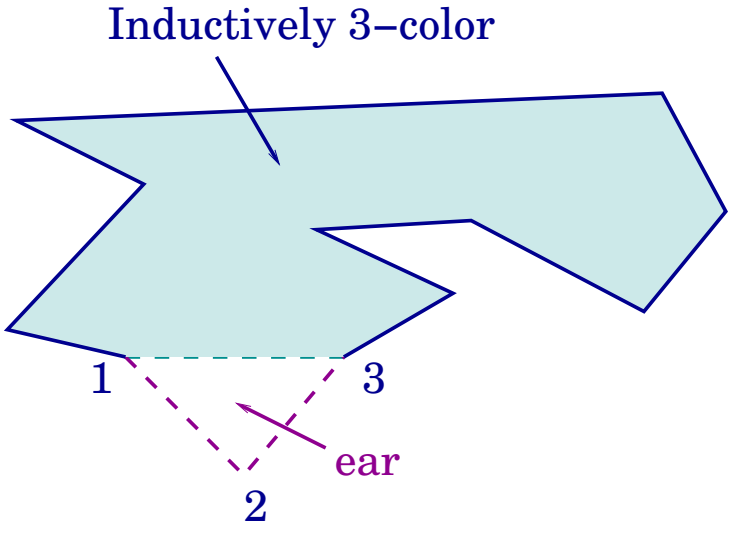


Necessity Construction

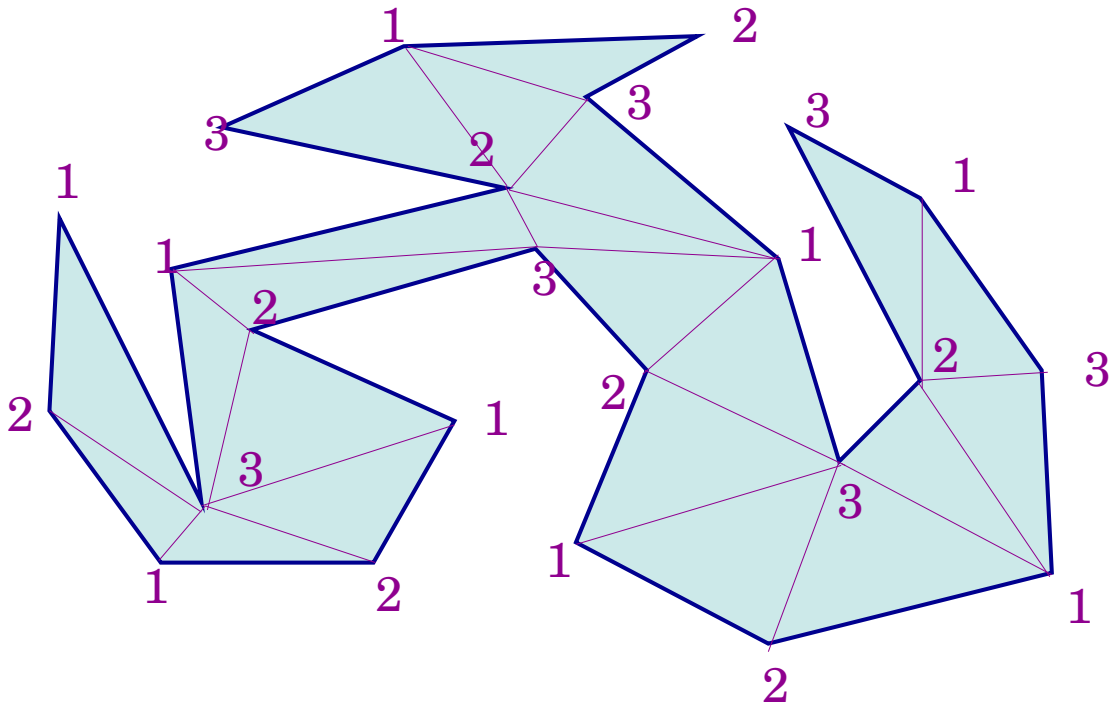
Fisk's Proof

Lemma: **Triangulation graph can be 3-colored.**

- P plus triangulation is a planar graph.
- **3-coloring means vertices can be labeled 1,2, or 3 so that no edge or diagonal has both endpoints with same label.**
- **Proof by Induction:**
  1. **Remove an ear.**
  2. **Inductively 3-color the rest.**
  3. **Put ear back, coloring new vertex with the label not used by the boundary diagonal.**



Proof

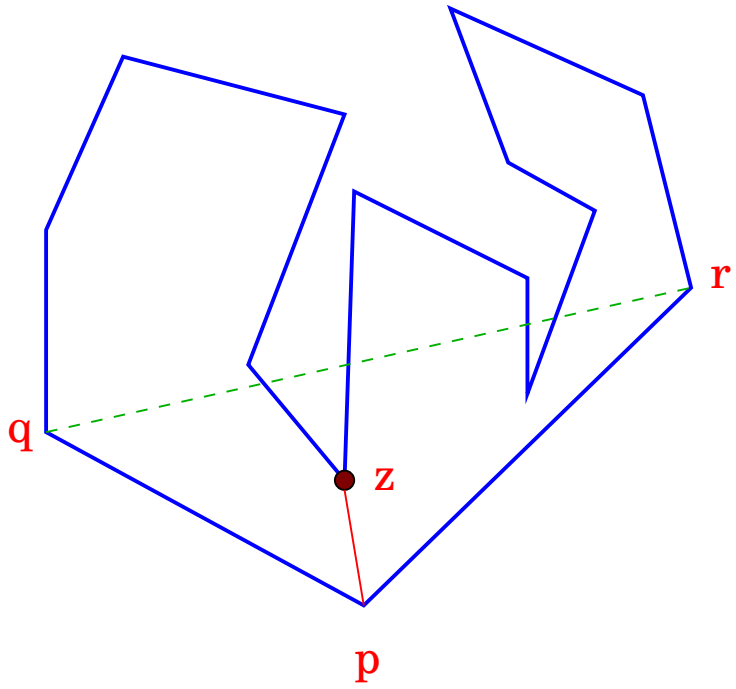


- **Triangulate P. 3-color it.**
- **Least frequent color appears at most $\lfloor n/3 \rfloor$ times.**
- **Place guards at this color positions—a triangle has all 3 colors, so seen by a gaurd.**
- **In example: Colors 1, 2, 3 appear 9, 8 and 7 times, resp. So, color 3 works.**

Triangulation: Theory

Theorem: Every polygon has a triangulation.

- **Proof by Induction. Base case $n = 3$.**

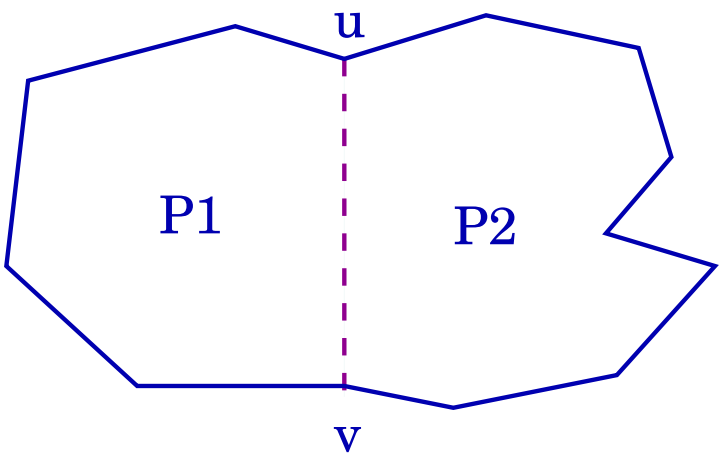


- **Pick a convex corner p. Let q and r be pred and succ vertices.**
- **If qr a diagonal, add it. By induction, the smaller polygon has a triangulation.**
- **If qr not a diagonal, let z be the reflex vertex farthest to qr inside $\triangle pqr$.**
- **Add diagonal pz; subpolygons on both sides have triangulations.**

Triangulation: Theory

Theorem: Every triangulation of an n-gon has $n-2$ triangles.

- **Proof by Induction. Base case $n = 3$.**



- **Let $t(P)$ denote the number of triangles in any triangulation of P.**
- **Pick a diagonal uv in the given triangulation, which divides P into P_1, P_2.**
- $t(P) = t(P_1) + t(P_2) = n_1 - 2 + n_2 - 2$.
- **Since $n_1 + n_2 = n + 2$, we get $t(P) = n - 2$.**

Triangulation History

1. **A really naive algorithm is $O(n^4)$: check all n^2 choices for a diagonal, each in $O(n)$ time. Repeat this $n-1$ times.**

2. **A better naive algorithm is $O(n^2)$; find an ear in $O(n)$ time; then recurse.**

3. **First non-trivial algorithm: $O(n \log n)$ [GJPT-78]**

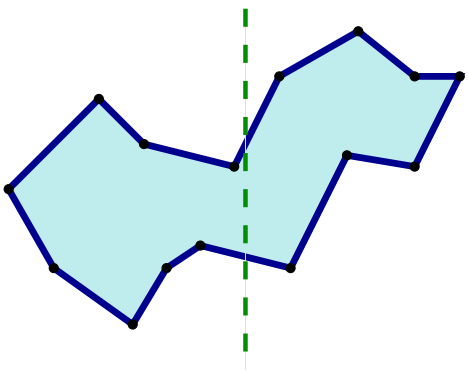
4. **A long series of papers and algorithms in 80s until Chazelle produced an optimal $O(n)$ algorithm in 1991.**

5. **Linear time algorithm insanely complicated; there are randomized, expected linear time that are more accessible.**

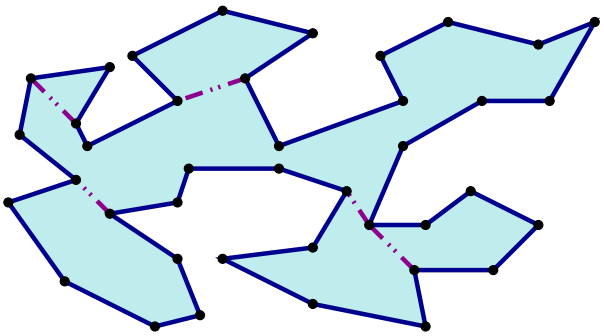
6. **We content ourselves with $O(n \log n)$ algorithm.**

Algorithm Outline

1. **Partition polygon into trapezoids.**
2. **Convert trapezoids into monotone subdivision.**
3. **Triangulate each monotone piece.**



x−monotone polygon

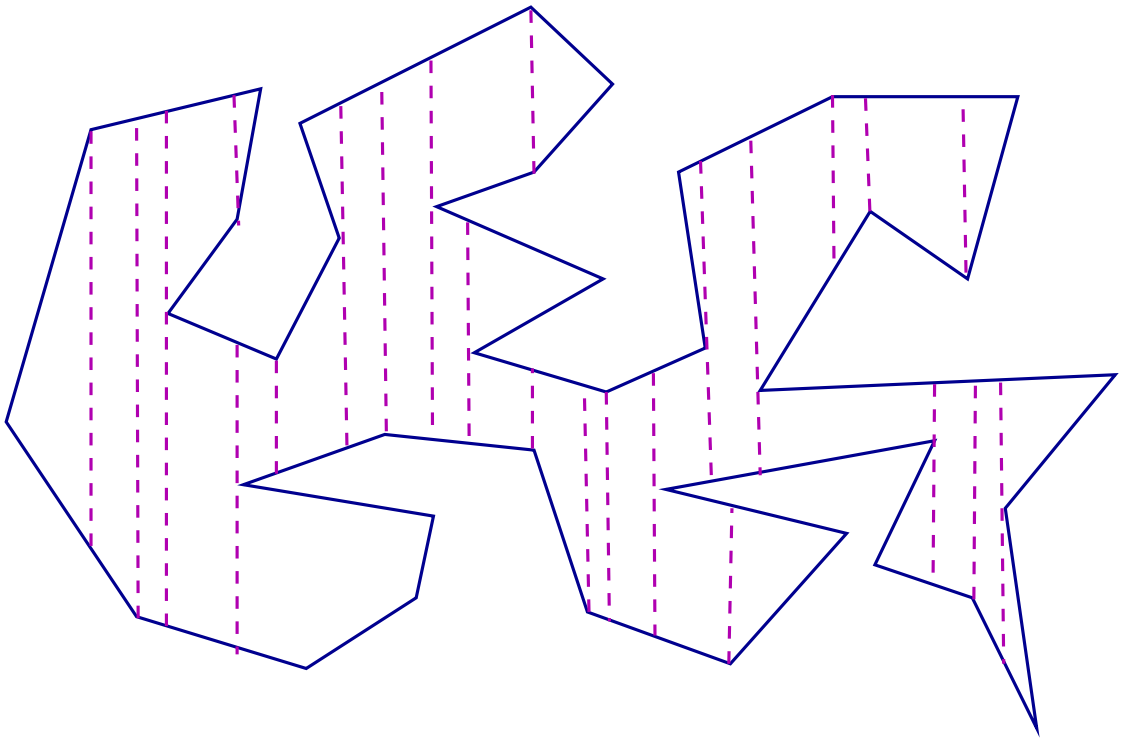


Monotone decomposition

4. **A polygonal chain C is monotone w.r.t. line L if any line orthogonal to L intersects C in at most one point.**
5. **A polygon is monotone w.r.t. L if it can be decomposed into two chains, each monotone w.r.t. L.**
6. **In the Figure, L is x-axis.**

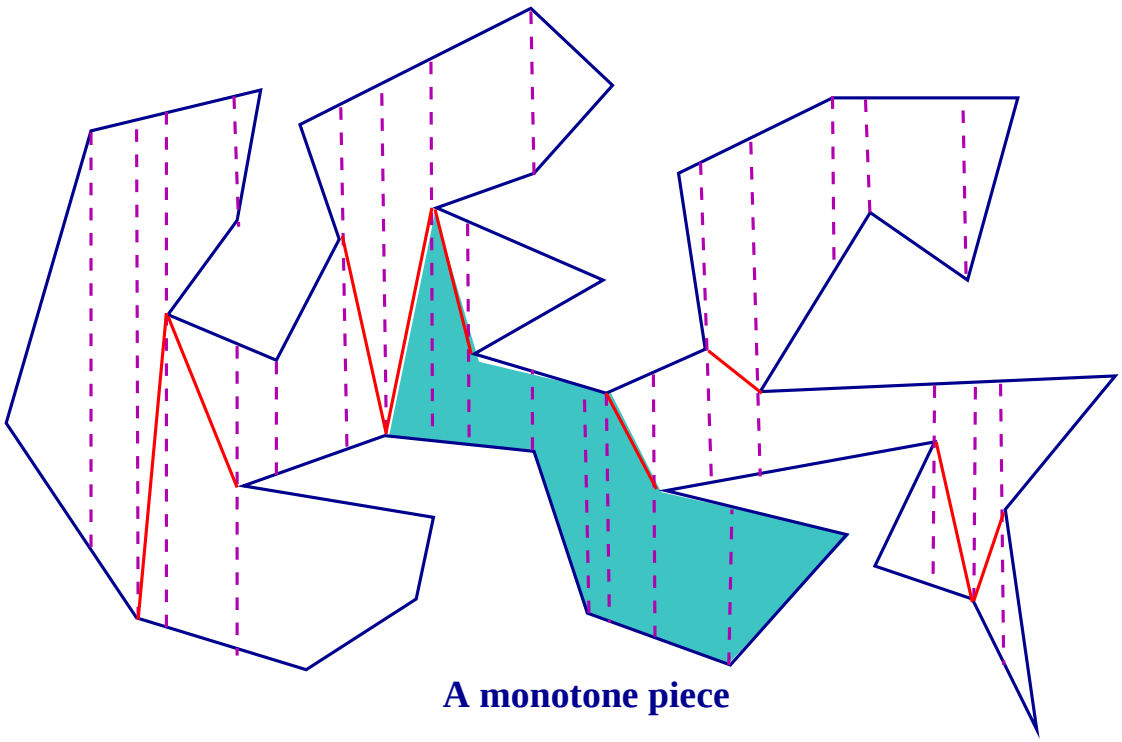
Trapezoidal Decomposition

- **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 $O(n \log n)$.**



Monotone Subdivision

- 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.

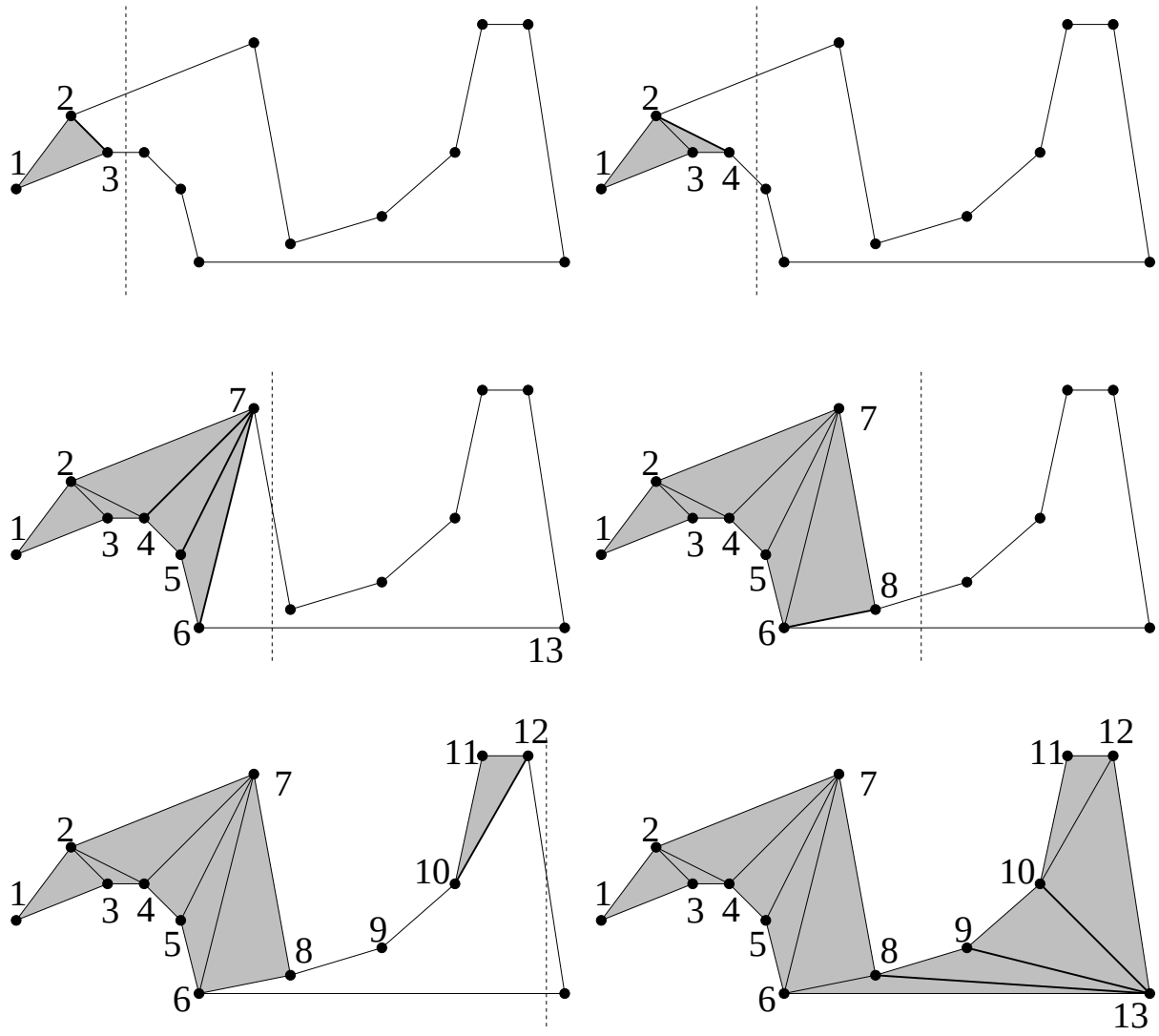


A monotone piece

Monotone Subdivision

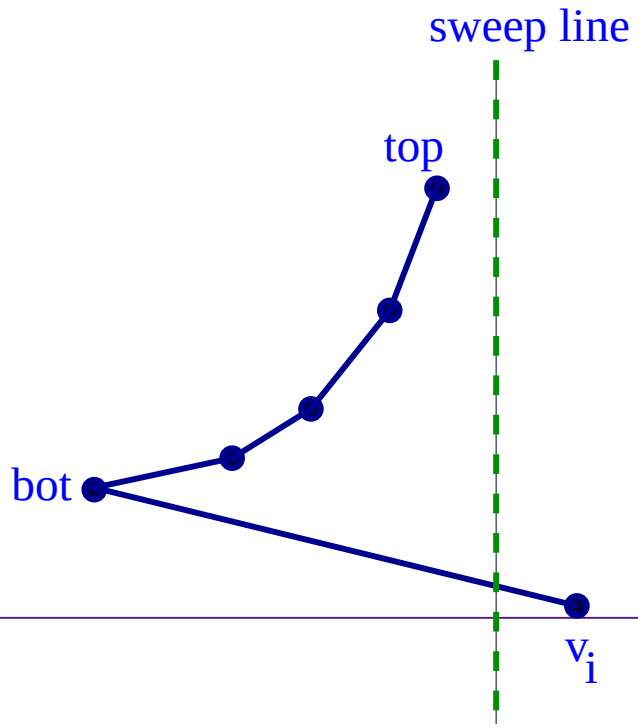
- Assume that trap decomposition represented by DCEL.
- Then, matching vertex for split and merge vertex can be found in $O(1)$ time.
- Remove all trapezoidal edges. The polygon boundary plus new split/merge edges form the monotone subdivision.
- The intermediate trap decomposition is only for presentation clarity—in practice, you can do monotone subdivision directly during the plane sweep.

Triangulation



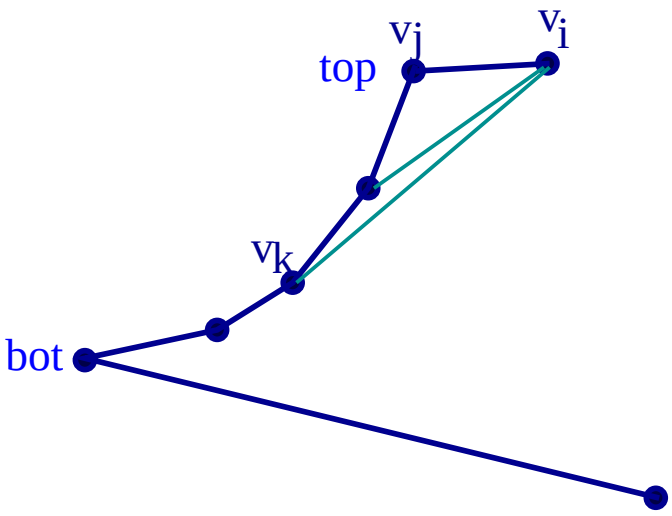
Triangulation

- $\langle v_1, v_2, \ldots, v_n \rangle$ **sorted left to right.**
- **Push** v_1, v_2 **onto stack.**
- **for** $i = 3$ **to** n **do**
  - **if** v_i **and** $top(stack)$ **on same chain**
    - **Add diagonals** $v_i v_j, \ldots, v_i v_k$**, where** v_k **is last to admit legal diagonal**
    - **Pop** $v_j, \ldots, v_{k-1}$ **and Push** v_i
  - **else**
    - **Add diagonals from** v_i **to all vertices on the stack and pop them**
    - **Save** v_{top}**; Push** v_{top} **and** v_i



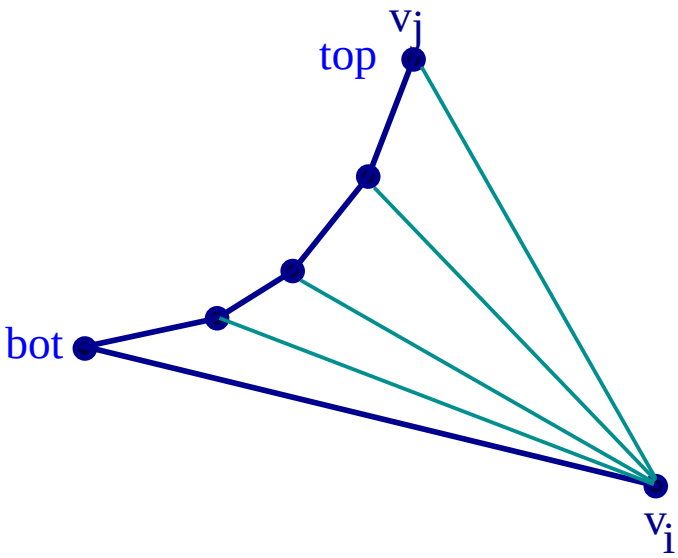
Correctness

- **Invariant:** Vertices on current stack form a single reflex chain. The leftmost unscanned vertex in the other chain is to the right of the current scan line.



New stack: (bot, ..., vk, vi)

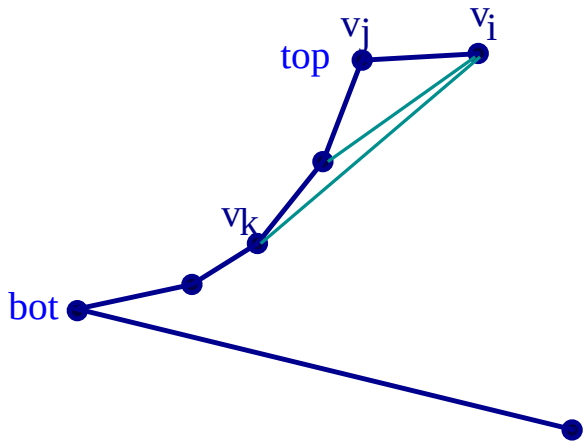
Case I



New stack: (vj, vi)

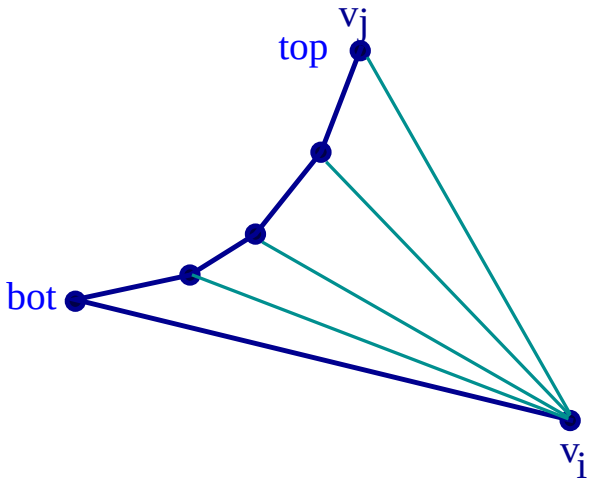
Case II

Time Complexity



New stack: (bot, ..., vk, vi)

Case I

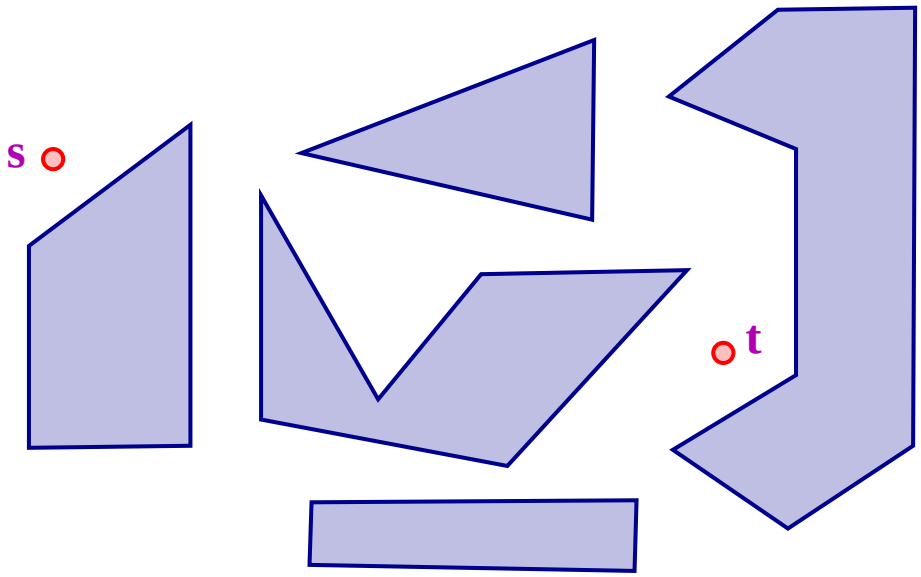


New stack: (vj, vi)

Case II

- **A vertex is added to stack once. Once it's visited during a scan, it's removed from the stack.**
- **In each step, at least one diagonal is added; or the reflex stack chain is extended by one vertex.**
- **Total time is $O(n)$.**
- **Total time for polygon triangulation is therefore $O(n \log n)$.**

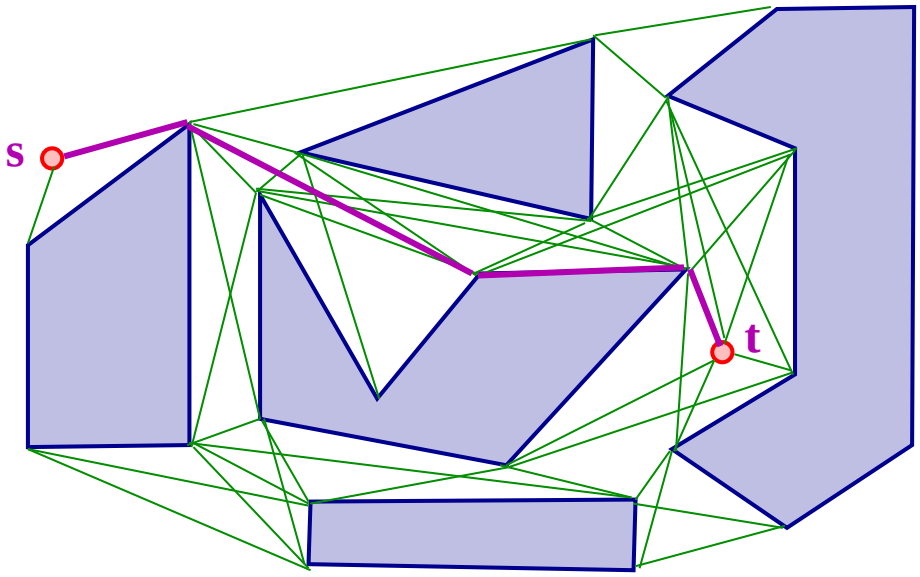
Shortest Paths



- A workspace with polygonal obstacles.
- Find shortest obstacle-avoiding path from s to t.
- Properties of Shortest Path:
  - Uses straight line segments.
  - No self-intersection.
  - Turns at convex vertices only.

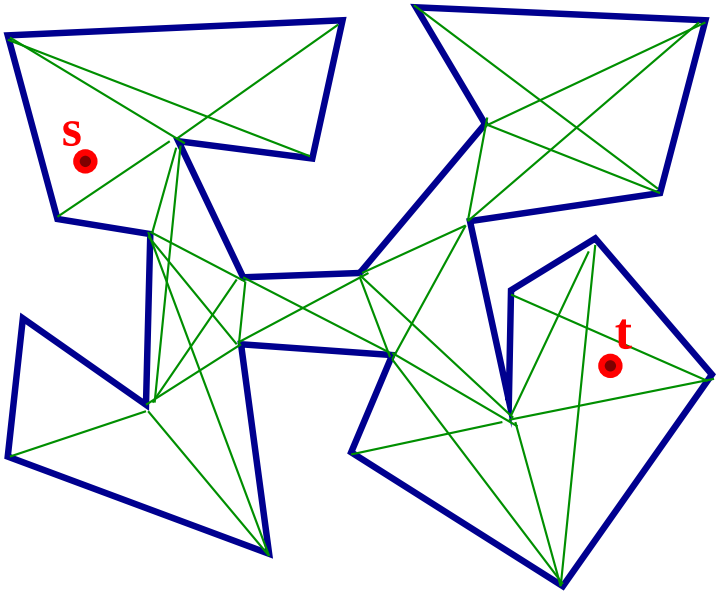
Visibility Graph

- **Construct a visibility graph $G = (V, E)$, where V is set of polygon vertices (and s, t), E is pairs of nodes that are mutually "visible".**
- **Give each edge (u, v) the weight equal to the Euclidean distance between u and v.**
- **The shortest path from s to t in this graph is the obstacle avoiding shortest path.**
- **G can have between $c_1 n$ and $c_2 n^2$ edges. Run Dijkstra's algorithm.**



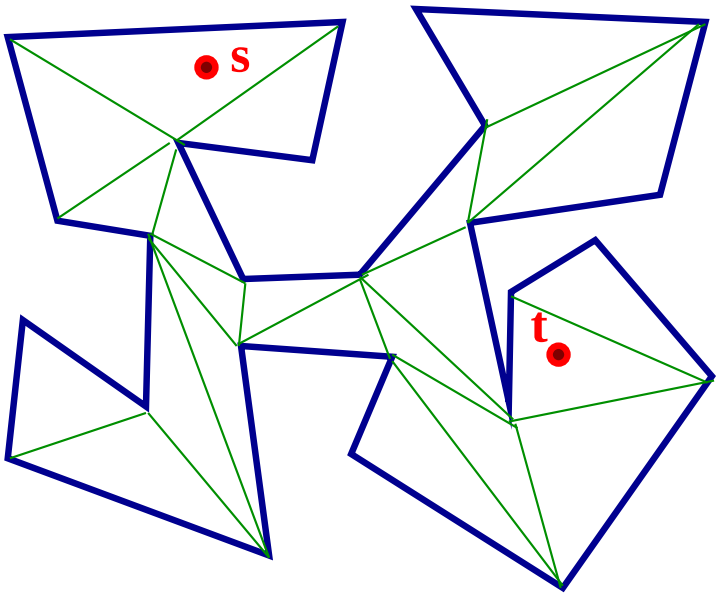
Paths in a Polygon

- **Workspace interior of a simple polygon.**
- **Can we compute a shortest path faster?**
- **The visibility graph can still have $\Theta(n^2)$ edges.**



- **Using polygon triangulation, we show an $O(n \log n)$ time algorithm.**

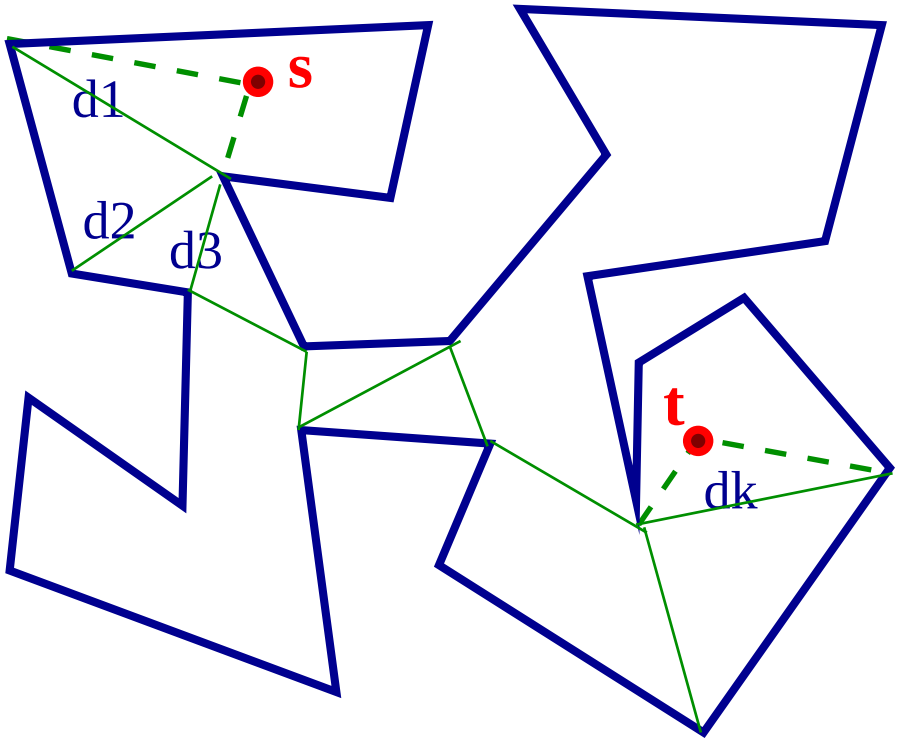
Fast Algorithm



- Let P be a simple polygon and s, t be source and target points.
- Let T be a triangulation of P.
- Call a diagonal d of T essential if s, t lie on opposite sides of d.
- Let $d_1, d_2, \ldots, d_k$ be ordered list of essential diagonal.

Algorithm

- **Essential diagonals $d_1, d_2, \ldots, d_k$.**

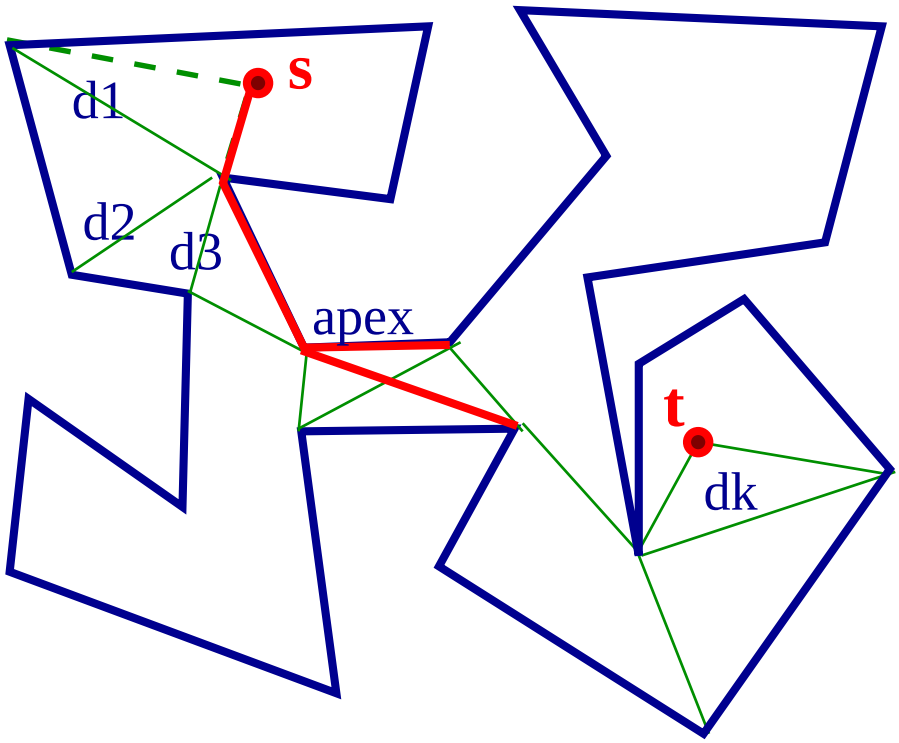


- **The algorithm works as follows:**

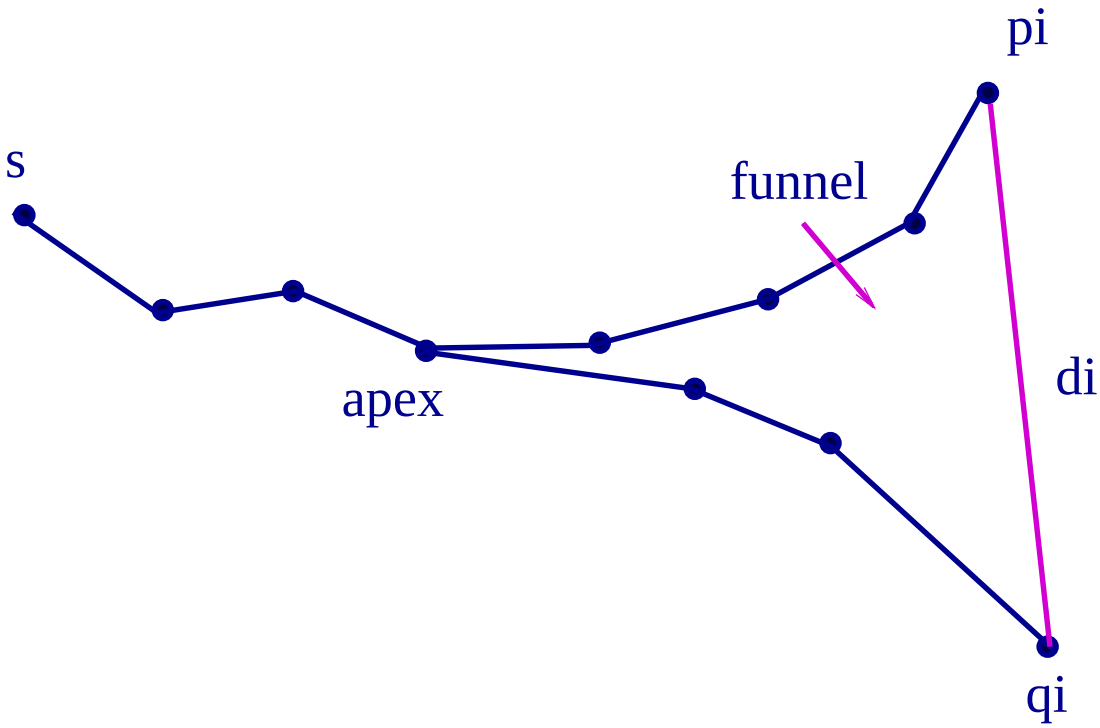
1. **Start with $d_0 = s$.**
2. **for $i = 1$ to $k + 1$ do**
3. **Extend path from s to both endpoints of d_i**

Path Extending: Funnel

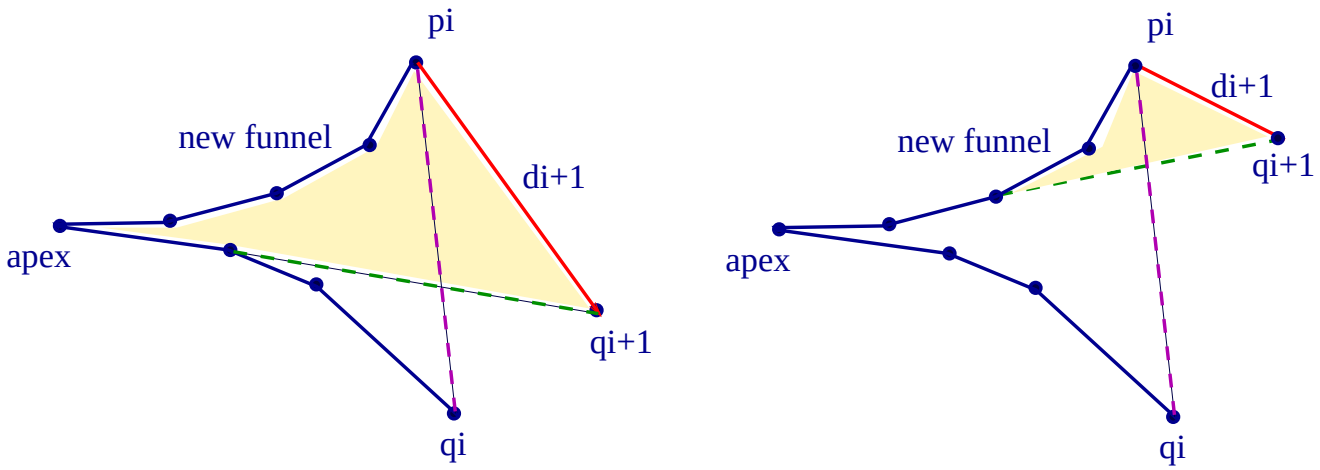
- **Union of $path(s, p_i)$ and $path(s, q_i)$ forms a funnel.**
- **The vertex where paths diverge is called apex.**



Funnel

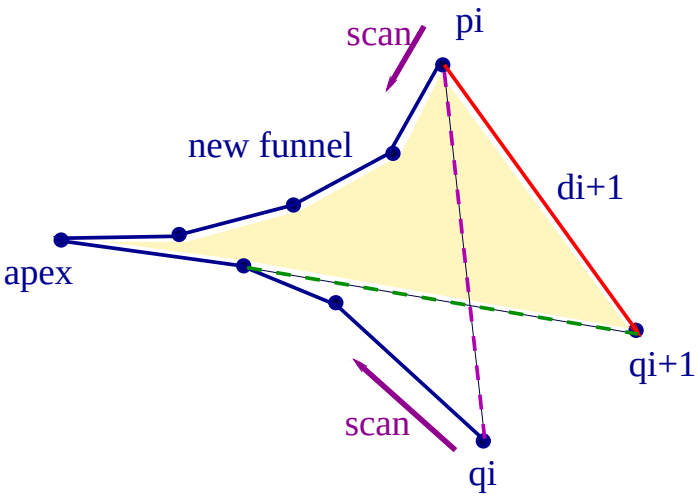


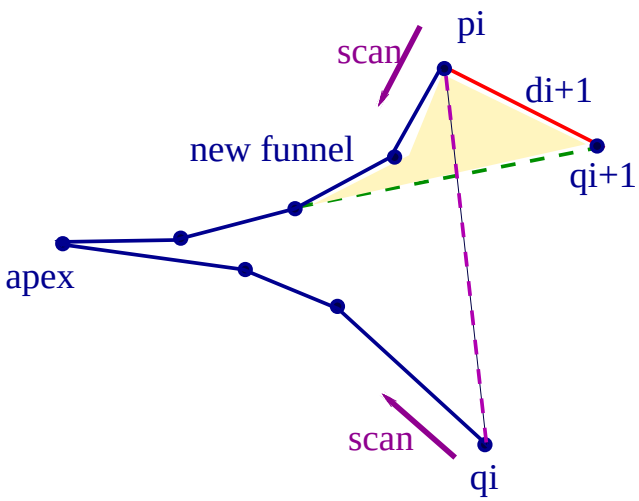
Path Extending



- **Two cases of how to extend the path.**
- **In case I, funnel contracts.**
- **In case II, apex shifts, tail extends, funnel contracts.**
- **In each case, funnel property maintained.**

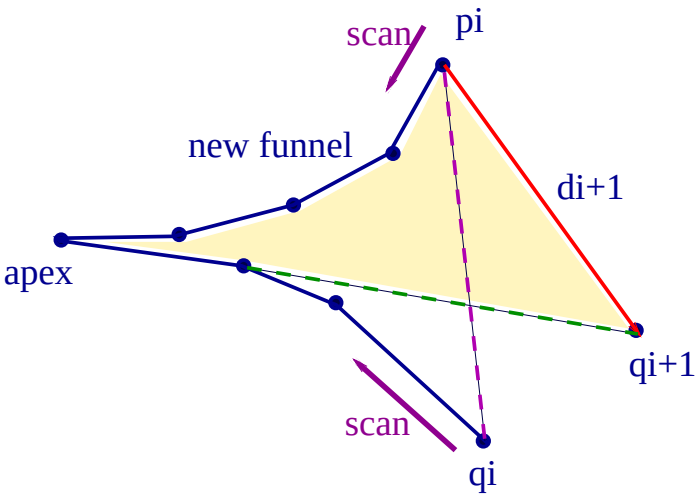
Data Structure & Update

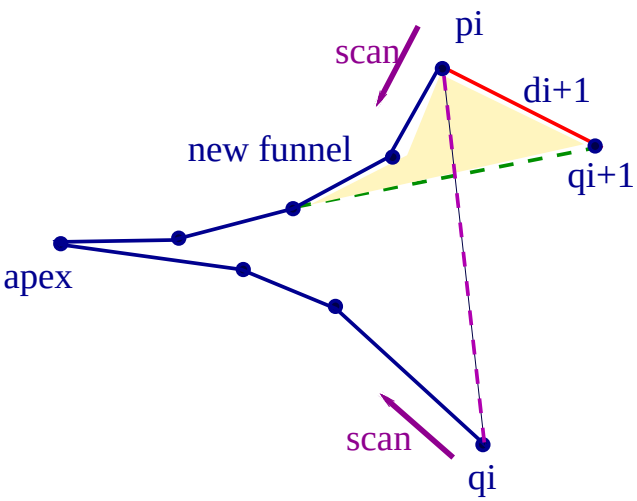




- **How to determine tangent to funnel?**
- **Can't afford to spend $O(n)$ time for each tangent.**
- **Idea: If x edges of funnel are removed by the new tangent, spend $O(x)$ time for finding the tangent.**
- **How to tell a tangent?**

Data Structure & Update





- **Start scanning the funnel from both ends, until tangent determined.**
- **At most $2x + 2$ vertices scanned.**
- **Since each vertex inserted once, and deleted once, total cost for all the tangents is $O(n)$.**
- **Data structure for the funnel: Double-ended queue. Insert/delete in $O(1)$ time.**