CS633 Lecture 03
Polygon Triangulation

Jyh-Ming Lien

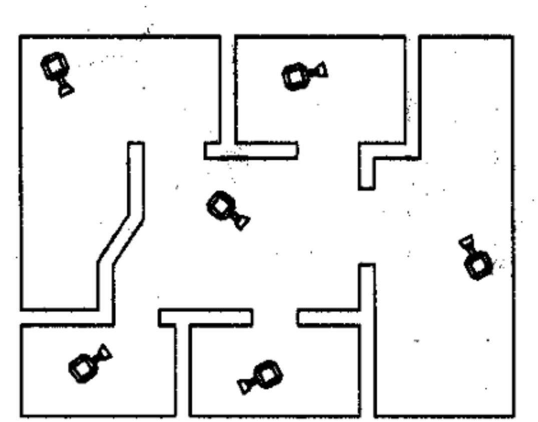
Dept. of Computer Science

George Mason University

Based on Chapter 3 of the textbook

Triangulation

- Chapter 3 of the Textbook



- Driving Applications
  - Guarding an art gallery
  - Rendering
  - Collision detection
  - Simulation (finite element method)
  - …

Guarding an Art Gallery

- Place as few cameras as possible
- Each part of the gallery must be visible to at least one of them
- Problems: how many cameras and where should they be located?

Art Gallery: Transform to Geometric Problem

- Floor plan may be sufficient and can be approximated as a simple polygon.
  - A simple polygon is a region enclosed by single closed polygonal chain that doesn't self-intersect
- A camera's position corresponds to a point in the polygon
- A camera sees those points in the polygon to which it can be connected with an open segment that lies in the interior of the polygon
  - assuming we have omni-cam that sees all directions

Art Gallery: Problem Analysis

- Bound the number of cameras needed in terms of *n*, number of vertices in the polygon
- 2 polygons with the same number of vertices may not be equally easy to guard
  - A convex polygon can always be guarded by 1
- Note: Find the minimum number of cameras for a specific polygon is NP-hard

Art Gallery: Our Plan

- Triangulate the polygon P
  - Decompose P into a set of simpler shapes
  - Decompose each shape to triangle

- Place a camera in each triangle

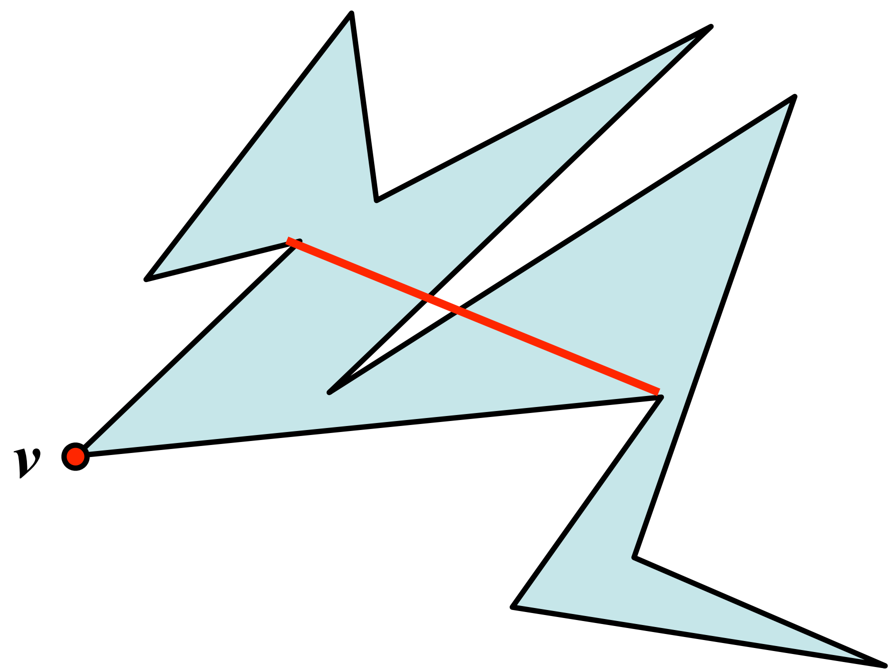
Triangulation of a Polygon

- **Definition**: A decomposition of a polygon into triangles by a maximal set of non-intersecting diagonals

- Triangulations are usually **NOT** unique

Can Any Polygon Be Triangulated?

- **Yes, but how?**



Size of Triangulation

- **Any triangulation of a simple polygon with n vertices consists of exactly $n-2$ triangles**

- **How many diagonals?**

Polygon Triangulation

- **Brute force**: Find a diagonal and triangulate the two resulting sub-polygons recursively: $O(n^2)$
- Ear clipping/trimming: $O(n^2)$

Clearly we need to do this more efficiently

Polygon Triangulation

- Triangulation of a convex polygon: $O(n)$
- First decompose a nonconvex polygon into convex pieces and then triangulate the pieces.
  - But, it is as hard to do a convex decomposition as to triangulate the polygon

=> Decompose a polygon into monotone pieces

Polygon Triangulations

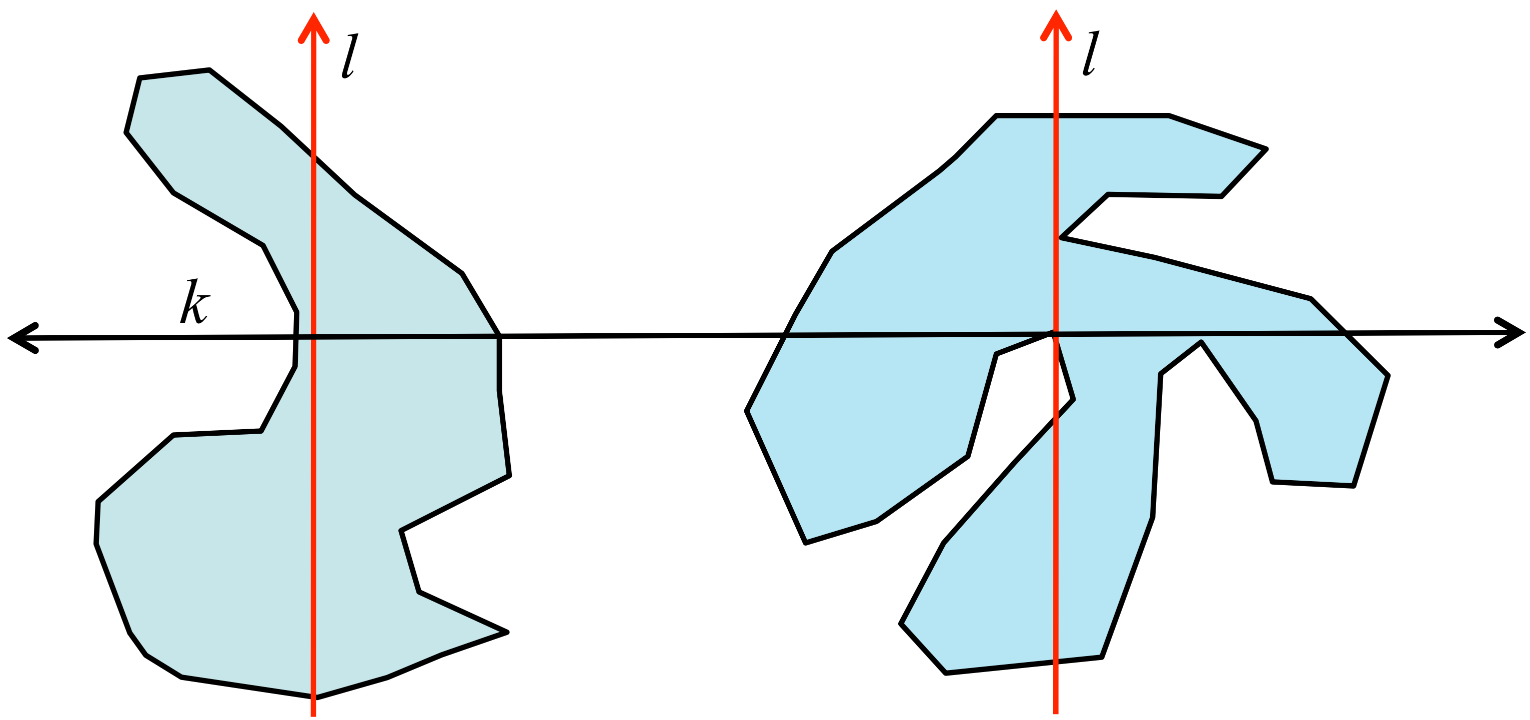
- Decompose a simple polygon into a monotone polygon: O($n$log$n$)
  - Plane sweep algorithm

- Triangulation of a monotone polygon: O(n)

$\boxed{\text{Total time to compute a triangulation: O}(n\log n)}$

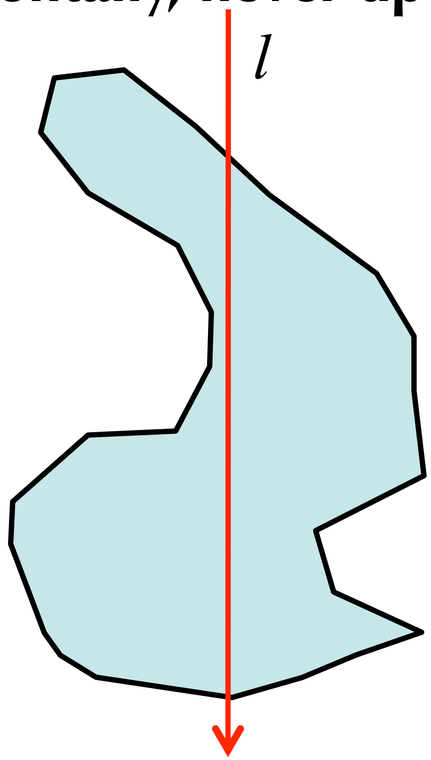
Partition a Polygon into Monotone Pieces

- A simple polygon is monotone w.r.t. a line *l* if for any line *l'* perpendicular to *l* **the intersection of the polygon with *l'* is connected**



Partition a Polygon into Monotone Pieces

- **Property: If we walk from a topmost to a bottom-most vertex along the left (or right) boundary chain, then we always move downwards or horizontally, never upwards**

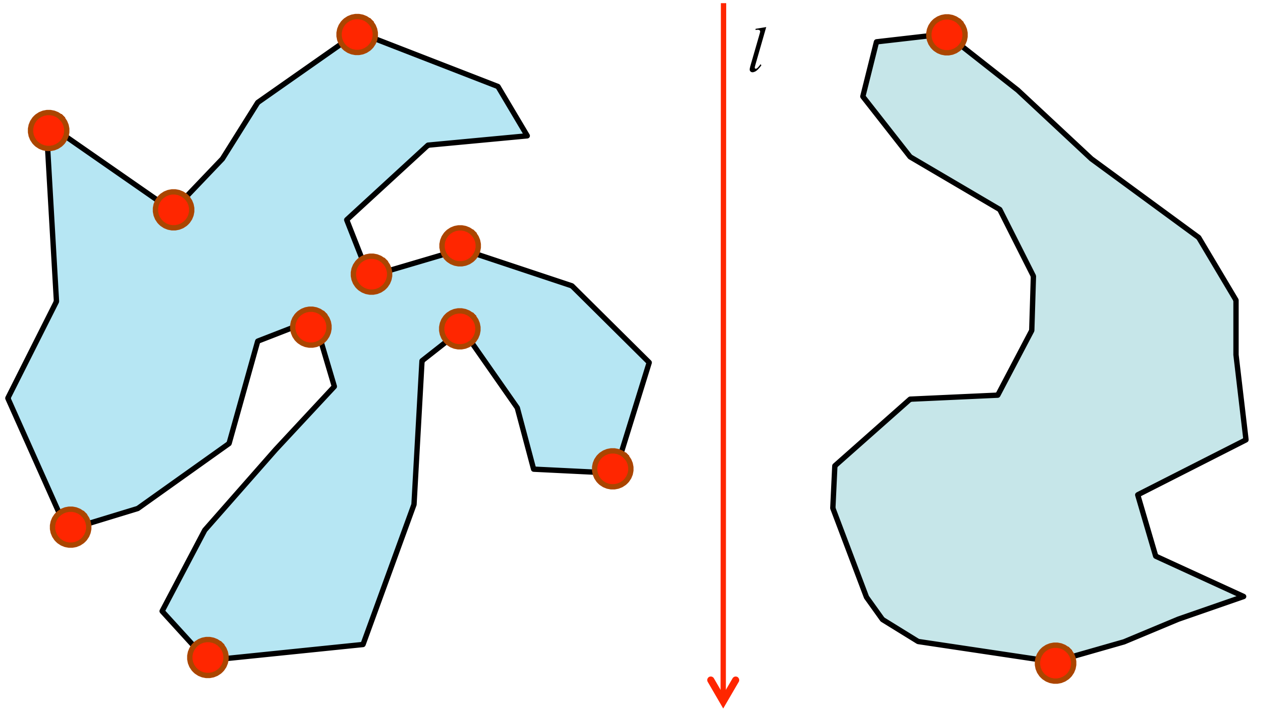


Turn Vertex

Imagine walking from the topmost vertex of P to the bottommost vertex on the left/right boundary chain……

- **Definition:** A vertex where the direction in which we walk switches from downward to upward or vice versa

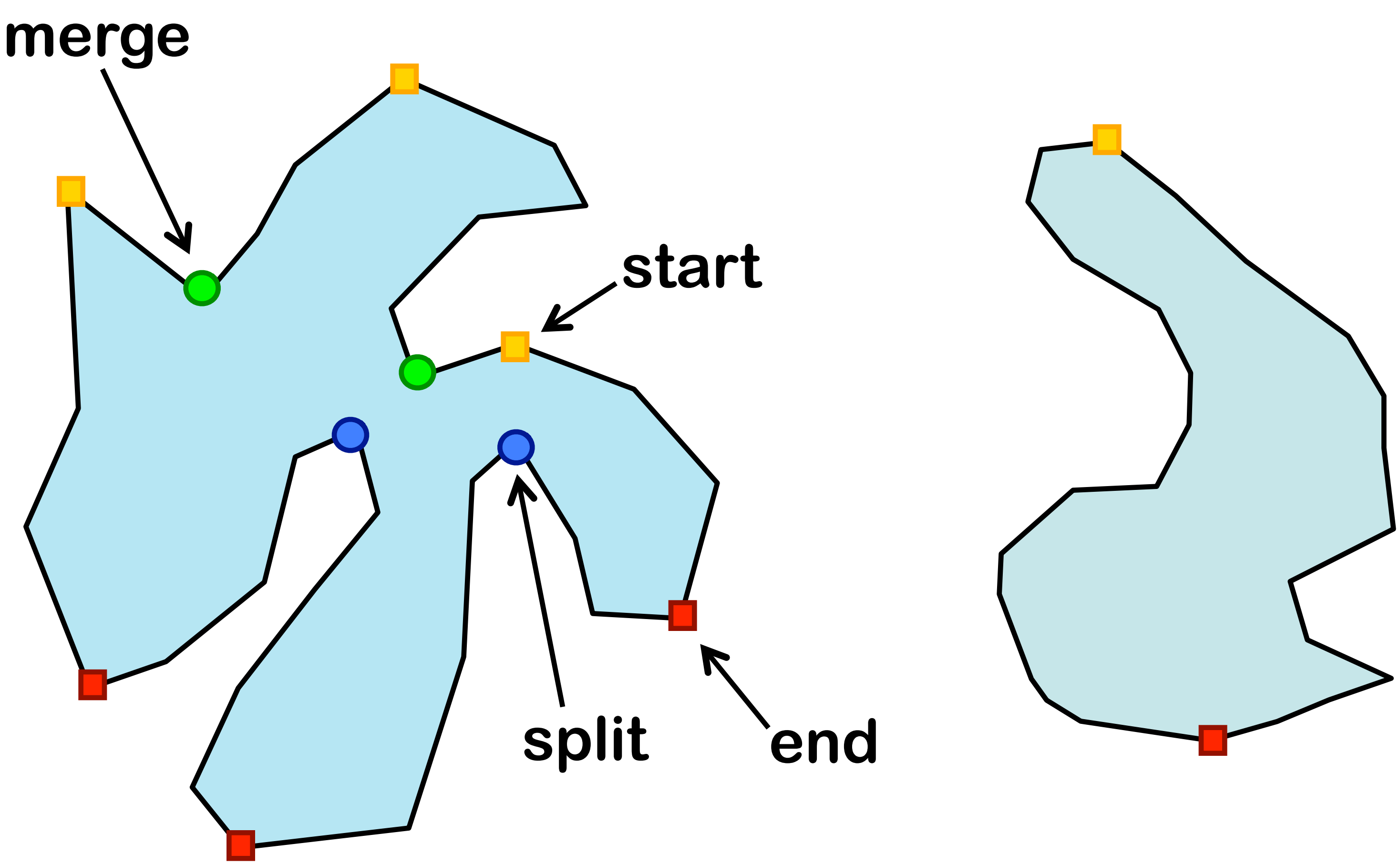
Turn Vertex



Types of Turn Vertices

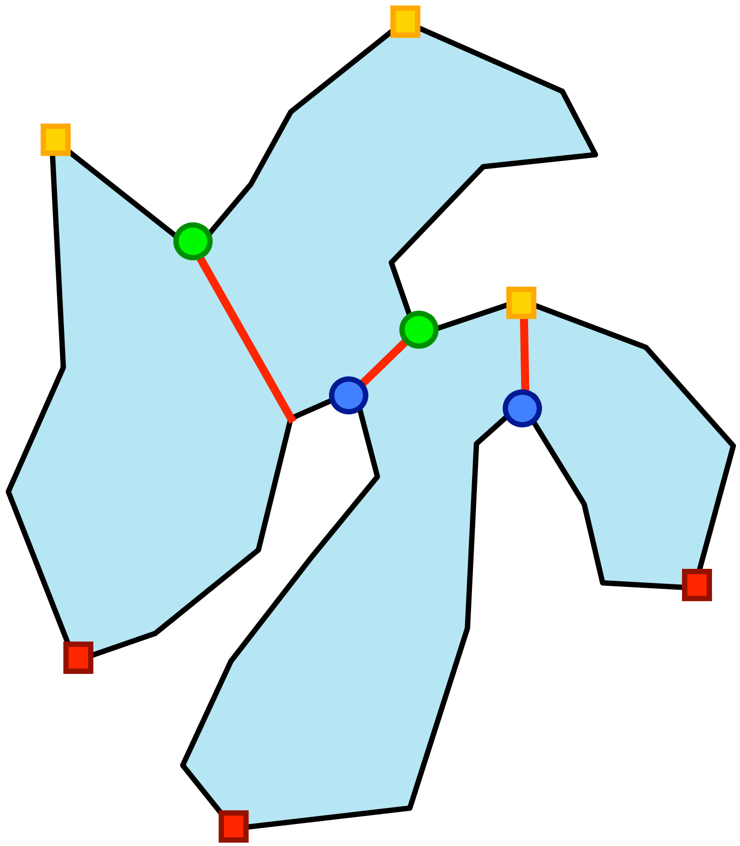
- **Start** Vertex - its two neighbors lie below it and the interior angle < 180°
- **End** Vertex - its two neighbors lie above it and the interior angle < 180°
- **Split** Vertex - its two neighbors lie below it and the interior angle > 180°
- **Merge** Vertex - its two neighbors lie above it and the interior angle > 180°

Types of Turn Vertices



Turn Vertex

- To partition a polygon into y-monotone pieces, get rid of split and merge vertices by adding diagonals

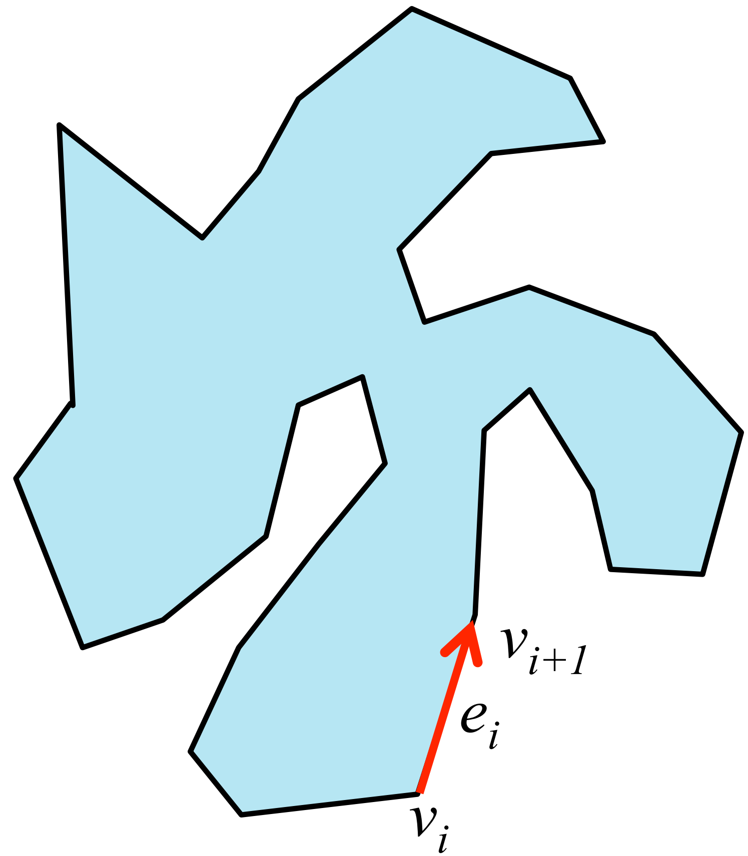


Property Summary

- The split and merge vertices are sources of local non-monotonicity
- A polygon is y-monotone if it has no split or merge vertices
- Use the plane-sweep method to remove split & merge vertices

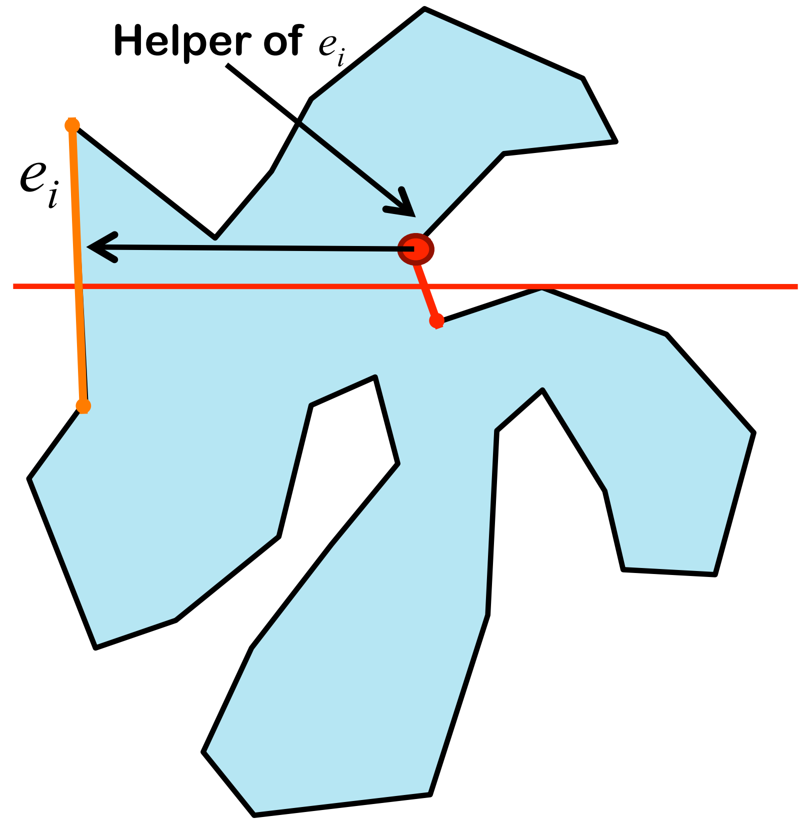
Plane Sweep

- **Input: A simple polygon P**
  - $v_1 \ldots v_n$: **a counter-clockwise enumeration of vertices of** P
  - $e_1 \ldots e_n$: **a set of edges of** P, **where** $e_i = segment(v_i, v_{i+1})$
- **Events (places where the sweep line status changes)**
  - **Polygon vertices**
  - **Sorted from top to bottom**



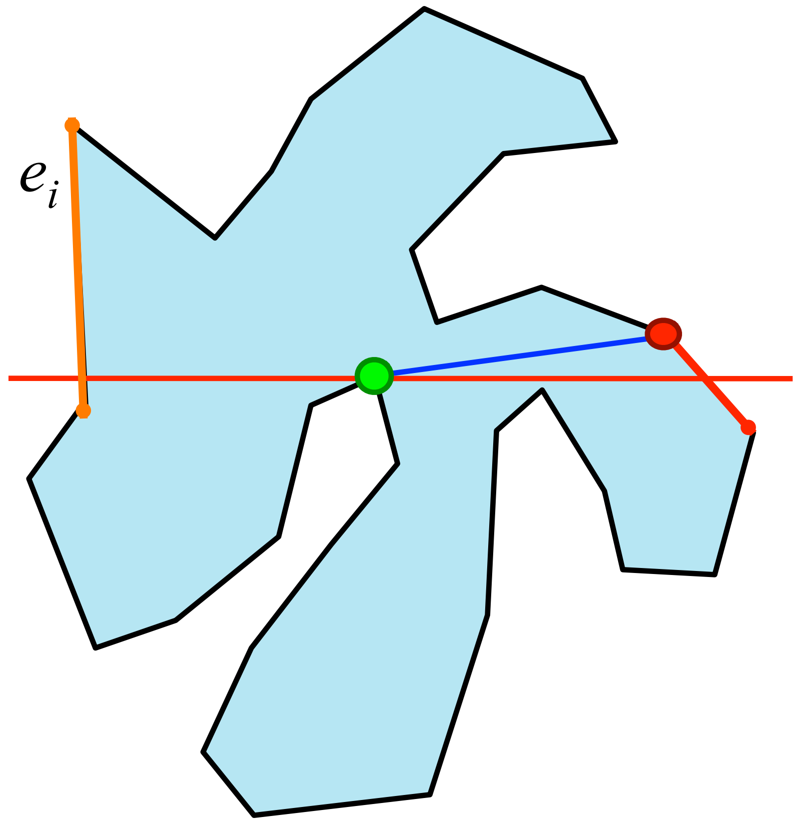
Plane Sweep

- **Status of the sweep line**
  - Intersecting edges
    - Ordered from left to right
    - Only store edges that P is on the right (Should be clear later)
  - Helper of the edge
- **The helper of edge e_i**
  - Is a vertex
  - The lowest vertex above l that can see e_i



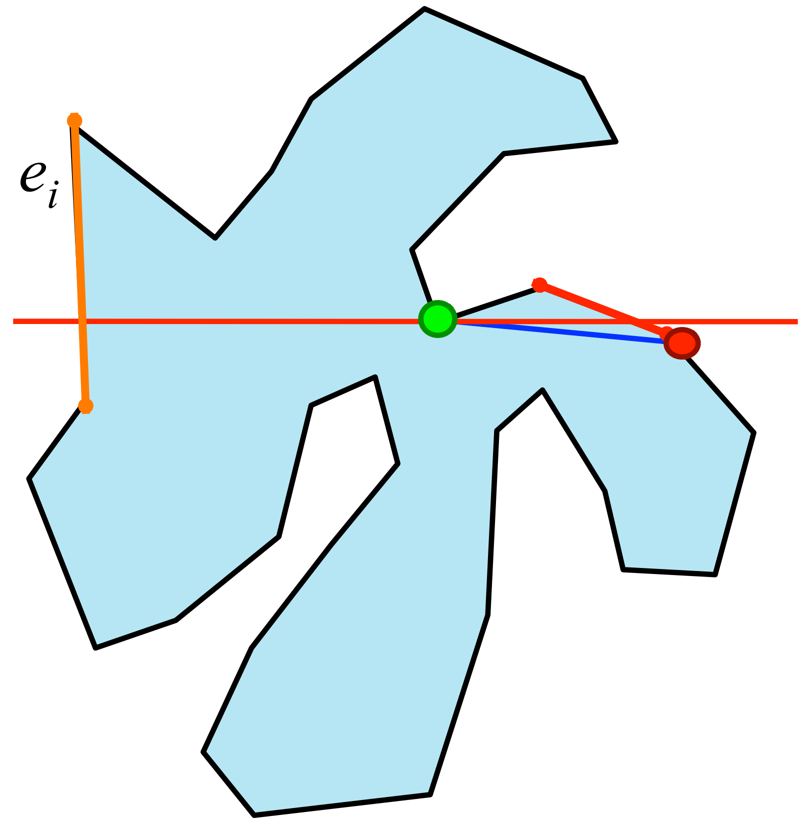
Remove Split Point

- If the sweep line stops at a split point
  - add a diagonal
  - from the split point
  - To the lowest point (above *l*) between its left and right segment (in the status)
  - this is exactly the helper of the segment



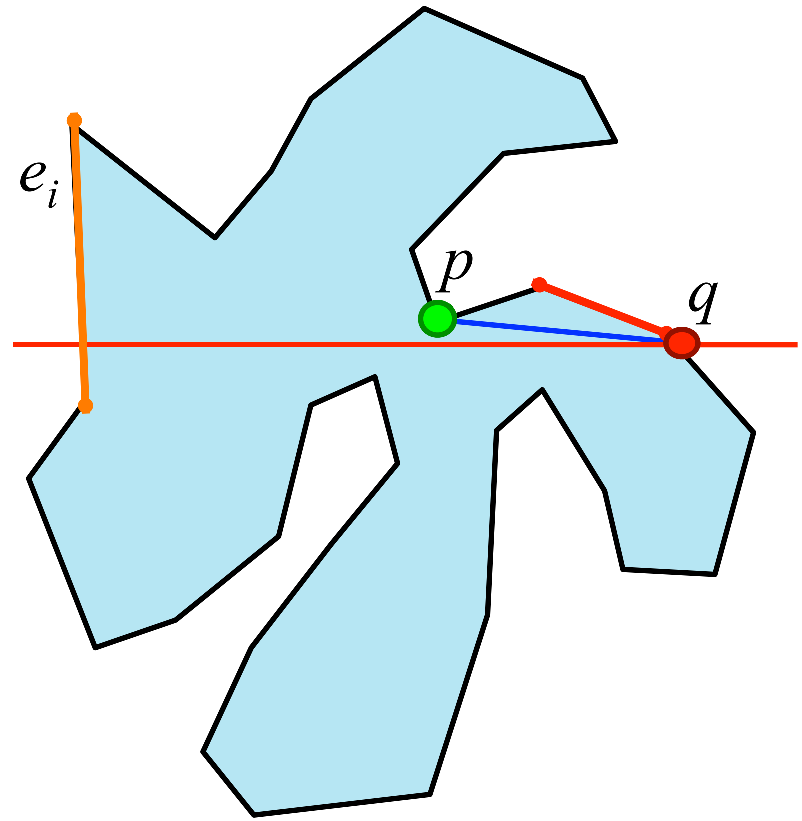
Remove Merge Point

- If the sweep line stops at a merge point
  - add a diagonal
  - from the merge point
  - To the highest point (below *l*) between its left and right segment (in the status)



Remove Merge Point

- Merge point can be also handled using helper!
  - When the sweep line is at q, the helper of e_i is p
  - After at q, the helper of e_i is q
  - When a merge point is replaced we add a diagonal



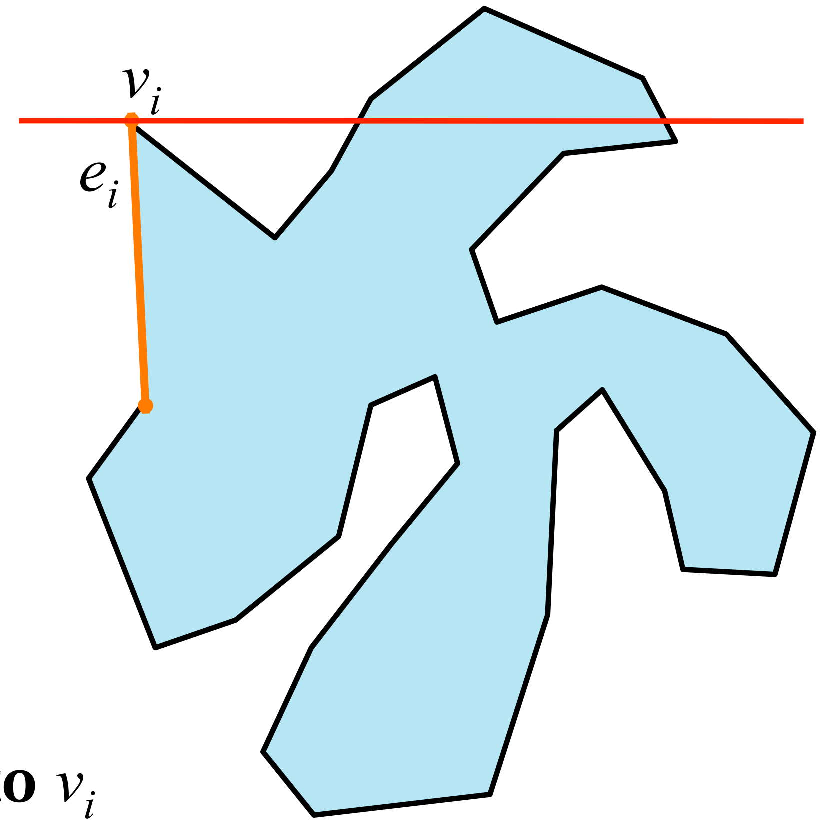
Make Monotone: Algorithm

Input: A simple polygon P

Output: A partitioning of P into monotone subpolygons

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

Start Vertex



(Insert e_i)

Insert e_i in T and set $helper(e_i)$ to v_i

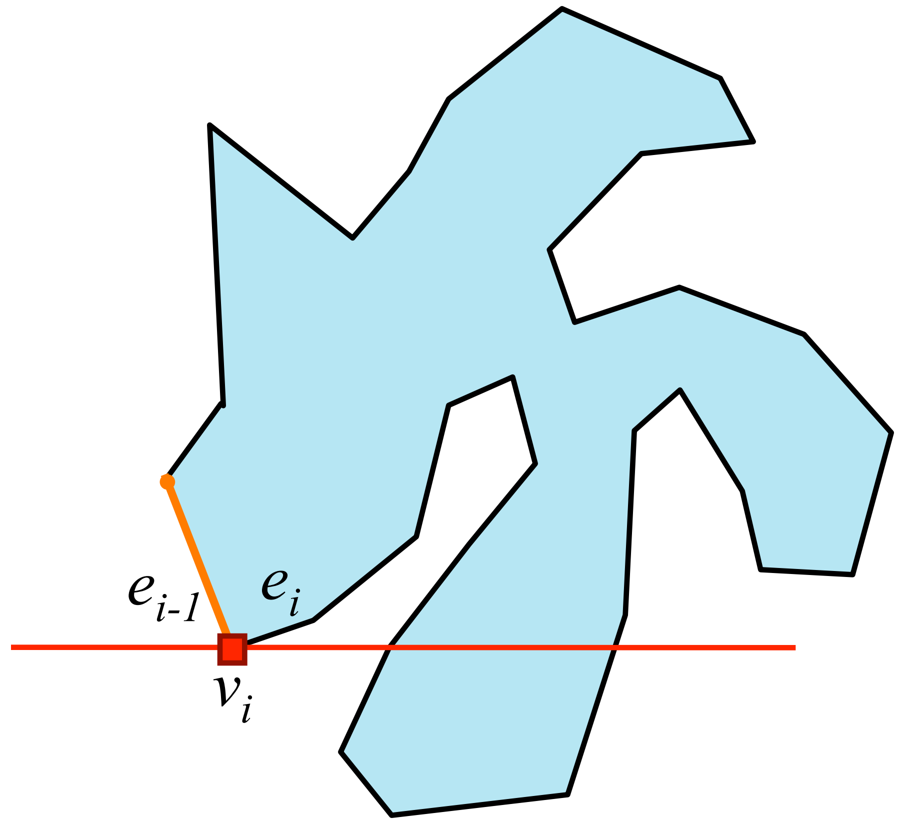
End Vertex

(Delete e_{i-1})

1. **if** *$helper(e_{i-1})$* **is a merge vertex**

   then Insert diagonal connecting v_i **to** *$helper(e_{i-1})$* **in** D

2. **Delete** e_{i-1} **from** T



Split Vertex

(Update e_j)

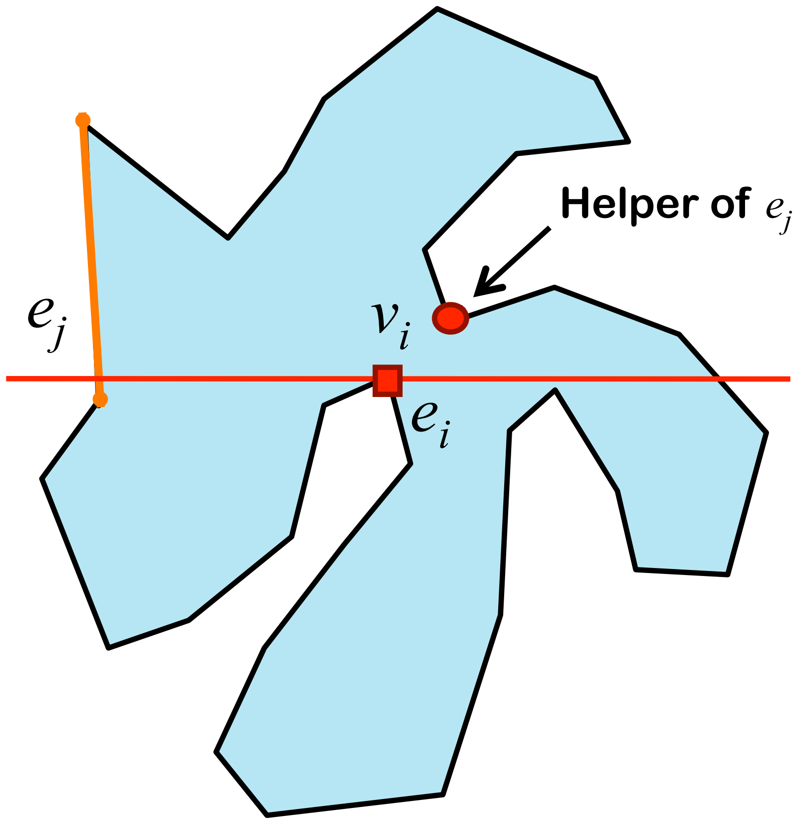
Search in T to find the edge e_j directly left of v_i

Insert diagonal connecting v_i to $helper(e_j)$ in D

$helper(e_j) \leftarrow v_i$

(Insert e_i)

Insert e_i in T and set $helper(e_i)$ to v_i



Merge Vertex

(Delete e_{i-1})

if $helper(e_{i-1})$ is a merge vertex

then Insert diagonal connecting v_i to $helper(e_{i-1})$ in D

Delete e_{i-1} from T

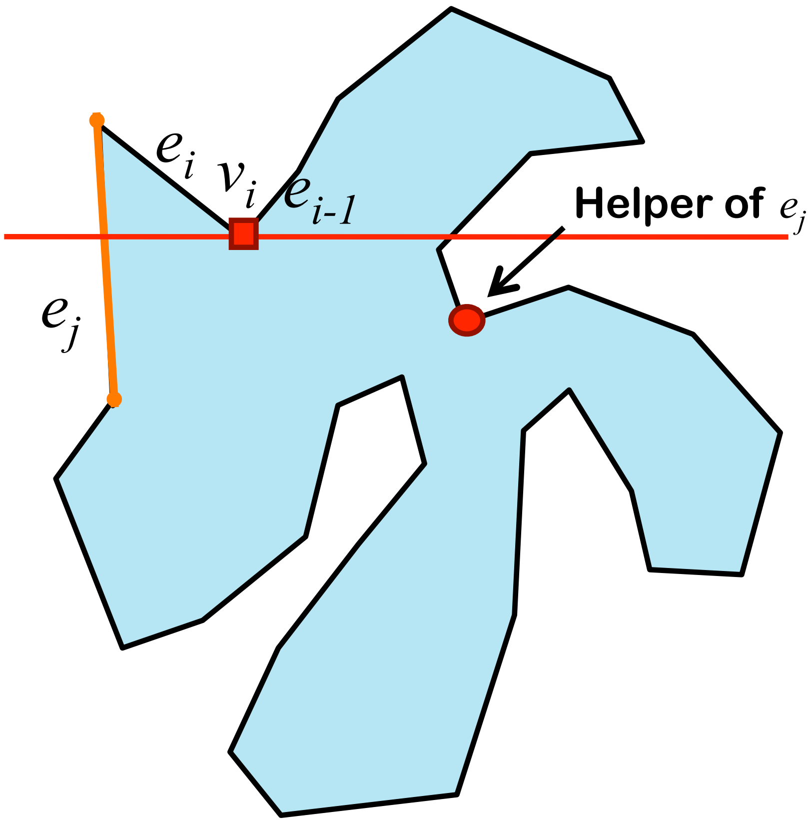
(Update e_j)

Search in T to find the edge e_j directly left of v_i

if $helper(e_j)$ is a merge vertex

then Insert diagonal connecting v_i to $helper(e_j)$ in D

$helper(e_j) \leftarrow v_i$



Regular Vertex

- the interior of P lies to the right of v_i

  (Delete e_{i-1})

  if $helper(e_{i-1})$ is a merge vertex

  then Insert diag. connect v_i to $helper(e_{i-1})$ in D

  Delete e_{i-1} from T

  (Insert e_i)

  Insert e_i in T and set $helper(e_i)$ to v_i

- the interior of P lies to the left of v_i

  (Update e_j)

  Search in T to find the edge e_j directly left of v_i

  if $helper(e_j)$ is a merge vertex

  then Insert diag. connect v_i to $helper(e_j)$ in D

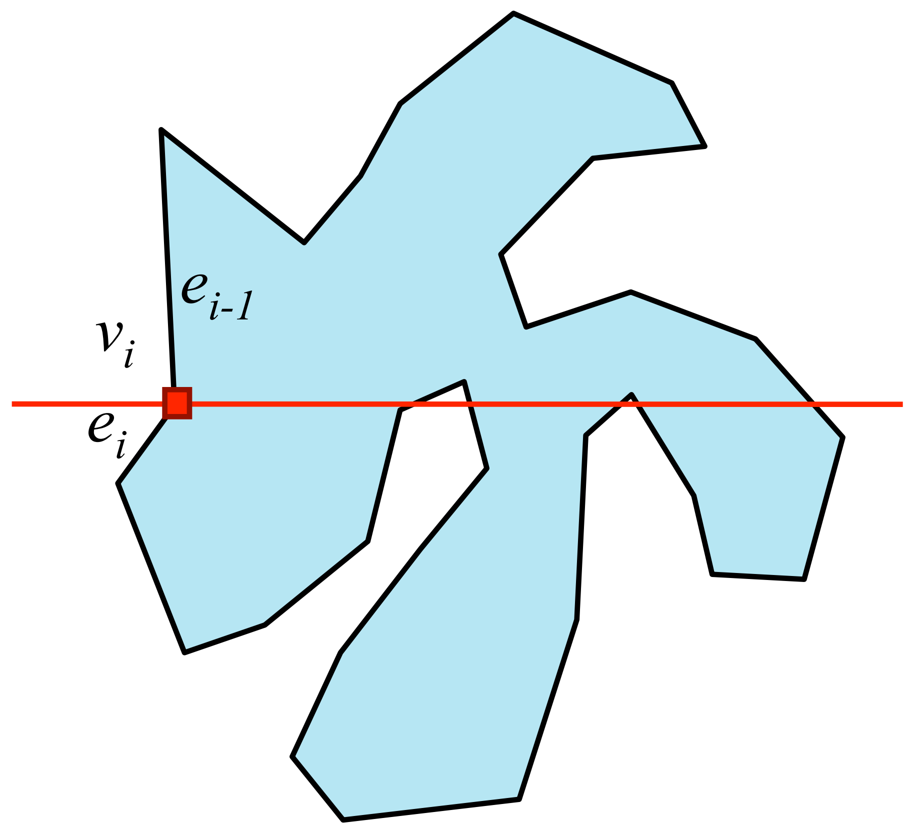
  $helper(e_j) \leftarrow v_i$

Regular Vertex

- the interior of P lies to the right of v_i

(Delete e_{i-1})

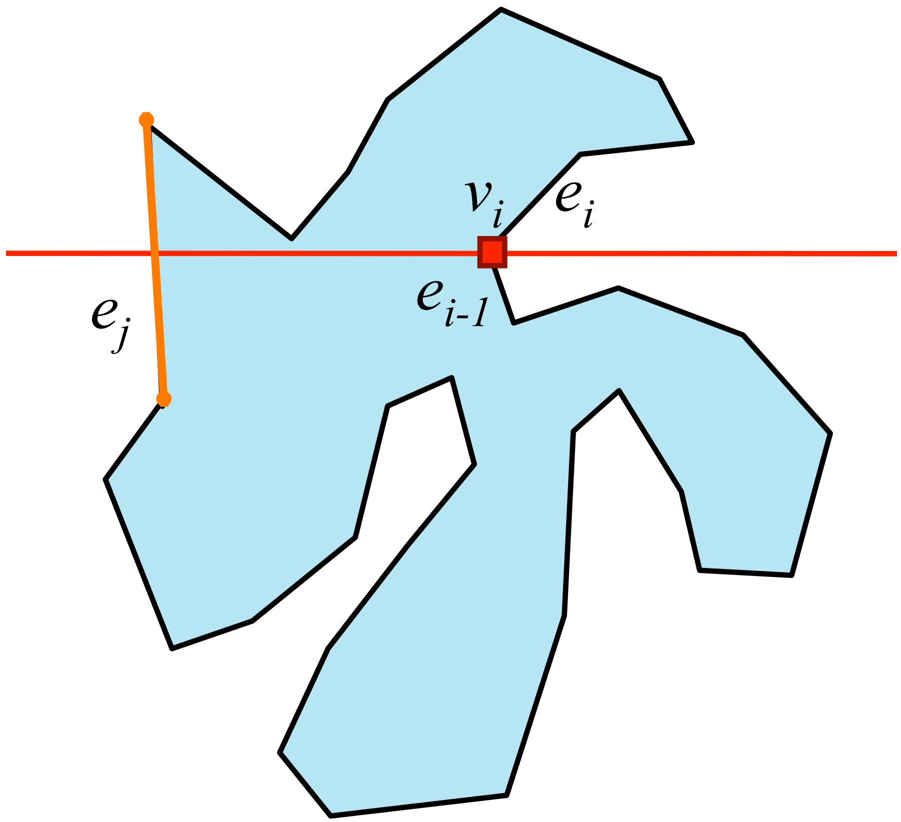
(Insert e_i)



Regular Vertex

- the interior of P lies to the left of v_i

(Update e_j)

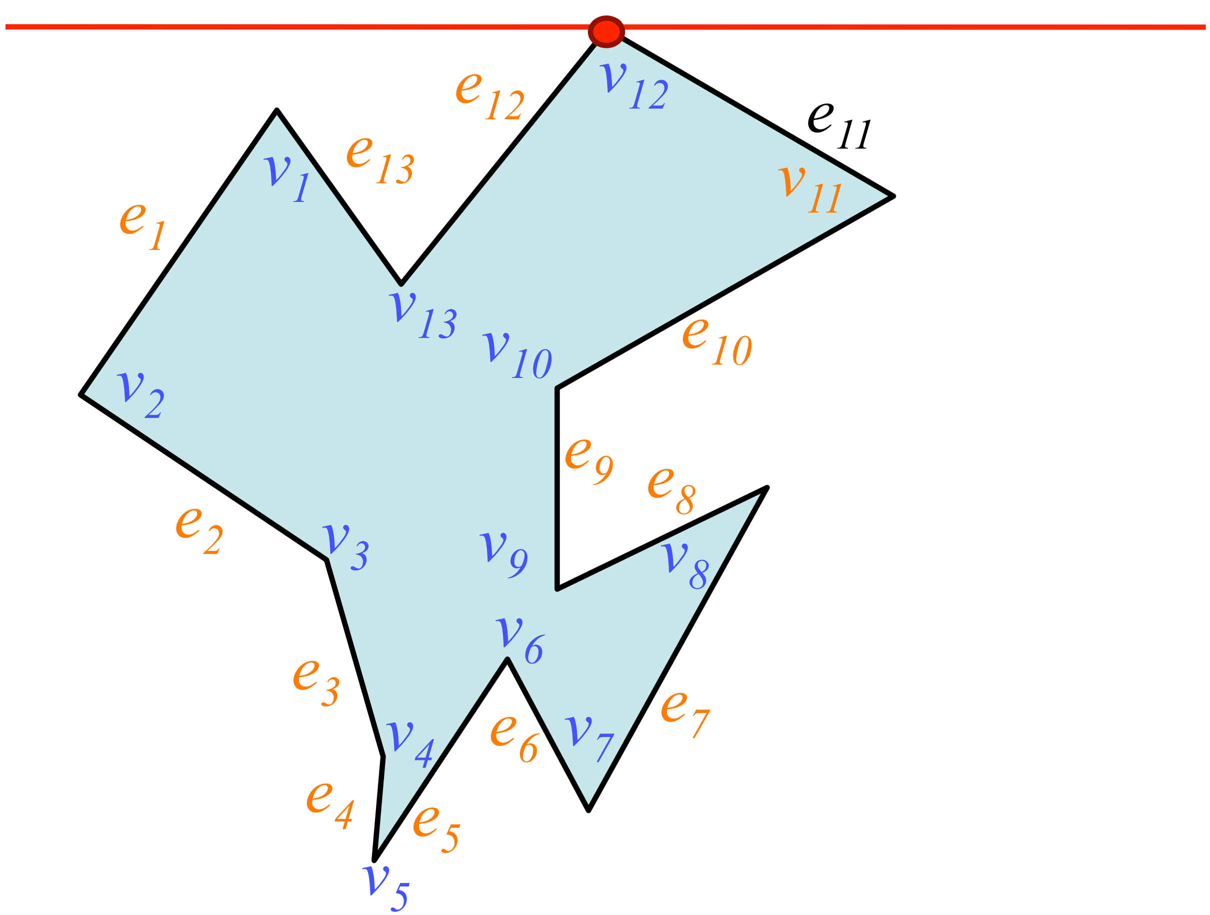


Partitioning Analysis

- **Construct priority queue:** $O(nlogn)$
- **Initialize T:** $O(1)$
- **Handle an event:** $O(\log n)$
  - **one operation on** Q: $O(logn)$
  - **at most 1 query, 1 insertion & 1 deletion on** T**:** $O(logn)$

- **Total run time:** $O(n \log n)$
- **Storage:** $O(n)$

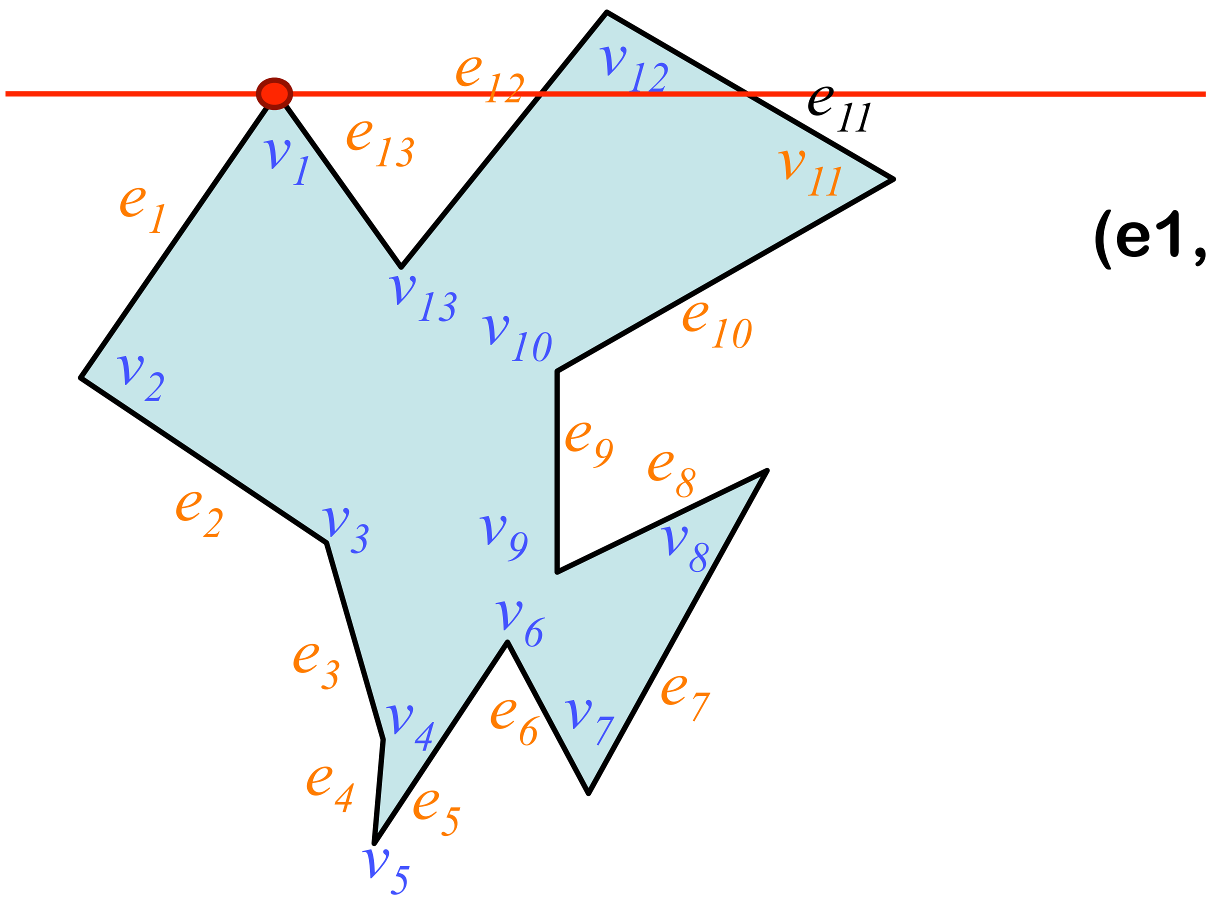
Example



T=

(e12, v12)

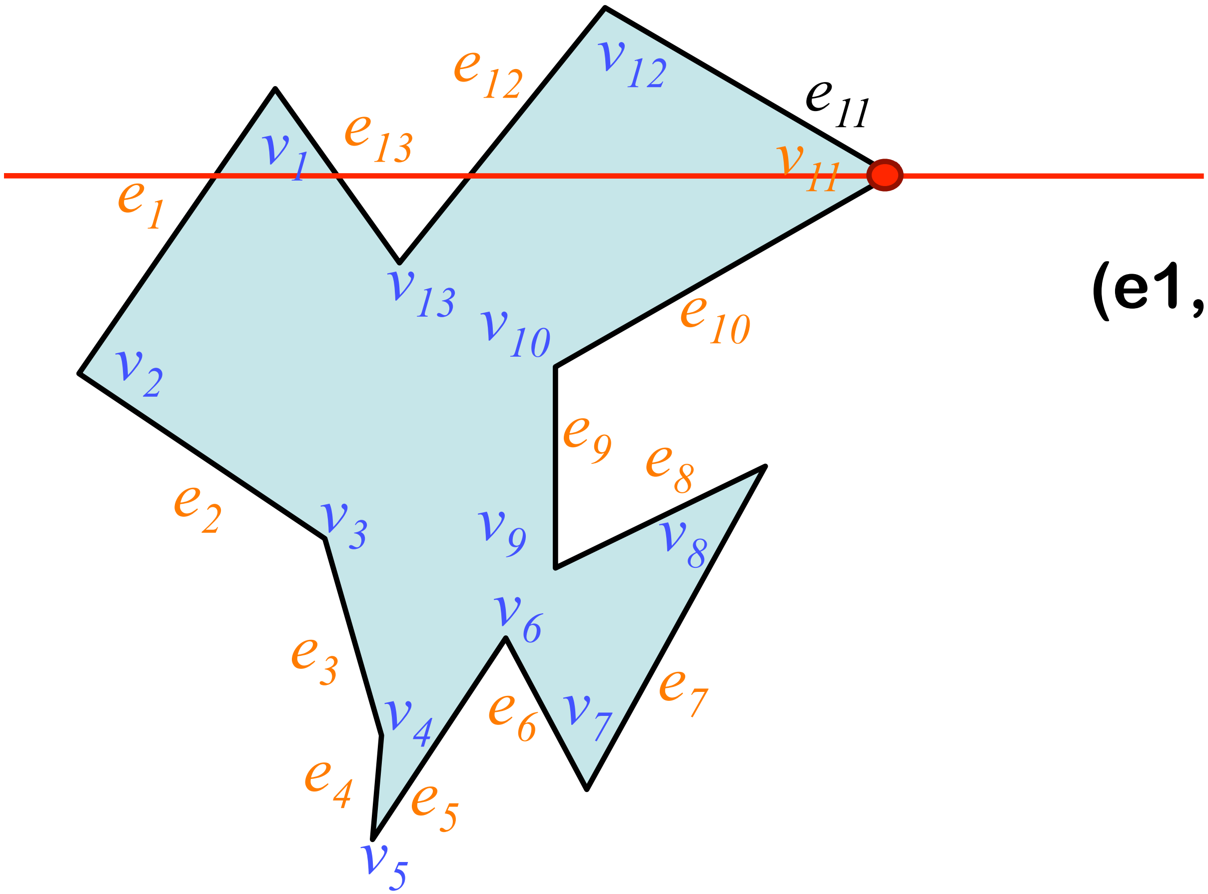
Example



T=

(e1, v1) (e12, v12)

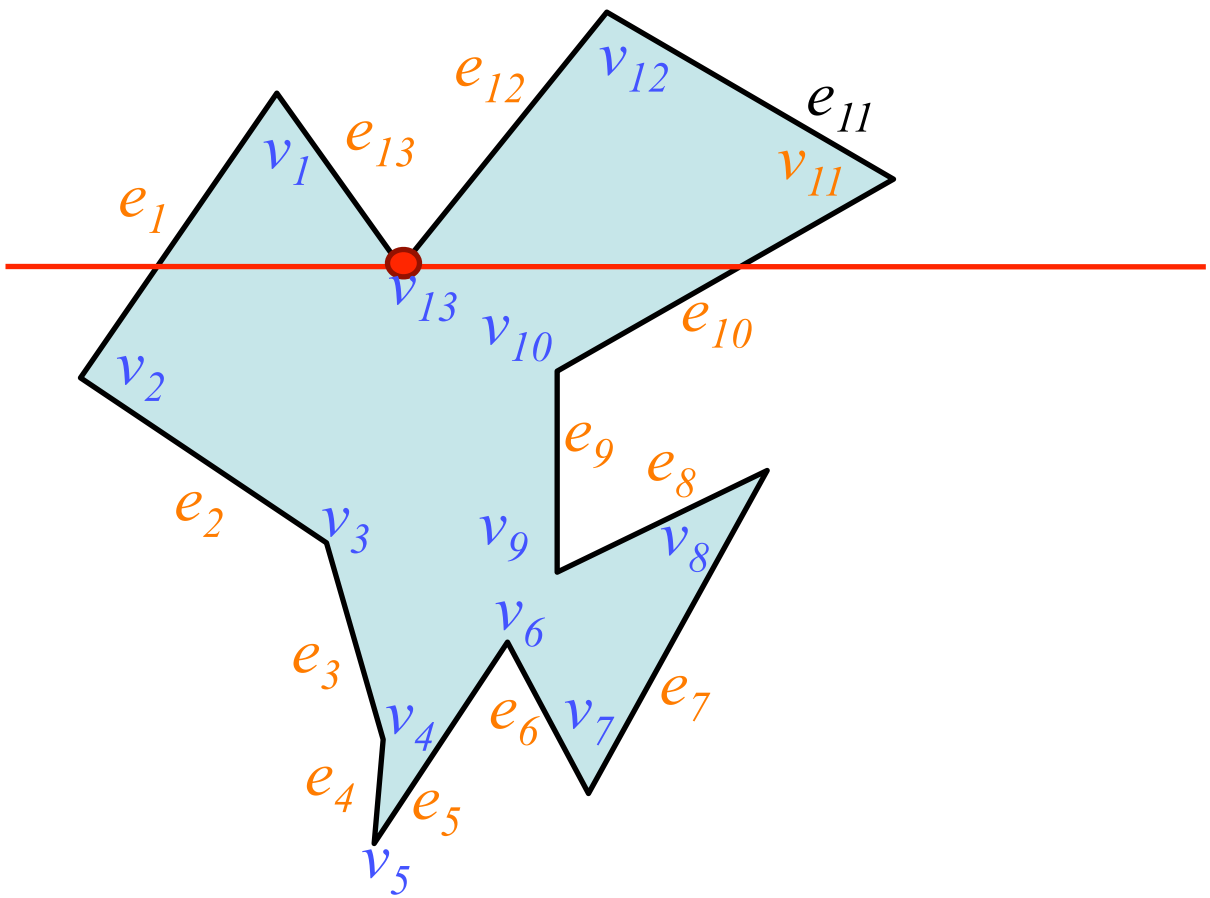
Example



T=

(e1, v1) (e12, v11)

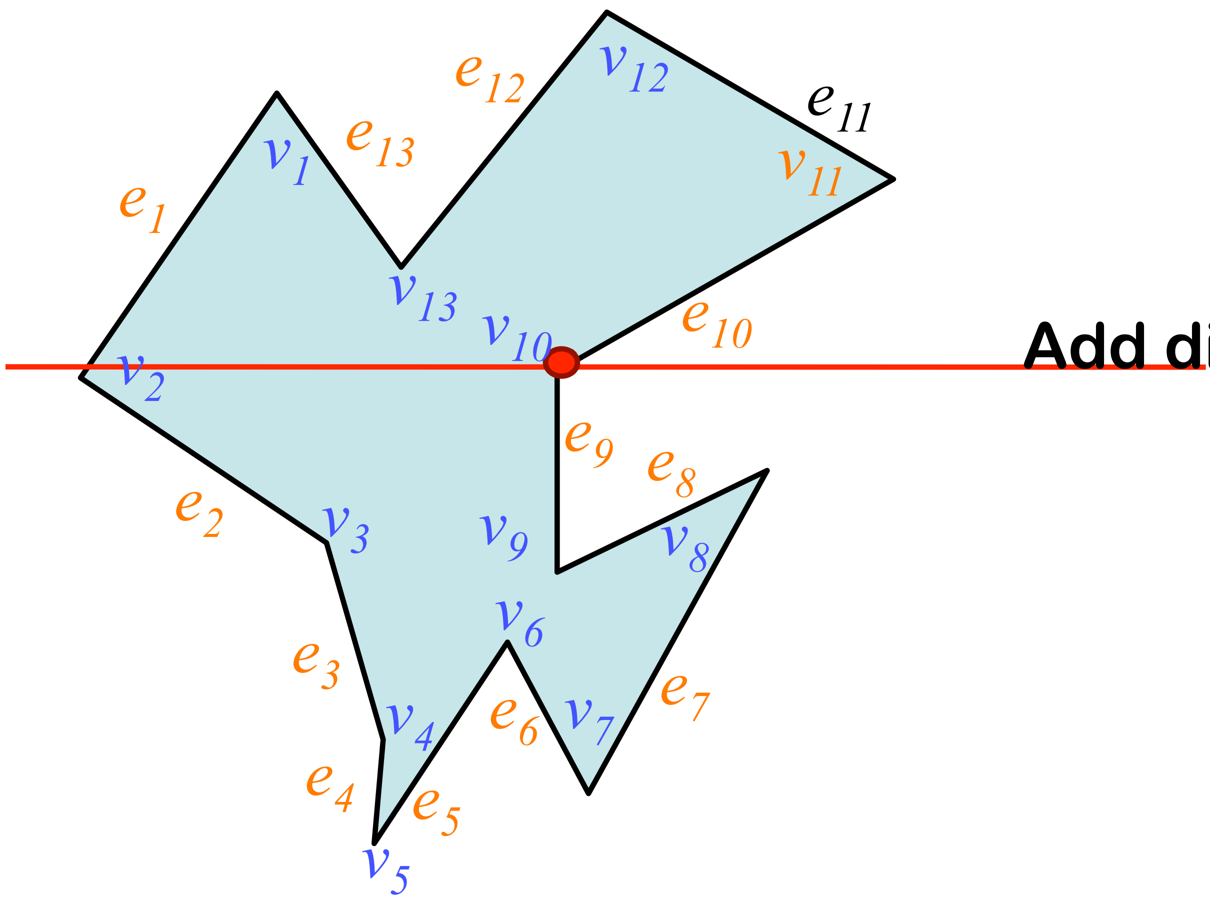
Example



T=

(e1, v13)

Example

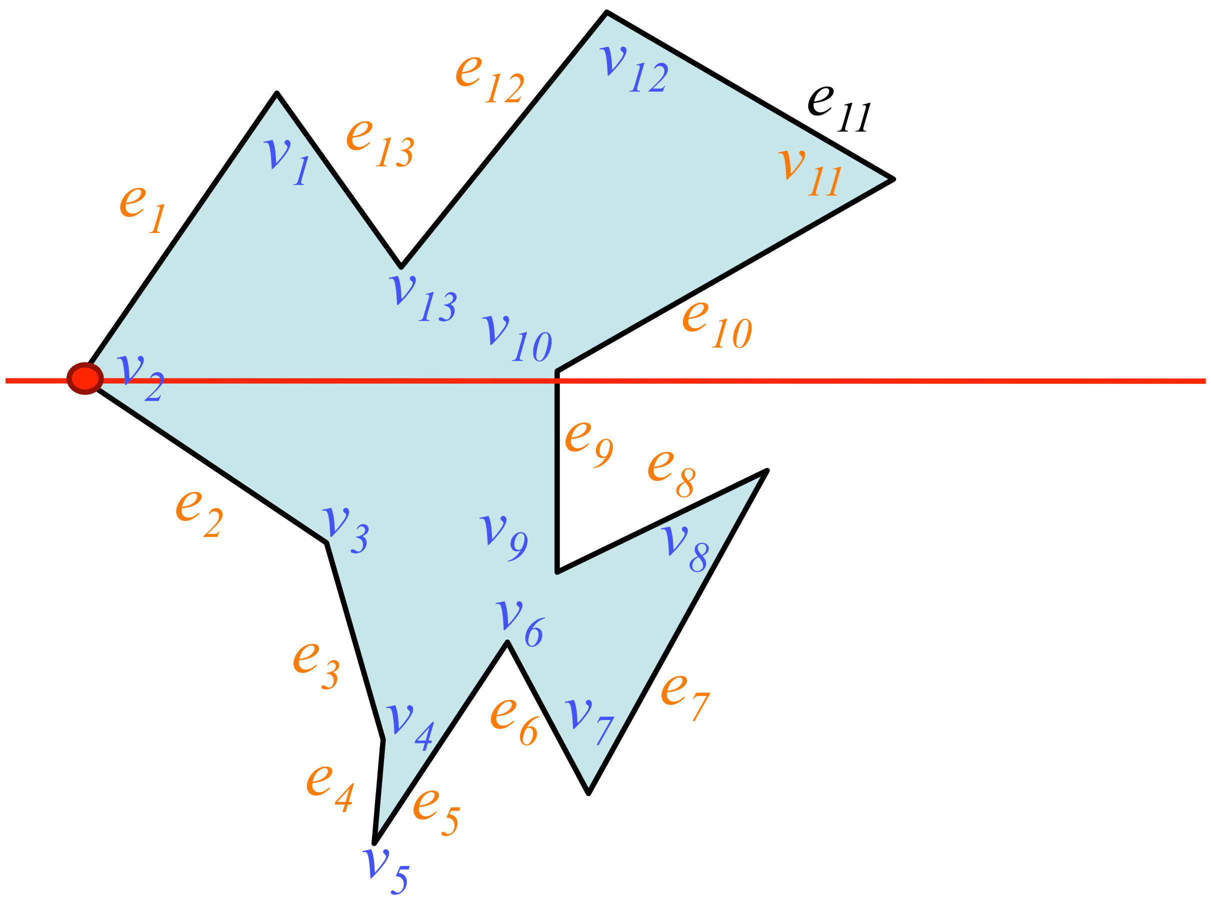


T=

Add diagonal v13v10

(e1, v10)

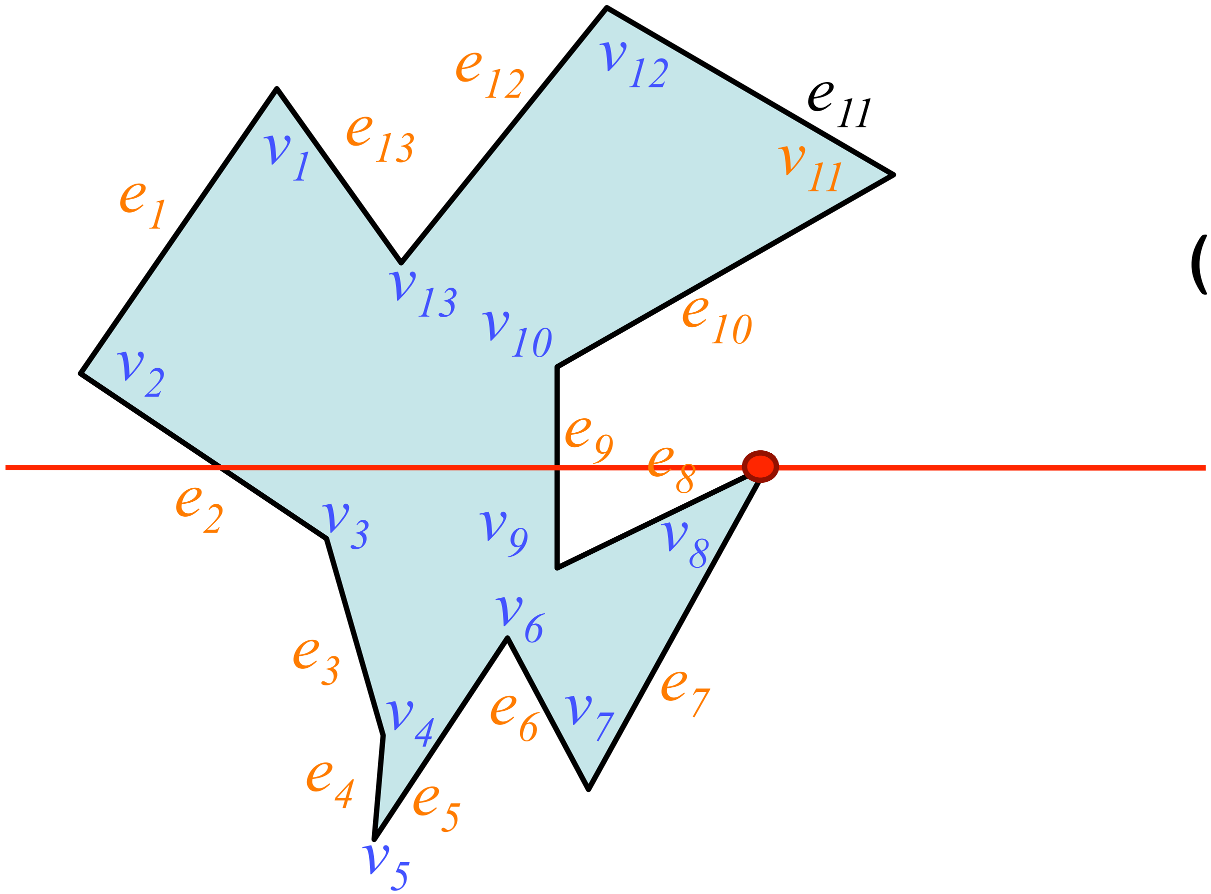
Example



T=

(e2, v2)

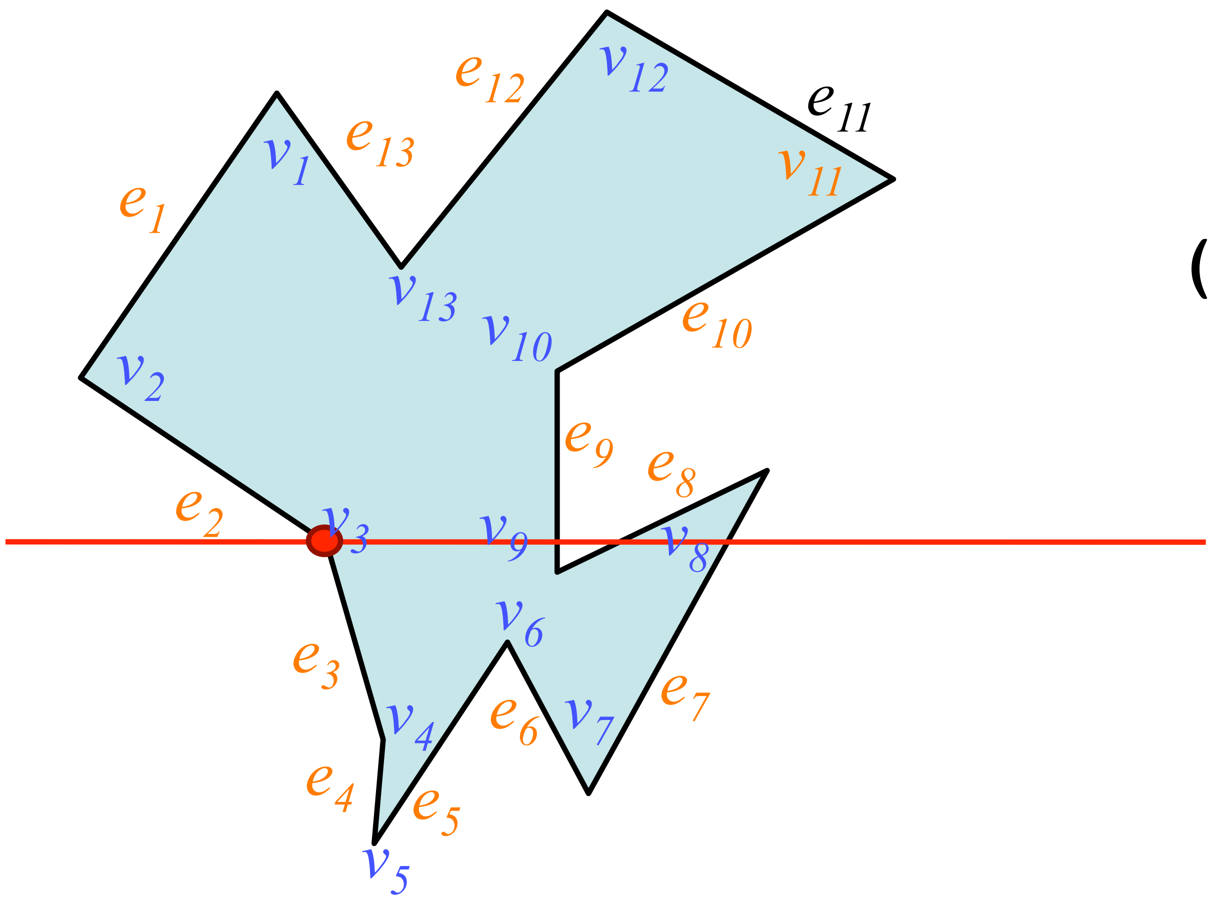
Example



T=

(e2, v2) (e8,v8)

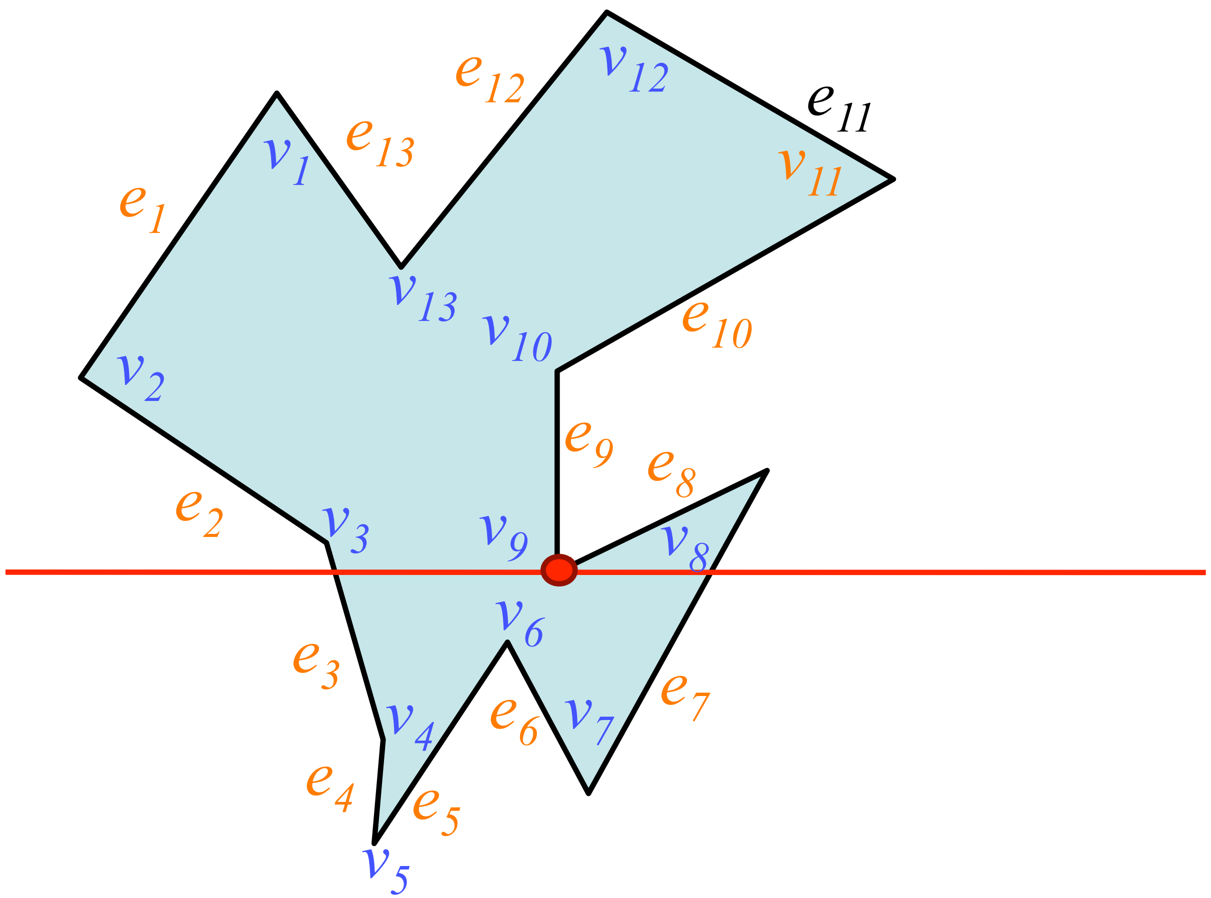
Example



T=

(e3, v3) (e8,v8)

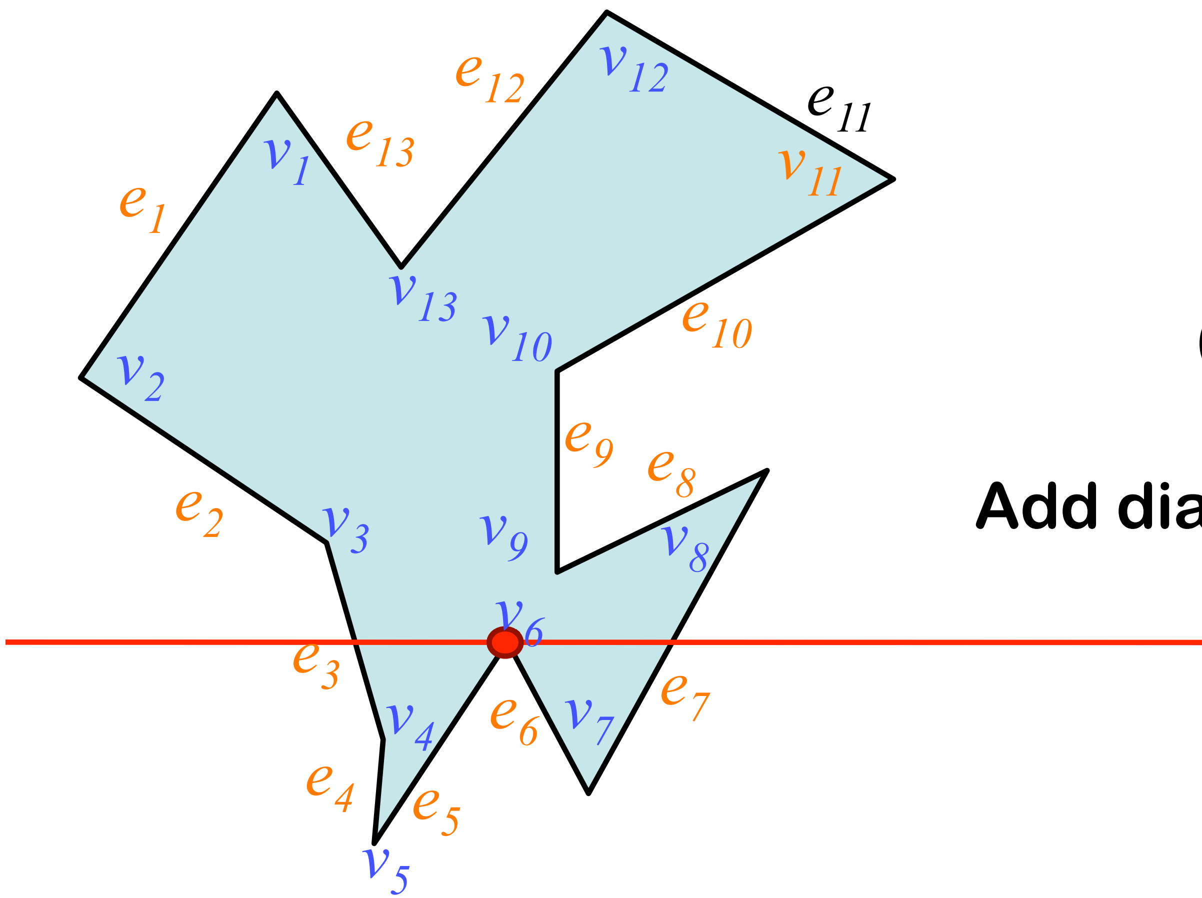
Example



T=

(e3, v9)

Example

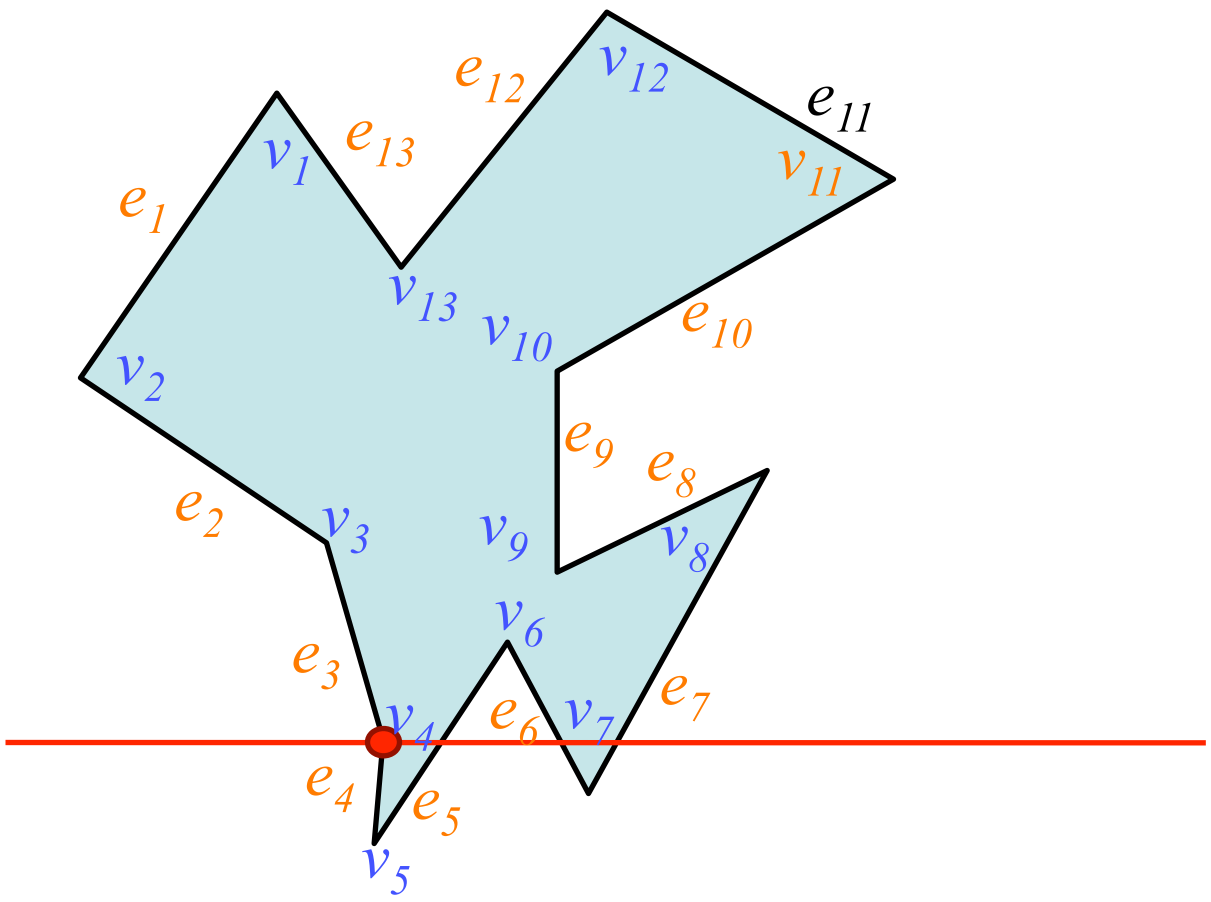


T=

(e3, v6) (e6,v6)

Add diagonal v6v9

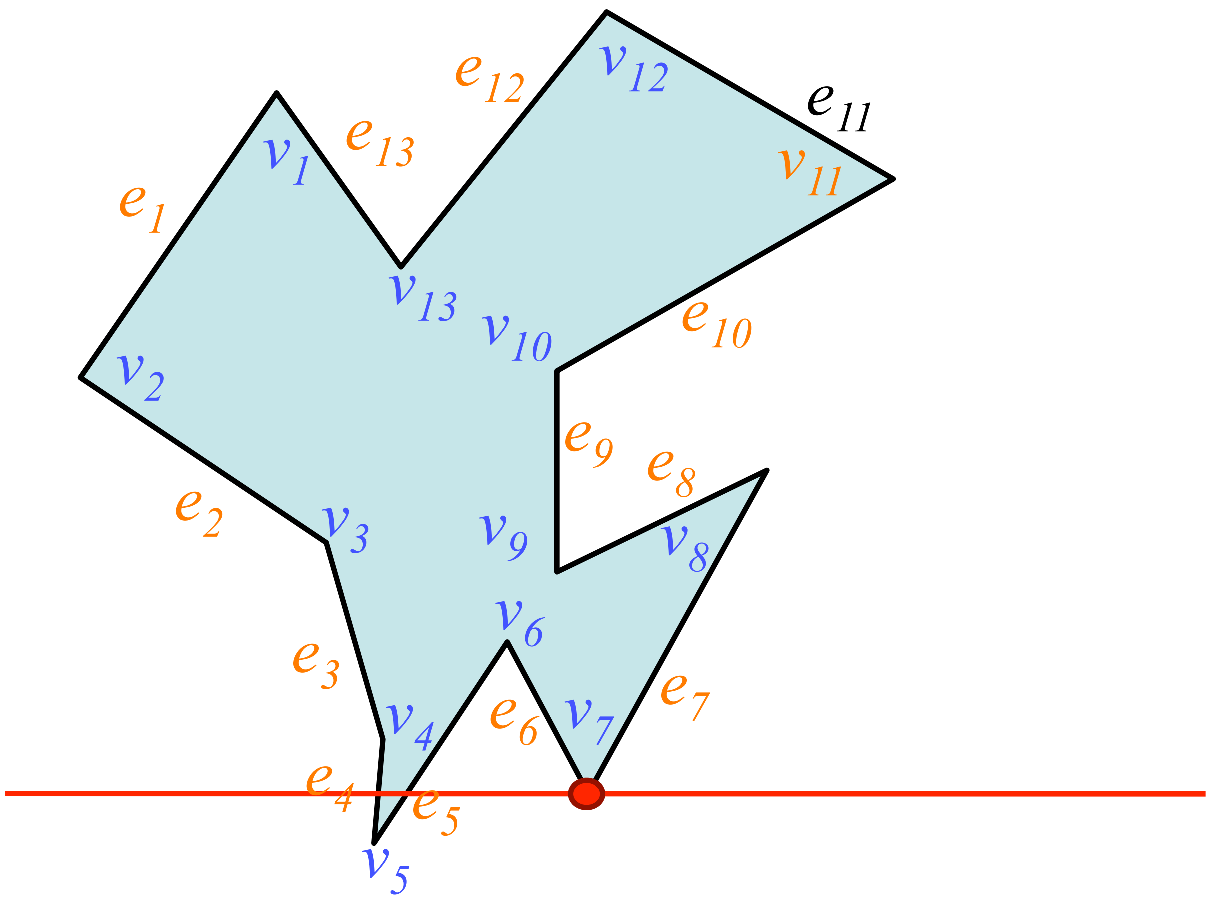
Example



e_{12}
v_{12}
e_{11}
e_{13}
v_1
v_{11}
e_1
v_{13}
v_{10}
e_{10}
v_2
e_9
e_8
e_2
v_3
v_9
v_8
v_6
e_3
e_7
v_4
e_6
v_7
e_4
e_5
v_5

T=

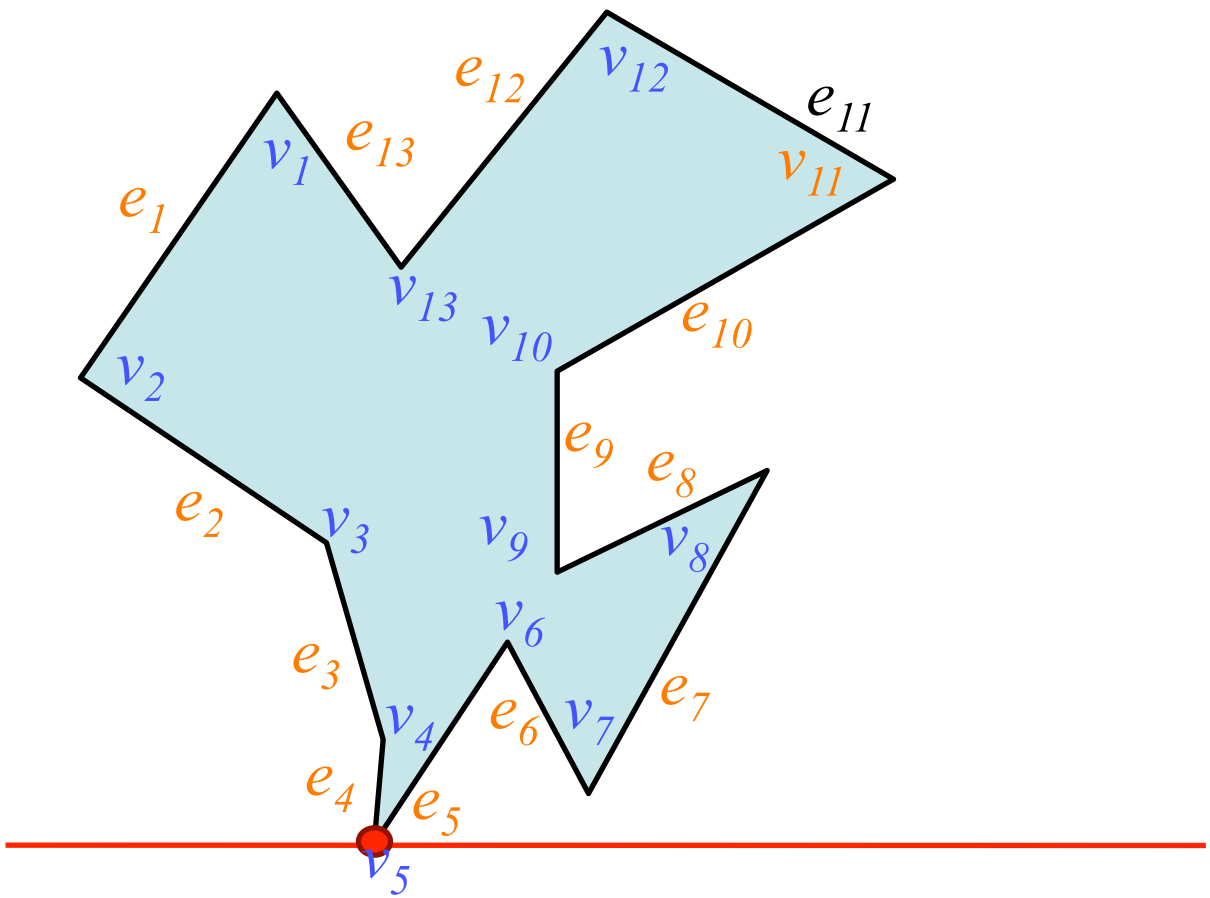
Example



e_{12}
v_{12}
e_{11}
e_{13}
v_1
v_{11}
e_1
v_{13}
v_{10}
e_{10}
v_2
e_9
e_8
e_2
v_3
v_9
v_8
v_6
e_3
e_7
e_6
v_7
v_4
e_4
e_5
v_5

T=

Example



e_{12}
v_{12}
e_{11}
v_{11}
e_{13}
v_1
e_1
v_{13}
v_{10}
e_{10}
v_2
e_9
e_8
e_2
v_3
v_9
v_8
v_6
e_3
e_7
e_6
v_7
v_4
e_4
e_5
v_5

T=

Polygon Triangulation

- Decompose a simply polygon into a monotone polygon: O($n$log$n$)
  - Plane sweep algorithm

- **Triangulation of a monotone polygon: O(n)**

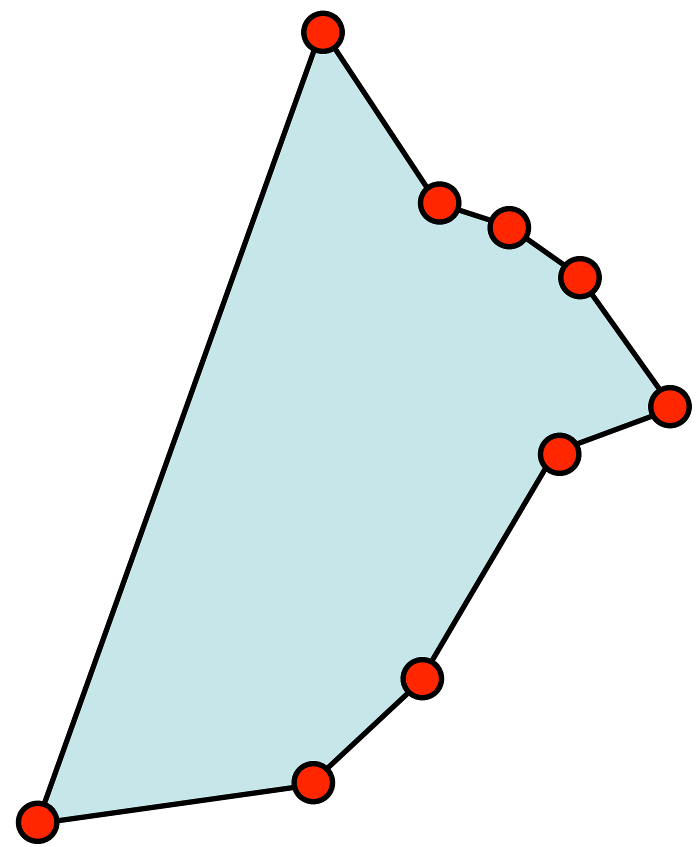
Total time to compute a triangulation: O($n$log$n$)

Triangulate a Monotone Polygon

- Walk from top to bottom on both chains (Sweep line, again)
- Greedy algorithm. Add as many diagonals as possible from each vertex

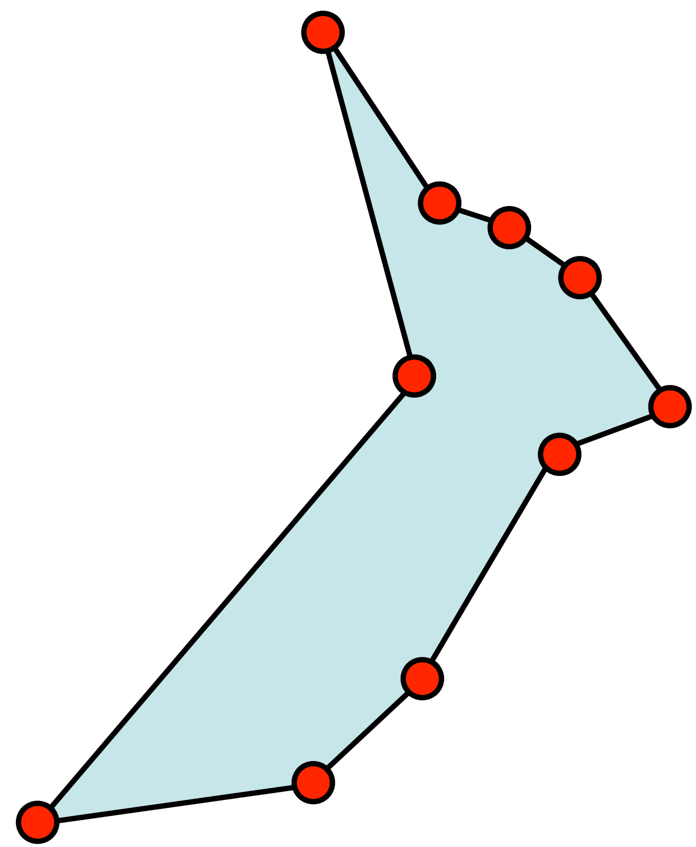
Triangulate a Monotone Polygon

- Assuming all vertices are one the same side
- We maintain a stack S
- S contains vertices
  - Above the sweep line
  - Not be triangulated
  - Forms an upside-down funnel



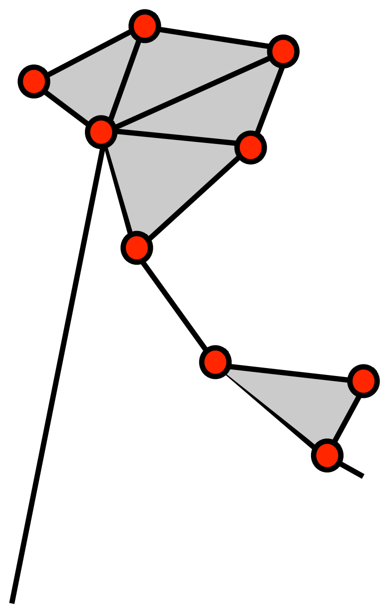
Triangulate a Monotone Polygon

- Now there is a vertex on the other side of the chain
- Maintain the same stack S
- When the sweep line stops at this new vertex, add diagonals from it to all the vertices in S



Triangulate a Monotone Polygon

- This funnel is an invariant of the algorithm
  - consisted of a singe edge & a chain of reflex vertices
  - only the highest vertex (at the bottom of *S*) is convex



Summary

When the sweep line is at a vertex V_j

On the single edge side

- must be the lower end point of the edge: add diagonals to all reflex edges, except last one.
- This vertex and first are pushed back to stack

On the chain of reflex vertices

- pop one; this one is already connected to V_j
- pop vertices from stack till not possible

Triangulate a Monotone Polygon

Input: A strictly y-monotone polygon P stored in a d.-c. e. list D

Output: A triangulation of P stored in doubly-connected edge list D

1. Merge the vertices on the left and right chains of P into one sequence, sorted on decreasing y-coordinate, with the leftmost comes first. Let $u_1 \ldots u_n$ denote sorted sequence
2. Push u_1 and u_2 onto the stack S
3. for j ← 3 to n ← 1
4. if u_j and vertex on top of S are on different chains
5. Add diagonals from u_j to all vertices in S
6. if u_j and vertex on top of S are on same chains
7. Add diagonals from u_j to vertices in S until you cannot do so
8. Add diagonals from u_n to all stack vertices except the

Triangulation Algorithm Analysis

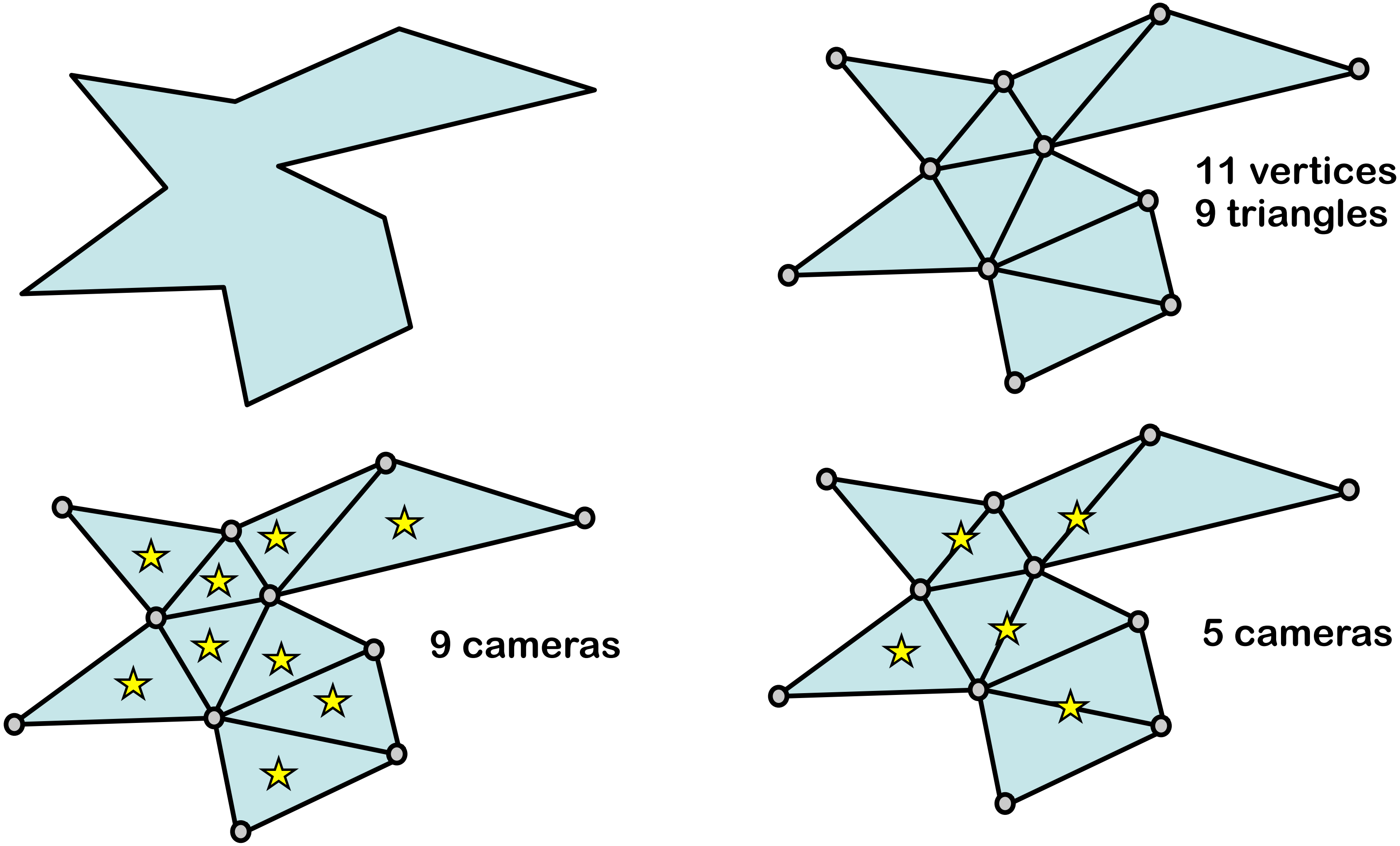
- A strictly y-monotone polygon with n vertices can be triangulated in linear time

- A simple polygon with n vertices can be triangulated in $O(n \log n)$ time with an algorithm that uses $O(n)$ storage

Art Gallery Problem

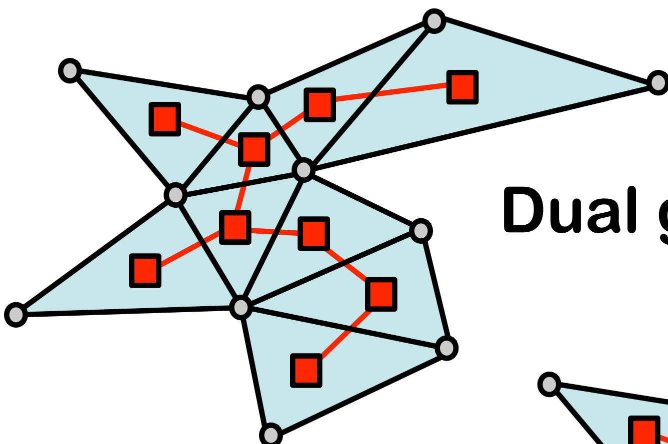
- We can guard a gallery by $n-2$ cameras
- We can do better by placing cameras at the diagonals, then we only need $n/2$
- Even better by placing cameras at vertices of the polygons => $\lfloor n/3 \rfloor$ needed by using 3-coloring scheme of a triangulated polygon (ex) comb-shape like polygon
  - 3-coloring of a polygon always exists

Art Gallery Problem

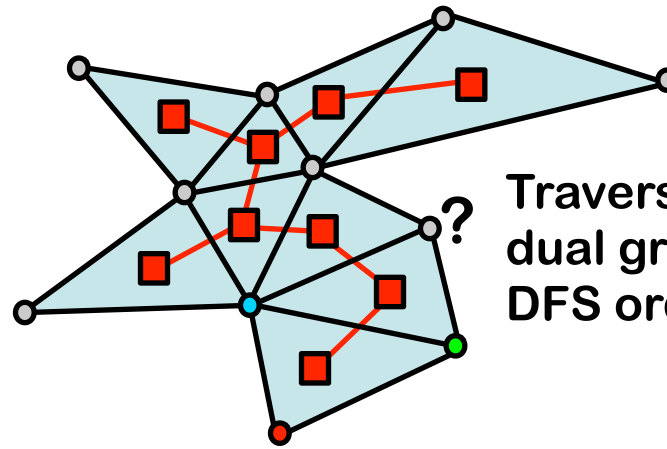


Art Gallery Problem

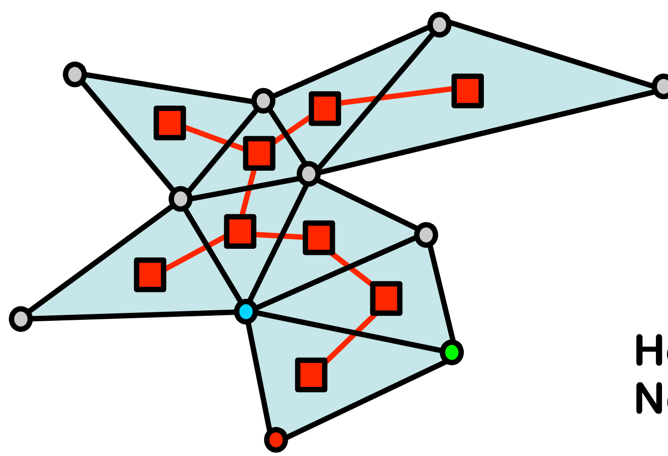
3-coloring



Dual graph



Traverse the dual graph in DFS order

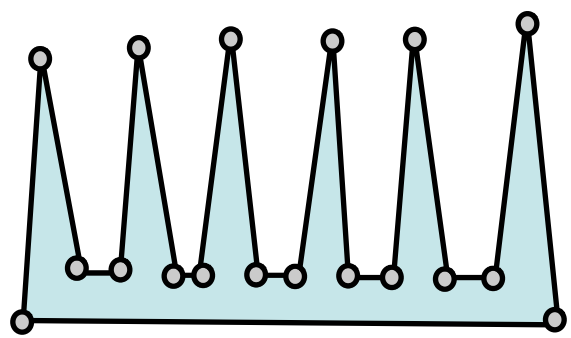


☆ ? cameras

How many cameras are really Needed?

Art Gallery Theorem

- **For a simple polygon with n vertices, $\lfloor n/3 \rfloor$ cameras are occasionally necessary and always sufficient to have every point in the polygon visible from at least one of the cameras**



Chvátal's Comb

Conclusion

- Triangulation in $O(n \log n)$ time
  - n is the number of vertices
  - Decompose a polygon into monotone subpolygons: $O(n \log n)$ time (plane-sweep algorithm)
  - Triangulate each subpolygons: $O(n)$ time
- Art gallery problem
  - Represent the floor plan as a polygon
  - Triangulate the polygon
  - 3 coloring the vertices of the "graph of the triangulation"
  - Place cameras at the color with fewest vertices
  - Art gallery theorem: $\lfloor n/3 \rfloor$ cameras is always sufficient but sometime necessary

Practice!

- **Exercises 3.6 & 3.13.**