

CS60064

Spring 2022

Computational Geometry

Instructors

Bhargab B. Bhattacharya (BBB)

Partha Bhowmick (PB)

Lecture 07

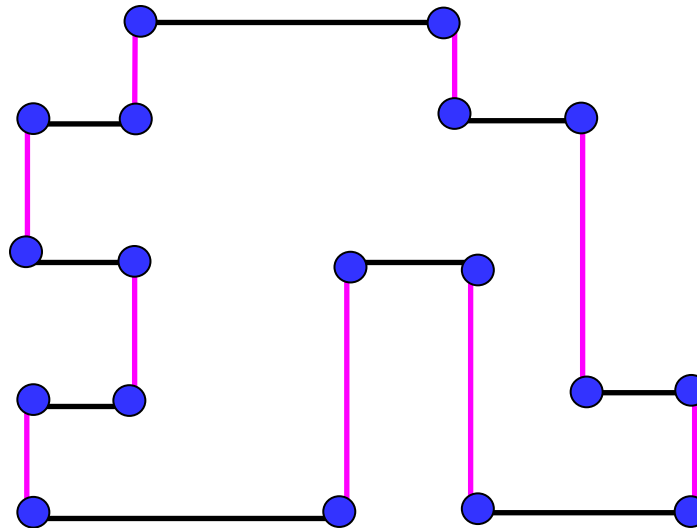
19 January 2022

Indian Institute of Technology Kharagpur
Computer Science and Engineering

Orthogonal Polygonization

All edges are axis-parallel

In other words, internal turn angles are either $\pi/2$ or $3\pi/2$



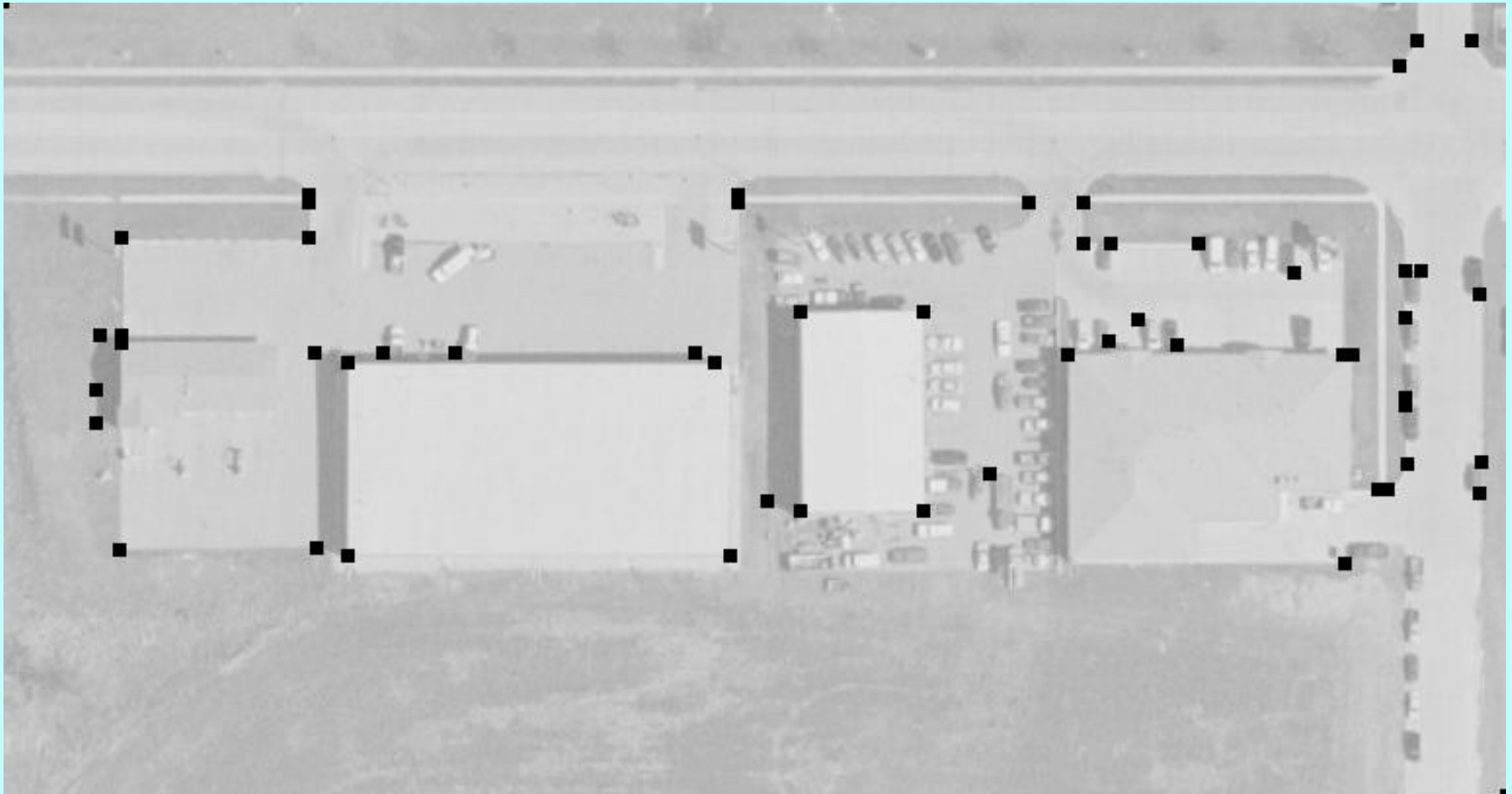
orthogonal polygonization
(from unlabelled vertices to polygon)

Applications of Orthogonal Polygonization: Remote Sensing and GIS



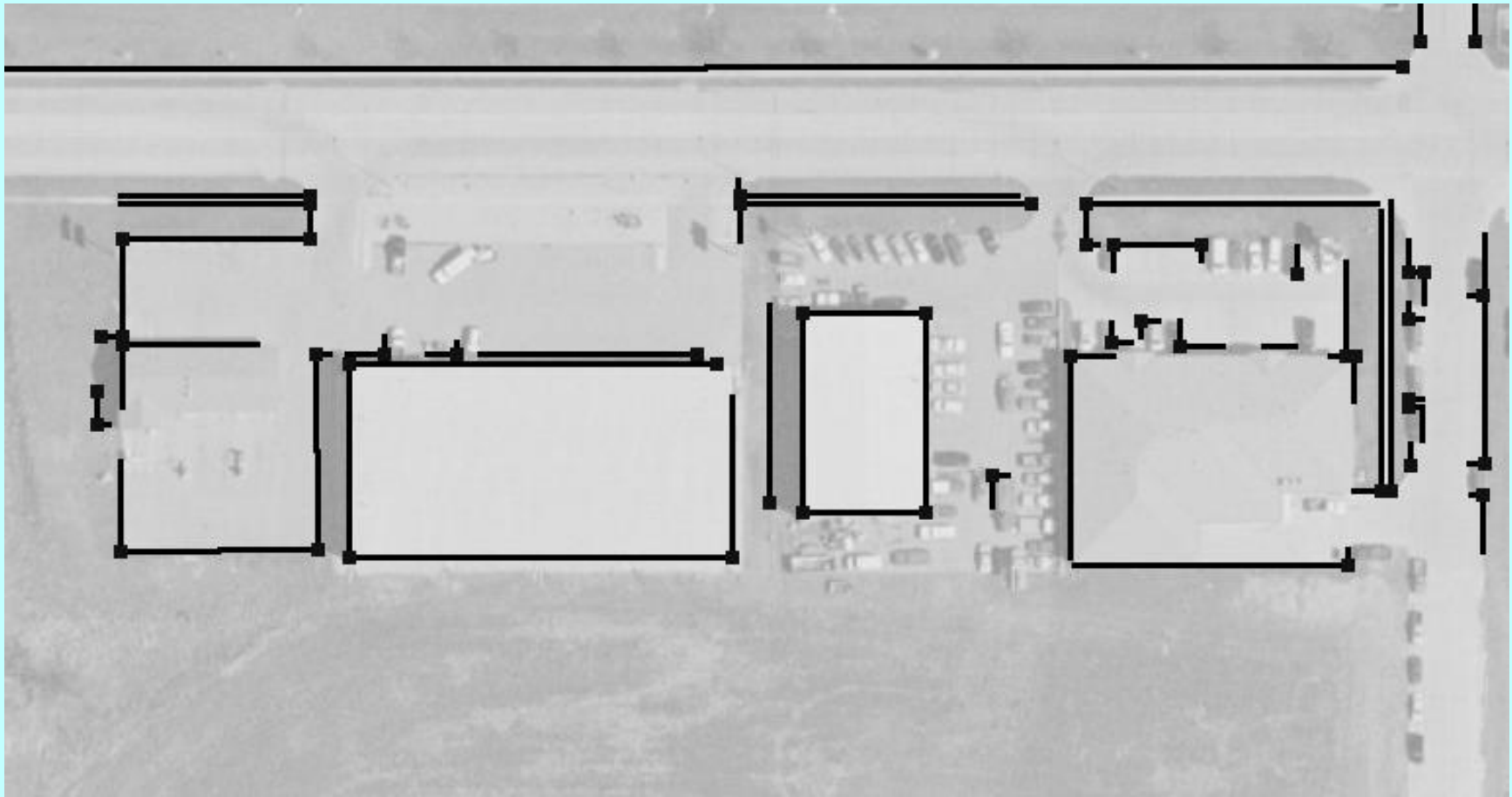
Aerial Image

Image to Corner Data



Data Compression

Reconstruction via Orthogonal Polygonization



Aerial Image



Image to Corner Data

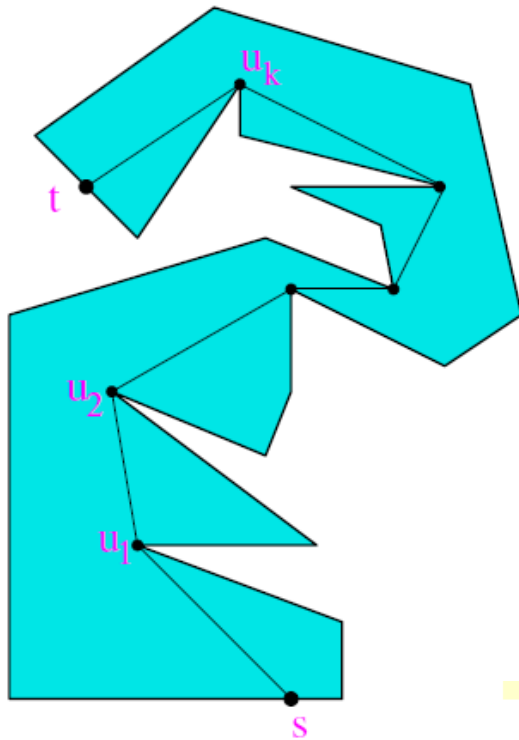


Reconstruction via Orthogonal Polygonization

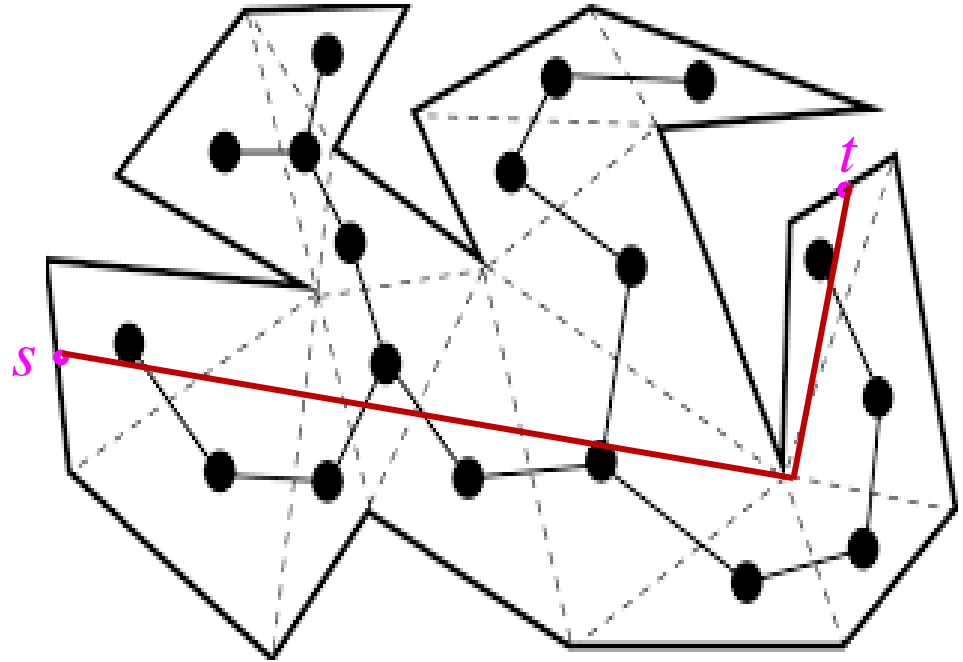


Problem of the Day

Euclidean Shortest Path (ESP) in a Polygon



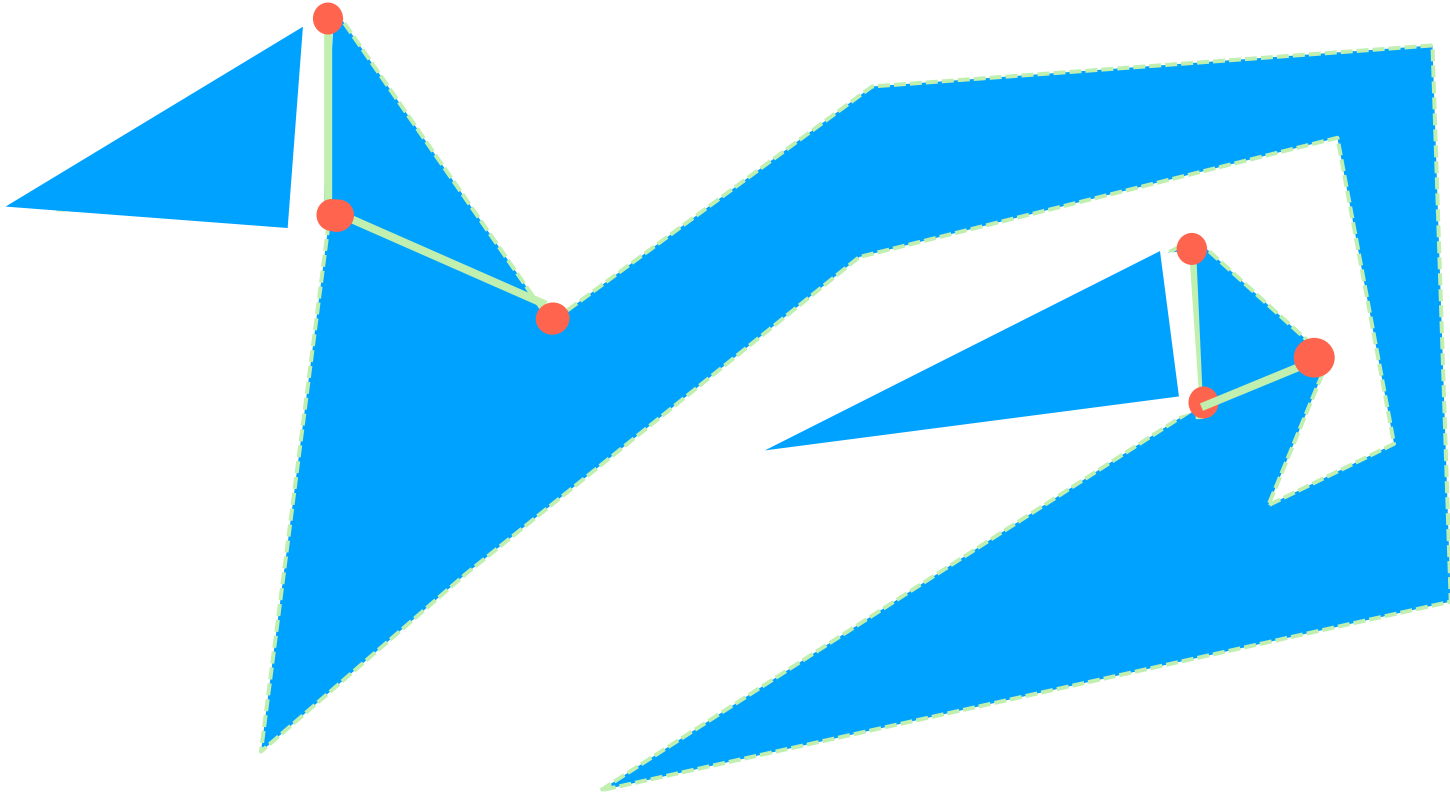
ESP between s and t



- The dual of a triangulated polygon is a tree

- ESP comprises a sequence of straight edges, joining only at the vertices of the polygon;
- The triangles intercepted by the ESP appear as vertices on the path of the dual (tree) that connects the two triangles corresponding to s and t

Recap: Triangulation via Ear-Clipping Algorithm



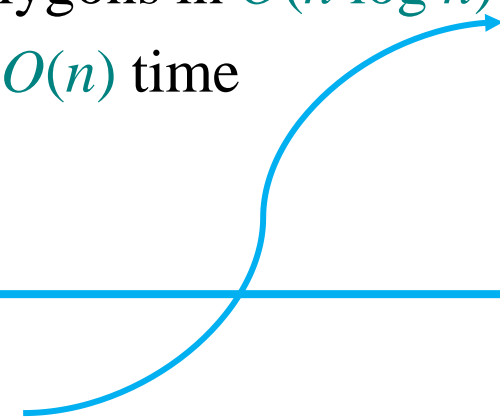
Every polygon has at least two ears!

Find an ear, fix a diagonal, chop the ear and iterate

Triangulating a polygon: Improvements

- Simple $O(n^2)$ time algorithm based on Meister's two-ear theorem
- Long-standing open problem: can it be done in $O(n)$ time?
- [Chazelle 1991] proposed an $O(n)$ time algorithm, very intricate and impractical to implement
- A more practical $O(n \log n)$ time algorithm \rightarrow needs two steps:
 1. Partition the polygon into *monotone* polygons in $O(n \log n)$ time
 2. Triangulate each monotone polygon in $O(n)$ time \Rightarrow total time complexity: $O(n \log n)$

Today's agenda



Monotone Curves



y

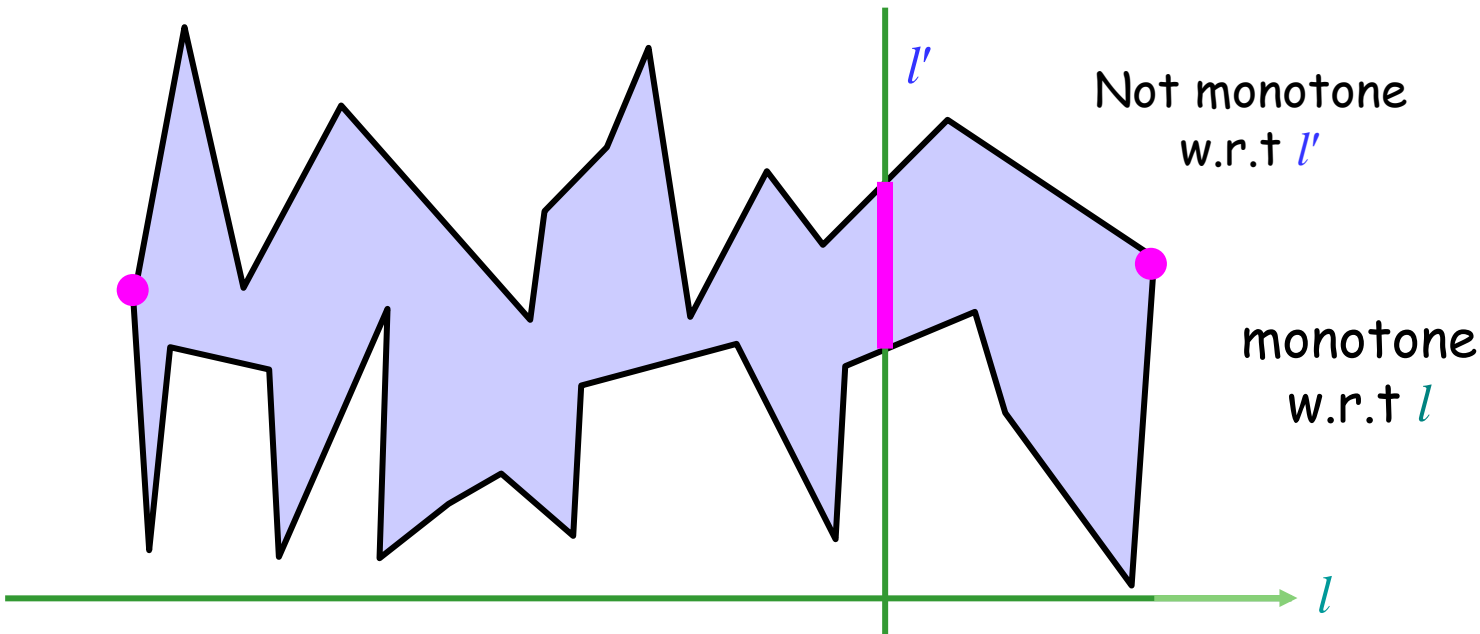


Monotone Polygons

- Two monotone chains between two extremal points along l ;
- l -coordinates are *non-decreasing* when you travel along the chains of P in the direction of l

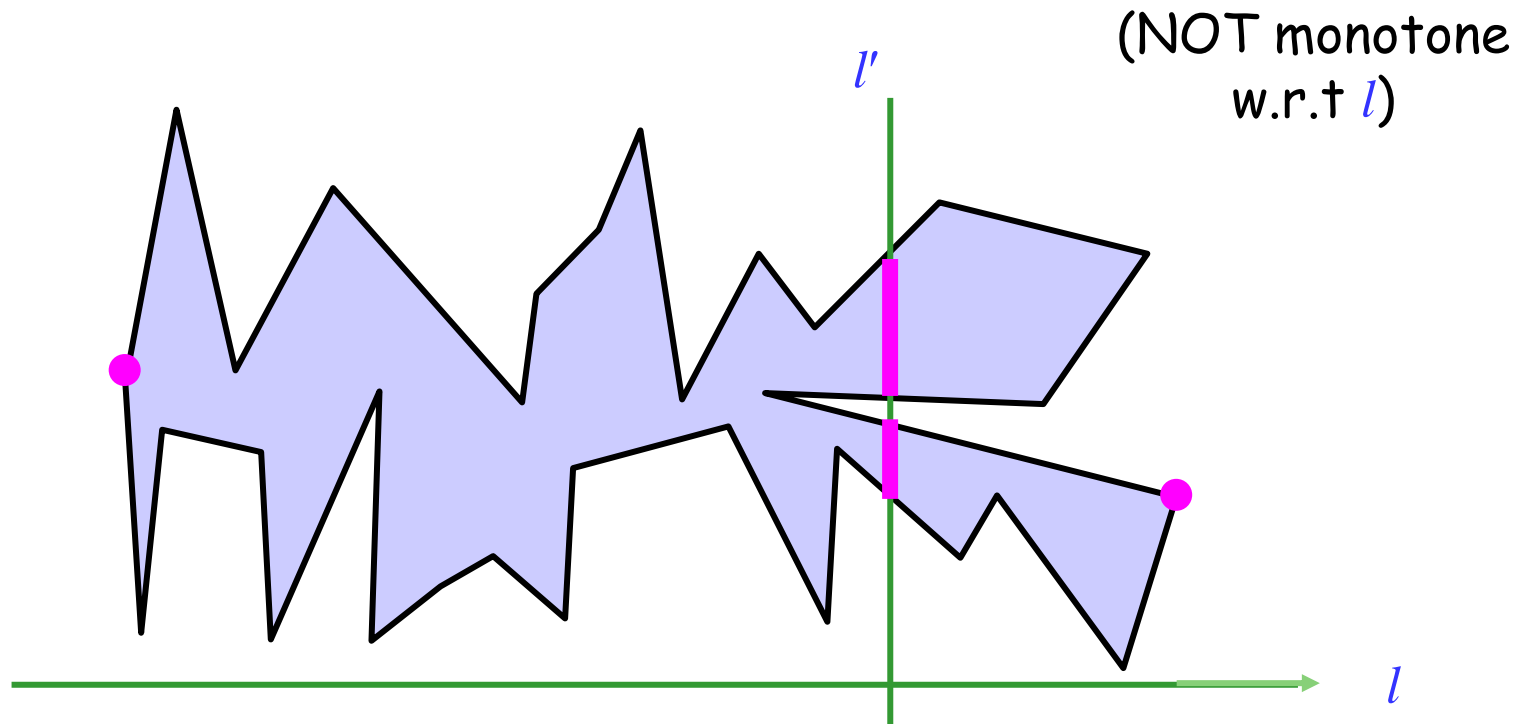
• Equivalently:

A simple polygon P is called **monotone with respect to a line l** if for every line l' perpendicular to l , the intersection of P with l' is connected (i.e., l' intersects with P exactly at zero or two points)

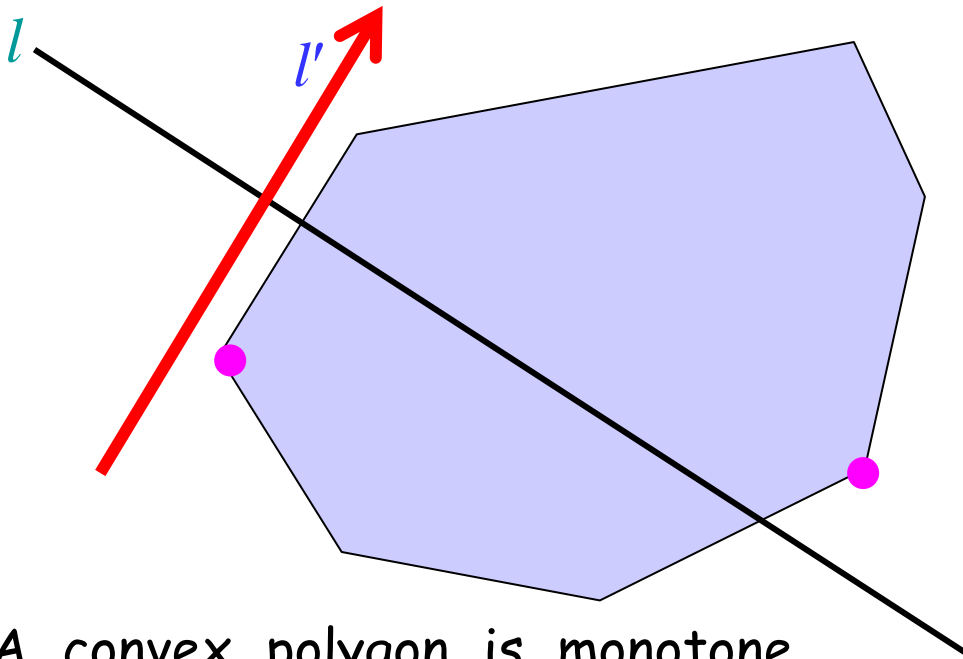


Monotone Polygons

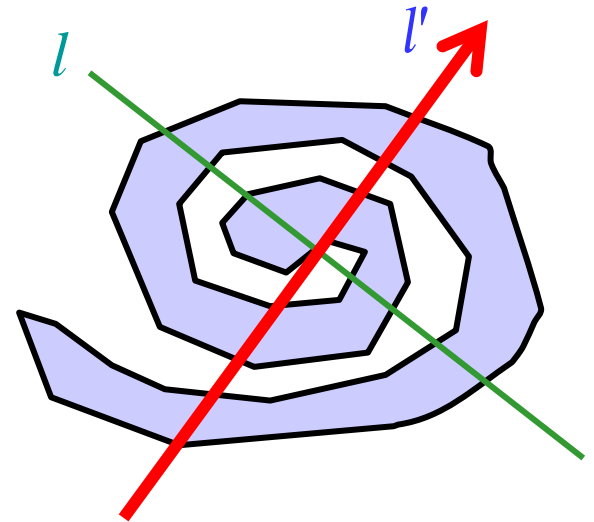
- Two monotone chains
- l -coordinates are *non-decreasing* when you travel along the chains of P in the direction of l



Monotone Polygons



A convex polygon is monotone
w.r.t any line l

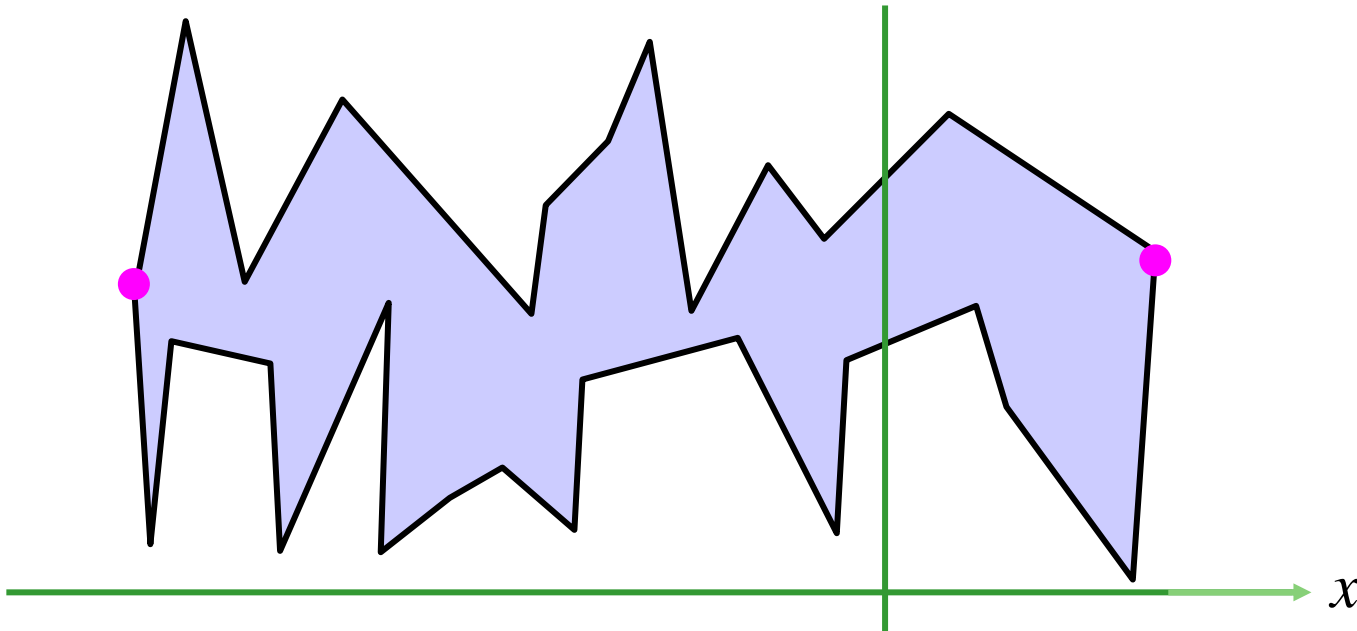


NOT monotone w.r.t
any line l

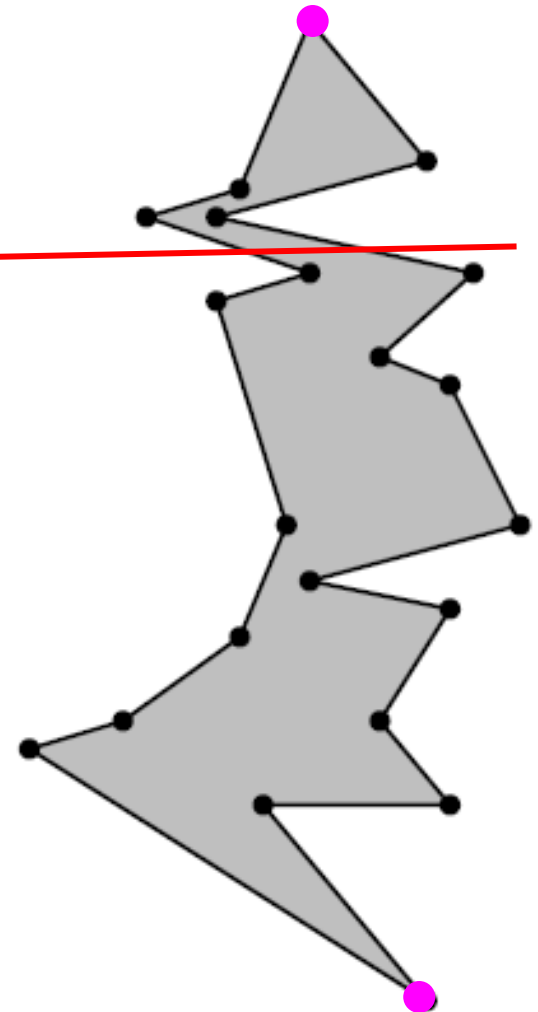
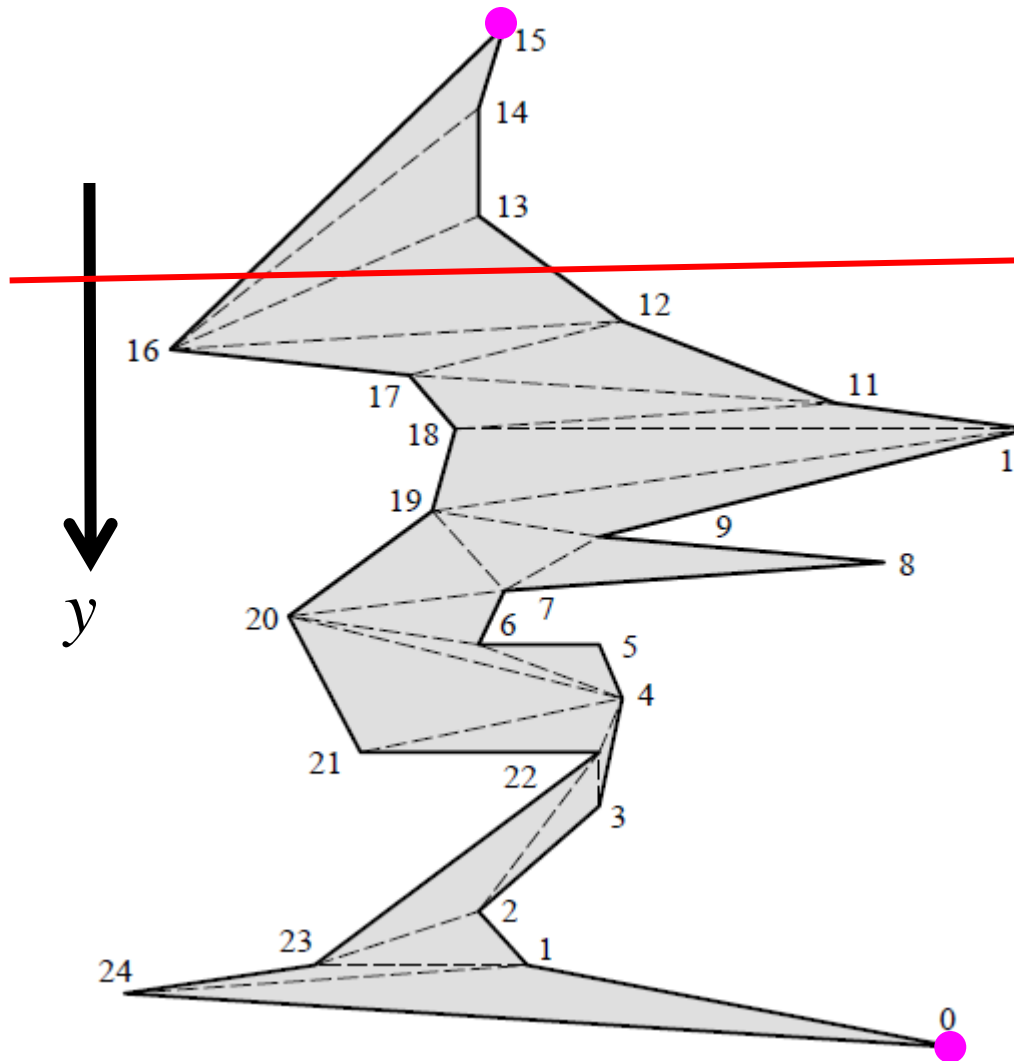
Test Monotonicity

How to test if a polygon is x -monotone?

- Find leftmost and rightmost vertices $\Rightarrow O(n)$ 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 $\Rightarrow O(n)$ time



y-Monotone Polygon

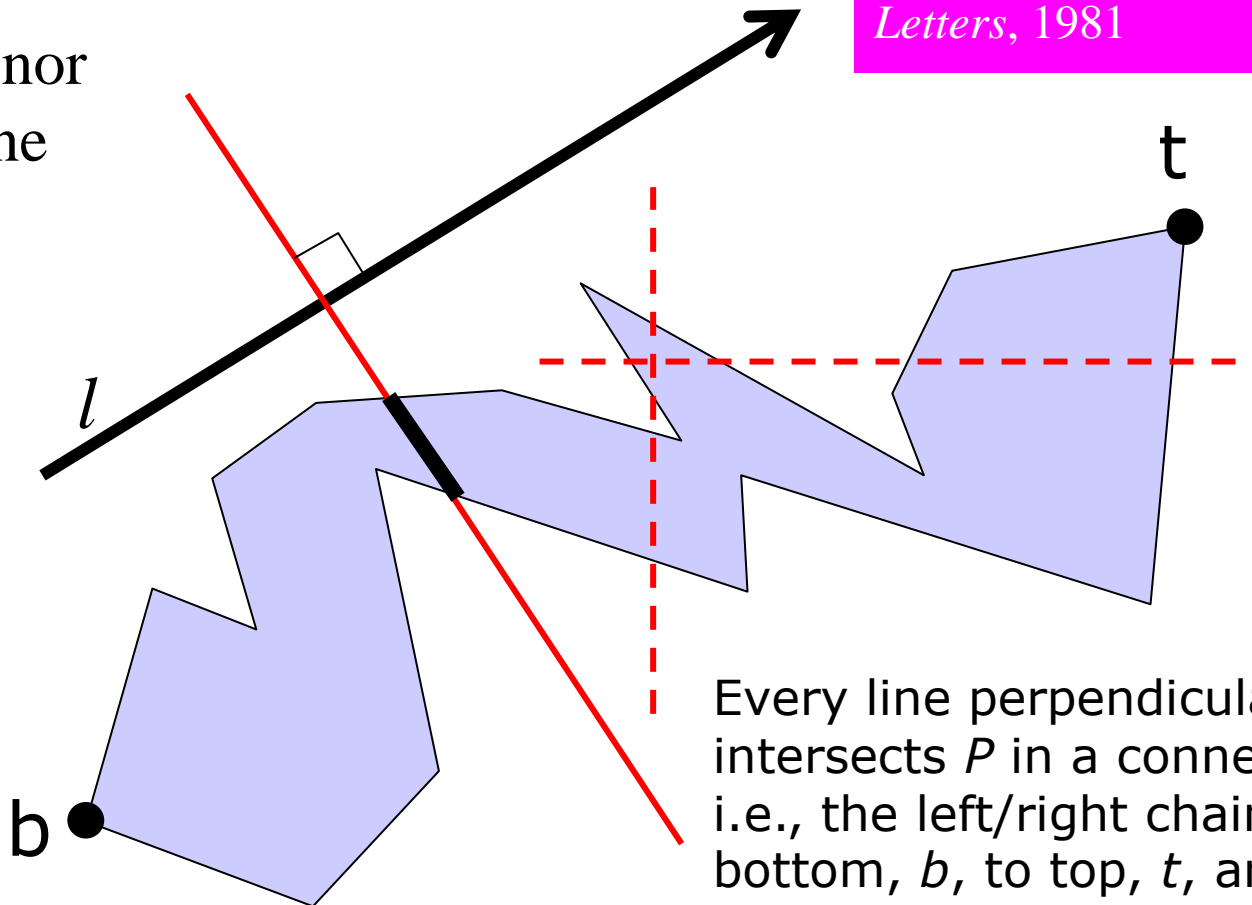


Monotone polygon may contain many reflex vertices

Monotone Polygons

- P is monotone in direction l

neither x - nor
 y -monotone



Every line perpendicular to l intersects P in a connected set; i.e., the left/right chains from bottom, b , to top, t , are each l -monotone

How do you test monotonicity of a polygon if no direction is given? Does there exist any direction *w.r.t.* which P is monotone?

Preparata and Supowit: $O(n)$ -time algorithm, *Information Processing Letters*, 1981

Partitioning a polygon with n vertices into monotone pieces and triangulation

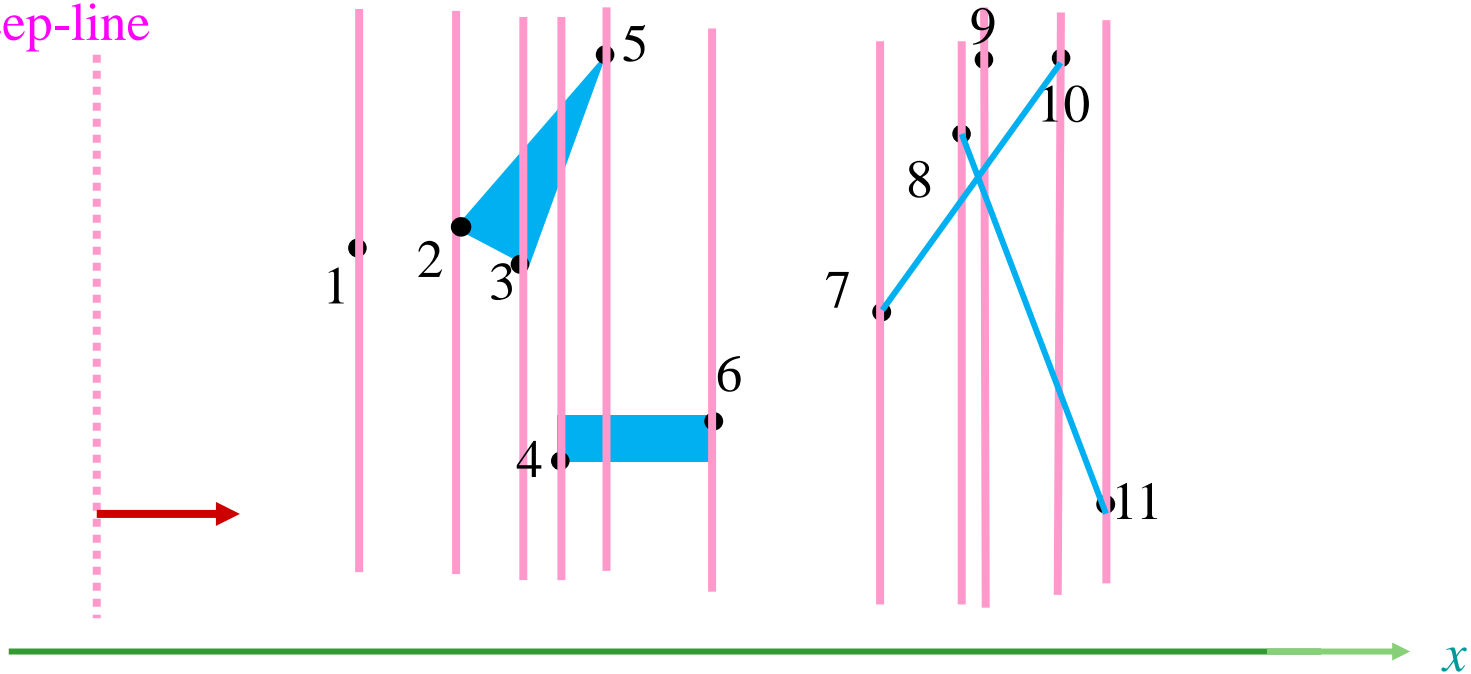
Partitioning:

- (i) Method via trapezoidation
- (ii) Method via “split” and “merge” vertices
- Both require plane-sweep techniques
- Both take $O(n \log n)$ time

Sweep-Line Paradigm (Horizontal Sweep)

- Sort characteristic points of objects by increasing x -coordinates;
- Perform plane-sweep along x -axis with a vertical line, halting at vertices in order;
- Update sweep-line status (SLS);
- Take appropriate actions depending on applications

sweep-line



Ref: David Mount's Lecture Notes;
4A Textbook

CS60064

Spring 2022

Computational Geometry

Instructors

Bhargab B. Bhattacharya (BBB)

Partha Bhowmick (PB)

Lecture 08 & Lecture 09

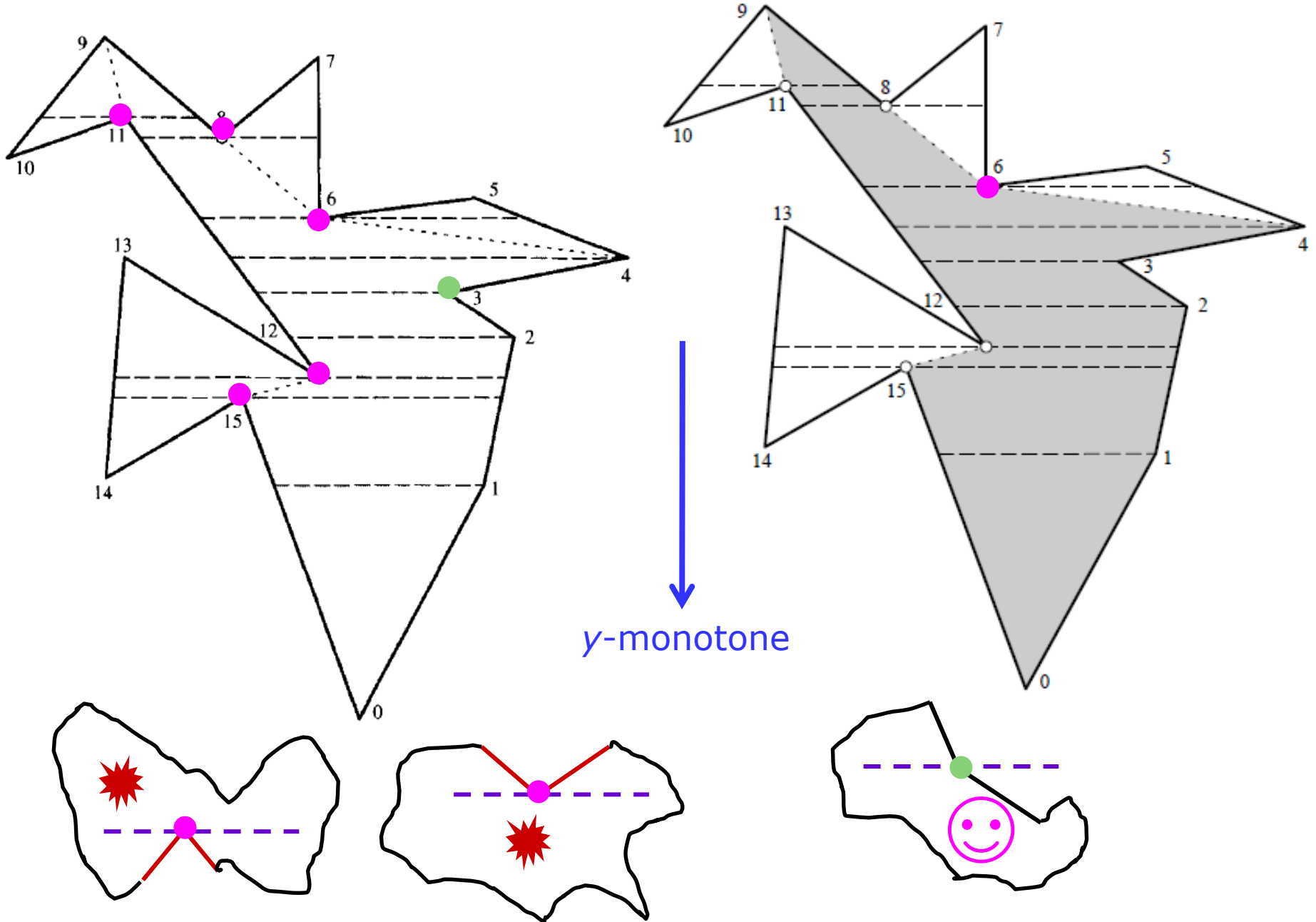
21 January 2022

Indian Institute of Technology Kharagpur
Computer Science and Engineering

Partitioning a polygon with n vertices into monotone pieces

- Method via trapezoidation

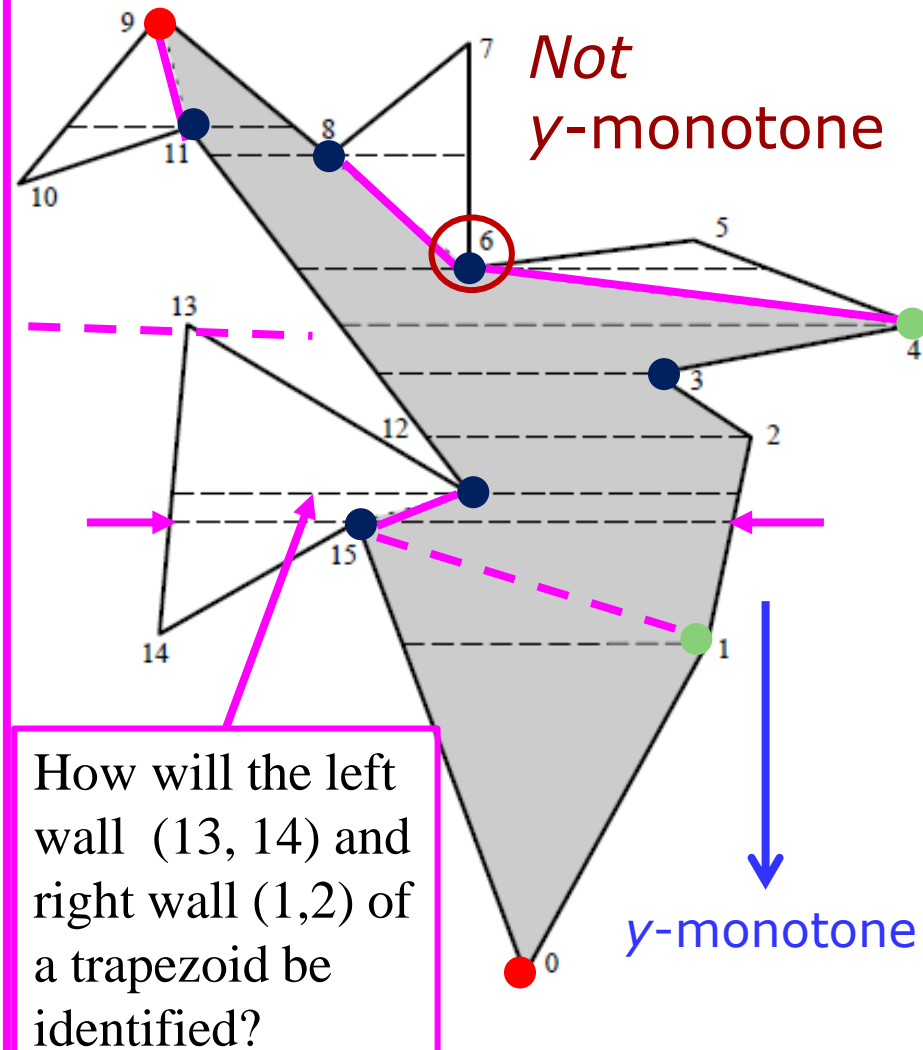
Partitioning a polygon into y-monotone pieces



Partitioning a polygon into y-monotone pieces

Trapezoidal space partitioning

- Process vertices in sorted order from top-to-bottom;
- Draw horizontal lines through each *reflex* vertex of P , and through those *convex* vertices whose both neighbors lie in *opposite* sides of the horizontal; this divides the interior of P into trapezoidal “slabs”;
- Identify each trapezoid t whose support vertex v is reflex and lies somewhere in the middle of the corresponding horizontal side;
- Join v with the other support vertex k of the same trapezoid t .

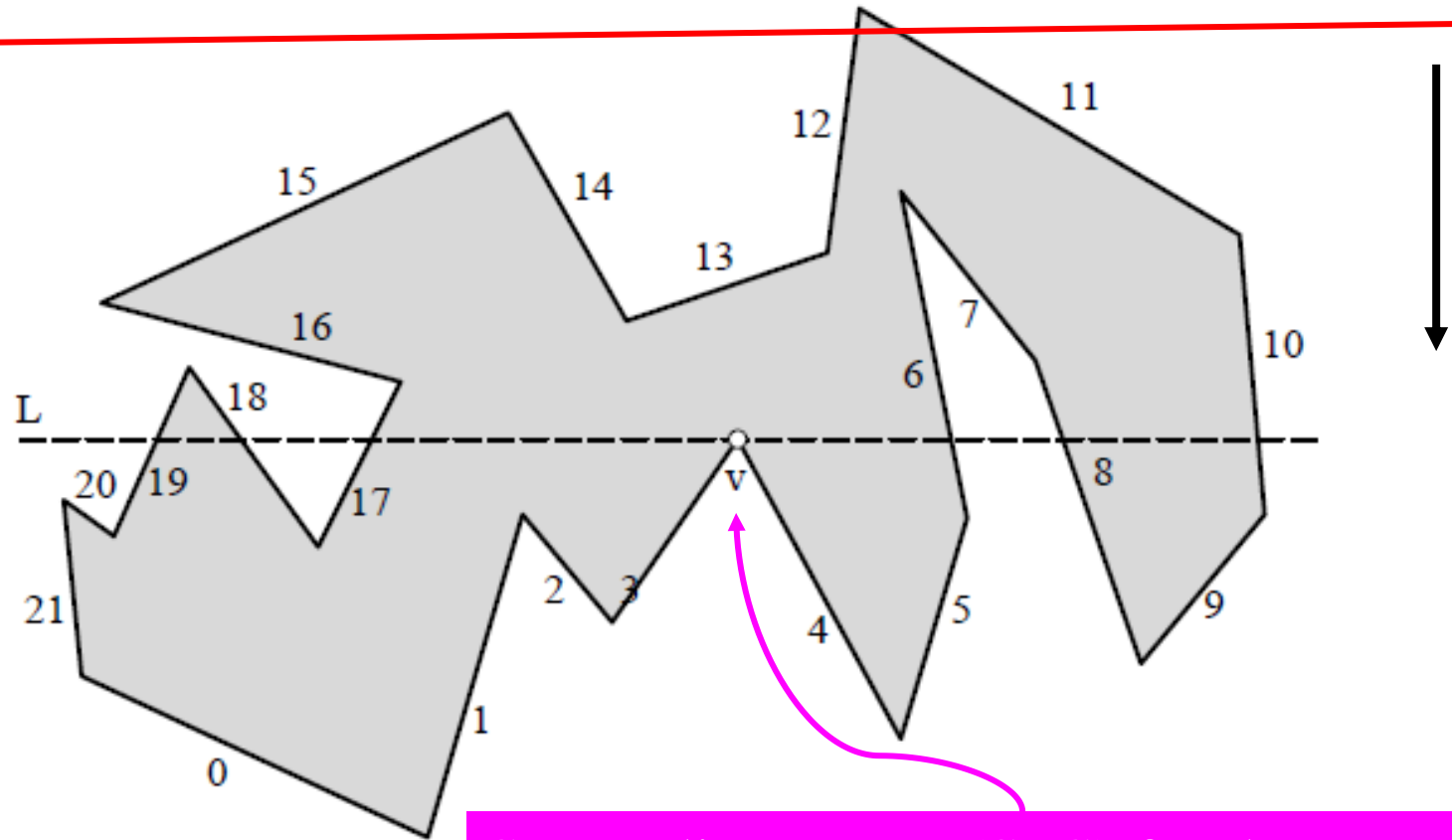


Partitioned into five y-monotone pieces

Implementation: Plane-Sweep Technique

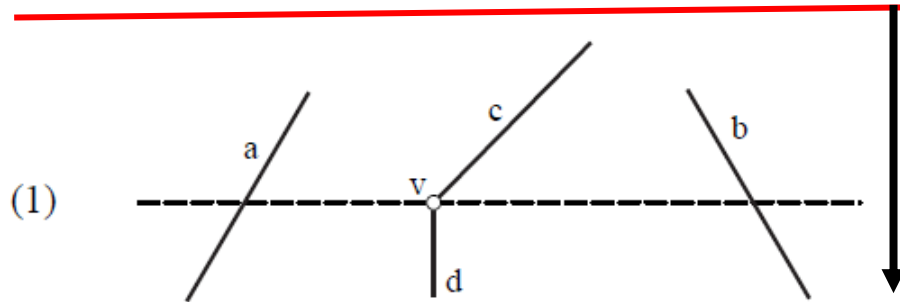
- Sort the vertices from top-to-bottom
(Assume for simplicity, all y -coordinates are distinct)
- Sweep a horizontal line downward starting from the top-most vertex; stop at discrete points, i.e. at vertices
- Two key ingredients of any sweep algorithm:
 - (i) Maintain “Sweep Line Status” (SLS): a “combinatorial description” of the “slice” seen by the sweeping line
 - Often stored in a balanced binary tree
 - (ii) Event Queue: Actions to be taken, sometimes prioritized, using appropriate data structures

Example: Partitioning a polygon into y-monotone pieces via trapezoid processing



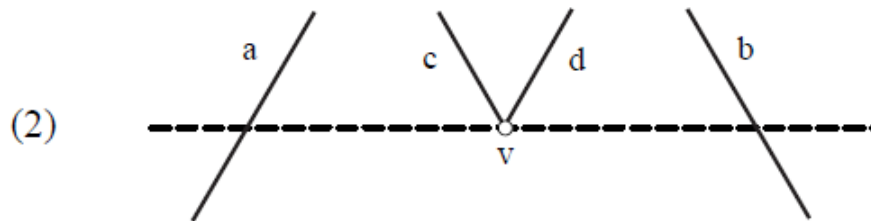
Sweep-line status (SLS) for the current v :
Ordered edge-list: (e19, e18, e17, e6, e8, e10)

Actions Needed



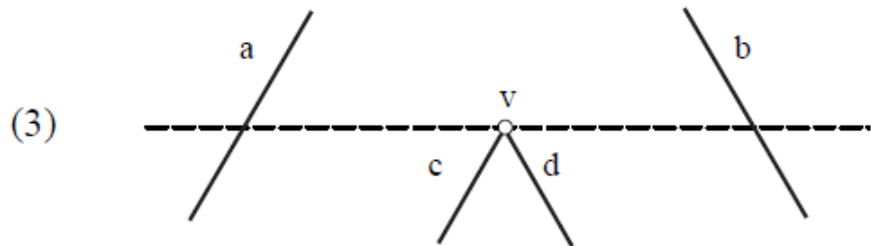
Delete c from the list, insert d in place of c :

$\dots, a, c, b \dots \rightarrow \dots, a, d, b, \dots$



- Delete c, d from the list

$\dots, a, c, d, b \dots \rightarrow \dots, a, b, \dots$

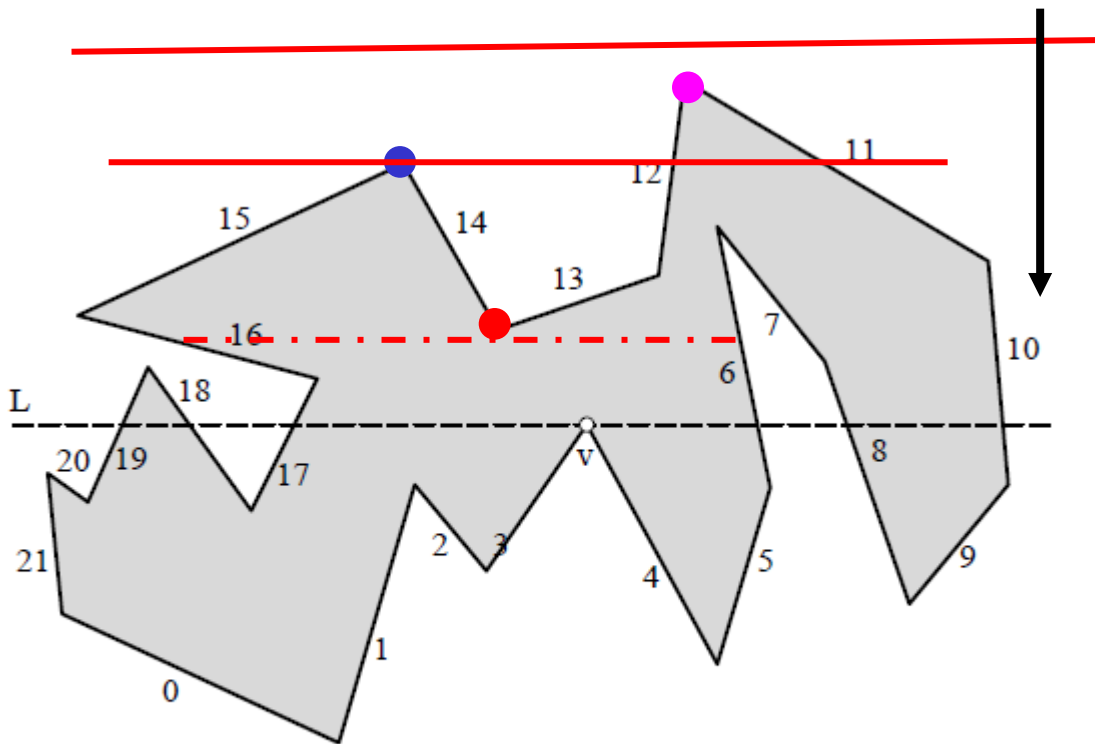


- Insert c, d in appropriate positions of the list

$\dots, a, b \dots \rightarrow \dots, a, c, d, b, \dots$

We do $O(1)$ updates to the SLS, each taking time $O(\log n)$, since the SLS is stored in a balanced binary search tree

Processing: SLS keeps track of edge-lists in an height-balanced binary tree



(e_{12}, e_{11}) ●
 $(\underline{e_{15}}, e_{14}, e_{12}, e_{11})$ ●
 $(e_{15}, e_{14}, e_{12}, e_6, e_7, e_{11})$
 $(e_{15}, e_{14}, e_{13}, e_6, e_7, e_{10})$
 $(\textcircled{e_{16}}, \textcolor{red}{\cancel{e_{14}}}, \textcolor{red}{\cancel{e_{13}}}, \textcircled{e_6}, e_7, e_{10})$ ●
 $(e_{16}, e_6, e_7, e_{10})$
 $(e_{16}, e_6, e_8, e_{10})$
 $(e_{19}, e_{18}, e_{16}, e_6, e_8, e_{10})$
 $(e_{19}, e_{18}, e_{17}, e_6, e_8, e_{10})$.

Insertion, deletion of edges, and identifying left and right walls of a trapezoid can be accomplished in $O(\log n)$ time, per vertex

Complexity of monotone partitioning

- Sorting of y -coordinates of n vertices: $O(n \log n)$
- Horizontal sweep-line stops at n points
- At each vertex, update and trapezoid formation:
 $O(\log n)$ using AVL-tree of $O(n)$ -size edge lists
- Finding support vertices of each trapezoid: $O(1)$
- Total number of trapezoids: $O(n)$
- Total partitioning cost into monotone pieces:
 $O(n \log n)$

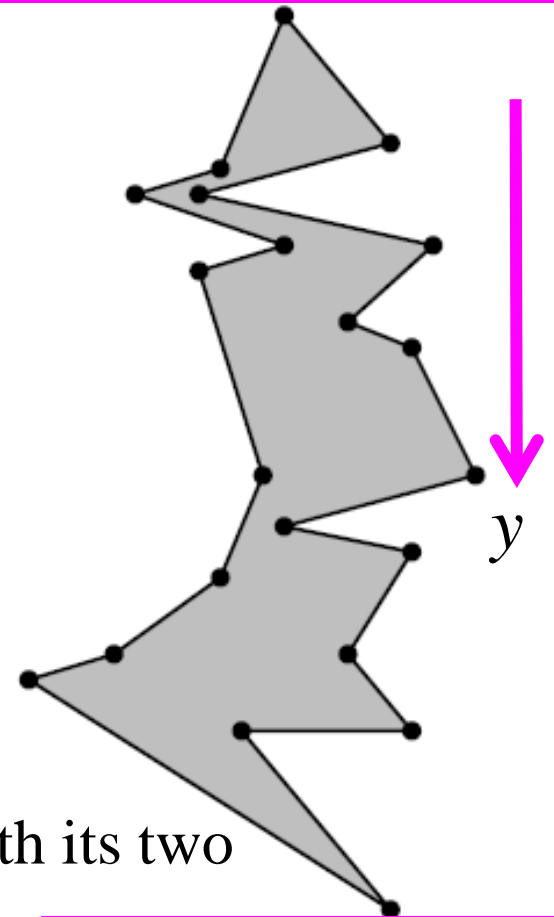
Monotone Partitioning via “Spilt and Merge” Vertices

Monotone polygons

top-vertex (start-vertex)

A y -monotone polygon has a top vertex, a bottom vertex, and two y -monotone chains between top and bottom as its boundary

Any simple polygon with one top vertex and one bottom vertex is y -monotone



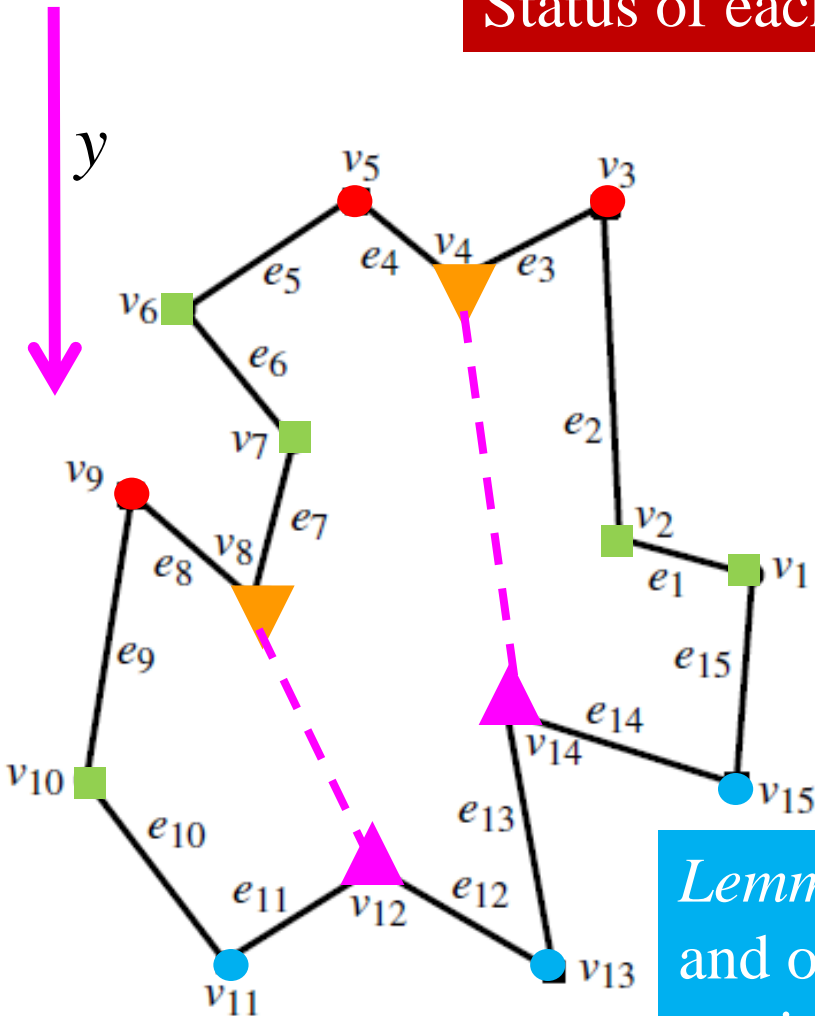
bottom (end)-vertex

top vertex v_t : a strictly convex vertex such that both its two neighbors lie below the horizontal line through v_t

bottom vertex v_b : a strictly convex vertex such that both its two neighbors lie above the horizontal line through v_b

Vertex classification in a polygon *w.r.t.* y -axis

Status of each vertex can be determined in $O(1)$ time



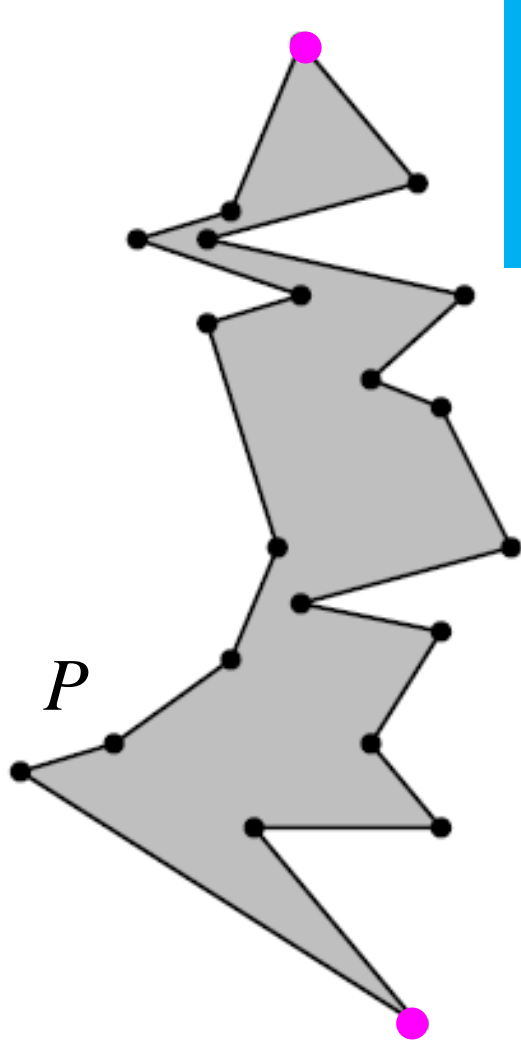
- start vertex convex, both neighbors ↓
- end vertex convex, both neighbors ↑
- regular vertex one neighbor ↓, other ↑
- ▲ split vertex reflex, both neighbors ↓
- ▼ merge vertex reflex, both neighbors ↑

Lemma: A simple polygon P is y -monotone if and only if P is devoid of any merge or split vertices
 $\Rightarrow P$ will have only one start- and one end-vertex

P is devoid of any merge or split vertices;

P has only one start- and one end-vertex

P is y -monotone

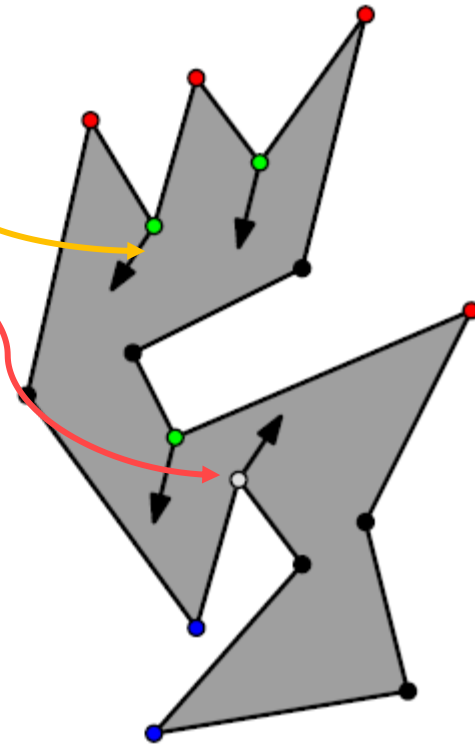


□ Proof: Since there are only start/end/regular vertices, the polygon must consist of two y -monotone chains.

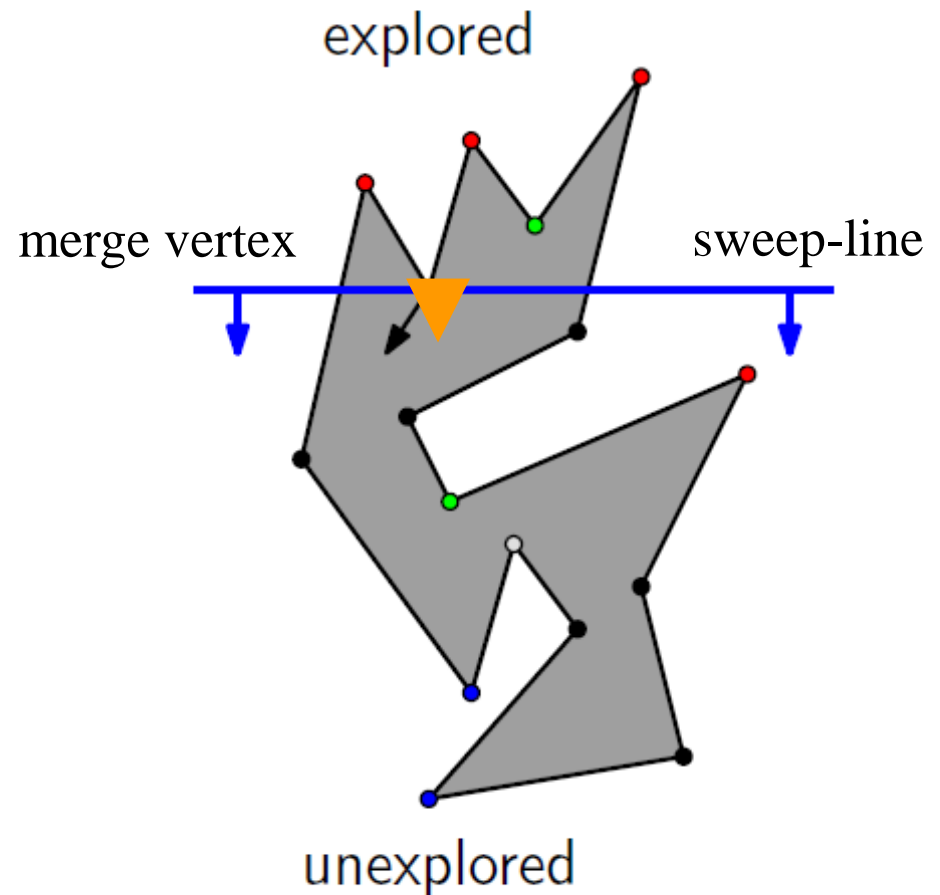
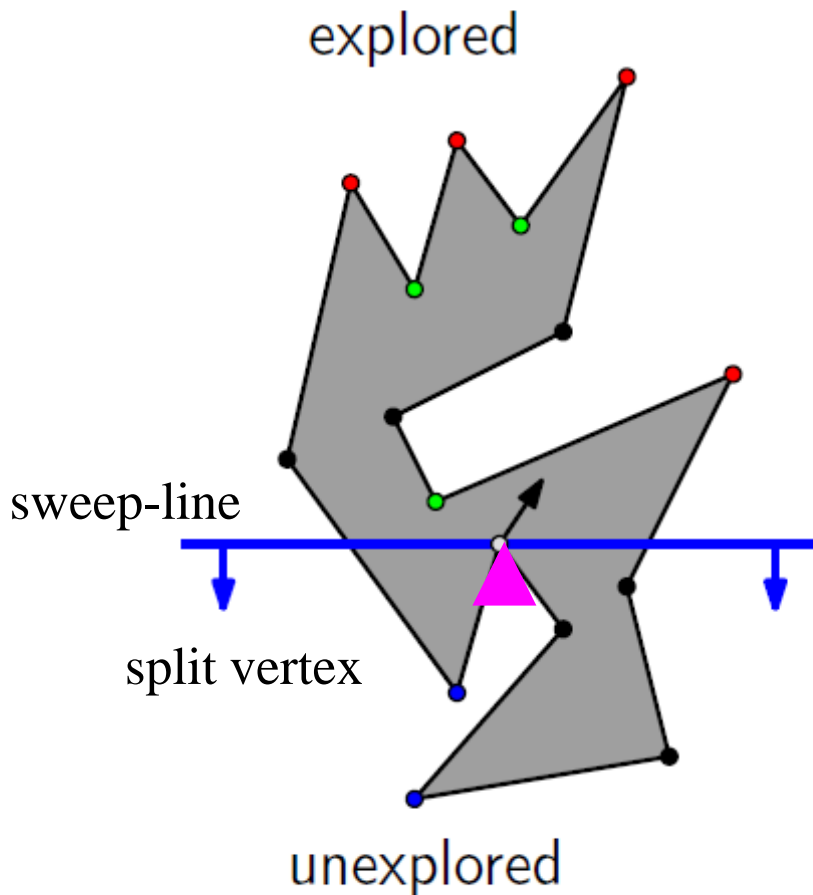
Sweep ideas

Find diagonals from each merge vertex down, and from each split vertex up

A simple polygon with no split or merge vertices can have at most one start and one stop vertex, so it is y -monotone



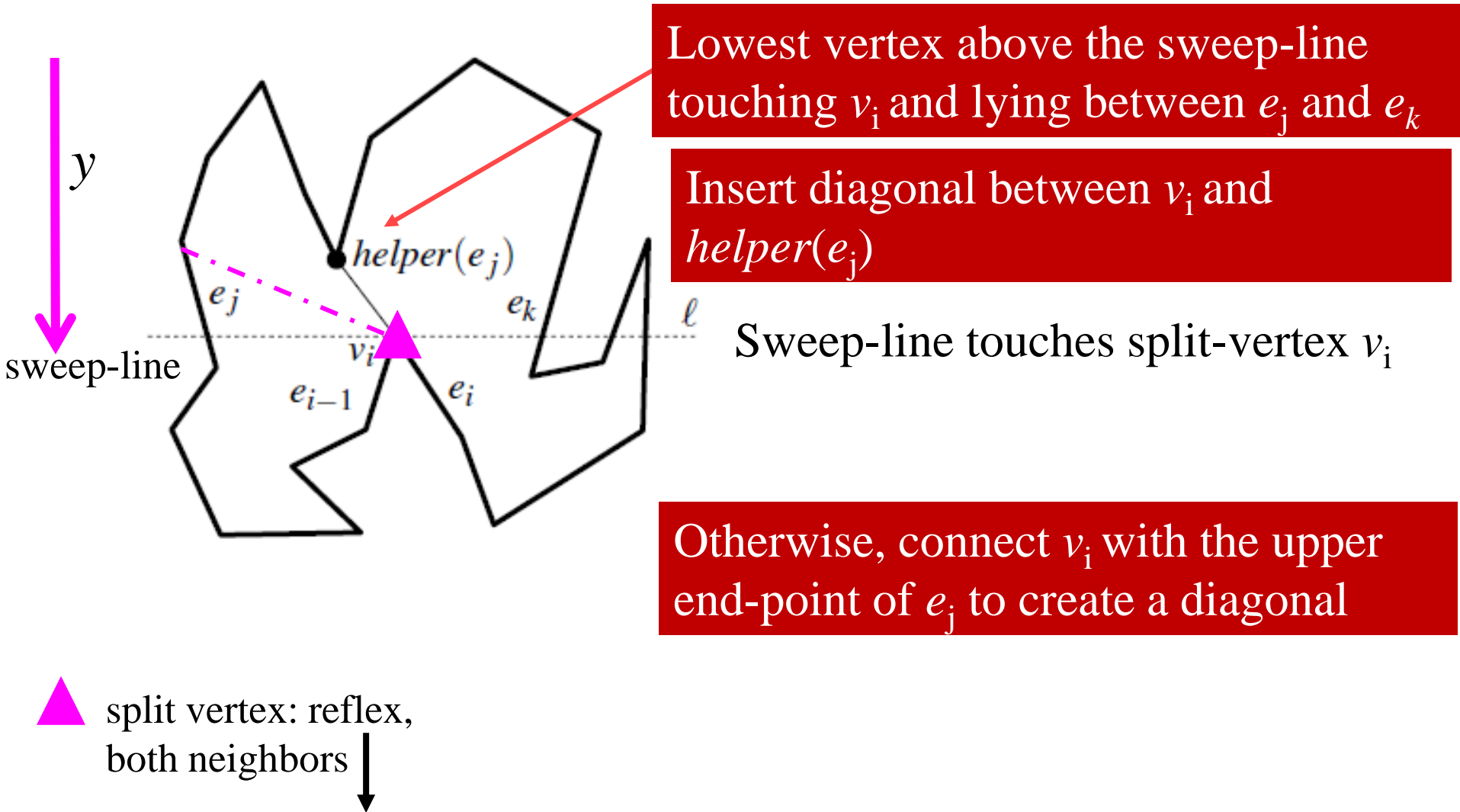
Sweep ideas



Handling “split” is fine as its possible matching vertices are already explored by the sweep-line. How about “merge”?

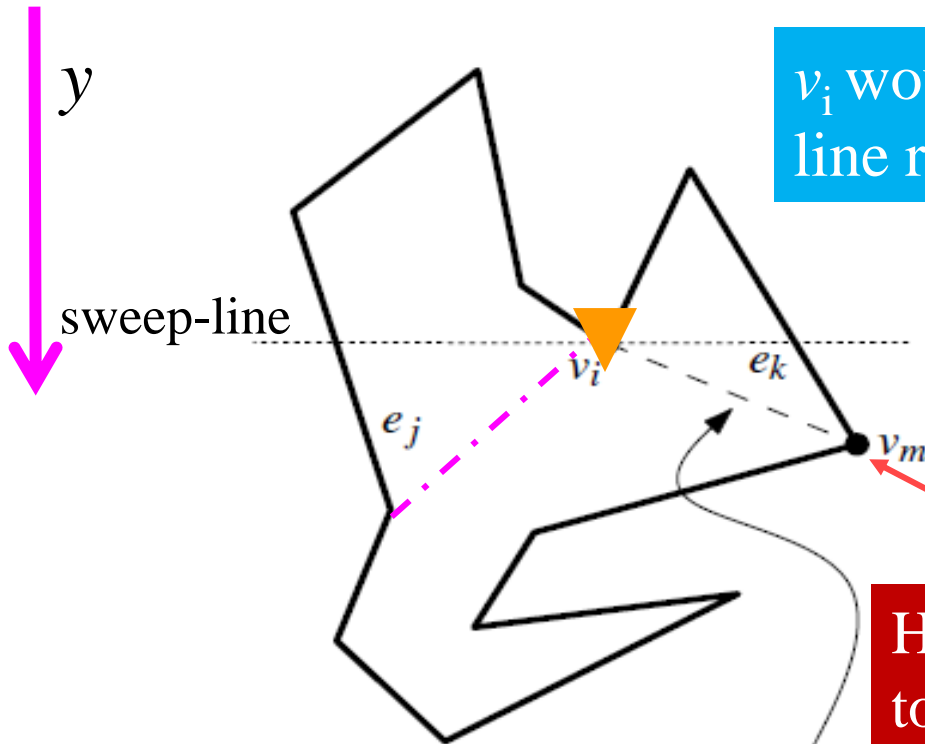
Management of “merge” needs a reverse sweep at the end

Handling Split Vertices



Handling Merge Vertices

How do you identify v_m ?



v_i would become *helper*(e_j) when sweep-line reaches v_m later

Insert diagonal between v_i and v_m

Sweep-line touches split-vertex v_i

Highest vertex below the sweep-line touching v_i and lying between e_j and e_k

diagonal will be added when the sweep line reaches v_m

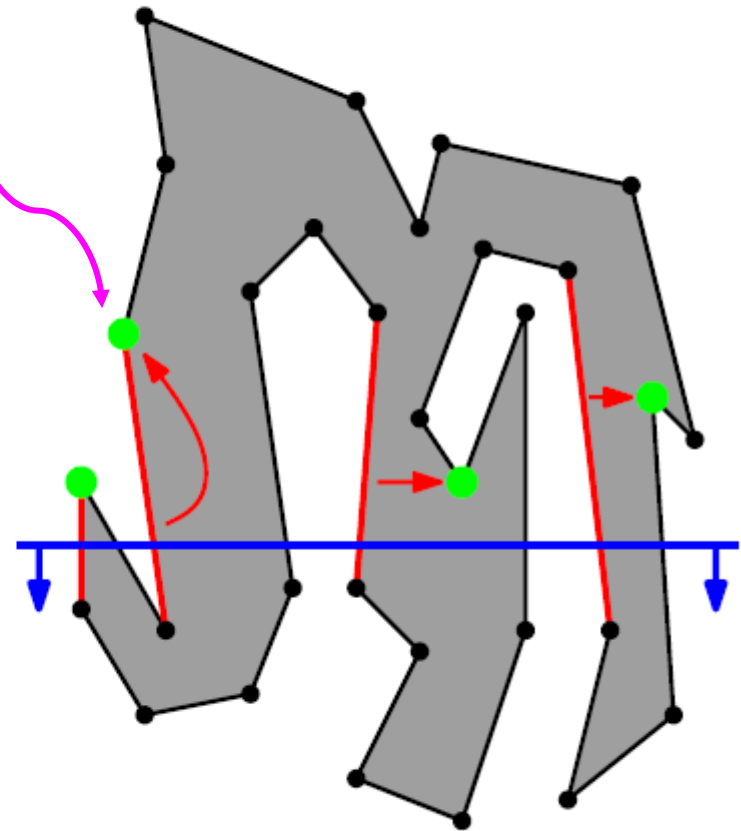
Otherwise, connect v_i with the lower end-point of e_j to create a diagonal

merge vertex: reflex,
both neighbors ↑

Status of sweep

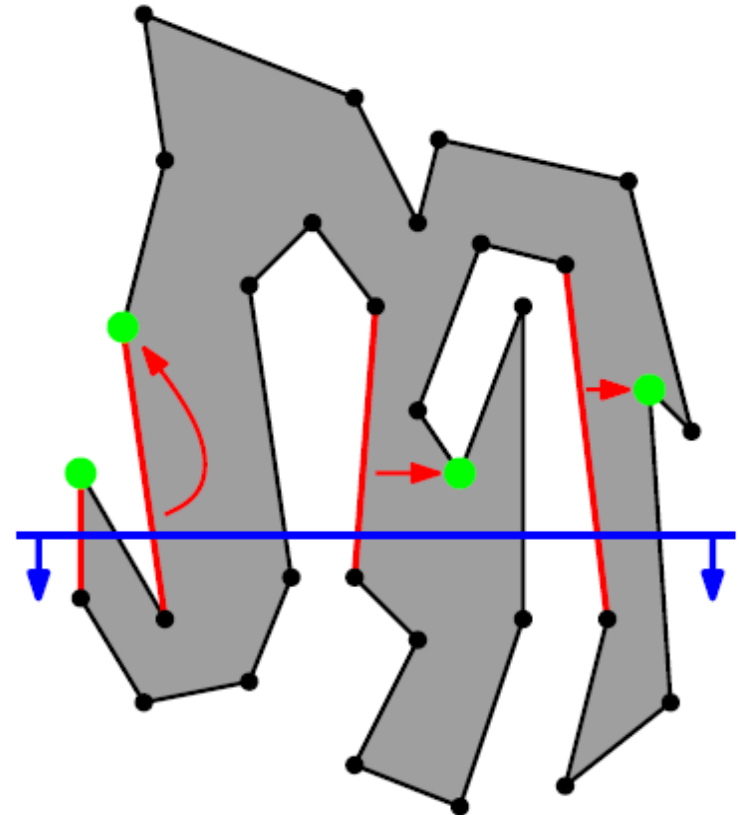
The **status** is the set of edges intersecting the sweep line that have the polygon to their right, sorted from left to right, and each with their *helper*: the last vertex passed in that component

helper



Helpers of edges

The **helper** for an edge e that has the polygon right of it, and a position of the sweep line, is the lowest vertex v above the sweep line such that the horizontal line segment connecting e and v is inside the polygon



Status structure, event list

The **status structure** stores all edges that have the polygon to the right, with their helper, sorted from left to right in the leaves of a balanced binary search tree T

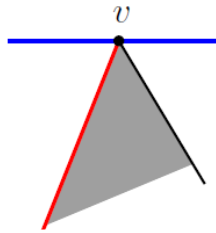
The events happen only at the vertices: sort them by y -coordinate and put them in a list (or array, or tree)

Main algorithm

- Sort all vertices by y -coordinate and put them in a list.
Initialize an empty status structure T
- **while** the event list is not empty
 do remove the first event and handle it

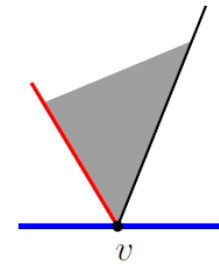
Sweep-Line Algorithm

- **Events:** Vertices of polygon, sorted in increasing order by y -coordinates (no new events will be dynamically added)
- **Sweep-line status:** Balanced *binary search tree* storing the list of edges intersecting sweep line, sorted by x -coordinates; Also, $\text{helper}(e)$ for every edge intersecting the sweep line
- Event processing of vertex v :
 1. start vertices
 2. end vertices
 3. regular vertices
 4. merge vertices
 5. split vertices



Start vertex v :

- Insert the left incident edge in T with v as the helper

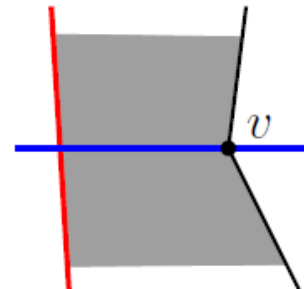
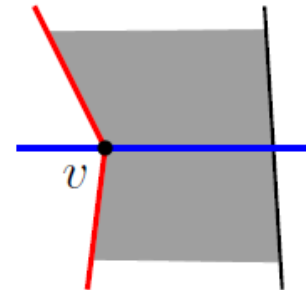


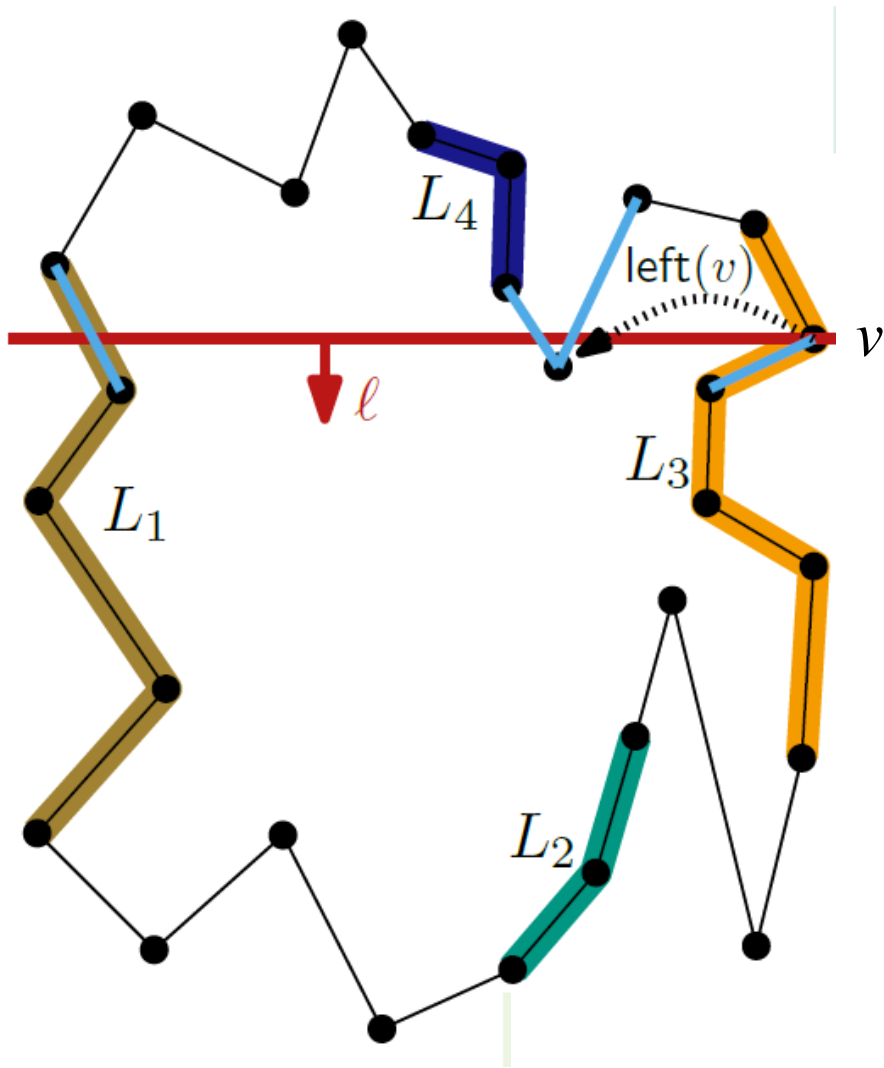
End vertex v :

- Delete the left incident edge and its helper from T

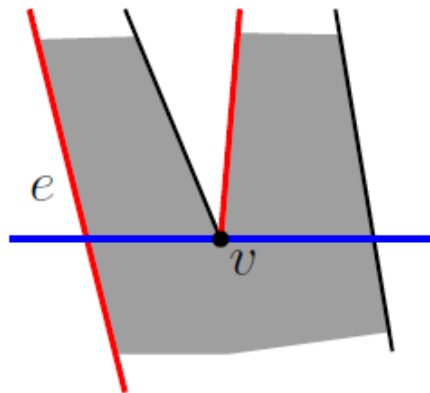
Regular vertex v :

- If the polygon is right of the two incident edges, then replace the upper edge by the lower edge in T , and make v the helper
- If the polygon is left of the two incident edges, then find the edge e directly left of v , and replace its helper by v



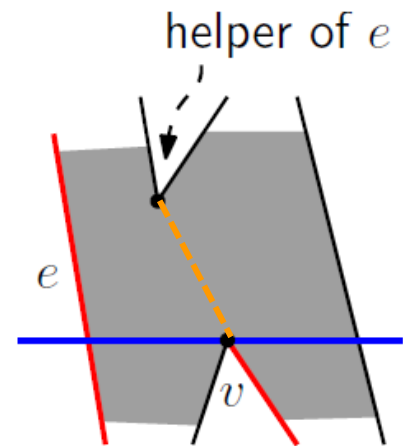


Determine for vertex v the edge $\text{left}(v)$ directly left to v



Merge vertex v :

- Remove the edge clockwise from v from T
- Find the edge e directly left of v , and replace its helper by v



Split vertex v :

- Find the edge e directly left of v , and choose as a diagonal the edge between its helper and v
- Replace the helper of e by v
- Insert the edge counterclockwise from v in T , with v as its helper

Sorting all events by y -coordinate takes $O(n \log n)$ time

Every event takes $O(\log n)$ time, because it only involves querying, inserting and deleting in T

Sweep Line Algorithm

- Insert diagonals for merge vertices with “reverse” sweep
 - Each update takes $O(\log n)$ time
 - There are n events
- Run-time to compute a monotone subdivision is $O(n \log n)$

Algorithm: Monotone Partitioning

A data structure for storing planar maps. Allows insertion of a diagonal in $O(1)$ time

Algorithm MAKEMONOTONE(\mathcal{P})

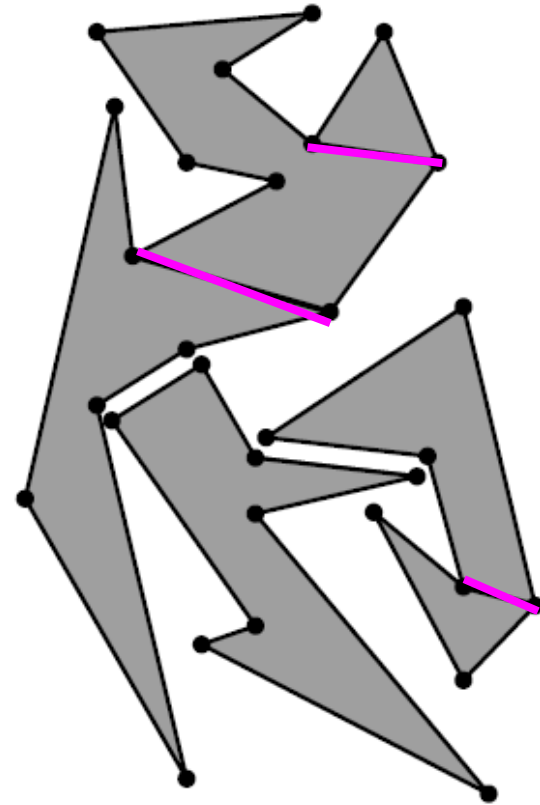
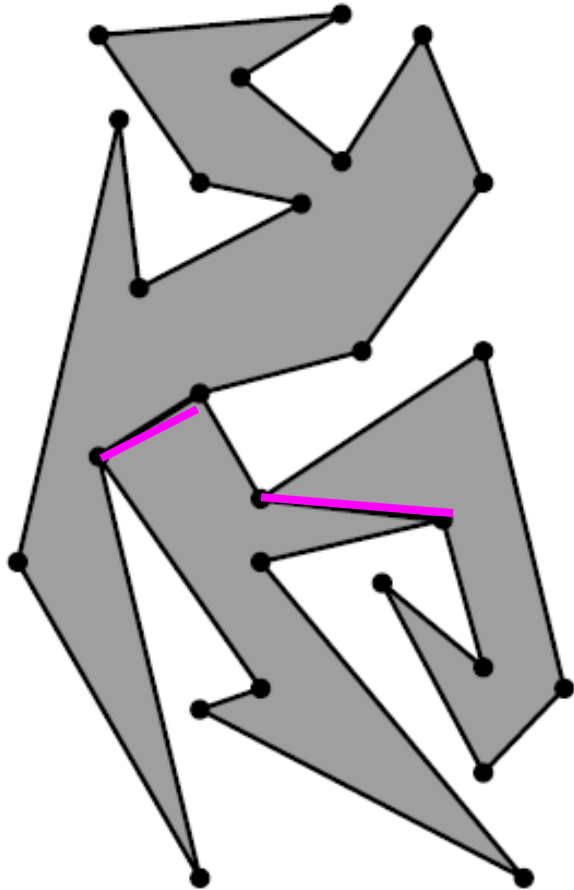
Input. A simple polygon \mathcal{P} stored in a doubly-connected edge list \mathcal{D} .

Output. A partitioning of \mathcal{P} into monotone subpolygons, stored in \mathcal{D} .

1. Construct a priority queue \mathcal{Q} on the vertices of \mathcal{P} , using their y -coordinates as priority. If two points have the same y -coordinate, the one with smaller x -coordinate has higher priority.
2. Initialize an empty binary search tree \mathcal{T} .
3. **while** \mathcal{Q} is not empty
4. **do** Remove the vertex v_i with the highest priority from \mathcal{Q} .
5. Call the appropriate procedure to handle the vertex, depending on its type.

- *Theorem:* A simple polygon with n vertices can be partitioned into y -monotone polygons in $O(n \log n)$ time with an algorithm that uses $O(n)$ storage

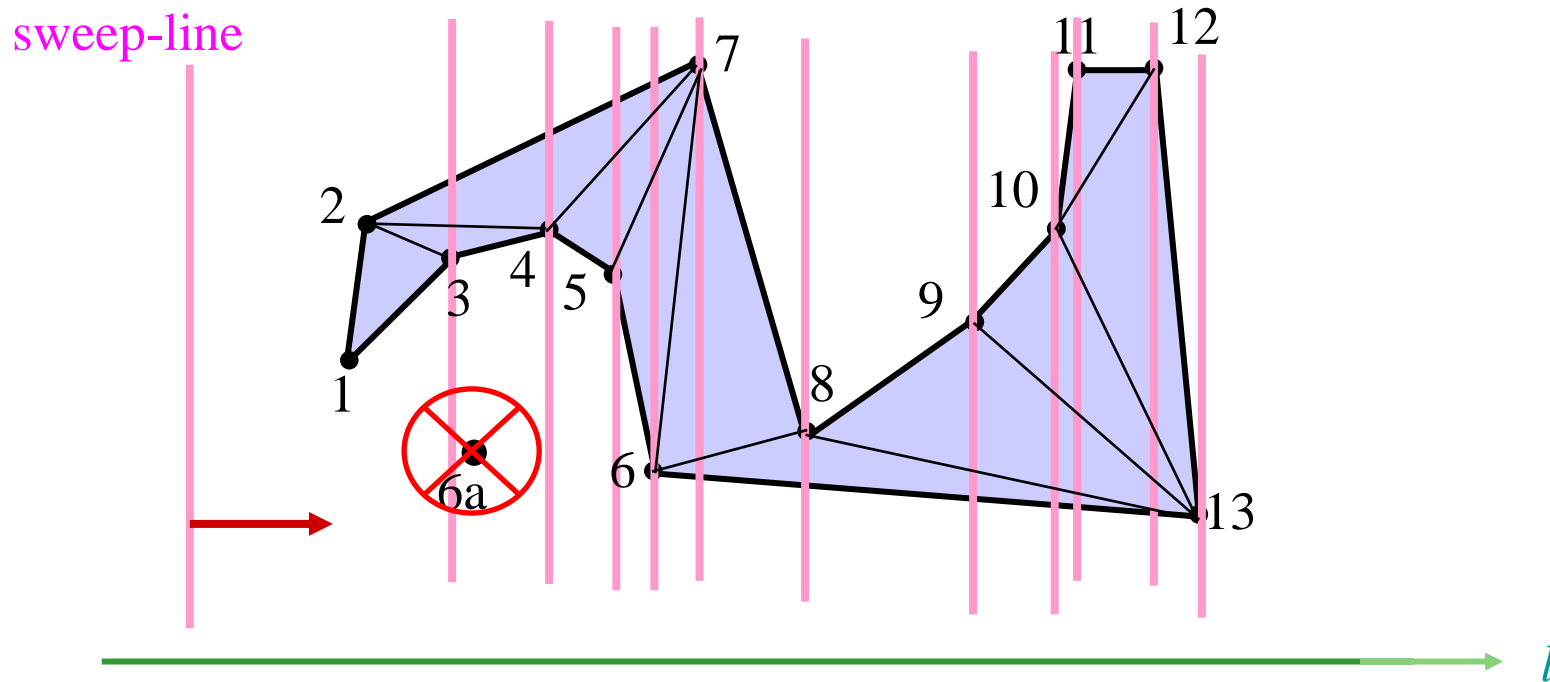
Example



Partitioned into six monotone pieces: Can you reduce the number?

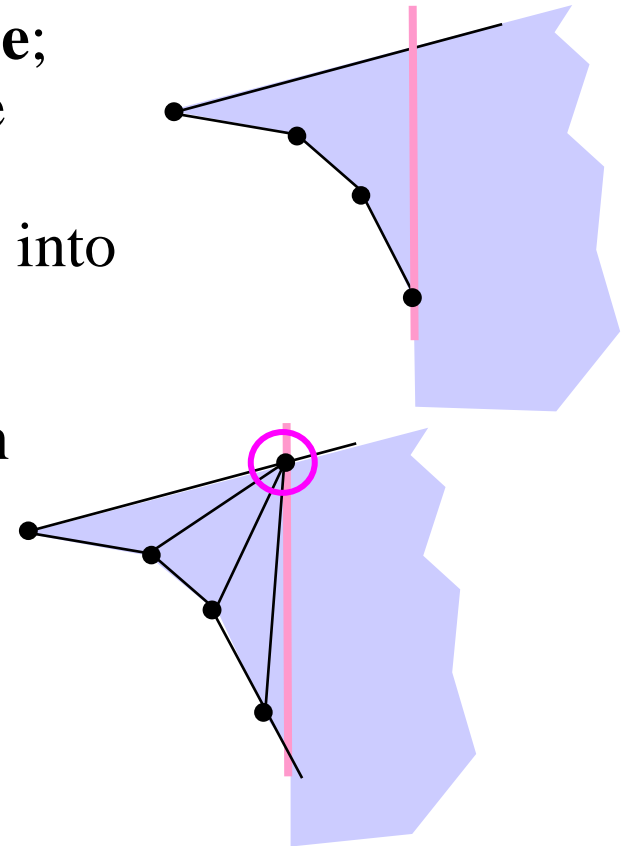
Triangulate an l -Monotone Polygon

- Sort vertices by increasing l -coordinates (merging the upper and lower chains in $O(n)$ time)
- Perform plane-sweep in direction l , halting at vertices in order
- Triangulate everything by adding valid diagonals to the left of the sweep line; maintain a stack and use orientation tests



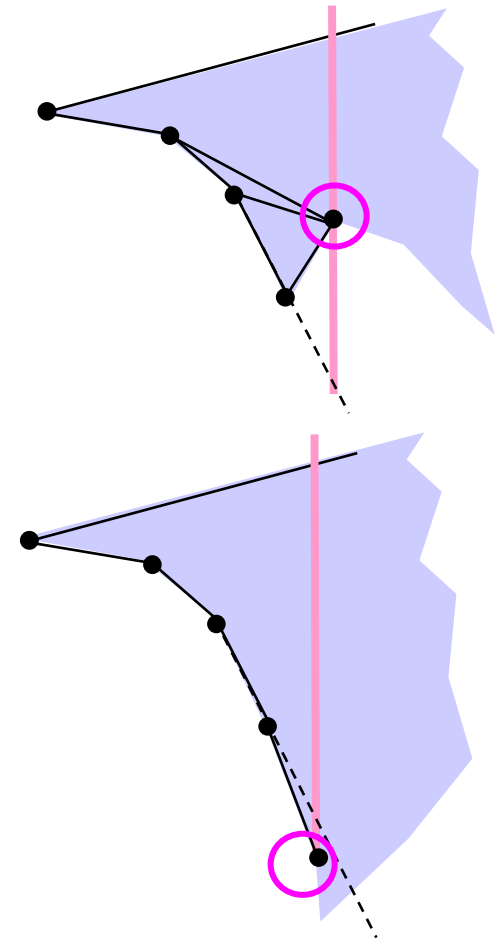
Triangulate an *l*-Monotone Polygon

- Store in a **stack** (sweep line status) the vertices that have been encountered but may need more diagonals
- Un-triangulated region has a **funnel shape**;
The funnel comprises a line segment (one side) and a **reflex chain** (interior angles $>180^\circ$) on the other side, which is pushed into the stack
- **Update:** *Case 1*: new vertex lies on chain opposite of reflex chain; triangulate; pop-off occluded vertices



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 as far as possible (via orientation tests); pop-off occluded vertices
 - **Case *b*:** The new vertex lies below line through previous two vertices; add to the reflex chain (stack)



The algorithm

- Sort the vertices by a merge of the two chains
- Initialize a stack. Push the first two vertices
- Take the next vertex v , and triangulate as much as possible, top-down, while popping the stack
- Push v onto the stack

Triangulate an l -Monotone Polygon

- Distinguish cases in constant time using chain information and orientation 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 add a diagonal) at most once

⇒ Constant time per vertex

⇒ $O(n)$ total run-time

Please convince yourself how monotonicity helps!