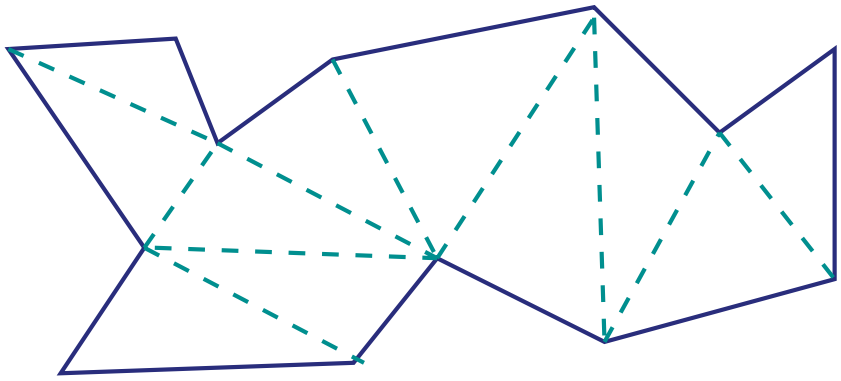
Triangulation

- 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?

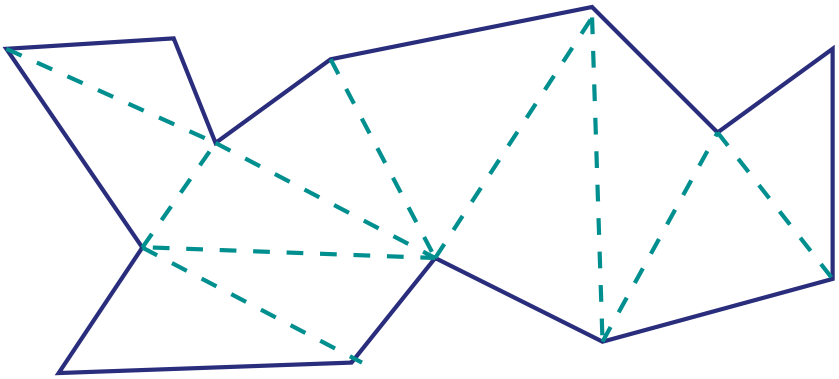


- Triangulation reduces complex shapes to collection of simpler shapes. First step of many advanced algorithms.
- Many applications: visibility, robotics, mesh generation, point location etc.

Triangulation Theorem

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

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



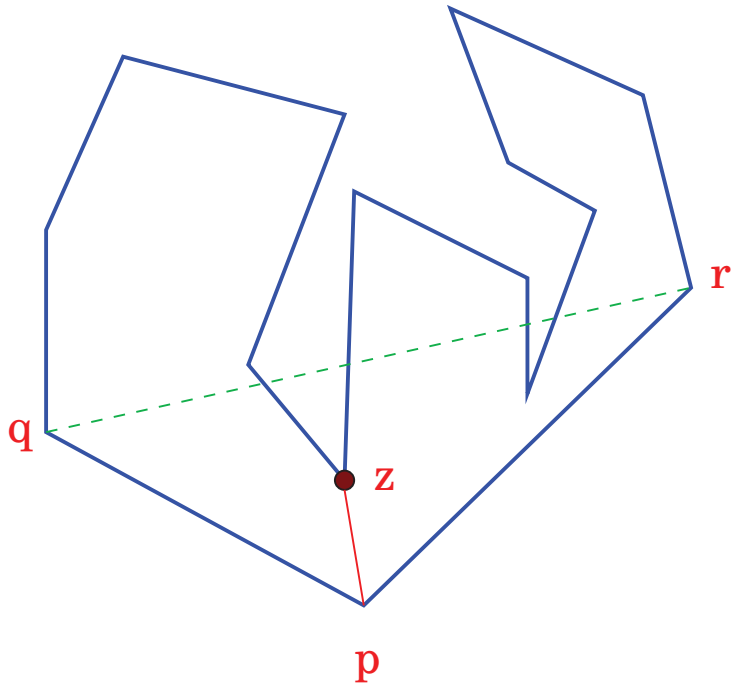
3. **Polygon in picture has $n = 13$, and 11 triangles.**

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

Triangulation: Theory

Theorem: Every polygon has a triangulation.

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

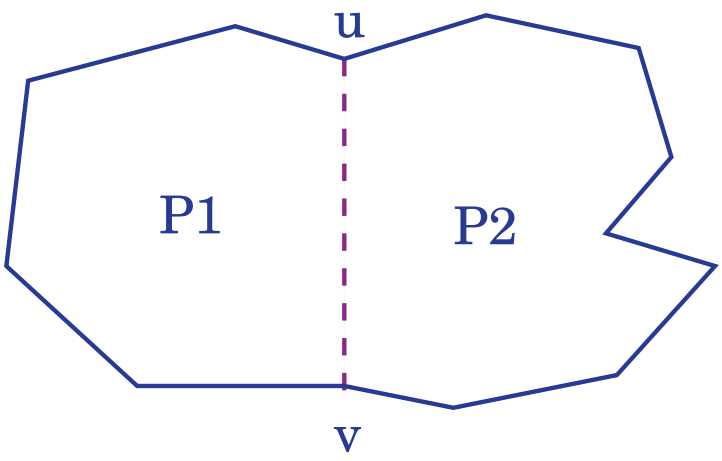


- **Pick a convex corner p. Let q and r be pred and succ vertices.**
- **If qr a diagonal, add it. By induction, the smaller polygon has a triangulation.**
- **If qr not a diagonal, let z be the reflex vertex farthest to qr inside $\triangle pqr$.**
- **Add diagonal pz; subpolygons on both sides have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

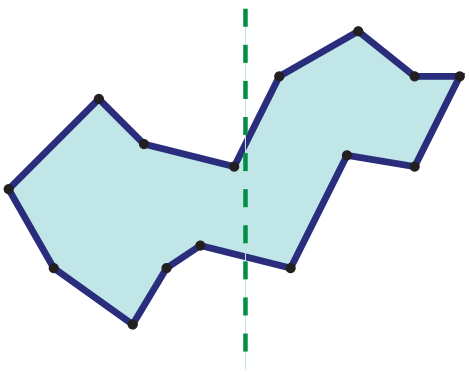
4. **A long series of papers and algorithms in 80s until Chazelle produced an optimal $O(n)$ algorithm in 1991.**

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

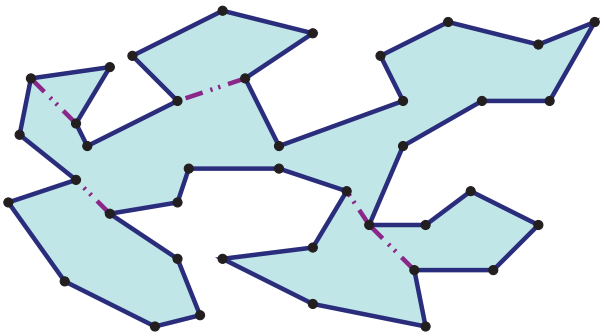
6. **We content ourselves with $O(n \log n)$ algorithm.**

Algorithm Outline

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



x−monotone polygon

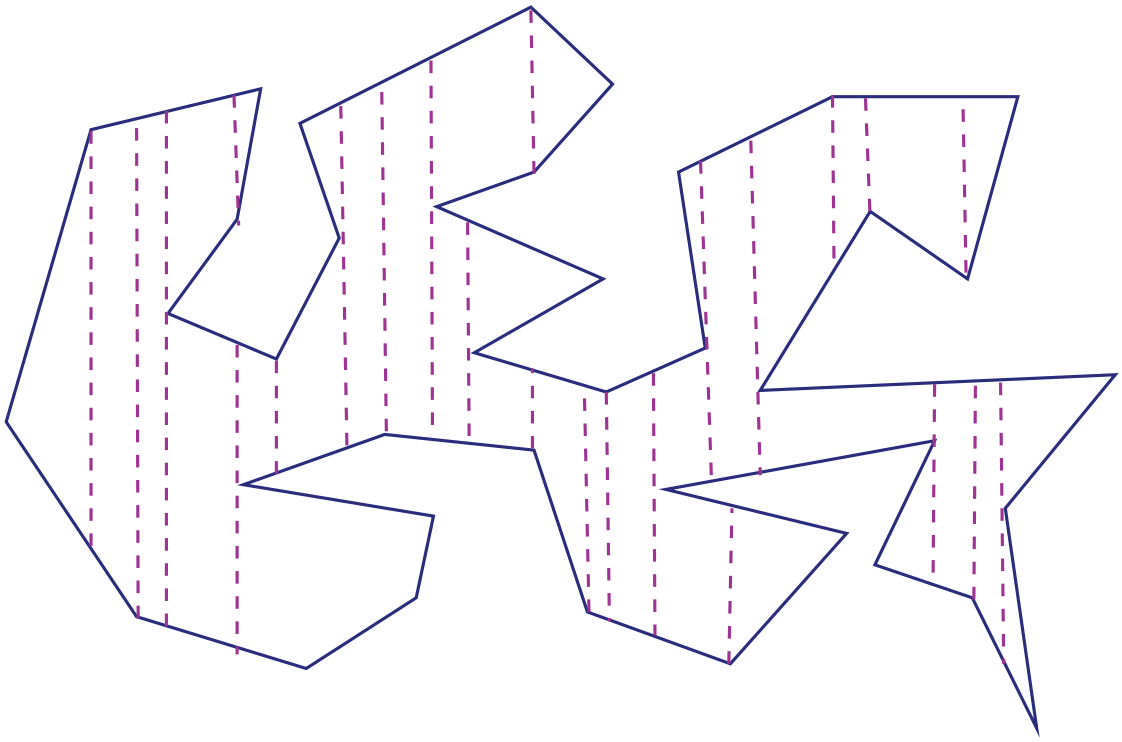


Monotone decomposition

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.**

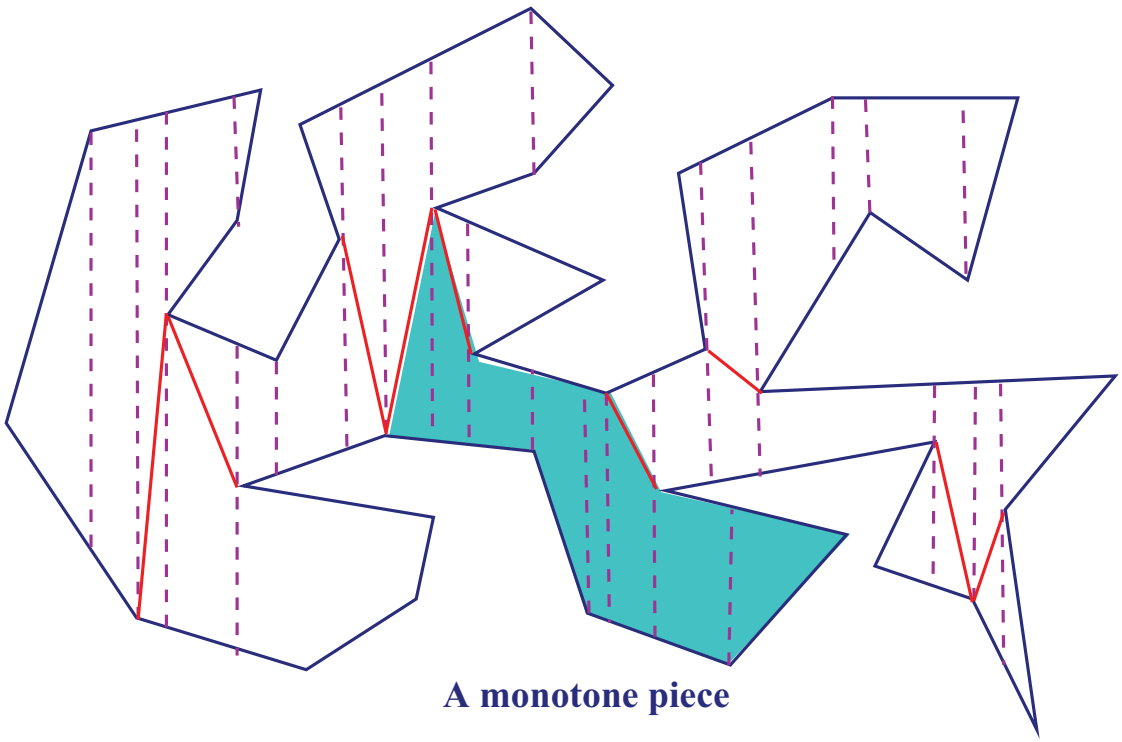
Trapezoidal Decomposition

- **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.

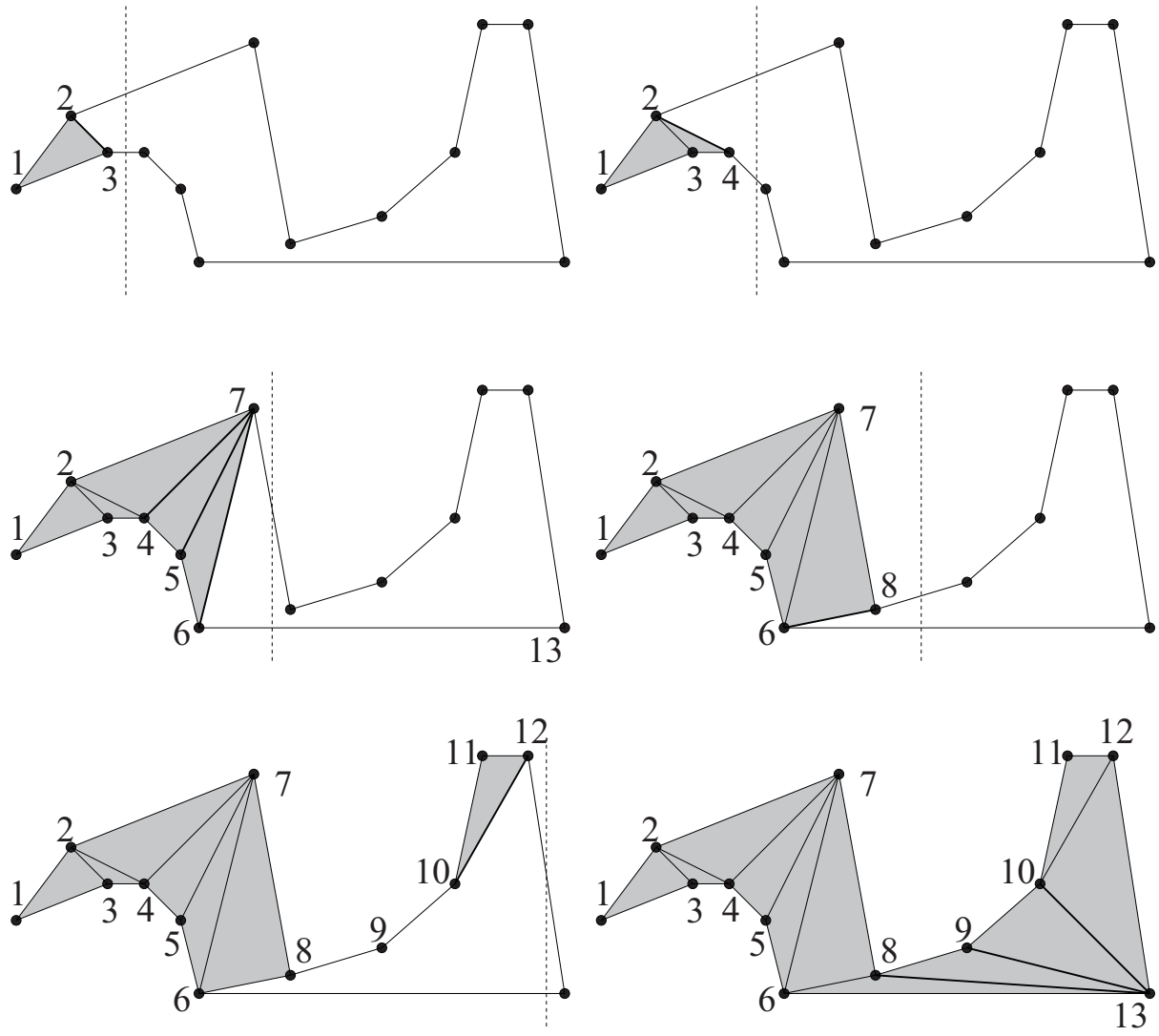


A monotone piece

Monotone Subdivision

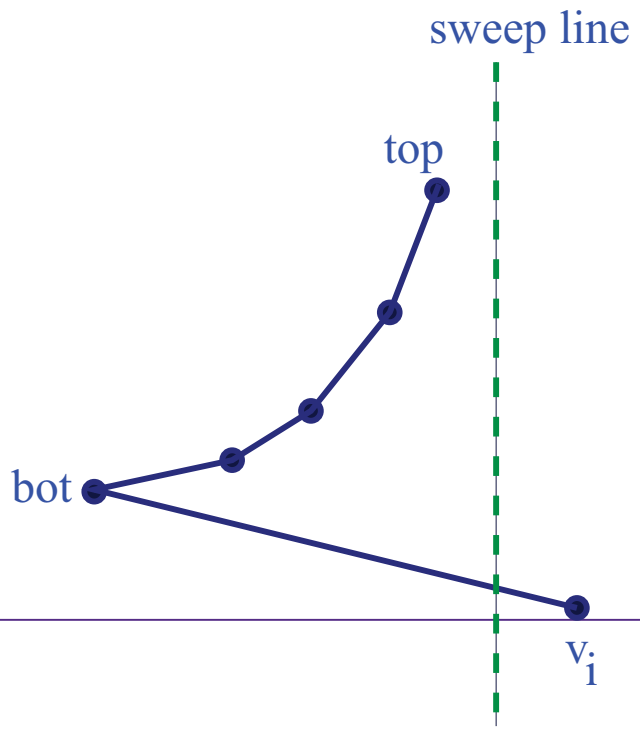
- Assume that trap decomposition represented by DCEL.
- Then, matching vertex for split and merge vertex can be found in $O(1)$ time.
- Remove all trapezoidal edges. The polygon boundary plus new split/merge edges form the monotone subdivision.
- The intermediate trap decomposition is only for presentation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

Triangulation



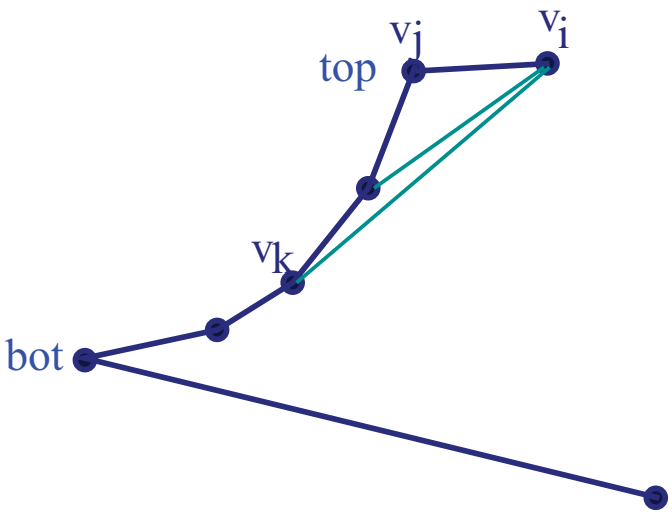
Triangulation

- $\langle v_1, v_2, \ldots, v_n \rangle$ **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



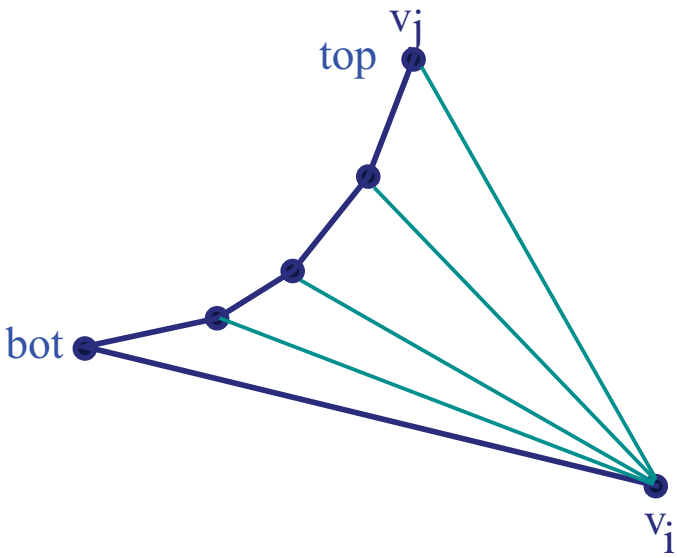
Correctness

- **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)

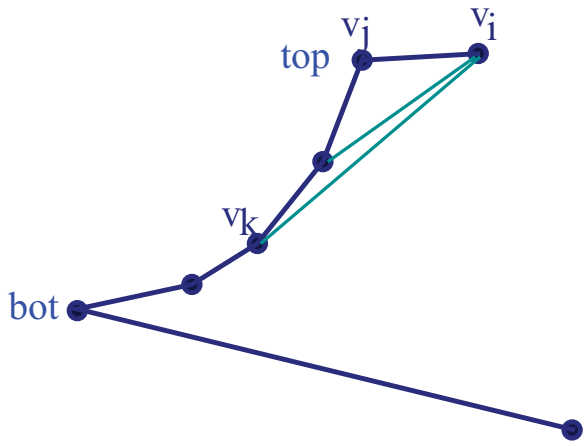
Case I



New stack: (vj, vi)

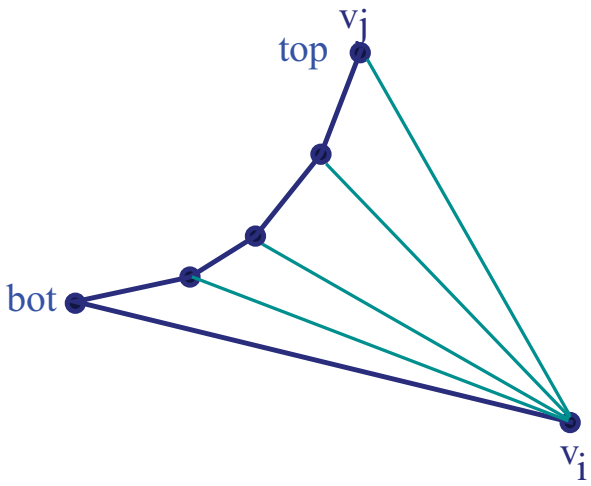
Case II

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