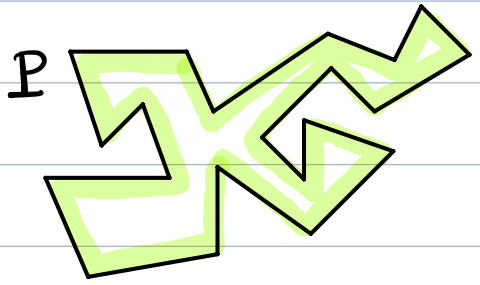
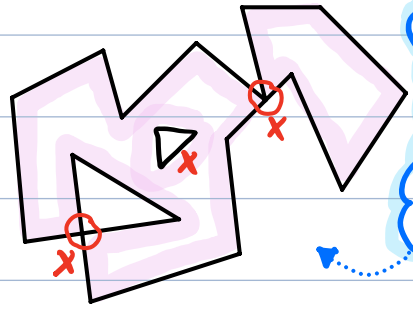
CMSC 754 - Computational Geometry
Lecture 5: Polygon Triangulation

Polygon Triangulation: Given a simple polygon P, (that is, a simple, closed polygonal chain)



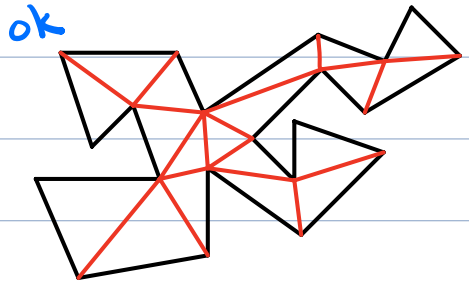
simple polygon

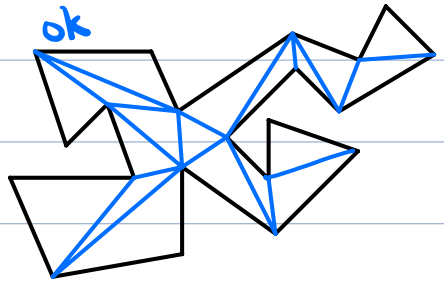


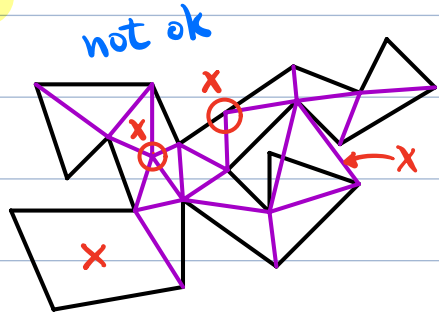
not simple

- boundary does not self intersect
- no "holes"

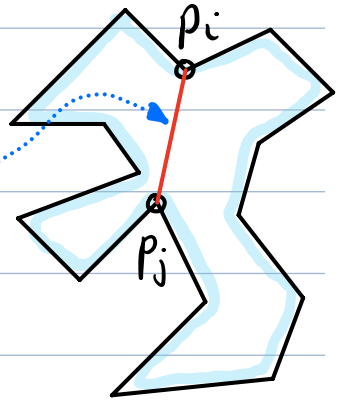
Subdivide P's interior into disjoint triangles using vertices drawn from P's vertices







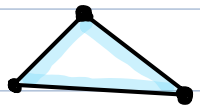
- P given as cyclic sequence of it vertices
- Vertices p_i & p_j are visible if the open segment $\overline{p_i p_j}$ lies within int(P)
- If p_i & p_j are visible, the segment $\overline{p_i p_j}$ is called a diagonal of P



Lemma: For any n-vertex simple polygon ($n \geq 3$):

- A triangulation exists
- Any triangulation has $n-3$ diagonals
- Any triangulation has $n-2$ triangles

Proof (Sketch):



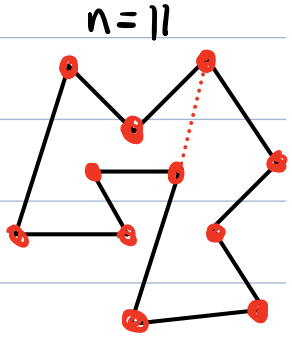
By induction on n.

Basis ($n=3$): All 3 claims are trivially true

Step ($n \geq 4$):

Claim: Every simple polygon with $n \geq 4$ vertices has at least one diagonal

Proof: (Exercise)



Adding this diagonal splits P into polygons $P_1 + P_2$ with $n_1 + n_2$ vertices where $n_1 + n_2 = n + 2$, $3 \leq n_1, n_2 < n$

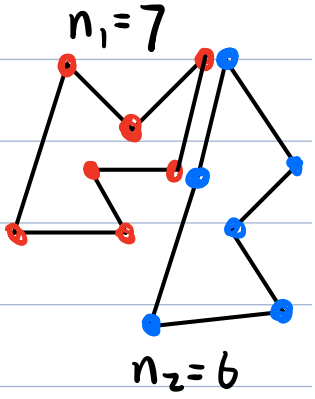
Applying induction we have:

Diagonals: $(n_1 - 3) + (n_2 - 3) + 1$

$= (n_1 + n_2 - 2) - 3 = n - 3$ ✓

Triangles: $(n_1 - 2) + (n_2 - 2)$

$= (n_1 + n_2 - 2) - 2 = n - 2$ ✓



□

This all seems rather obvious...

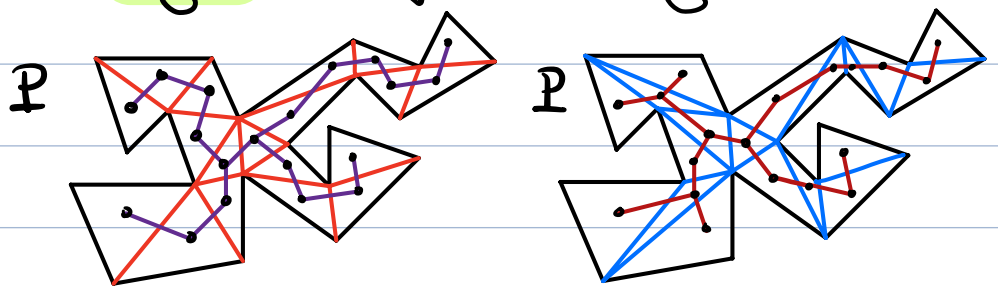
- But it all fails in $\mathbb{R}^3$!
  - Tetrahedralization may not exist (without adding additional vertices)
  - Different tetrahedralizations have different numbers of tetrahedra.

Try to see if you create one!

Dual Graph: A triangulation defines a graph structure

Vertices ← Triangles

Edges ← Adjacent triangles



The dual graph of a polygon triangulation

connected + acyclic (why?) ⇒ a tree

History of Polygon Triangulation

$O(n^2)$ - Easy - Find any diagonal in $O(n)$ + recurse

$O(n \log n)$ - Based on plane sweep

$O(n)$ - Chazelle (1991) - Very complicated !!

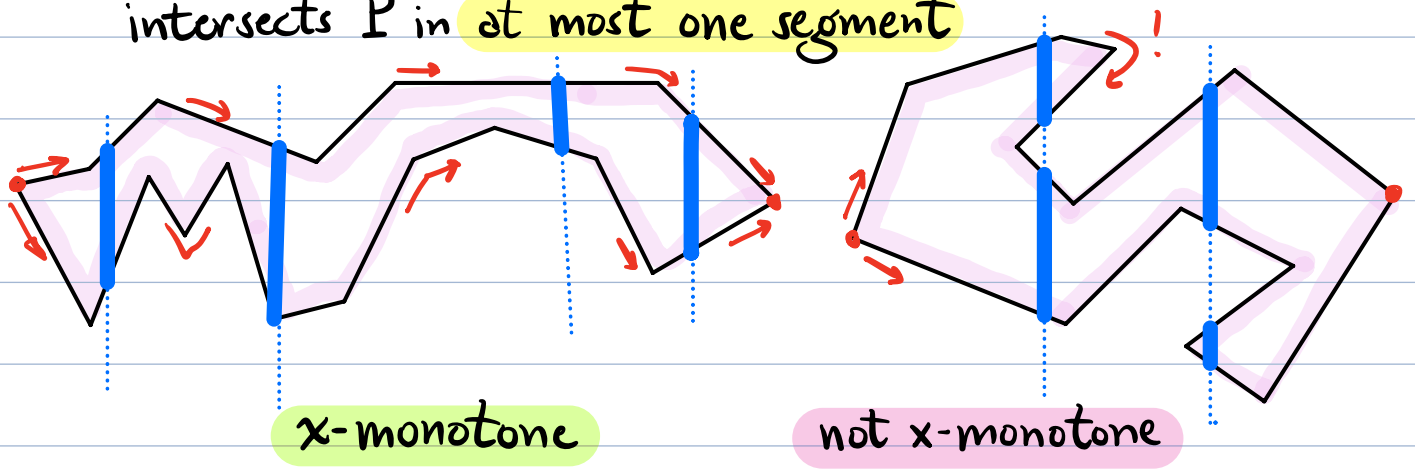
(Holy Grail of comp. geom. - find a simple algorithm)

Our 2-Step Approach:

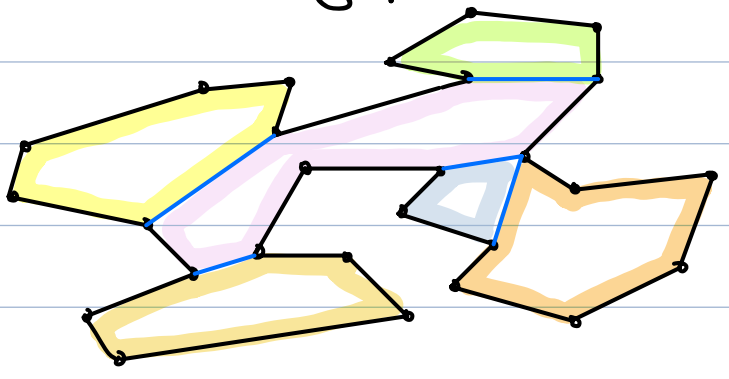
- Decompose P into simpler (x-monotone) pieces
  $O(n \log n)$ time
- Triangulate each monotone piece - $O(n)$ time

Output representation? A general graph structure for planar subdivisions - Doubly-Connected Edge List (DCEL) - (see text for details)

Def: A polygon P is x-monotone if any vertical line intersects P in at most one segment



Monotone Decomposition - Add non-intersecting diagonals so that resulting pieces are all x-monotone



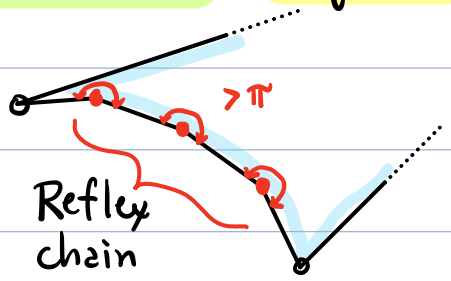
Stage 2: Triangulating an x-Monotone Polygon:

General position assumption - No duplicate x-coords

Terminology:

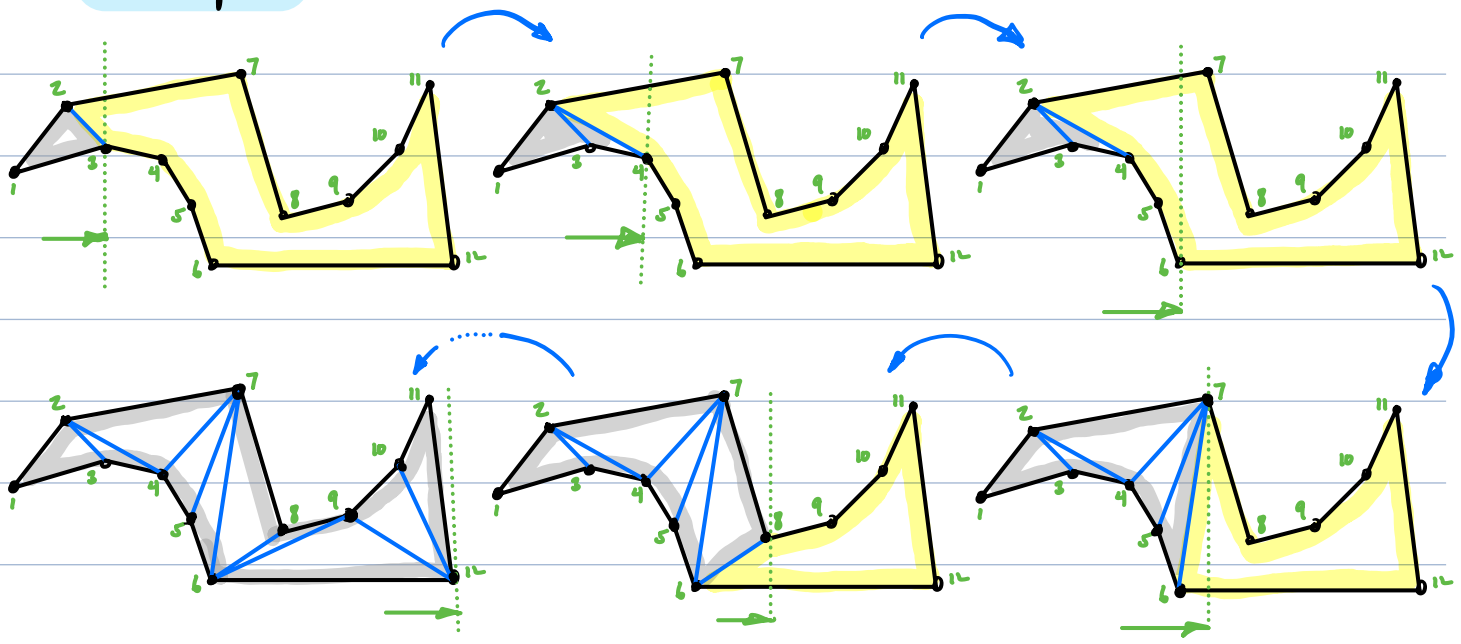
Reflex vertex: A vertex whose **internal angle** $> \pi$

Reflex chain: A **sequence** of (0 or more) reflex vertices



Approach: **Plane sweep** (left to right) + **triangulate as much as we can** (based on limited knowledge)

Example:

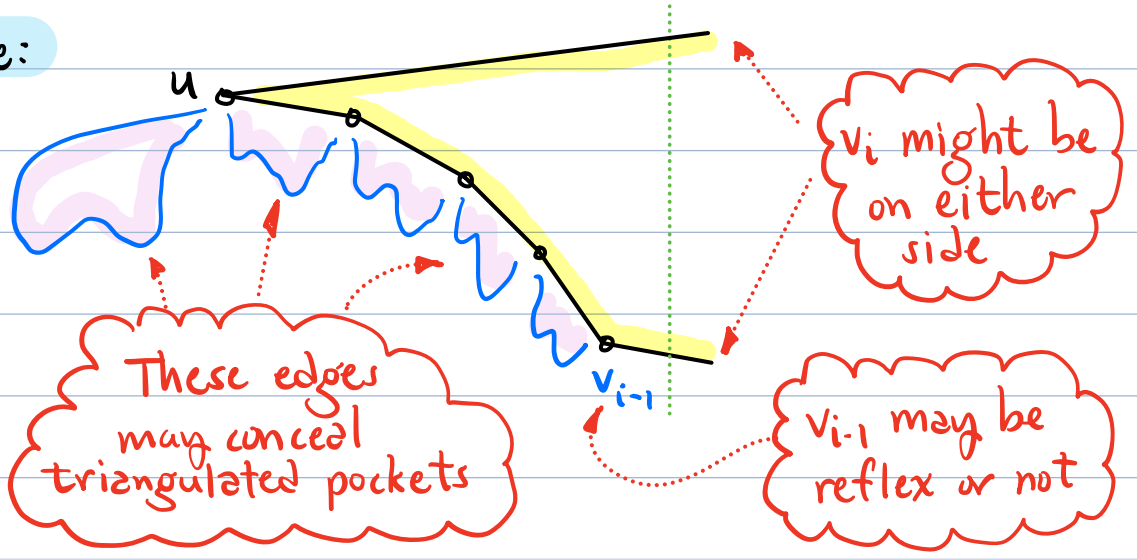


Key to correctness - Get the right **loop invariant**

Lemma: For $i \geq 2$, let v_i be the next vertex to process. The untriangulated region to the left of v_i consists of two x-monotone chains, both starting from a common vertex u:

- one is a single edge
- the other is a reflex chain (of ≥ 1 edges)

Picture:

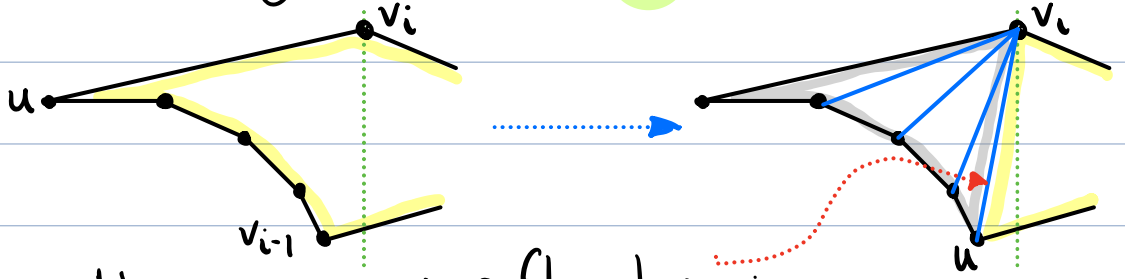


Processing:

Case 1: (v_i lies on the single edge side)

Claim: All vertices on the reflex chain are visible to v_i

- Add diagonals between v_i + all chain vertices

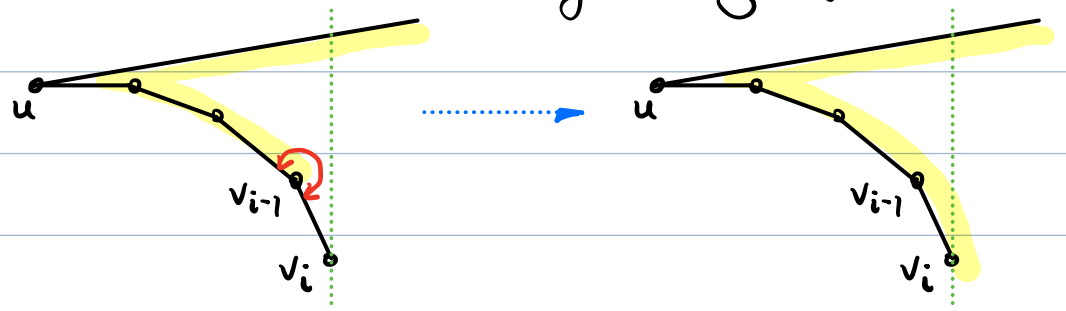


- Now, $u \leftarrow v_{i-1}$ + reflex chain is the single edge $u v_i$

Case 2: v_i lies along the reflex chain

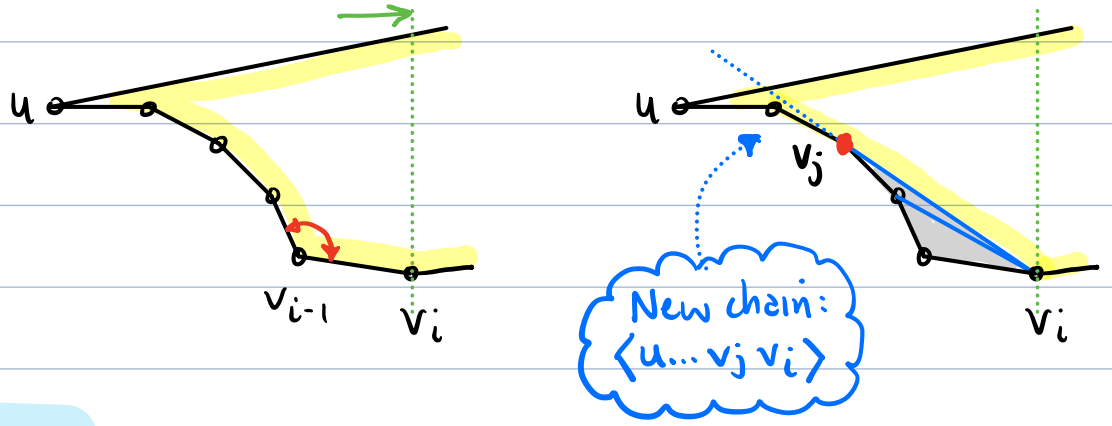
Subcase 2(b): v_{i-1} is reflex

- Extend the chain by adding v_i.



Subcase 2(a): v_{i-1} is non-reflex

- Walk backwards along the chain (as in Graham's algorithm) adding diagonals from v_i until finding a vertex v_j of tangency



Correctness:

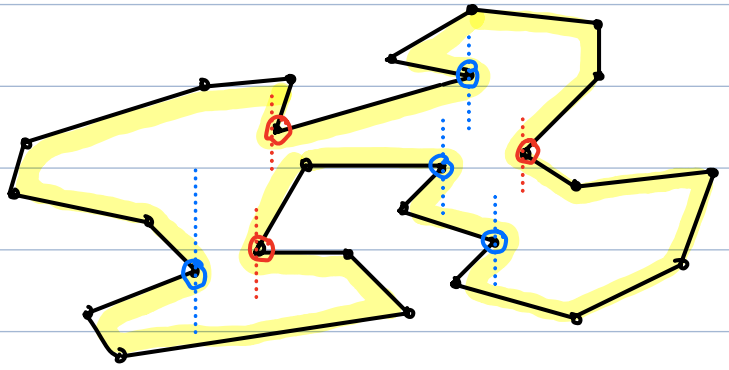
Claim: The main invariant (from previous lemma) holds after every vertex is added. The last vertex satisfies case 1 ($\Rightarrow$ Everything is triangulated.)

Running Time: Time per step ~ # no. of new diagonals

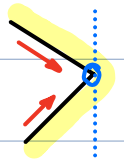
$= n + (n-3) = O(n)$.

Stage 1: Monotone Decomposition

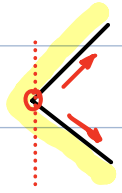
Scan reflex vertex - A reflex vertex (angle $> \pi$) where both edges are on same side of a vertical line through the vertex.



Types:

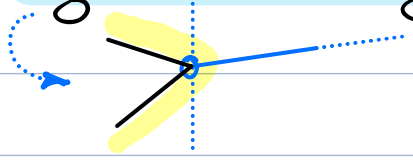


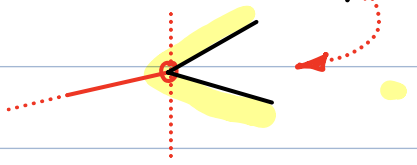
Merge vertex (edges converge)



Split vertex (edges diverge)

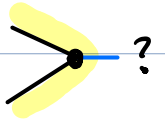
Plan: Add a set of disjoint diagonals, to the right of each merge and left of each split



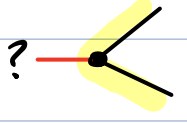


Main Issues:

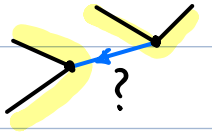
- When we see a merge vertex, we cannot see the future to know where to connect the diagonal



- When we see a split vertex, need to connect to some prior vertex - which one?



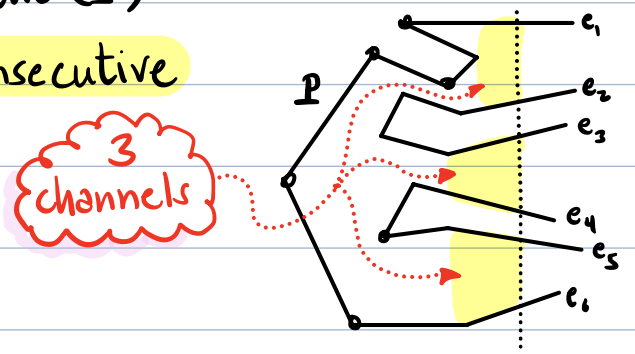
- When we see any vertex, we may have a pending merge vertex, waiting to connect.



- How do we manage all this efficiently?

- We need to save some auxilliary info to help identify diagonals.

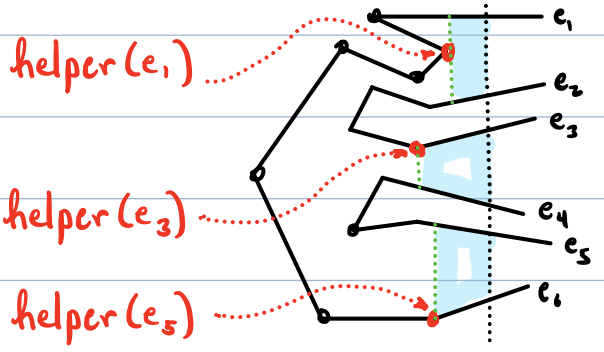
Channel - The region of int(P) lying between two consecutive edges on sweep line



Each channel is associated with its upper edge.

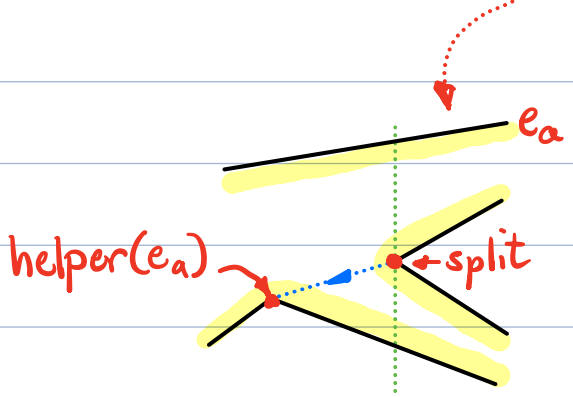
Helper -

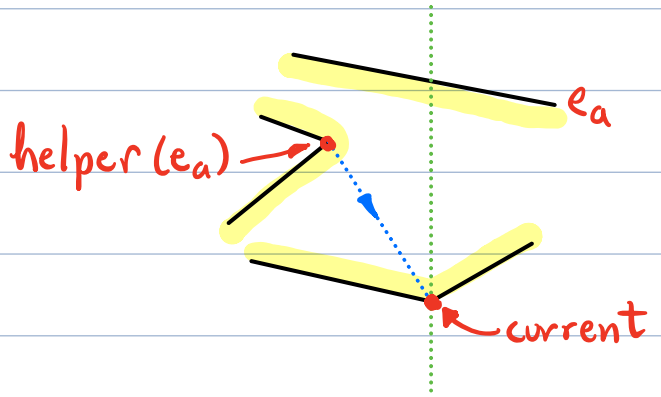
Given a channel with upper edge e, define helper(e) = rightmost vertex in the channel to the left of the current sweep line.



Why the helper helps?

- When we see a split vertex, we create a diagonal going back to the helper





- When the helper is a merge vertex, we add a diagonal to the next vertex in channel

Sweepline Algorithm for Monotone Subdivision:

Events: Polygon vertices (sorted by x)

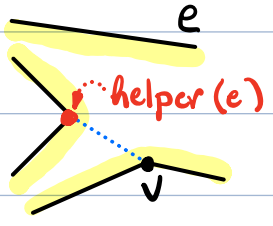
Sweep-line status:

- Edges crossing the sweep line
- Helper values for each upper edge

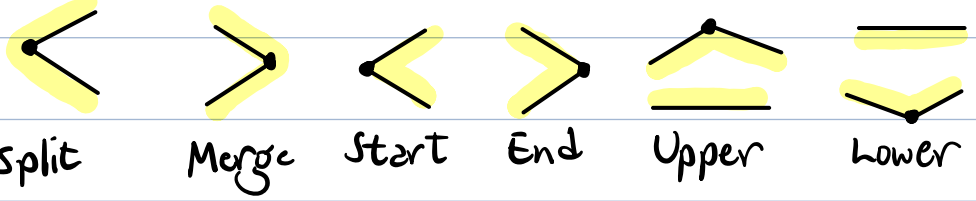
Utility function: Given vertex v + edge e above it

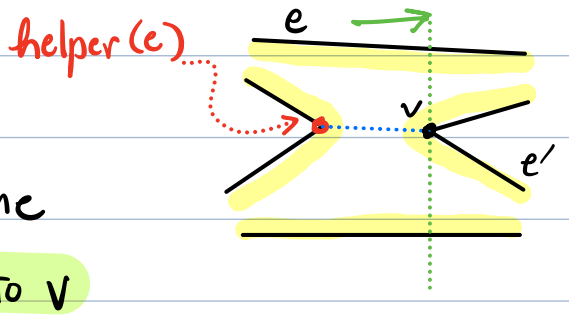
fix-up(v,e)

```
if (helper(e) is a merge vertex)
    add diagonal between helper(e) + v
```



Processing cases - Depends on type of vertex.



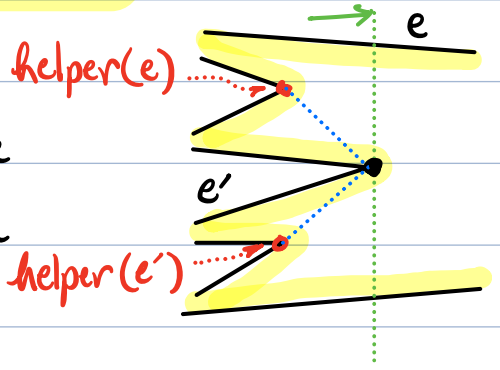


Split Vertex (v):

- e ← edge above v in sweep line
- add diagonal from helper(e) to v
- insert the two edges incident to v into sweep line
- letting e' denote the lower of these, set helper(e') ← v

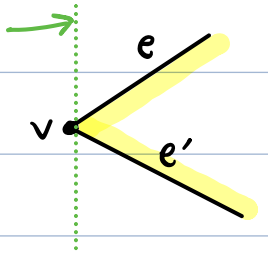
Merge Vertex (v):

- Consider the two edges incident to v.
  Let e' be the lower one
- Delete both edges from sweep line
- Let e be edge above v in sweep line
- fix-up (v, e) ; fix-up (v, e')



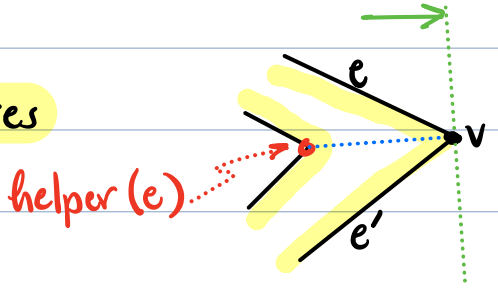
Start Vertex (v):

- Let e & e' be upper & lower edges
  incident to v, respectively
- Insert both into sweep line
- helper (e) ← v



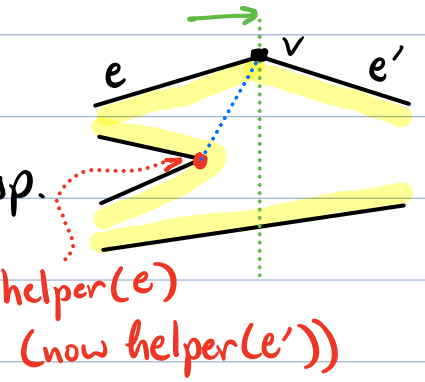
End Vertex (v):

- Let e & e' be upper & lower edges
  incident to v, respectively
- Delete both from sweep line
- fix-up (v, e)



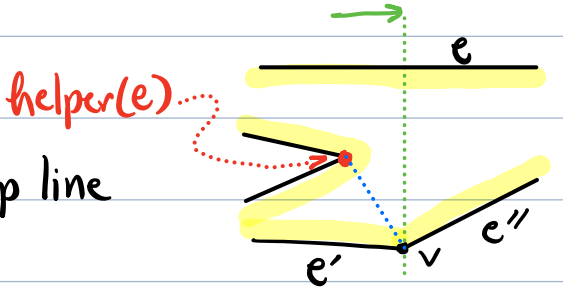
Upper Vertex (v):

- Let e + e′ be v's left + right edges, resp.
- fix-up(v, e)
- Replace e with e′ in sweep line
- helper(e′) ← v



Lower Vertex (v):

- Let e be edge above v in sweep line
- fix-up(v, e)
- Let e′ + e″ be v's left + right edges, resp.
- Replace e′ with e″ in sweep line



Correctness:

- As with any plane sweep this is a (tedious) induction proof showing that invariants are maintained:
  - Sweep line status contains all edges of P crossing the sweep line
  - Helper values for all upper edges are properly maintained
  - All merge vertices have diagonal added to a visible neighboring vertex (fix-up + helper properties)
  - All split vertices have diagonal added to a visible neighboring vertex (helper properties)

- Added diagonals are disjoint (helper properties)

Running time:

- n vertices ⇒ n events
- No need for priority queue, just sort vertices left to right → $O(n \log n)$
- Each event performs $O(1)$ dictionary operations (insert, delete, find, pred, succ) → $O(\log n)$ per event ⇒ $O(n \log n)$ total
- Total time = $O(n \log n)$

Summary:

- Simple polygons + triangulation
  - size + existence
- Triangulating an x-monotone polygon - $O(n)$
- Monotone subdivision - $O(n \log n)$
- Both based on plane sweep.

- Question:
  - If P is not simple (self intersecting), when (if ever) will this be discovered?