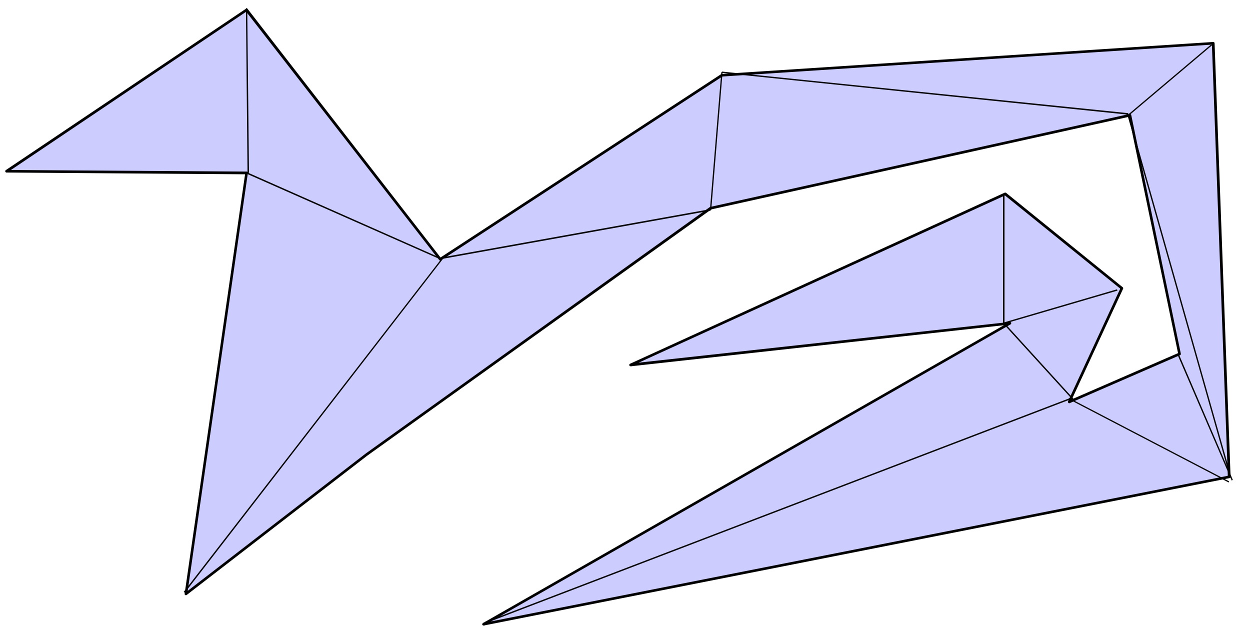
CMPS 3130/6130 Computational Geometry
Spring 2015



Triangulations and Guarding Art Galleries II

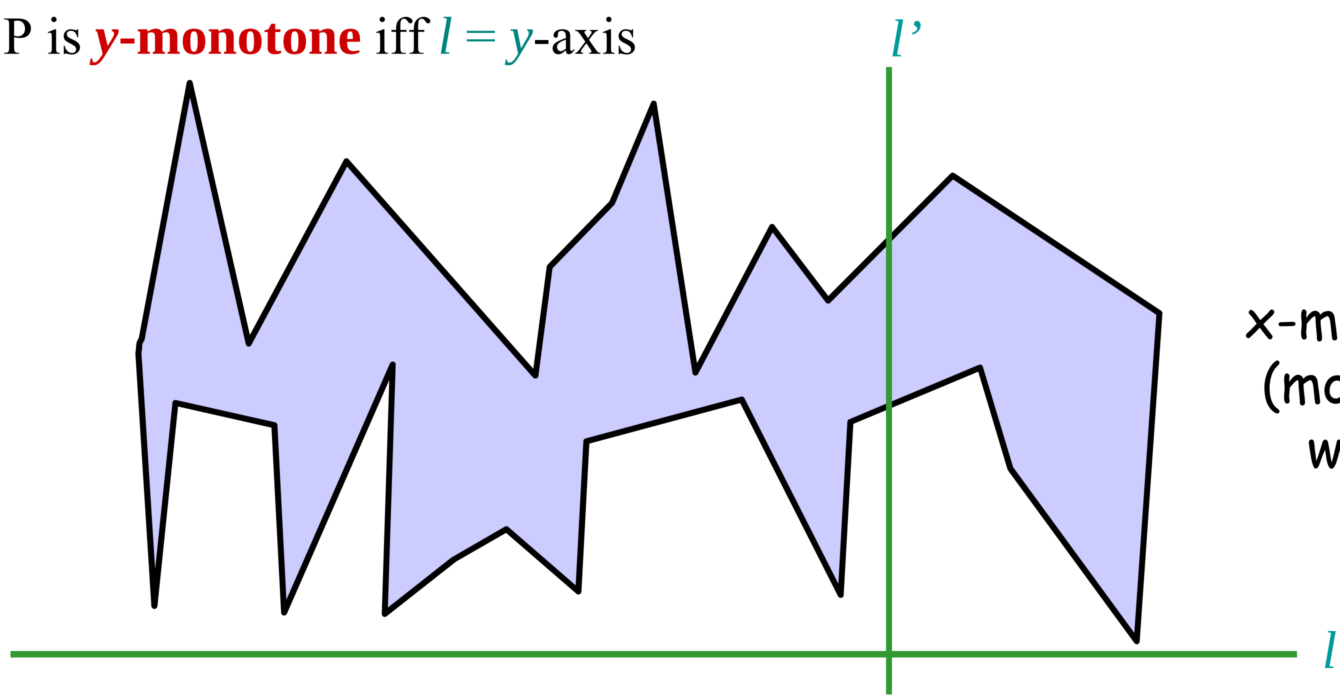
Carola Wenk

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

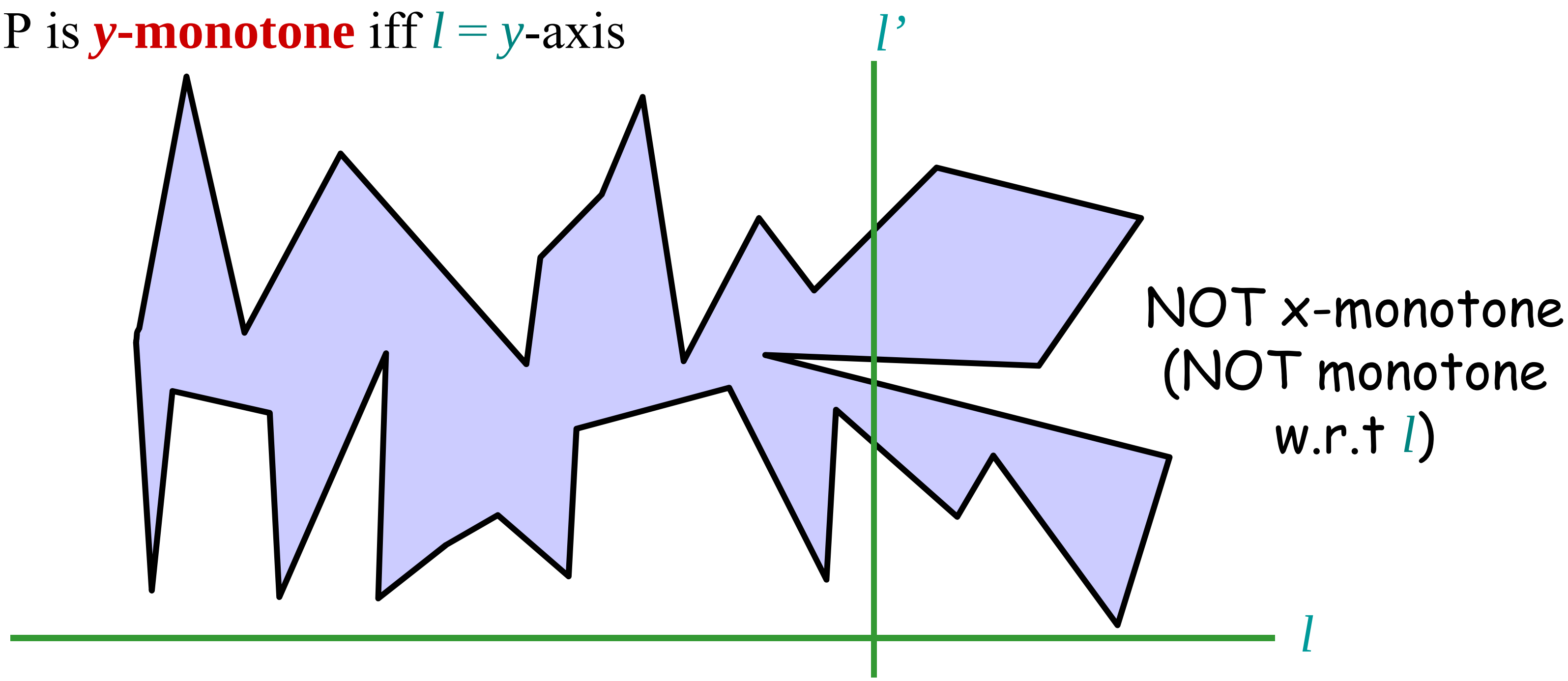
- A simple polygon *P* is called **monotone with respect to a line** *l* iff for every line *l'* perpendicular to *l* the intersection of *P* with *l'* is connected.
  - P is ***x*-monotone** iff *l* = *x*-axis
  - P is ***y*-monotone** iff *l* = *y*-axis



x-monotone (monotone w.r.t *l*)

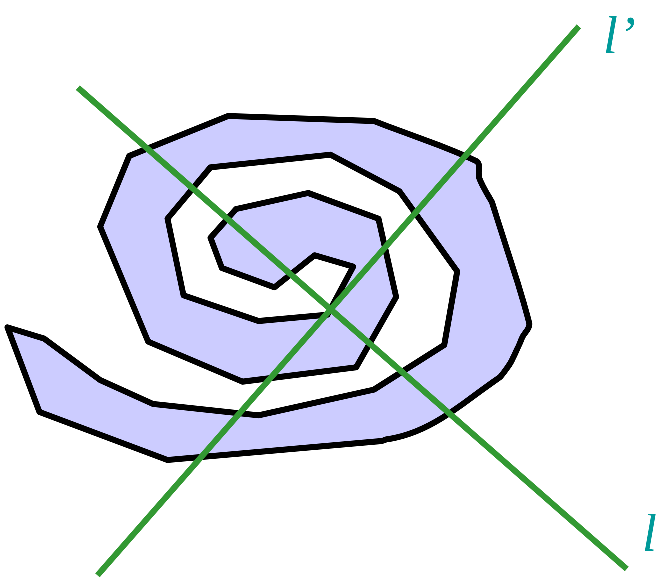
Monotone Polygons

- 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*

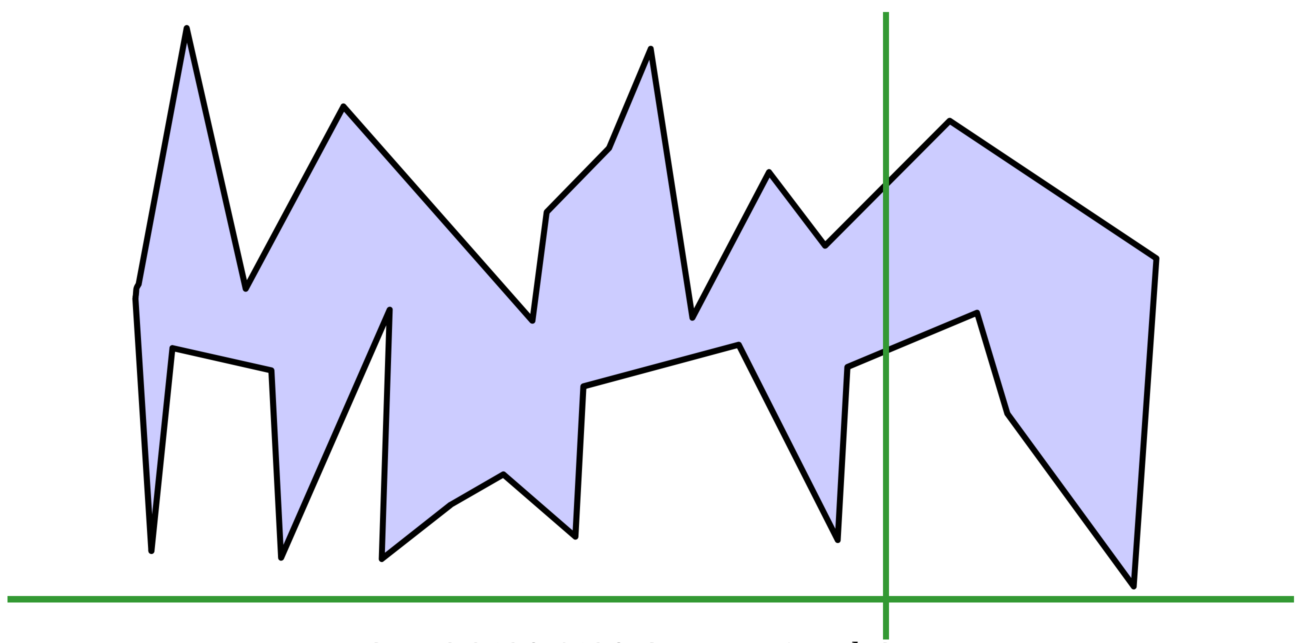
Test Monotonicity

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.

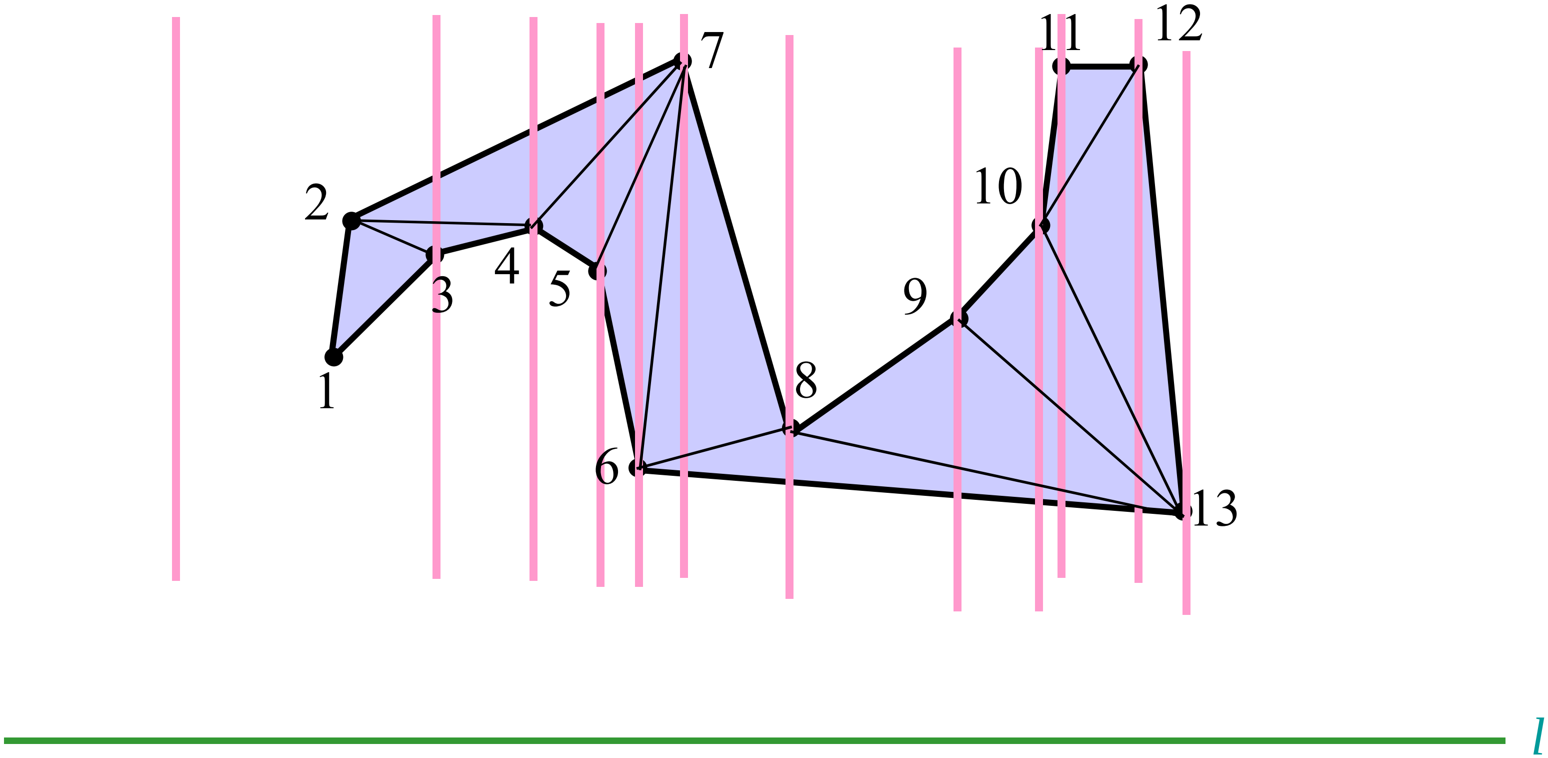


Triangulating a Polygon

- 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)

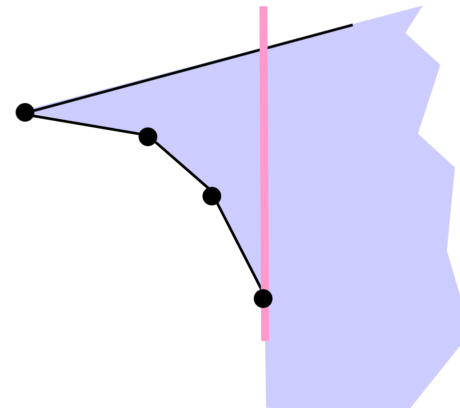
Triangulate an *l*-Monotone Polygon

- 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.

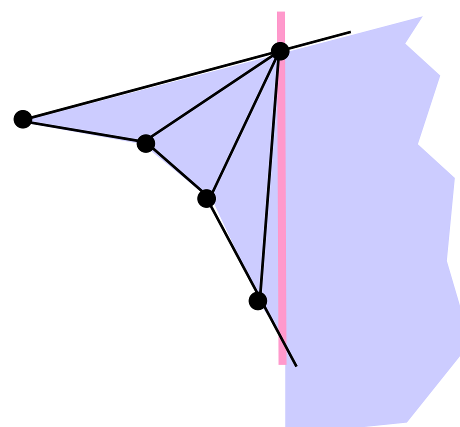


Triangulate an *l*-Monotone Polygon

- 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.

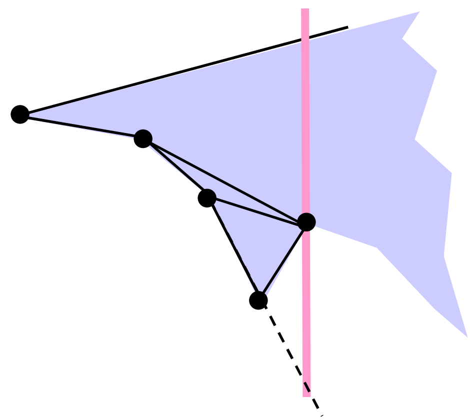


- Update, case 1: new vertex lies on chain opposite of reflex chain. Triangulate.

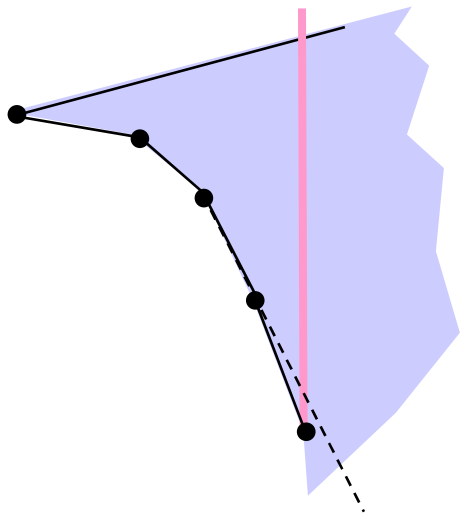


Triangulate an *l*-Monotone Polygon

- 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.

⇒ Constant time per vertex

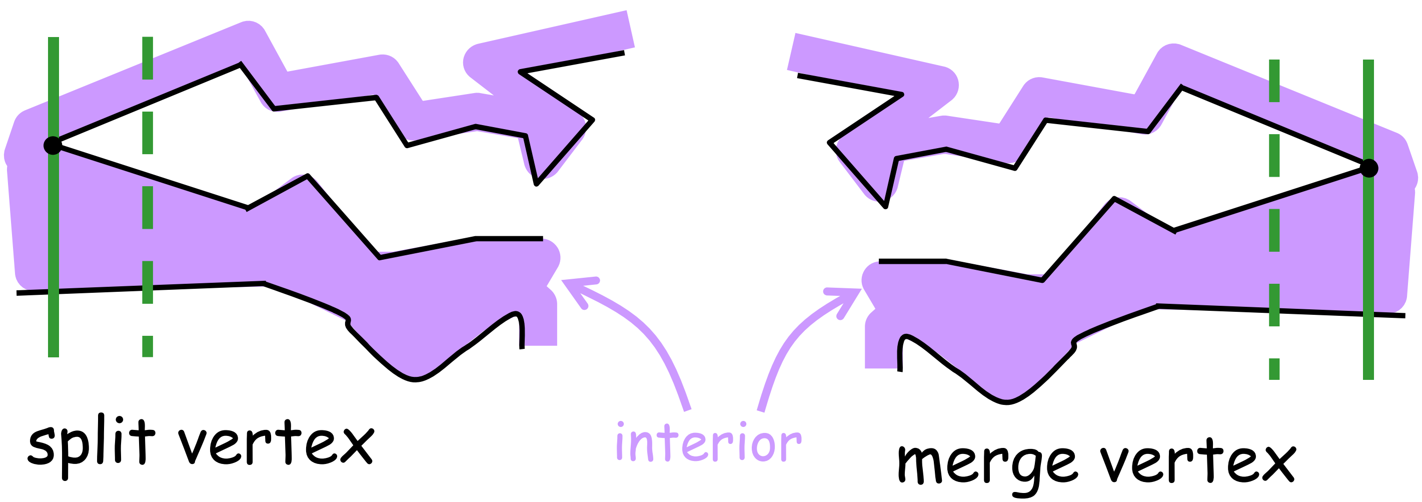
⇒ $O(n)$ total runtime

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)

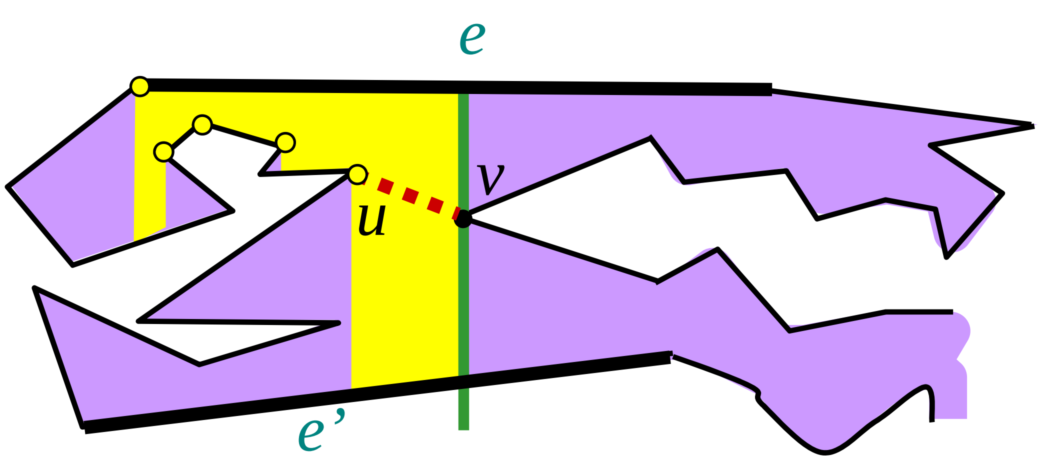
Finding a Monotone Subdivision

- **Monotone subdivision:** subdivision of the simple polygon *P* into monotone pieces
- Use plane sweep to add diagonals to *P* that partition *P* into monotone pieces
- Events at which violation of x-monotonicity occurs:



Helpers (for split vertices)

- **helper(e):** Rightmost vertically visible vertex below *e* on the polygonal chain (left of sweep line) between *e* and *e'*, where *e'* is the polygon edge below *e* on the sweep line.
- Draw diagonal between *v* and helper(*e*), where *e* is the edge immediately above *v*.

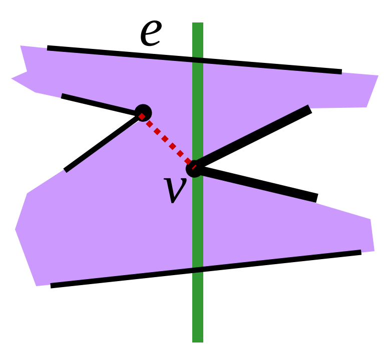


split vertex *v*
u = helper(*e*)

Sweep Line Algorithm

- **Events:** Vertices of polygon, sorted in increasing order by *x*-coordinate. (No new events will be added)
- **Sweep line status:** Balanced binary search tree storing the list of edges intersecting sweep line, sorted by *y*-coordinate. Also, helper(e) for every edge intersecting sweep line.
- Event processing of vertex *v*:

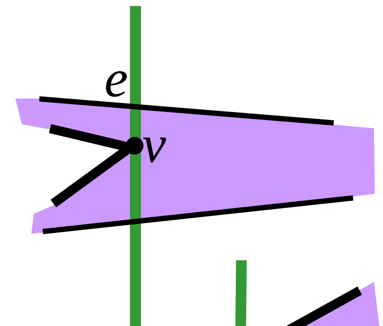
  1. **Split vertex:**
     - Find edge *e* lying immediately above *v*.
     - Add diagonal connecting *v* to helper(*e*).
     - Add two edges incident to *v* to sweep line status.
     - Make *v* helper of *e* and of the lower of the two edges



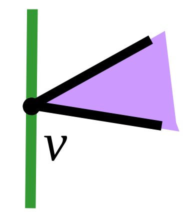
Sweep Line Algorithm

- Event processing of vertex *v* (continued):

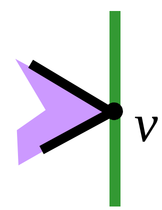
2. **Merge vertex:**
   - Delete two edges incident to *v*.
   - Find edge *e* immediately above *v* and set helper(*e*)=*v*.



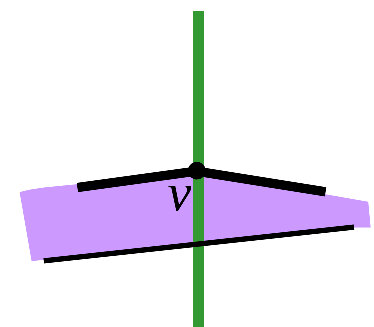
3. **Start vertex:**
   - Add two edges incident to *v* to sweep line status.
   - Set helper of upper edge to *v*.



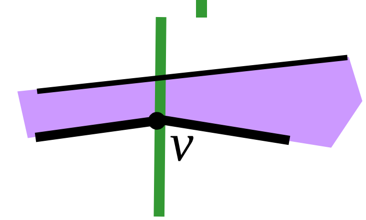
4. **End vertex:**
   - Delete both edges from sweep line status.



5. **Upper chain vertex:**
   - Replace left edge with right edge in sweep line status.
   - Make *v* helper of new edge.



6. **Lower chain vertex:**
   - Replace left edge with right edge in sweep line status.
   - Make *v* helper of the edge lying above *v*.



Sweep Line Algorithm

- Insert diagonals for merge vertices with “reverse” sweep
- Each update takes $O(\log n)$ time
- There are n events

→ Runtime to compute a monotone subdivision is $O(n \log n)$