# Partitioning a Polygon

1. *y*-Monotone Polygons
   - partitioning into monotone pieces
   - adding diagonals between turn vertices
2. A Sweep Line Method
   - the global algorithm
   - handling a split vertex
   - handling a merge vertex
   - handling start, end, and regular vertices
3. Correctness and Cost
   - MAKEMONOTONE is correct
   - cost of MAKEMONOTONE

MCS 481 Lecture 8
Computational Geometry
Jan Verschelde, 31 January 2025

# Partitioning a Polygon

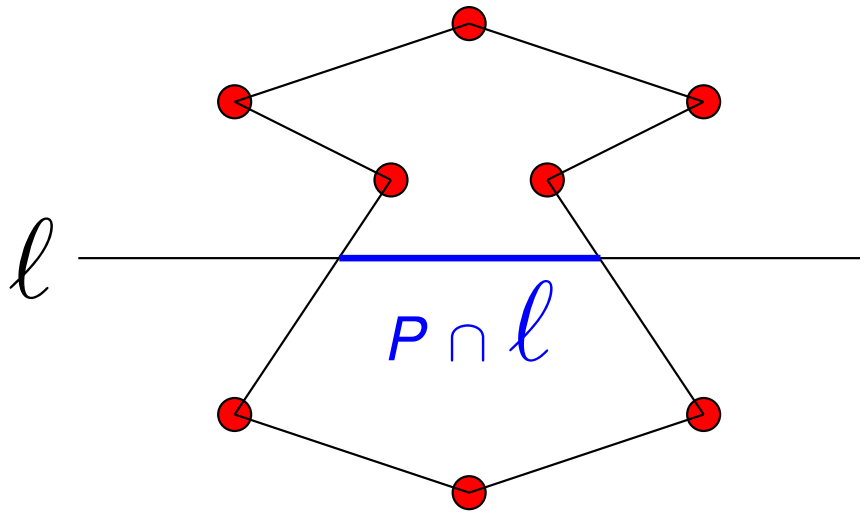
1. *y*-Monotone Polygons
   - partitioning into monotone pieces
   - adding diagonals between turn vertices
2. A Sweep Line Method
   - the global algorithm
   - handling a split vertex
   - handling a merge vertex
   - handling start, end, and regular vertices
3. Correctness and Cost
   - MAKEMONOTONE is correct
   - cost of MAKEMONOTONE

# $y$-monotone polygons

## Definition ($y$-monotone polygon)

A polygon $P$ is *$y$-monotone* if for any line $\ell$ perpendicular to the $y$-axis the intersection $P \cap \ell$ is connected.

Motivation: construct a triangulation of $P$ with $n$ vertices in $O(n \log(n))$.



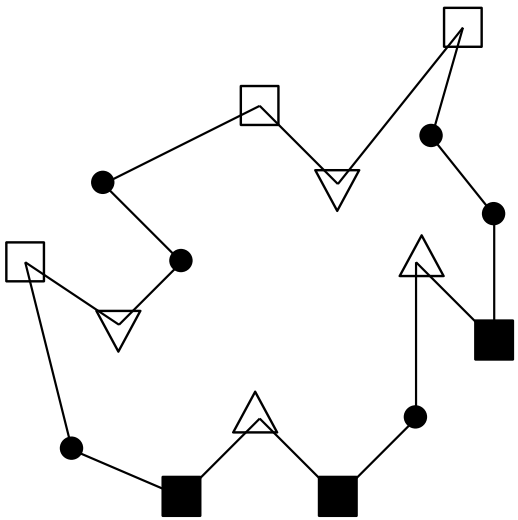

Any walk from top to bottom vertex is monotone decreasing, with no upgoing edges.

# five types of vertices

## Definition (a turn vertex)

A vertex where the orientation of the walk switches from down to up is *a turn vertex*.

We add diagonals at turn vertices to partition $P$ in $y$-monotone pieces.



□: startv : lower neighbors & inner angle $< \pi$

■: endv : higher neighbors & inner angle $< \pi$

▽: mergev : higher neighbors & inner angle $> \pi$

△: splitv : lower neighbors & inner angle $> \pi$

• : regularv : none of the above

# Partitioning a Polygon

1. *y*-Monotone Polygons
   - partitioning into monotone pieces
   - adding diagonals between turn vertices
2. A Sweep Line Method
   - the global algorithm
   - handling a split vertex
   - handling a merge vertex
   - handling start, end, and regular vertices
3. Correctness and Cost
   - MAKEMONOTONE is correct
   - cost of MAKEMONOTONE

# split and merge vertices

## Lemma (split and merge vertices)

*If $P$ contains no split and no merge vertices, then $P$ is $y$-monotone.*

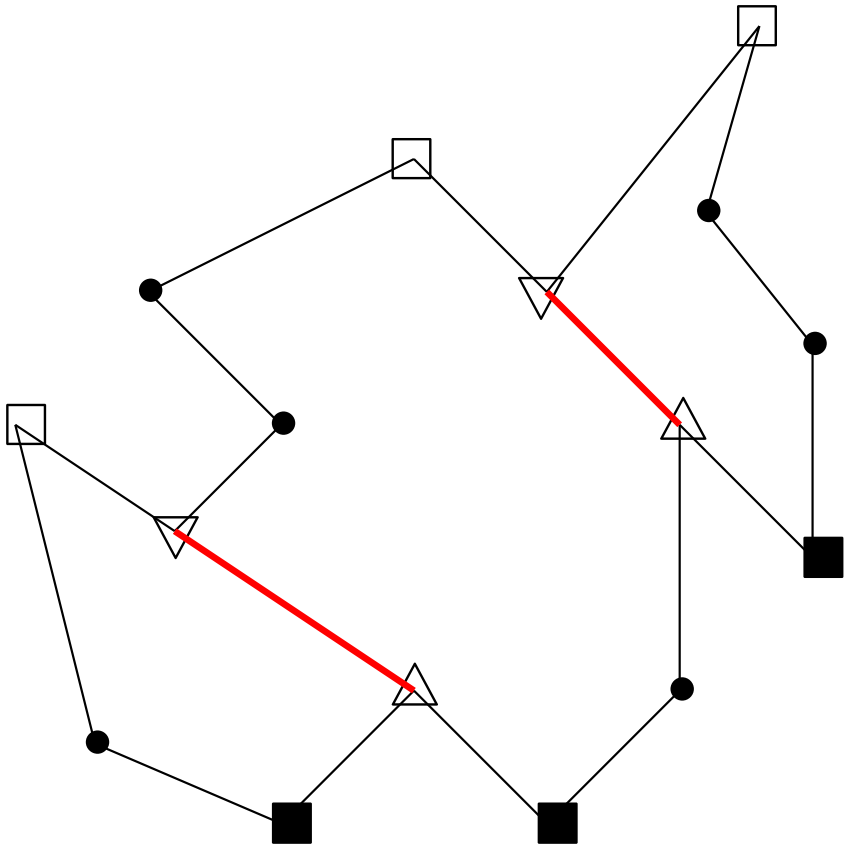
*Proof.* By contraposition: assuming $P$ is not $y$-monotone,
we will show $P$ contains a split or merge vertex.
Imagine a sweep line $\ell$ and $P \cap \ell$ is not connected.


As we move $\ell$ up or down, $P \cap \ell$ cannot stay disconnected.

- If we move $\ell$ down and $P \cap \ell$ becomes connected,
  then this happens at a split vertex.
- If we move $\ell$ up and $P \cap \ell$ becomes connected,
  then this happens at a merge vertex.

Q.E.D.

# adding diagonals between turn vertices



Observe, the above solution is not the only way to partition $P$.
Exercise 1: For the $P$ shown above, draw all possible ways to partition $P$ into $y$-monotone pieces. What do all those ways have in common?

# Partitioning a Polygon

1. *y*-Monotone Polygons
   - partitioning into monotone pieces
   - adding diagonals between turn vertices
2. A Sweep Line Method
   - the global algorithm
   - handling a split vertex
   - handling a merge vertex
   - handling start, end, and regular vertices
3. Correctness and Cost
   - MAKEMONOTONE is correct
   - cost of MAKEMONOTONE

# input, output, and data structures

The polygon $P$ is given by a doubly connected edge list $\mathcal{D}$.

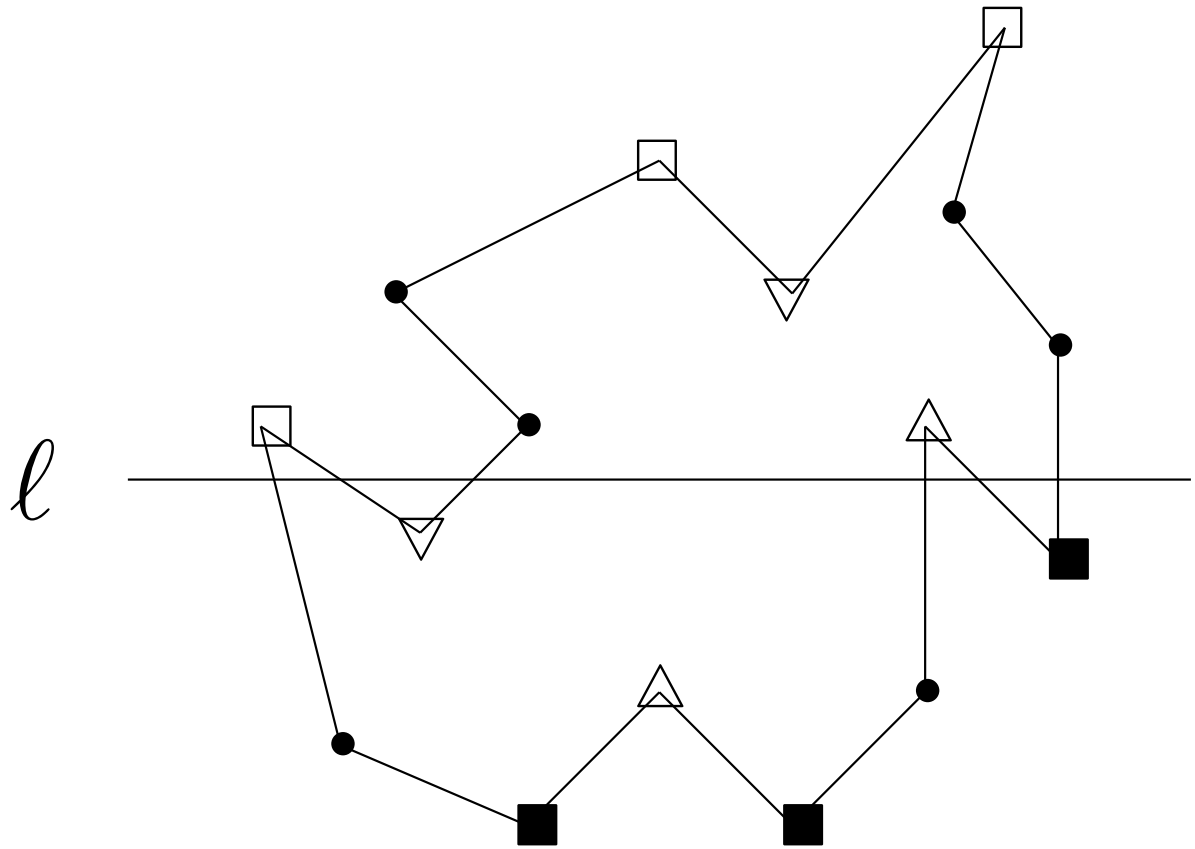
The $\mathcal{D}$ is updated to store the partitioning of $P$.

Two balanced binary search trees are used.

1. The event queue $Q$ stores the vertices of $P$,
   higher vertices have higher priority,
   when vertices are at equal height we prefer the leftmost vertex.
2. The status $T$ stores edges on the sweep line $\ell$,
   in the order, from left to right, in which they intersect $\ell$.

Updates to balanced binary search trees have logarithmic cost, proportional to the depth of the tree: $O(\log(n))$.

# define the status on the example




Exercise 2: For the $P$ and $\ell$ shown above, define the status $T$.
The status was introduced for the line segment intersection problem.

# the general outline

Algorithm MAKEMONOTONE($P$)

Input: a doubly connected edge list $\mathcal{D}$ stores a polygon $P$.
Output: updated $\mathcal{D}$ stores the partitioning of $P$
into $y$-monotone pieces.

1. Store the vertices of $P$ in the priority queue $Q$.
2. Initialize the status $T := \emptyset$.
3. while $Q \neq \emptyset$ do
4. $v = \text{pop}(Q)$
5. determine type($v$), the type of the vertex $v$
6. HANDLE($\mathcal{D}, v, \text{type}(v), T$)

We have five types of vertices, thus five types of handlers.

# five handler subroutines

Algorithm HANDLE($v$, type($v$), $T$)

Input: a doubly connected edge list $\mathcal{D}$ stores a polygon $P$,
a vertex $v$, its type, and the status $T$.
Output: updated $\mathcal{D}$, after handling the vertex.

1. If type($v$) is startv then HANDLESTART($\mathcal{D}$, $v$, $T$).
2. If type($v$) is endv then HANDLEEND($\mathcal{D}$, $v$, $T$).
3. If type($v$) is mergev then HANDLEMERGE($\mathcal{D}$, $v$, $T$).
4. If type($v$) is splitv then HANDLESPLIT($\mathcal{D}$, $v$, $T$).
5. If type($v$) is regularv then HANDLEREGULAR($\mathcal{D}$, $v$, $T$).

Exercise 3: Explain why type($v$) is a primitive operation,
that is: its cost is constant.

# Partitioning a Polygon

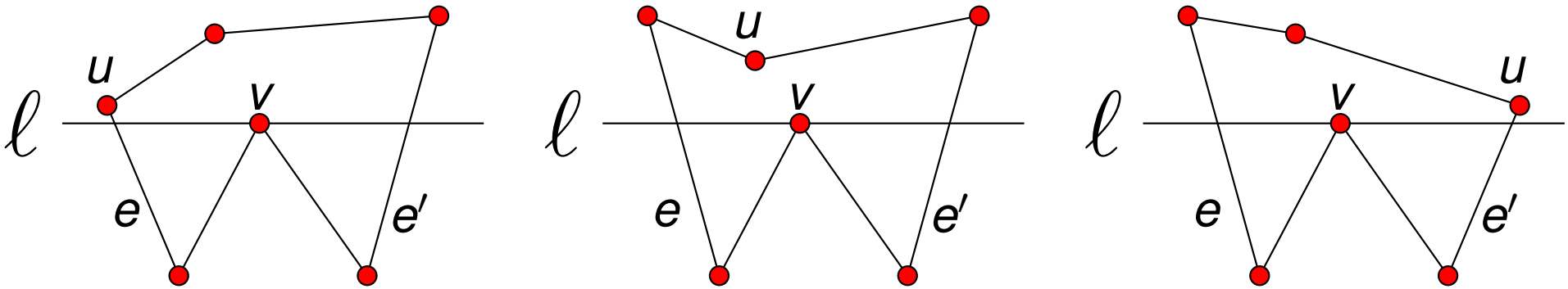
1. *y*-Monotone Polygons
   - partitioning into monotone pieces
   - adding diagonals between turn vertices
2. A Sweep Line Method
   - the global algorithm
   - handling a split vertex
   - handling a merge vertex
   - handling start, end, and regular vertices
3. Correctness and Cost
   - MAKEMONOTONE is correct
   - cost of MAKEMONOTONE

# handling a split vertex

Let the sweep line $\ell$ be at a split vertex $v$.

- Let $e$ be the first edge to the left of $v$, and
- let $e'$ be the first edge to the right of $v$.

Consider three different cases:




Let $u$ be the lowest vertex above the sweep line between $e$ and $e'$, $u$ may be the upper end point of $e$ or $e'$, $(u, v) \in P$, so connect $u$ to $v$.

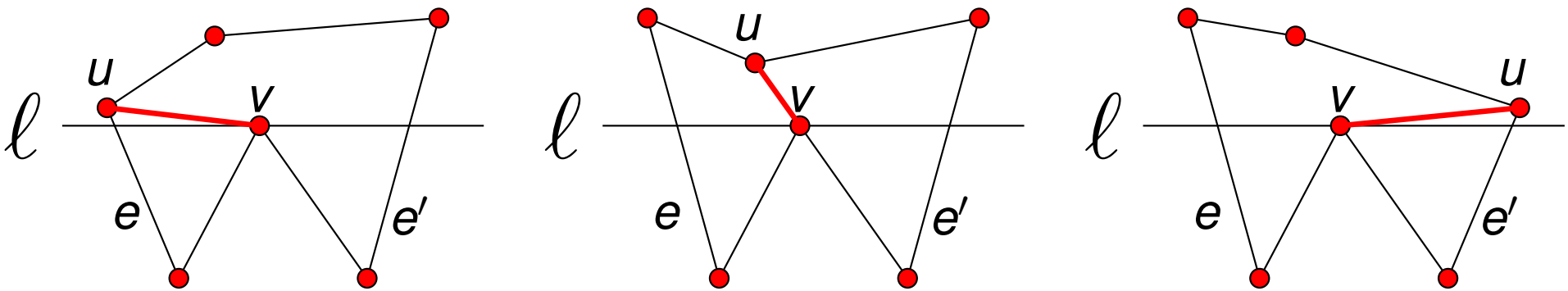
We call the vertex $u$ the helper of $e$, denoted by $helper(e)$.

# the status stores *helper*(*e*) for each edge *e*

## Definition (helper of an edge)

Given an edge *e* of a polygon *P*, and a sweep line $\ell$,
*helper(e)* is the lowest vertex above $\ell$ such that the horizontal segment connecting the vertex to *e* lies in *P*.

The status stores the edges in left-to-right order.




With each edge *e* in *T*, we store *helper*(*e*).

# the subroutine to handle a split vertex

Algorithm HANDLESPLIT($\mathcal{D}$, $v$, $T$).

Input: a doubly connected edge list $\mathcal{D}$ stores a polygon $P$,
a split vertex $v$ and the status $T$.
Output: updated $\mathcal{D}$, after handling the split vertex.

1. Search in $T$ the edge $e$ directly to the left of $v$.
2. Insert the diagonal $(helper(e), v)$ into $\mathcal{D}$.
3. Assign $v$ to $helper(e)$.
4. Let $e_v$ be the rightmost edge with top end point equal to $v$, insert $e_v$ in $T$.
5. Assign $v$ to $helper(e_v)$.

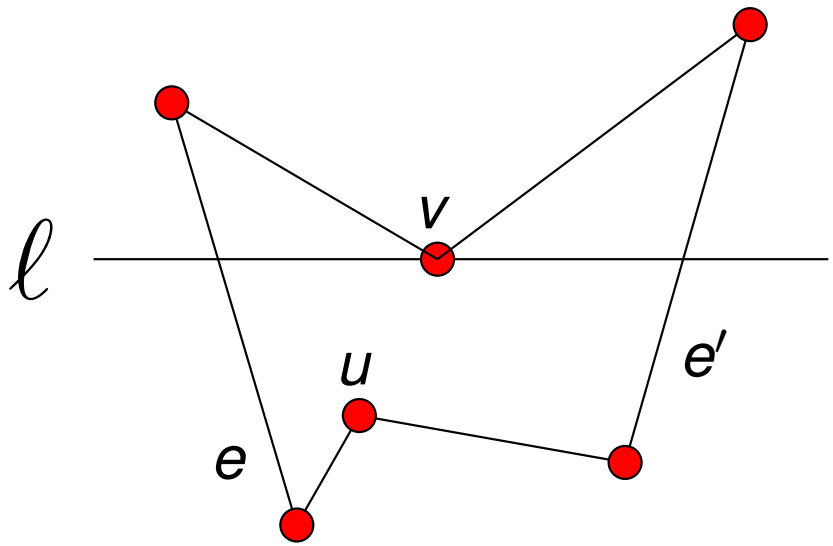
# Partitioning a Polygon

1. *y*-Monotone Polygons
   - partitioning into monotone pieces
   - adding diagonals between turn vertices
2. A Sweep Line Method
   - the global algorithm
   - handling a split vertex
   - handling a merge vertex
   - handling start, end, and regular vertices
3. Correctness and Cost
   - MAKEMONOTONE is correct
   - cost of MAKEMONOTONE

# handling a merge vertex

Let the sweep line $\ell$ be at a merge vertex $v$.

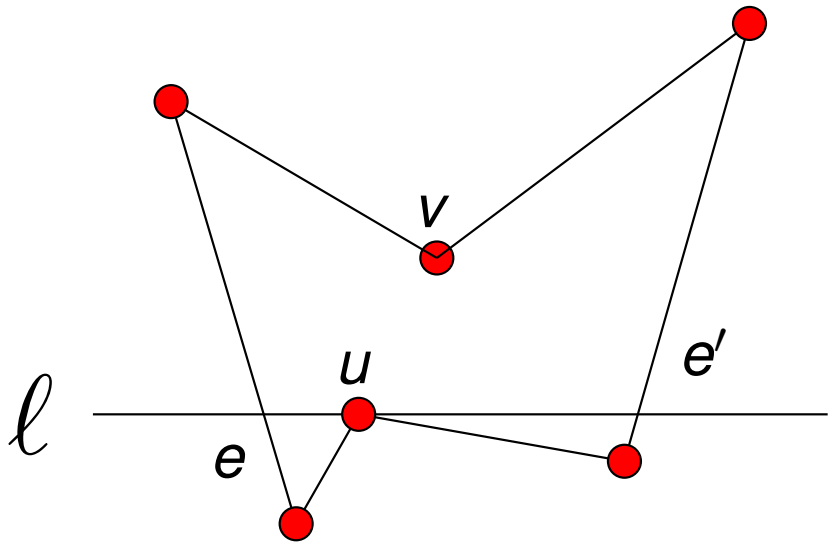
- Let $e$ be the first edge to the left of $v$, and
- let $e'$ be the first edge to the right of $v$.

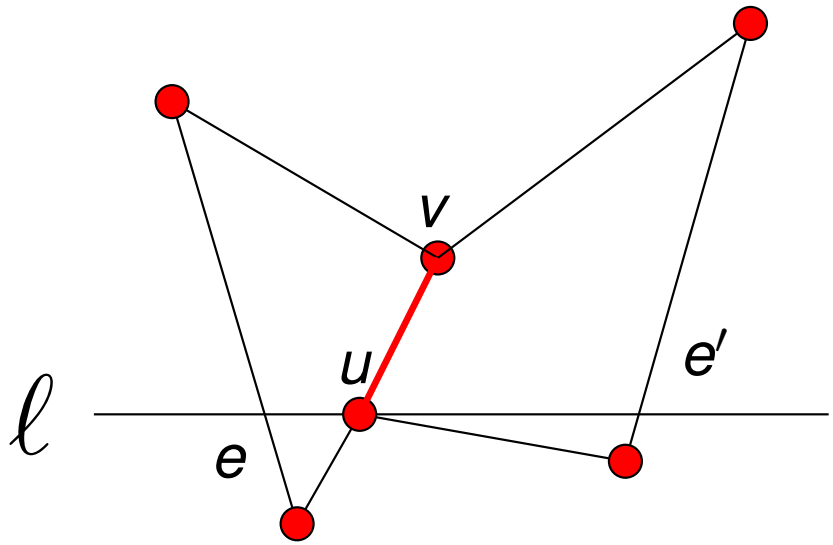



This seems to mirror the handling of a split vertex, however, when $\ell$ is at $v$, we do not know the highest vertex $u$, $(u, v) \in P$, to connect $u$ to $v$.

# partitioning at a merge vertex

When $\ell$ is at $u$:




*helper*$(e)$ changes from $v$ to $u$

$\Rightarrow$ if $v$ is a merge vertex, then add diagonal $(v, u)$ to $\mathcal{D}$.

# the subroutine to handle a merge vertex

Algorithm HANDLEMERGE($\mathcal{D}$, $v$, $T$).

Input: a doubly connected edge list $\mathcal{D}$ stores a polygon $P$,
a merge vertex $v$ and the status $T$.
Output: updated $\mathcal{D}$, after handling the merge vertex.

1. Let $e$ be the leftmost edge ending at $v$.
2. If $helper(e)$ is a merge vertex then
3. insert diagonal $(v, helper(e))$ in $\mathcal{D}$.
4. Delete $e$ from $T$.
5. Search in $T$ to find the edge $e$ directly left of $v$.
6. If $helper(e)$ is a merge vertex then
7. insert the diagonal $(v, helper(e))$ in $\mathcal{D}$.
8. Assign $v$ to $helper(e)$.

# Partitioning a Polygon

1. *y*-Monotone Polygons
   - partitioning into monotone pieces
   - adding diagonals between turn vertices
2. A Sweep Line Method
   - the global algorithm
   - handling a split vertex
   - handling a merge vertex
   - handling start, end, and regular vertices
3. Correctness and Cost
   - MAKEMONOTONE is correct
   - cost of MAKEMONOTONE

# the subroutine to handle a start vertex

Algorithm HANDLESTART($\mathcal{D}$, $v$, $T$).

Input: a doubly connected edge list $\mathcal{D}$ stores a polygon $P$,
a start vertex $v$ and the status $T$.
Output: updated $\mathcal{D}$, after handling the start vertex.

1. Let $e$ be the rightmost edge ending at $v$, insert $e$ in $T$.
2. Assign $v$ to $helper(e)$.

# the subroutine to handle an end vertex

Algorithm HANDLEEND($\mathcal{D}$, $v$, $T$).

Input: a doubly connected edge list $\mathcal{D}$ stores a polygon $P$,
an end vertex $v$ and the status $T$.
Output: updated $\mathcal{D}$, after handling the end vertex.

1. Let $e$ be the leftmost edge ending at $v$.
2. If $helper(e)$ is a merge vertex then
3. insert $(v, helper(e))$ into $\mathcal{D}$.
4. Delete $e$ from $T$.

# the subroutine to handle a regular vertex

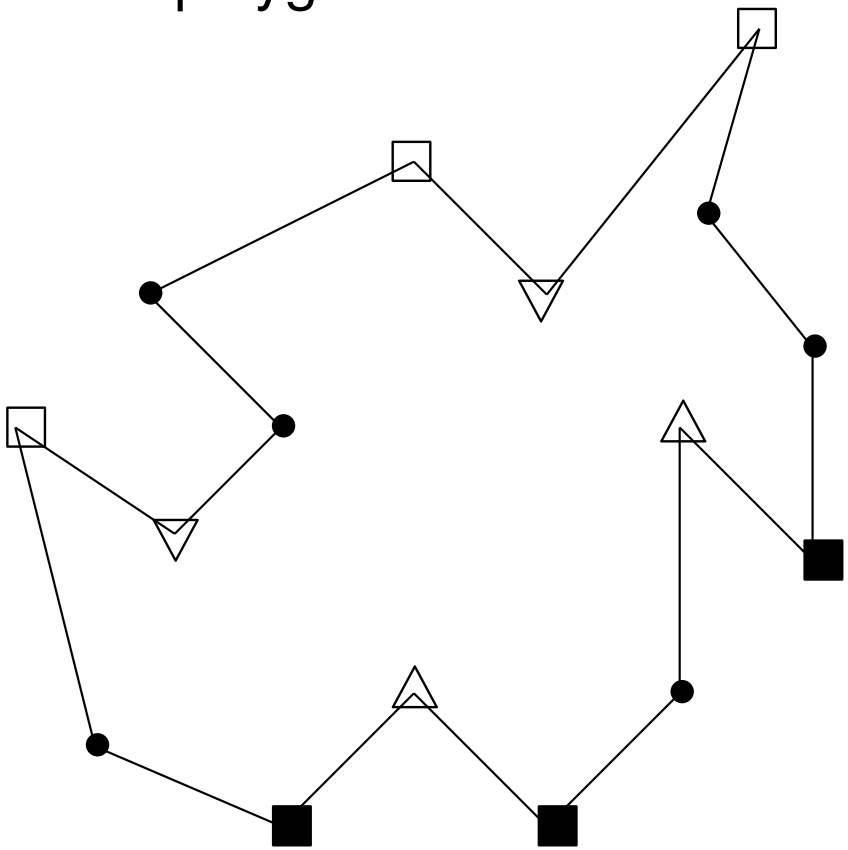
Algorithm HANDLEREGULAR($\mathcal{D}$, $v$, $T$).

Input: a doubly connected edge list $\mathcal{D}$ stores a polygon $P$,
a regular vertex $v$ and the status $T$.
Output: updated $\mathcal{D}$, after handling the regular vertex.

1. If the interior of $P$ lies to the right of $v$ then
2. let $e$ be the leftmost edge ending at $v$,
3. if $helper(e)$ is a merge vertex then
4. insert $(v, helpler(e))$ into $\mathcal{D}$
5. delete $e$ from $T$
6. let $e$ be the rightmost edge ending at $v$,
7. insert $e$ in $T$ and assign $v$ to $helper(e)$
8. else
9. search in $T$ to find the edge $e$ directly left of $v$
10. if $helper(e)$ is a merge vertex then
11. insert $(v, helper(e))$ into $\mathcal{D}$
12. assign $v$ to $helper(e)$.

# running the algorithm

Exercise 4: Consider the polygon $P$ below:



Run the algorithm on $P$ step by step.
For each new event point, describe the handling of the vertex.
Define the doubly connected edge lists of the input and the output.

# Partitioning a Polygon

1. *y*-Monotone Polygons
   - partitioning into monotone pieces
   - adding diagonals between turn vertices
2. A Sweep Line Method
   - the global algorithm
   - handling a split vertex
   - handling a merge vertex
   - handling start, end, and regular vertices
3. Correctness and Cost
   - MAKEMONOTONE is correct
   - cost of MAKEMONOTONE

# MAKEMONOTONE is correct

**Lemma (correctness of partitioning a polygon)**

*Algorithm* MAKEMONOTONE *add a set of nonintersecting diagonals that partitions $P$ into $y$-monotone polygons.*

Two parts of the proof:

1. MAKEMONOTONE removes all split and merge vertices.
2. All added diagonals are valid: do not intersect any edges of $P$.

# Partitioning a Polygon

1. $y$-Monotone Polygons
   - partitioning into monotone pieces
   - adding diagonals between turn vertices
2. A Sweep Line Method
   - the global algorithm
   - handling a split vertex
   - handling a merge vertex
   - handling start, end, and regular vertices
3. Correctness and Cost
   - MAKEMONOTONE is correct
   - cost of MAKEMONOTONE

# cost of MAKEMONOTONE

**Theorem (cost of partitioning a polygon)**

*A simple polygon with $n$ vertices can be partitioned into $y$-monotone polygons in $O(n \log(n))$ time using $O(n)$ storage.*

- Constructing the priority queue $Q$ takes $O(n)$, initializing $T$ runs in $O(1)$.
- Handling an event costs $O(\log(n))$ time, the cost to update a balanced binary search tree.

For $n$ points, the algorithm takes $O(n \log(n))$ time.

Storage is $O(n)$:

- every point is stored only once in $Q$,
- every edge is stored once in $T$.

# exercises

We continued the third chapter in the textbook.

Consider the following activities, listed below.

1. Write the solutions to exercises 1, 2, 3, and 4.
2. Consider the exercises 5, 6, 8 in the textbook.