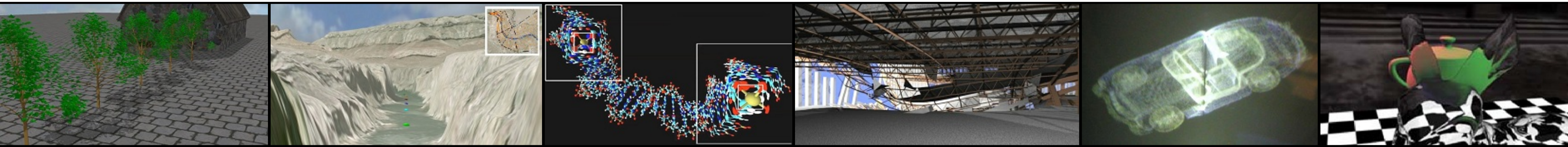
# COT 4521: Introduction to Computational Geometry

## Polygon Partitioning

Paul Rosen
Assistant Professor
University of South Florida

Some slide from Valentina Korzhova

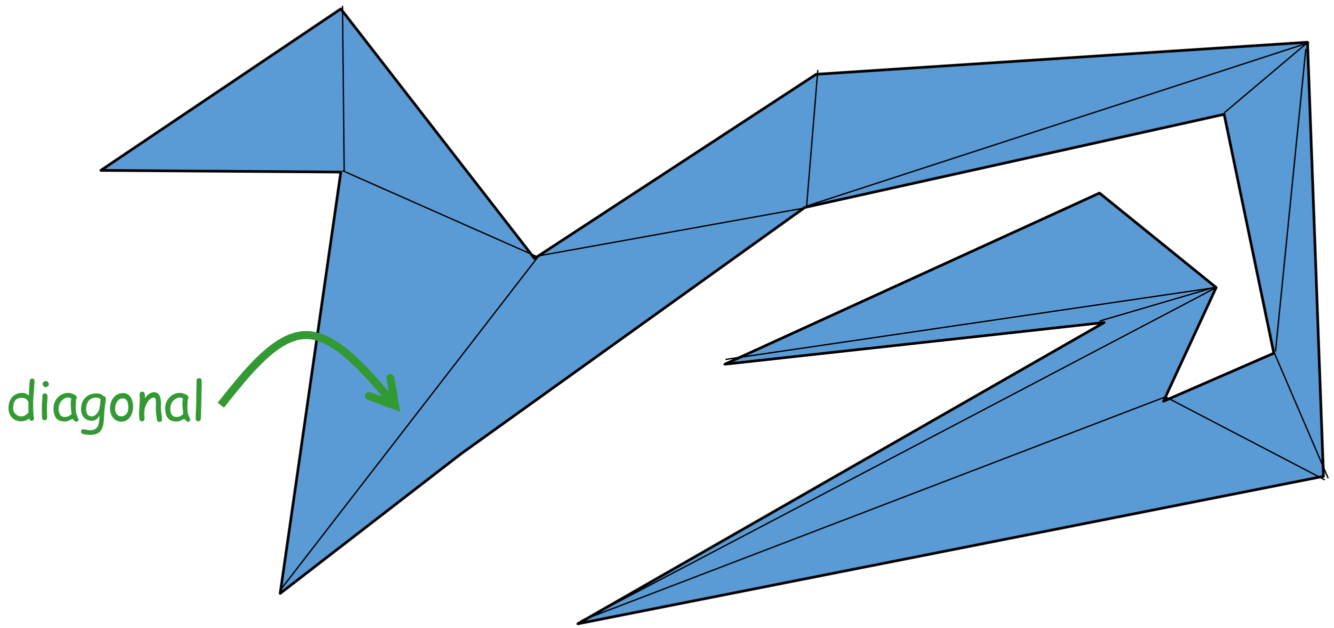


# Objectives

- Triangulation algorithms
  - Diagonal-based triangulation algorithm
  - Ear-based triangulation algorithm
- Complexity of the algorithms
- Definition of monotone polygon
  - Triangulation of a monotone polygon
- Partitioning into monotone polygons
- Triangulation of a polygon in n log n time

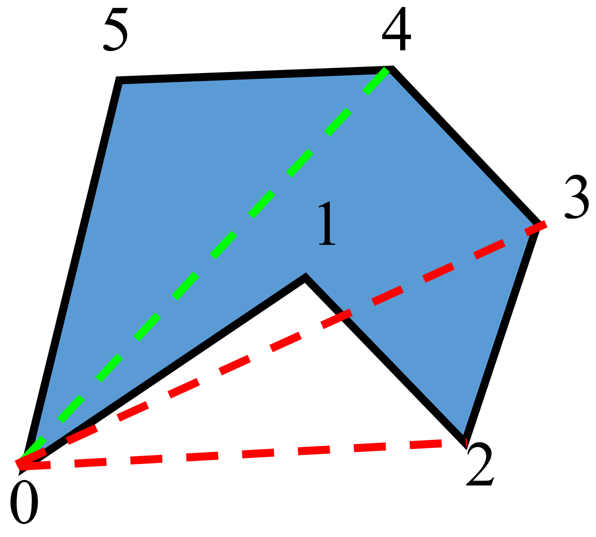
# Triangulation of polygons

- Decompose the polygon into shapes that are easier to handle: triangles
- A triangulation of a polygon P is a decomposition of P into triangles whose vertices are vertices of P. In other words, a triangulation is a maximal set of non-crossing diagonals.



# DIAGONAL-BASED TRIANGULATION

- DIAGONAL TEST
- THE SEGMENT $s = v_i v_j$ IS A DIAGONAL OF $P$ IFF
  - for all edges e of P that are not incident to either $v_i$ and $v_j$, $s$ and $e$ do not intersect.
  - $s$ is internal to $P$ in the neighborhood of $v_i$ and $v_j$.




- ALGORITHM: DIAGONAL TRIANGULATION

  REPEAT N-3 TIMES

  FOR EACH CANDIDATE DIAGONAL

  TEST EACH OF NEIGHBORHOODS

  OUTPUT PROPER DIAGONAL

# Brute Force Triangulation

- **Theorem**: Every polygon P of n vertices can be partitioned into triangle by the addition of (zero or more) diagonals.
- Complexity of diagonal-based algorithm:
  - $O(n^2)$ - # of diagonal candidates
  - $O(n)$ testing each of neighborhoods
  - Repeating this $O(n^3)$ computation for each of the $n-3$ diagonals yields $O(n^4)$
  - Can be made $O(n^3)$

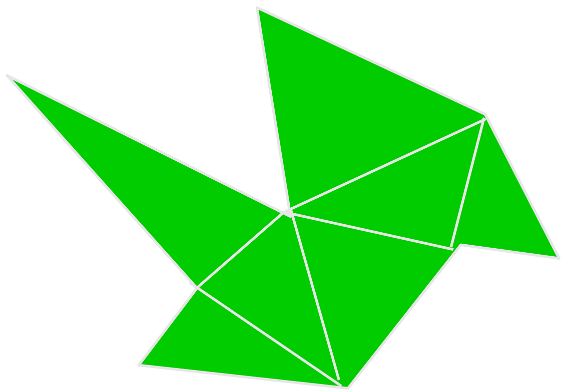
# Ear Based Idea….

- Locate an ear
- Output diagonal
- Clip the ear
- Repeat until a triangle is left

# Ear Based Idea….

- Definition of Ear: Three consecutive vertices, A, B, C form an ear if AC is a diagonal

- Meisters' Two Ears Theorem: Every polygon $(N \geq 4)$ has at least two non-overlapping ears.



# Ear Based Idea….

- Proof: Consider a triangulation of an n-polygon, with $n > 3$. The triangulation consists of $n - 2$ triangles. Since the polygon has $n$ edges but there are only $n - 2$ triangles, by the Pigeonhole Principle, there are at least two triangles with two polygon's edges. These are the ears.

- Another proof: It is known that a simple polygon can always be triangulated. Leaves in the dual-tree of the triangulated polygon correspond to ears and every tree of two or more nodes must have at least two leaves.

# TRIANGULATION: IMPLEMENTATION

**Algorithm**: **TRIANGULATION**

Initialize the ear tip status of each vertex.

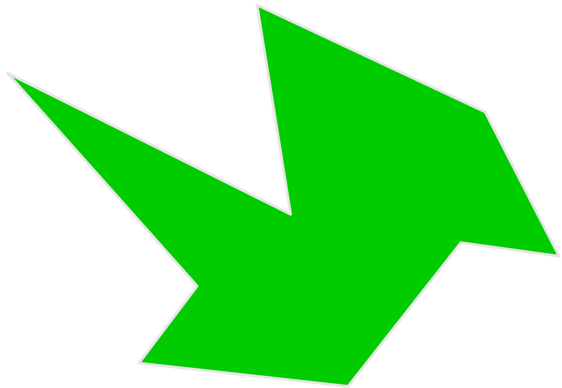
while n > 3 do

Locate an ear tip $v_2$.

Output diagonal $v_1 v_3$.

Delete $v_2$.

Update the ear tip status of $v_1$ and $v_3$.



# TRIANGULATION: IMPLEMENTATION

**Algorithm**: **TRIANGULATION**

Initialize the ear tip status of each vertex.

while n > 3 do

Locate an ear tip $v_2$.

Output diagonal $v_1 v_3$.

Delete $v_2$.

O(n) → Update the ear tip status of $v_1$ and $v_3$.

**O(n²)**

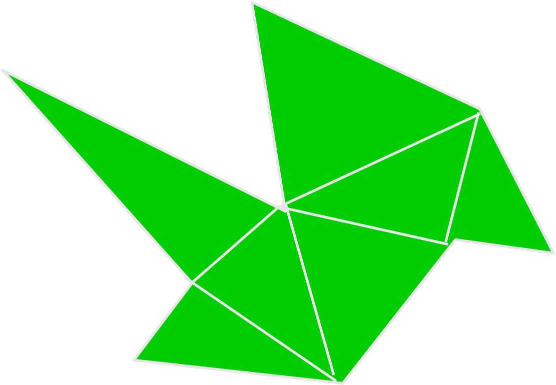
**+**

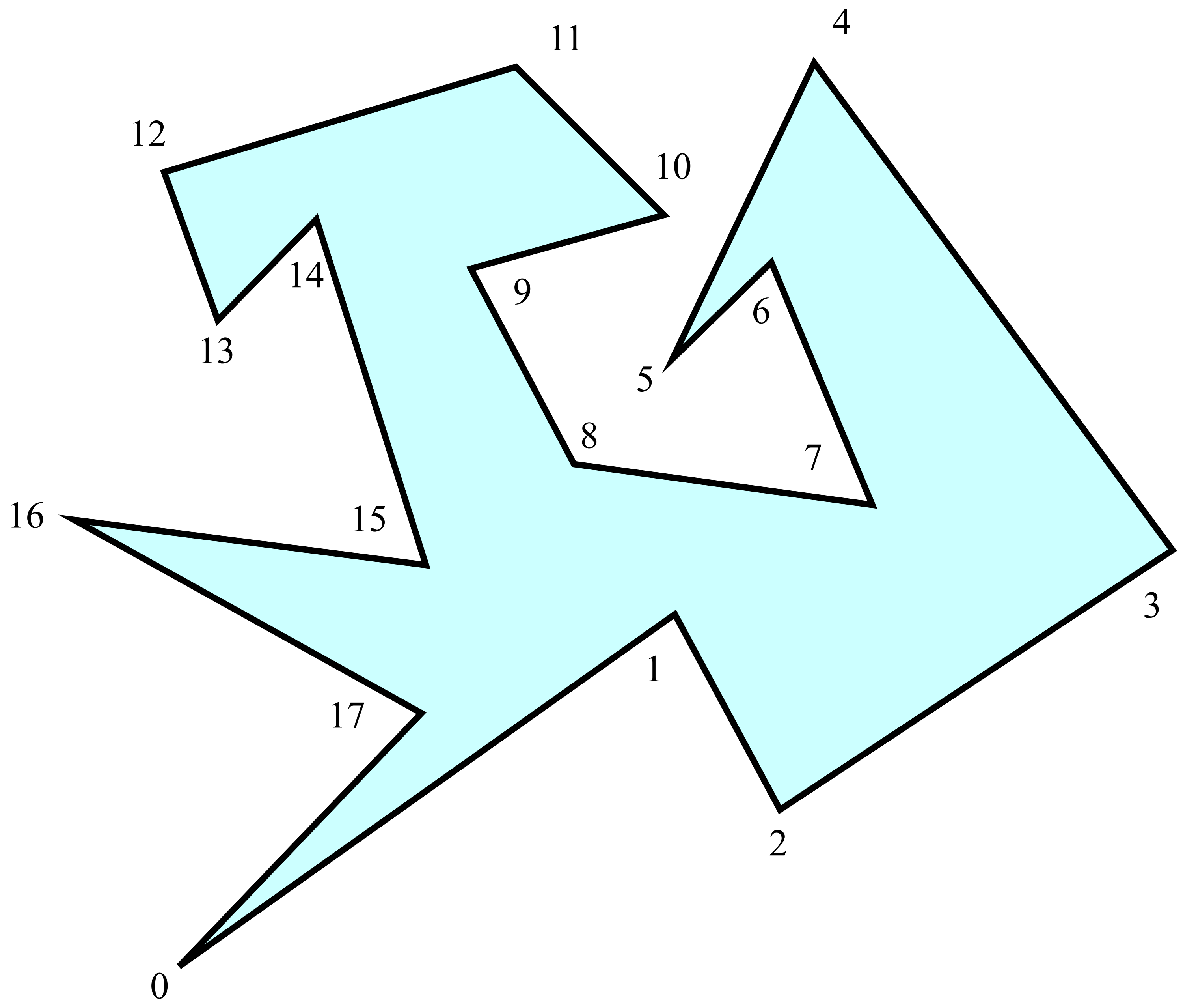
**O(n) iterations**

**O(n) iterations**

**Total: O(n³)**

**Can be made: O(n²)**

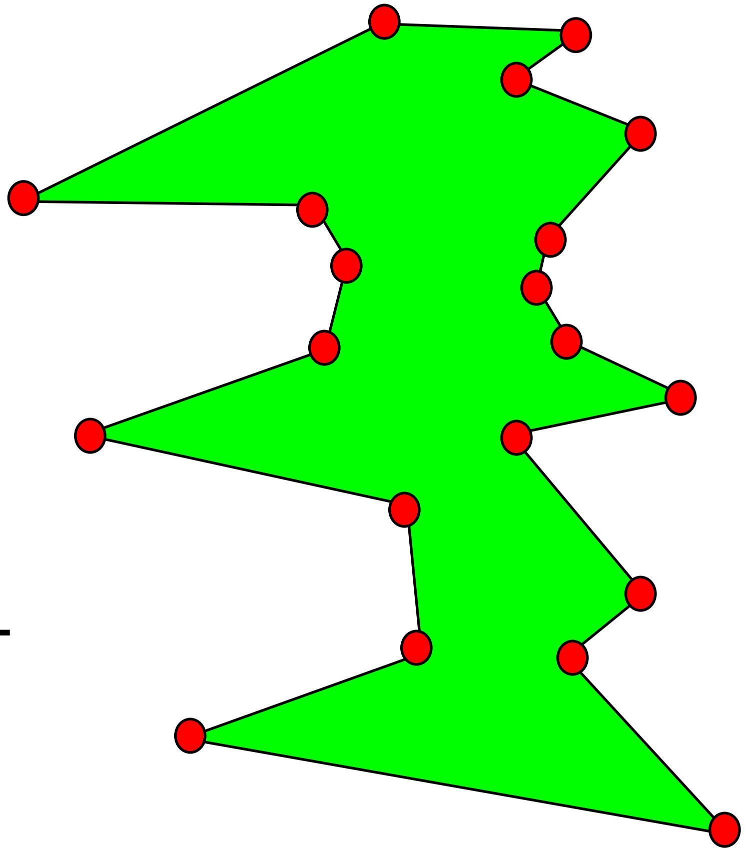




0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17

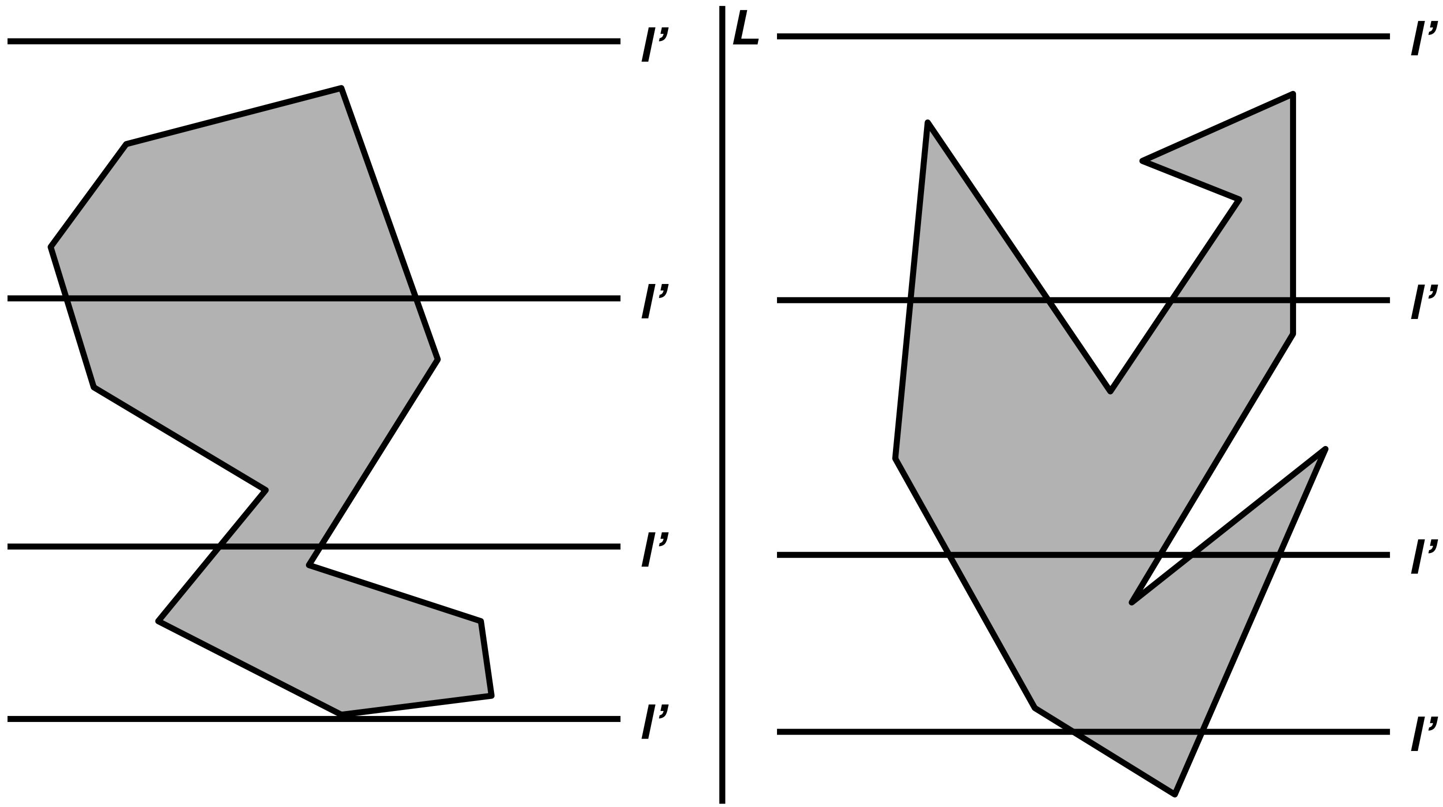
# Monotone Partitioning

- A **chain** is **monotone** with respect to a line L if every line orthogonal to L intersects the chain in at most 1 point
- **Polygon** is **monotone** with respect to a line L if boundary of P can be split into 2 polygonal chains A and B such that each chain is monotone with respect to L
- Monotonicity implies sorted order with respect to L
- Monotone polygon can be (greedily) triangulated in O(n) time!



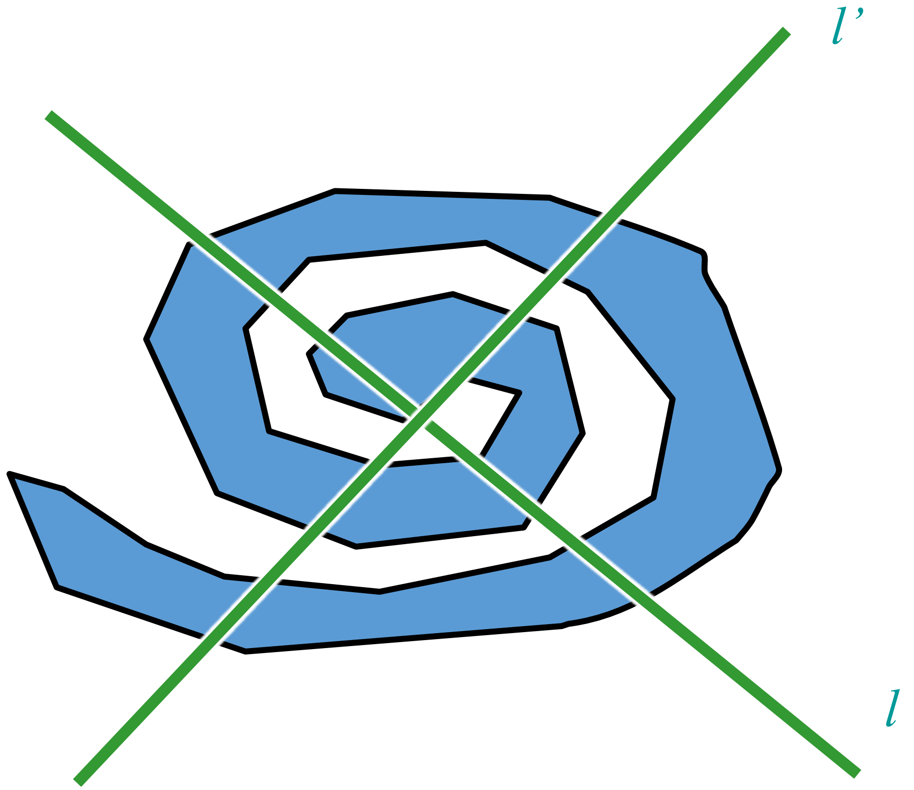
# Monotone Polygon

Monotone
w.r.t. line L




not monotone
w.r.t. L

# Monotone Polygon



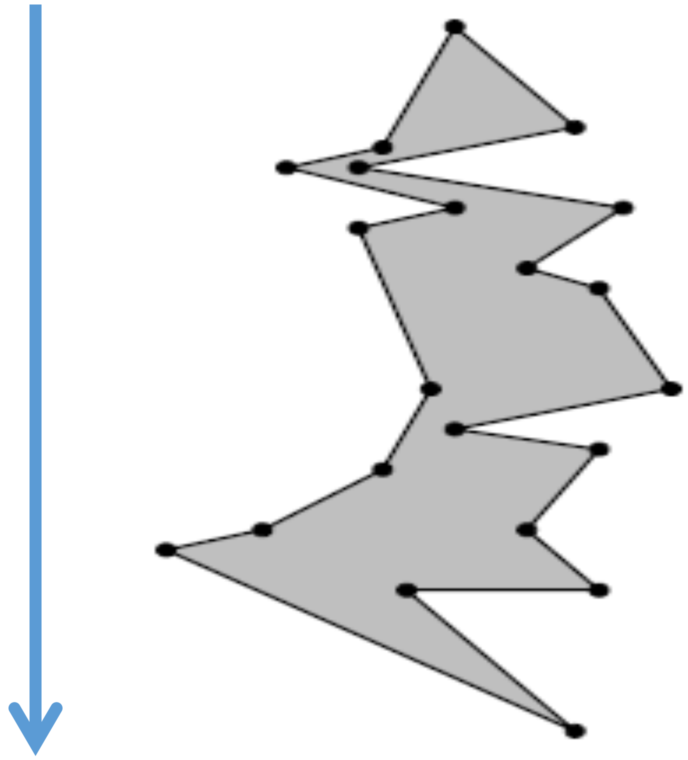

not monotone
w.r.t. any line

# POLYGON TRIANGULATION

- **ALGORITHM**: POLYGON TRIANGULATION: MONOTONE POLYGON WITH RESPECT TO Y-LINE
  - Partition into monotone polygons
  - Triangulate each monotone polygon

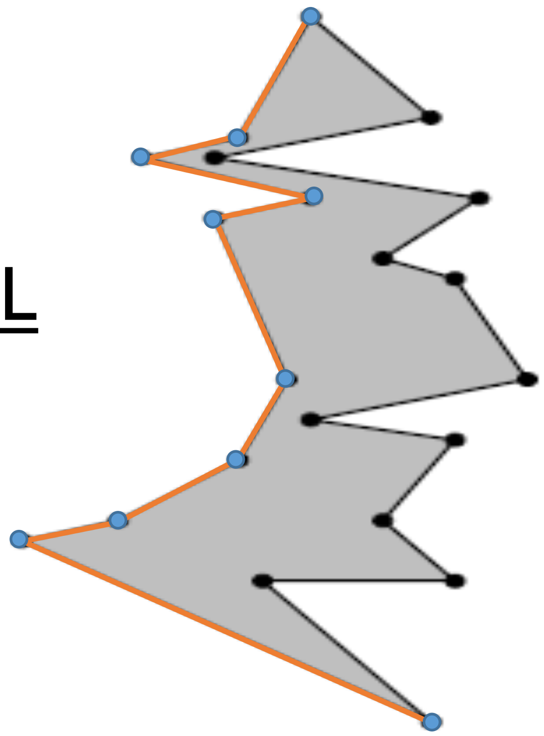
# MONOTONE POLYGON

- A Y-MONOTONE POLYGON HAS A TOP VERTEX, A BOTTOM VERTEX, AND TWO Y-MONOTONE CHAINS BETWEEN TOP AND BOTTOM AS ITS BOUNDARY



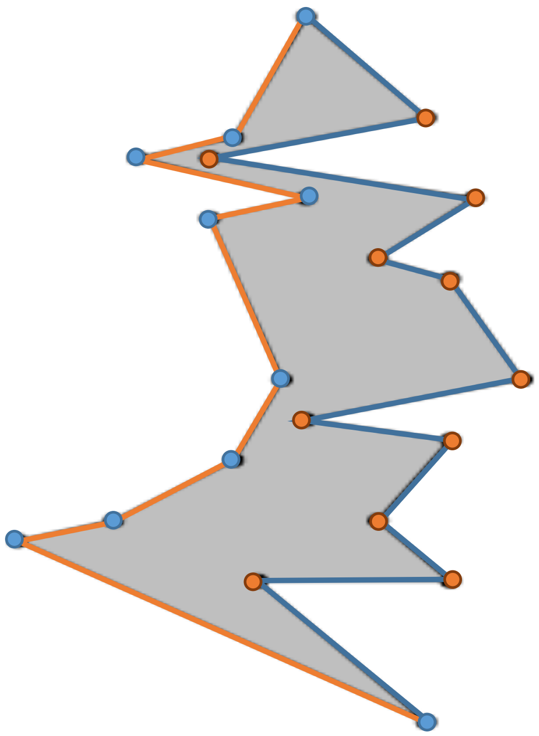
# Monotone Polygon

- A polygonal chain C is strictly monotone w.r.t. $L'$ if every L orthogonal to $L'$ meets C in at most one point.
  - Simply monotone if L∩C has at most one connected line segment.



# <u>Monotone Polygon</u>

- A <u>polygon P is said to be monotone</u> w.r.t. a line L if $\partial P$ can be split into two monotone chains w.r.t. L



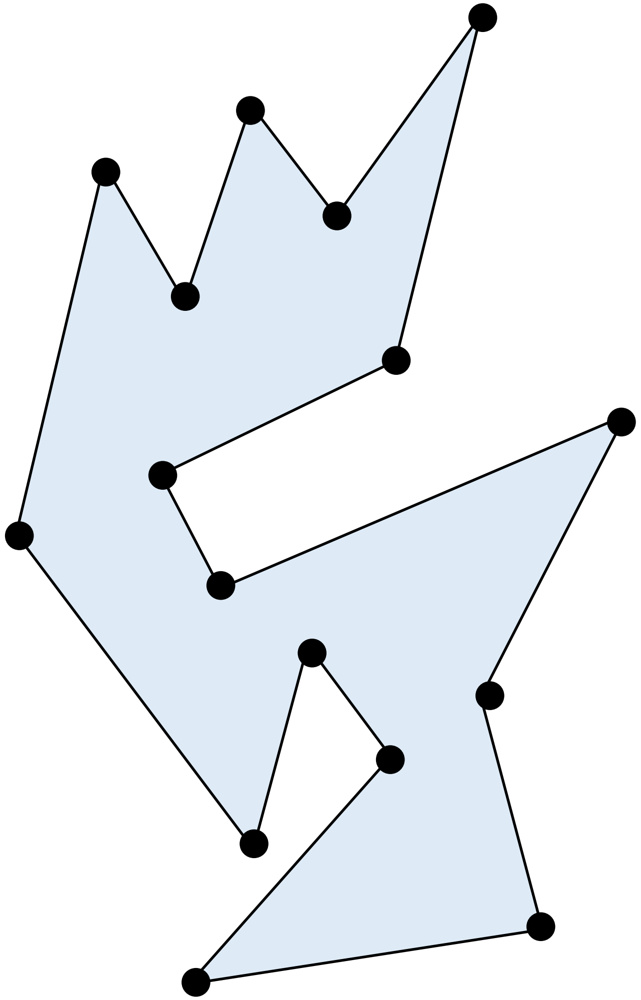
# Monotone Polygon

- The following property is characteristic for y-monotone polygons:
  - If we walk from a topmost to a bottommost vertex along the left (or the right) boundary chain, then we always move downwards or horizontally, never upwards.

# Triangulation of Monotone Polygon

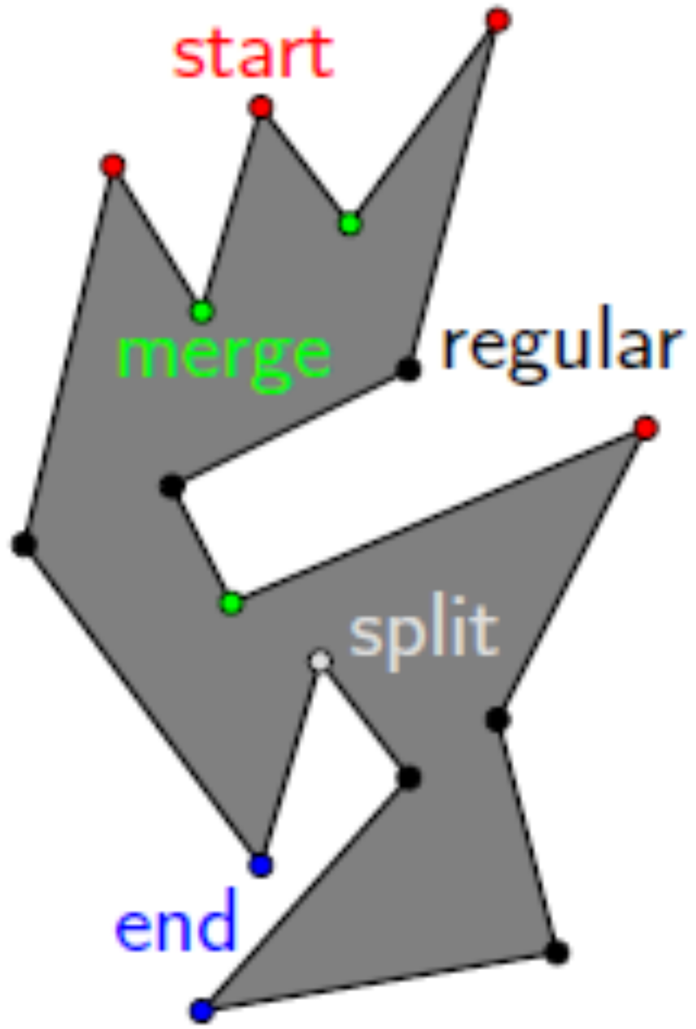
- The algorithm to triangulate a monotone polygon depends on its monotonicity.
- Developed in 1978 by Garey, Johnson, Preparata, and Tarjan
- described in both
  - Preparata pp. 239-241 (1985)
  - Laszlo pp. 128-135 (1996)
  - The former uses y-monotone polygons, the latter uses x-monotone.

# What kind of vertices does a Non-Y-Monotone Polygon have with respect to sweep of y?



# Properties of monotone polygon

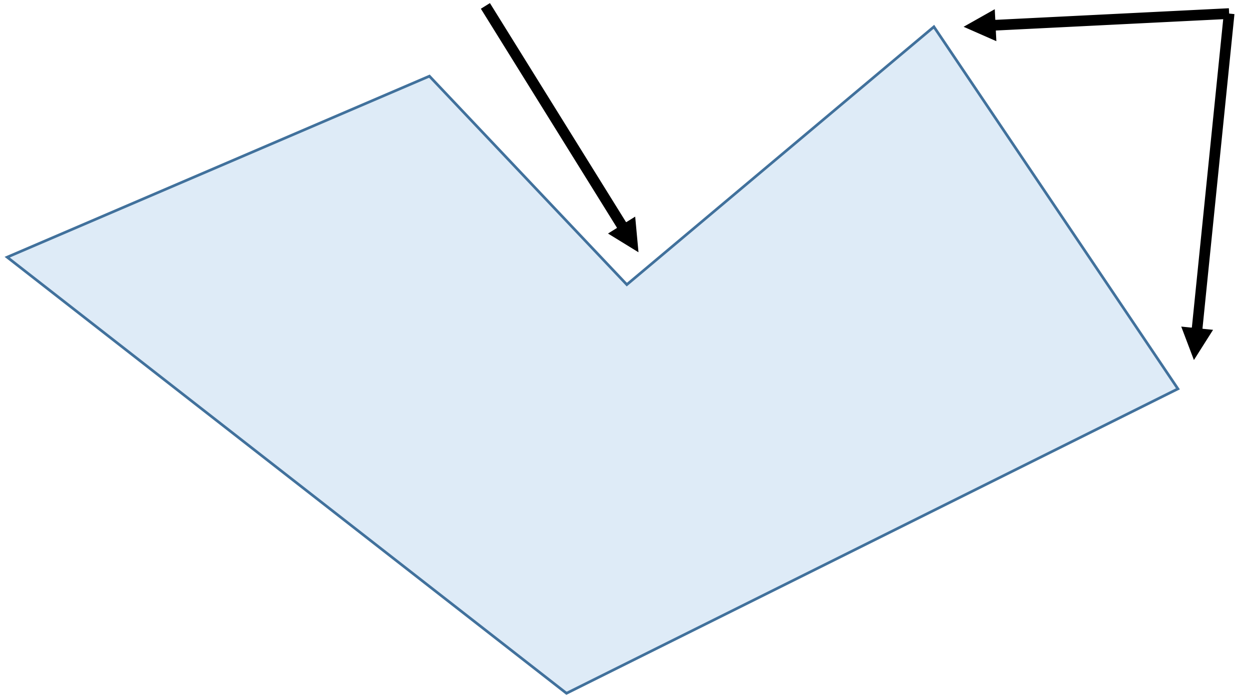
- If a polygon P has no interior cusps, then it is monotone
- What types of vertices does a simple polygon have?
  - start
  - end
  - split
  - merge
  - regular



# Reminder

Reflex Vertex
(internal angle $> 180°$)

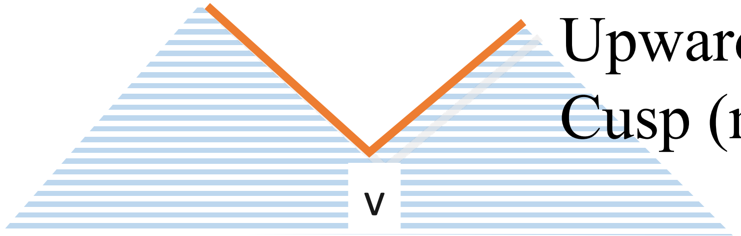
Convex Vertex
(internal angle $< 180°$)



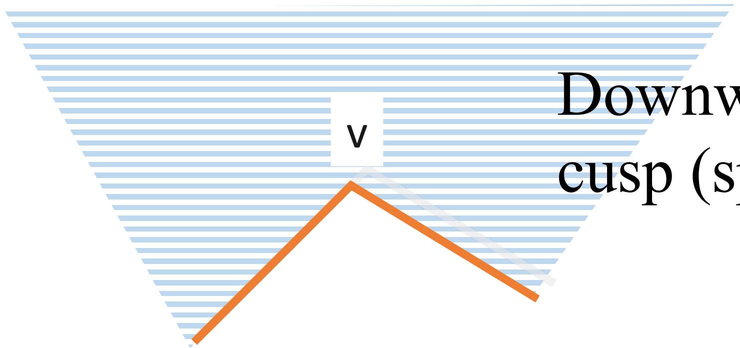
# PROPERTIES OF MONOTONE POLYGON

- REFLEX VERTEX WHOSE ADJACENT VERTICES ARE EITHER BOTH AT OR ABOVE V, OR BOTH AT OR BELOW IT.
- IF A POLYGON HAS NO INTERIOR CUSPS THEN IT IS MONOTONE WITH RESPECT TO THE VERTICAL LINE

Interior cusps



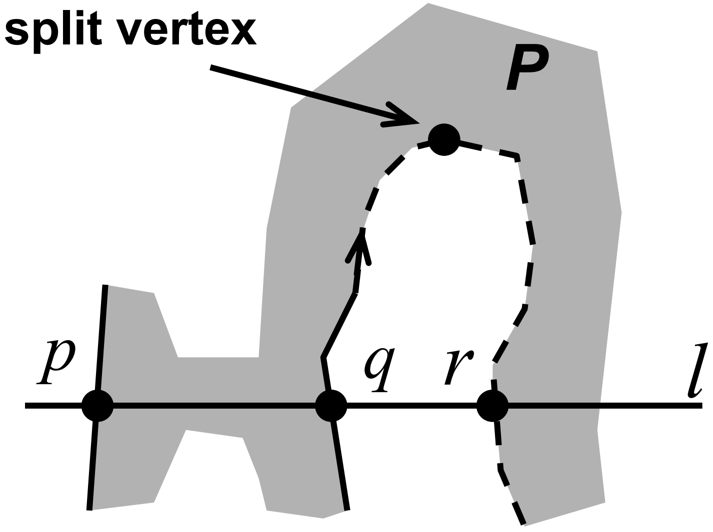

Upward
Cusp (merge)




Downward
cusp (split)

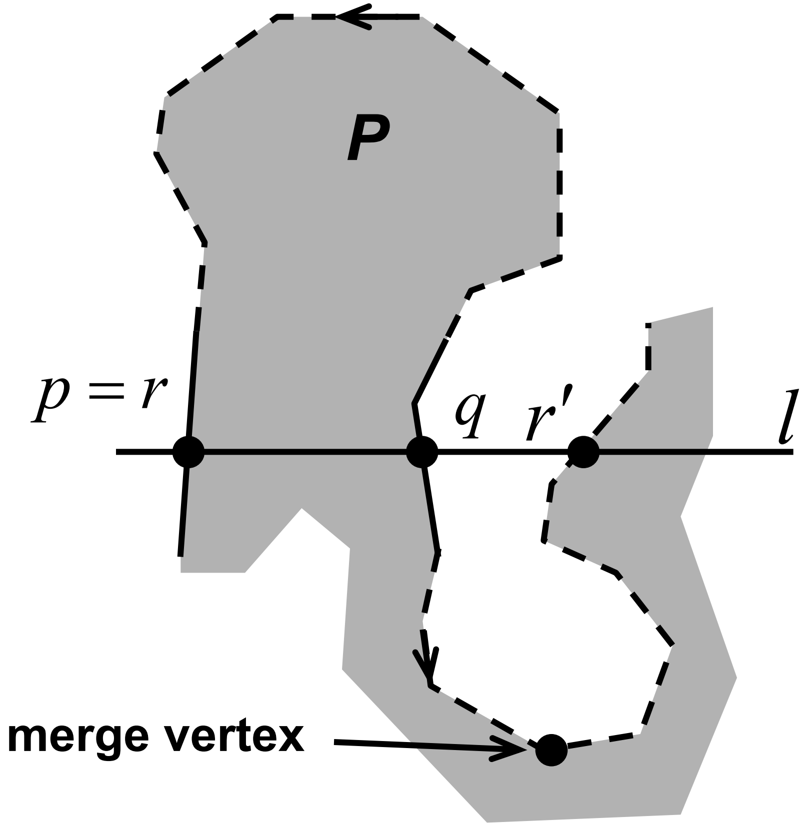
# Properties of monotone polygon

- **Lemma**: A polygon is y-monotone if it has neither split vertices nor merge vertices
- **Proof**: If P is not monotone, there must exist a line l intersecting P in more than a single segment.
  Let [p,q] be its leftmost sub segment.
- Follow the boundary of P starting at q, where P is on the left. At some point r we must cross l.
- If r ≠ p then the highest vertex must be a split one.



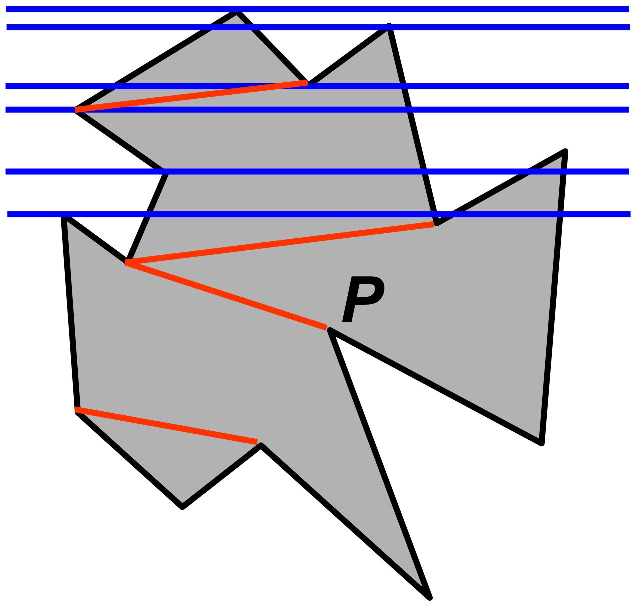
# Properties of monotone polygon

- If r = p we follow the boundary from q in opposite direction.
- At some point $r'$ we must cross l. $r' \neq p$ as otherwise it contradicts that P is not monotone.
- This implies that the lowest encountered vertex must be a merge one.



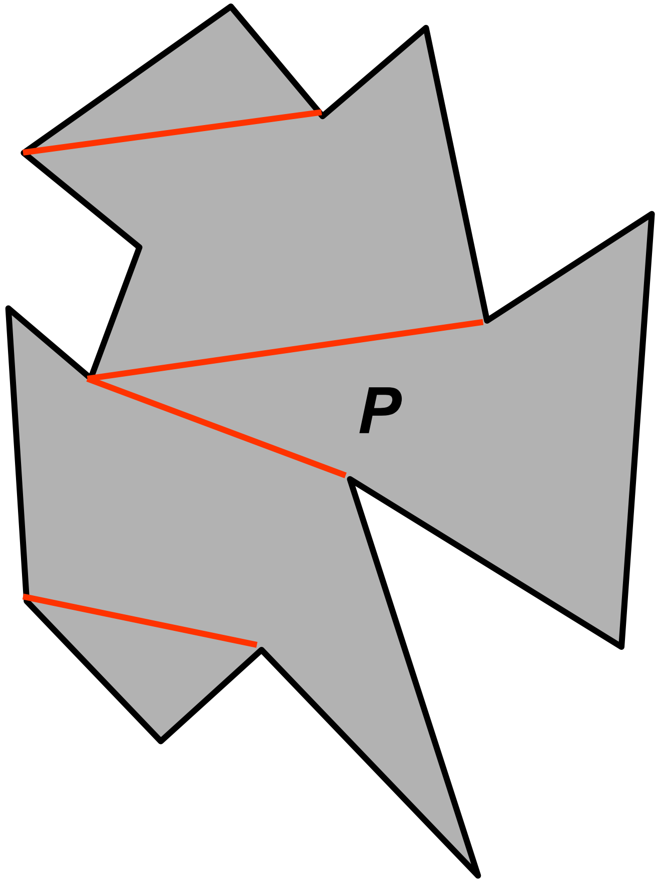
# Polygon Triangulation

- Algorithm: POLYGON TRIANGULATION: MONOTONE PARTITION
  - **Partition into monotone polygons**
  - Triangulate each monotone polygon



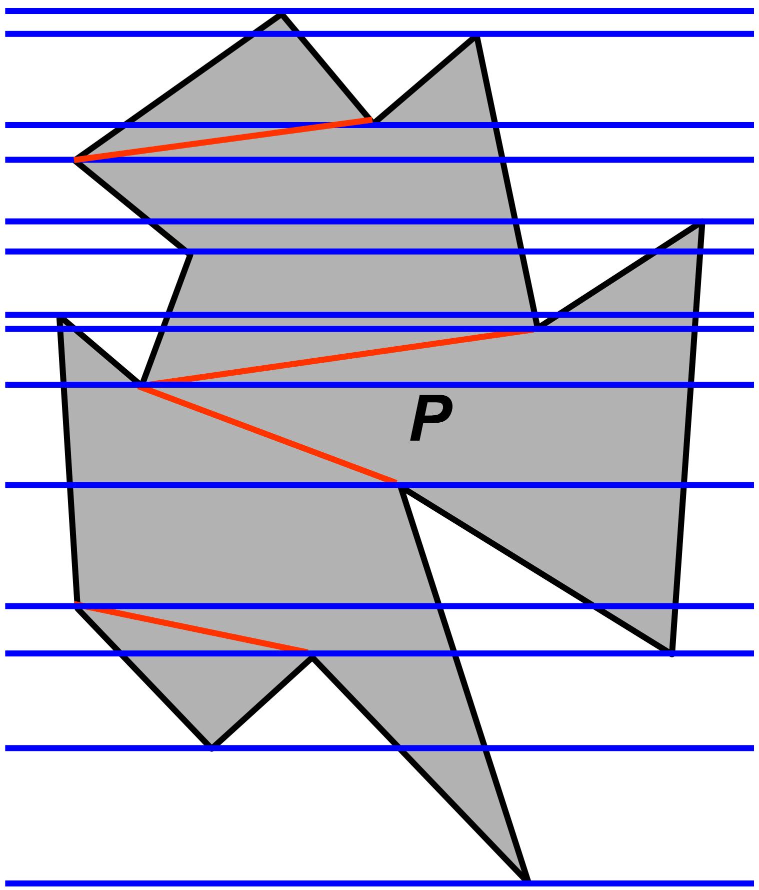
# Getting Rid of Split and Merge Vertices

- Sort P's vertices from top to bottom
  - takes $O(n \log n)$ time.
- Scan from top to bottom to encounter vertices.
- Diagonals are introduced at split and merge vertices.



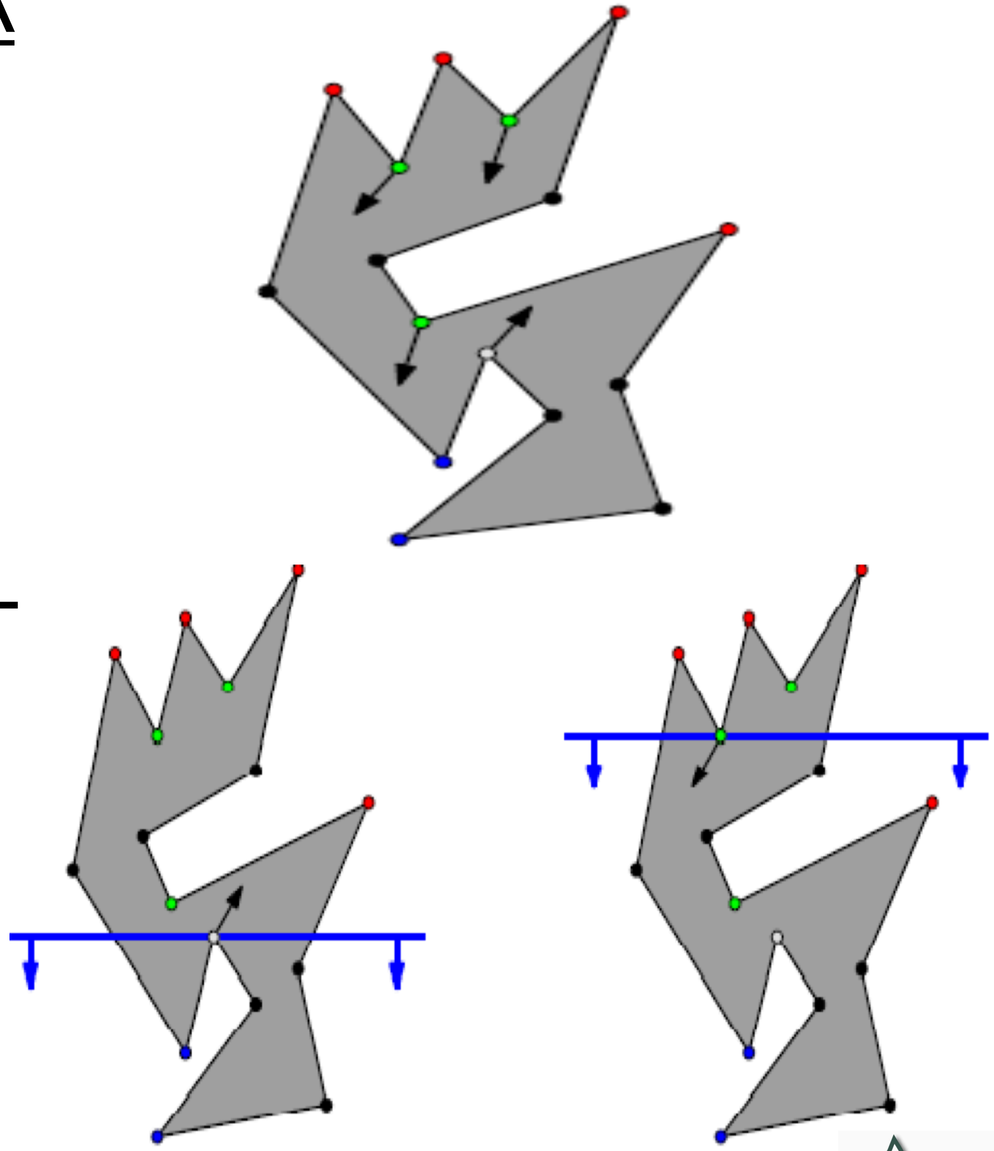
# TRAPEZOIDALIZATION

- "DRAW" HORIZONTAL LINE THROUGH EACH VERTEX
  - Consider only the connected segment inside the polygon containing the vertex
  - Two supporting vertices – top and bottom
- IF AN "INTERIOR" SUPPORTING VERTEX IS AN INTERIOR CUSP, BREAK IT
  - Connect downward for a upward cusp
  - Connect upward for an downward cusp
  - These connections partitions the polygon into monotone parts.



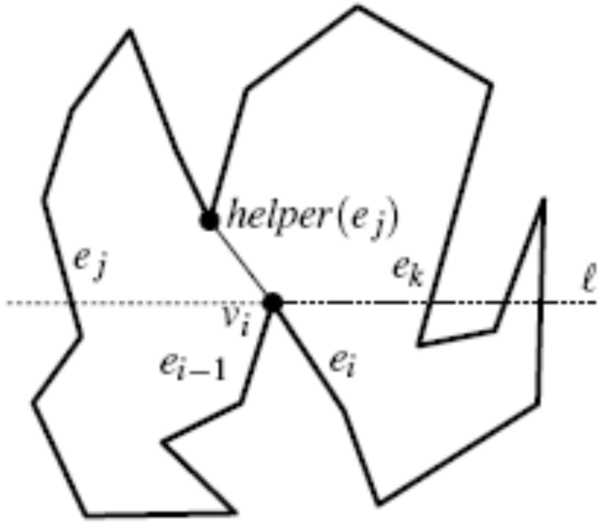
# SWEEP IDEA

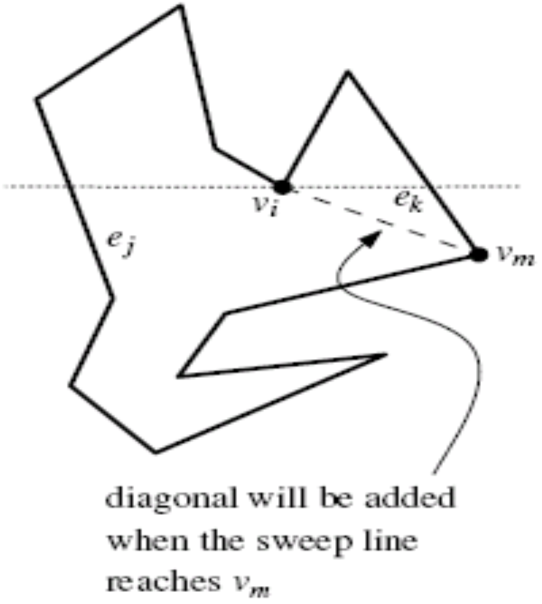
- FIND DIAGONALS FROM EACH MERGE VERTEX DOWN, AND FROM EACH SPLIT VERTEX UP
- A SIMPLE POLYGON WITH NO SPLIT OR MERGE VERTICES CAN HAVE AT MOST ONE START AND ONE END VERTEX, SO IT IS Y-MONOTONE



# SWEEP IDEA

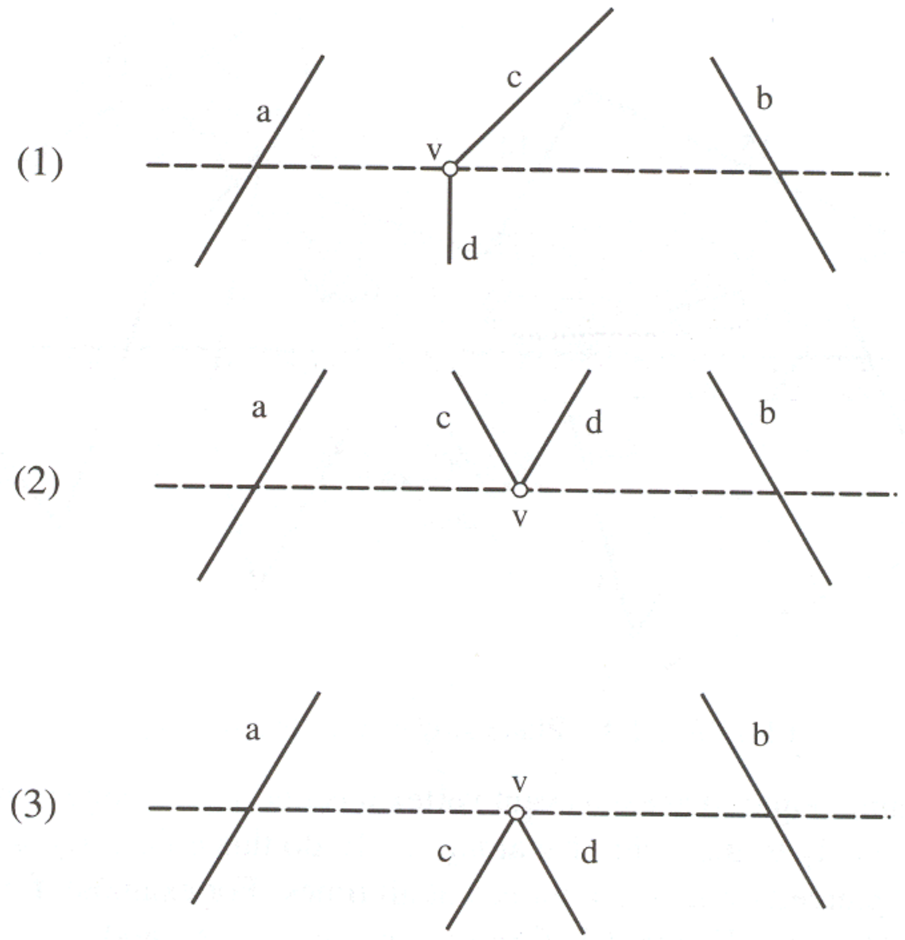
- FOR VERTEX OF INTEREST $v$, FIND THE CLOSEST VERTEX (IN THE Y DIRECTION) THAT IS BETWEEN THE EDGES TO THE LEFT AND RIGHT OF $v$



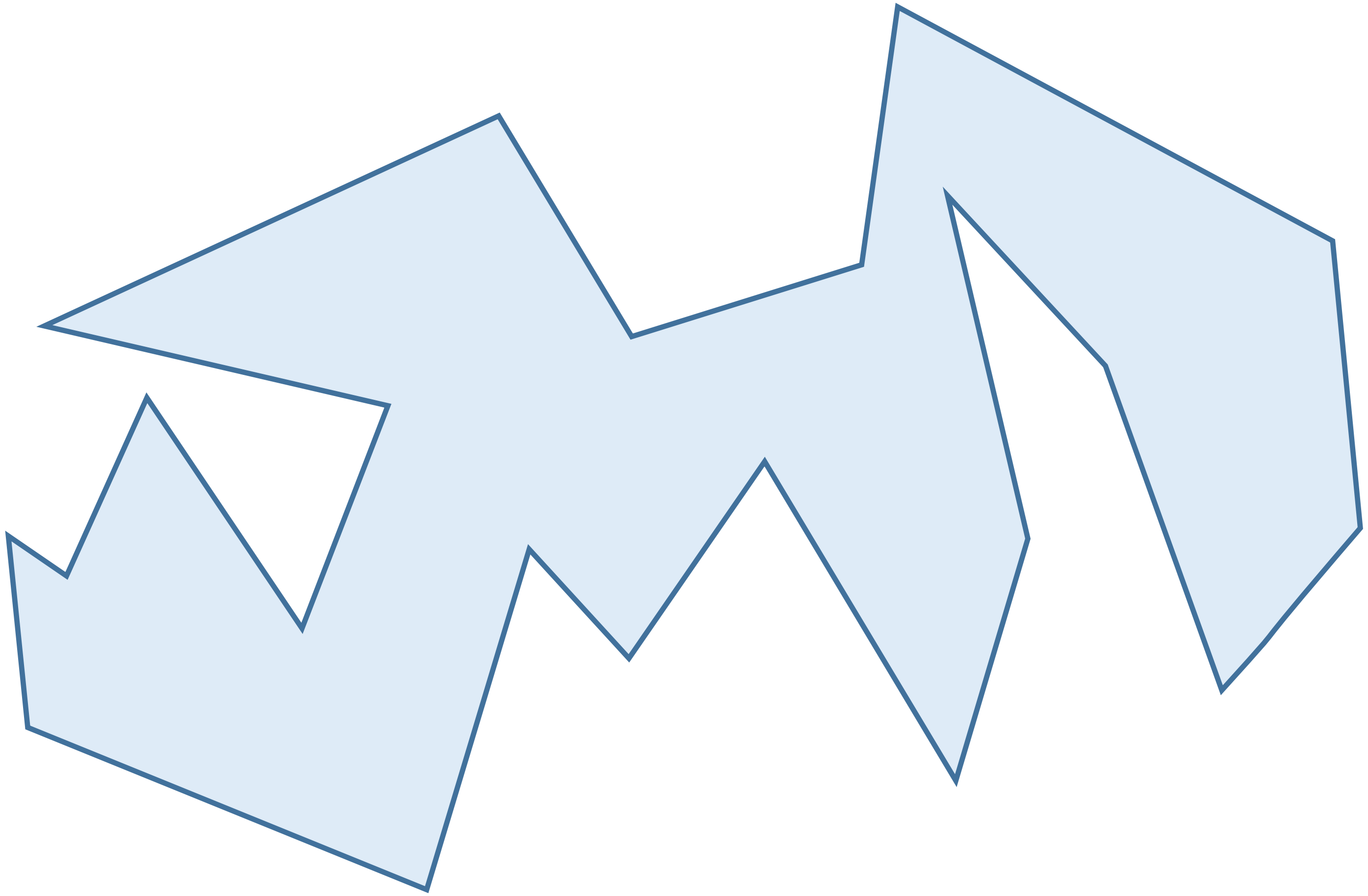



# Forming trapezoids

- Maintain a list of sides intersected by the sweeping line, sorted by the x-coord of intersection
- At each event, update the list
  - Can be done in $O(\log N)$ if the list is maintained as a balanced binary tree
- Overall: $O(N \log N)$

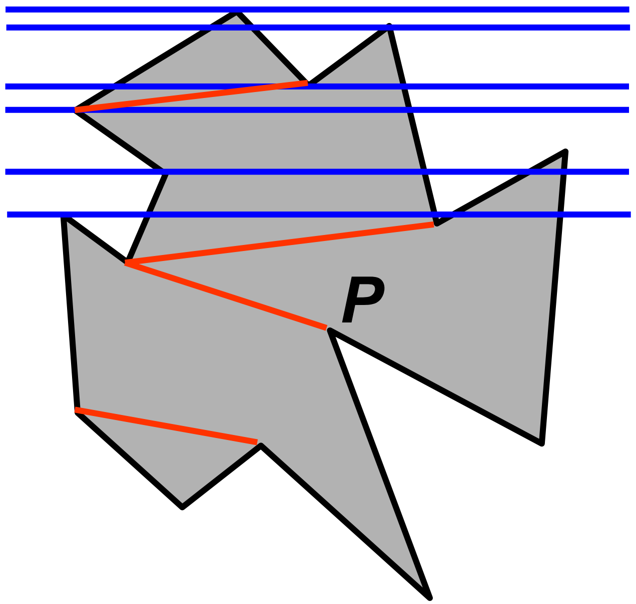


# Line Sweep Example



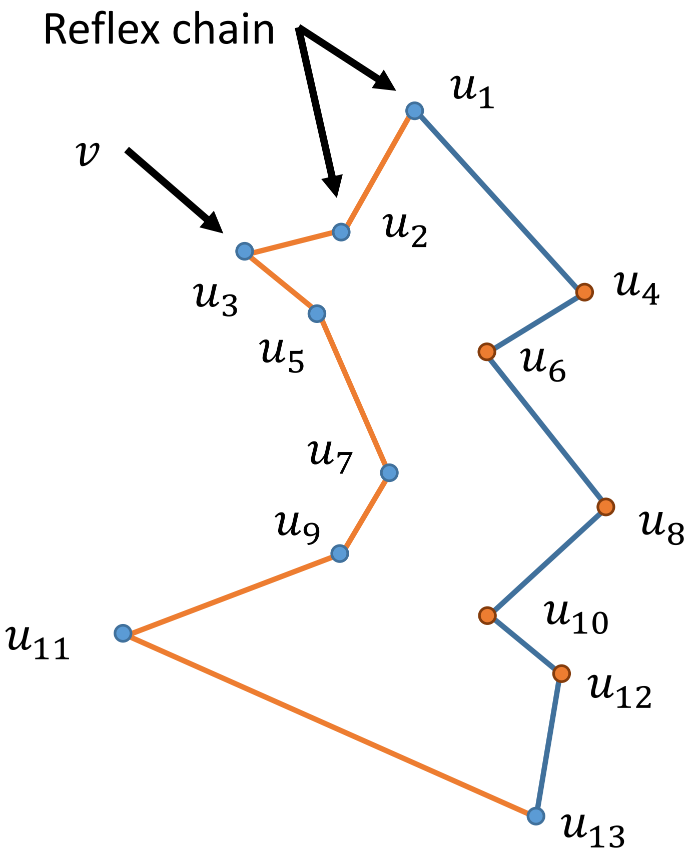
# POLYGON TRIANGULATION

- ALGORITHM: POLYGON TRIANGULATION: MONOTONE PARTITION
  - Partition into monotone polygons
  - **Triangulate each monotone polygon**



# TRIANGULATION OF MONOTONE POLYGON

- SORT VERTICES BY Y-COORDINATE BY A MERGE OF THE TWO CHAINS.
- LET $u_1, u_2, \dots, u_N$ BE THE SORTED SEQUENCE OF VERTICES, SO $y(u_1) > y(u_2) > \cdots > y(u_N)$.
- THE ALGORITHM PREFORMS OPERATIONS ON A REFLEX CHAIN, WHICH IS STORED AS A STACK
- INITIALIZATION
  - Reflex chain pushes two top vertices
  - Let v be the third highest vertex



# Description of the processing triangulation

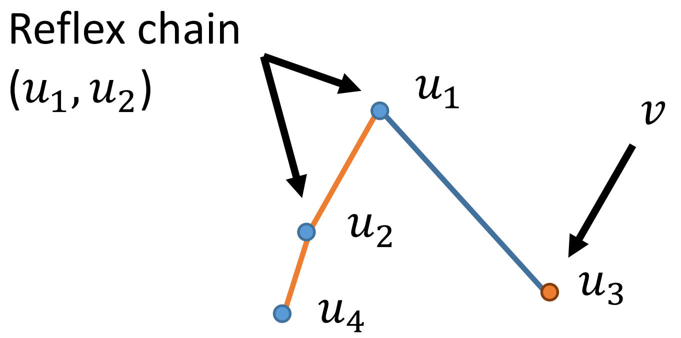
- The algorithm processes one vertex at a time in order of decreasing Y coordinate, creating diagonals of polygon P
  - At each step process 1 of 3 cases
- Each diagonal bounds a triangle, and leaves a polygon with one less side still to be triangulated

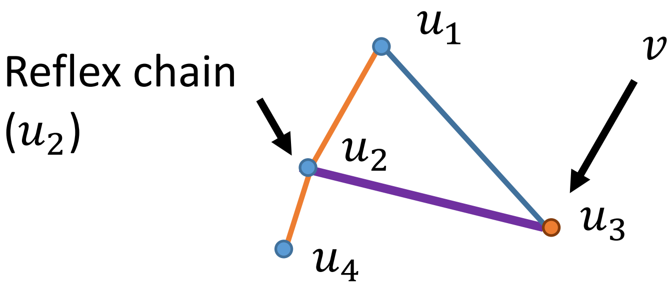
# TRIANGULATION OF MONOTONE POLYGON

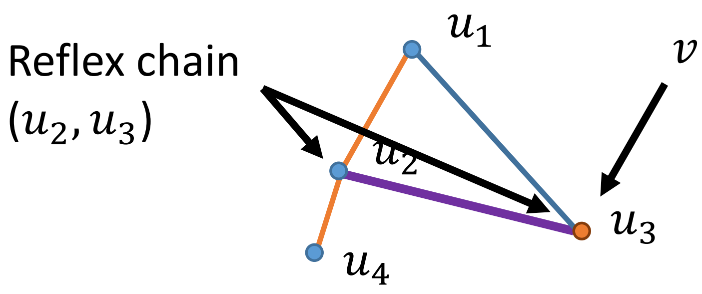
- WHILE V != LOWEST VERTEX DO:
  - Case 1: v is on chain opposite reflex chain
  - Case 2: v is adjacent to bottom of reflex chain and v+ is strictly convex
  - Case 3: v is adjacent to bottom of reflex chain and v+ is reflex or flat

# TRIANGULATION OF MONOTONE POLYGON

- CASE 1: V IS ON CHAIN OPPOSITE REFLEX CHAIN
  - Draw diagonal from v to second vertex from top of chain
  - Remove top of chain
  - If chain has one element then add v, advance v

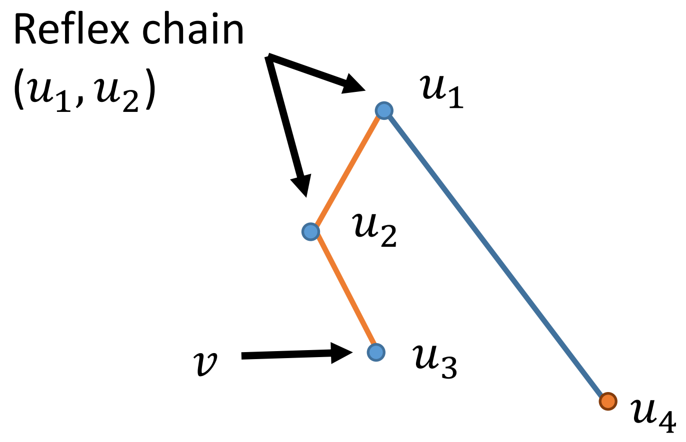


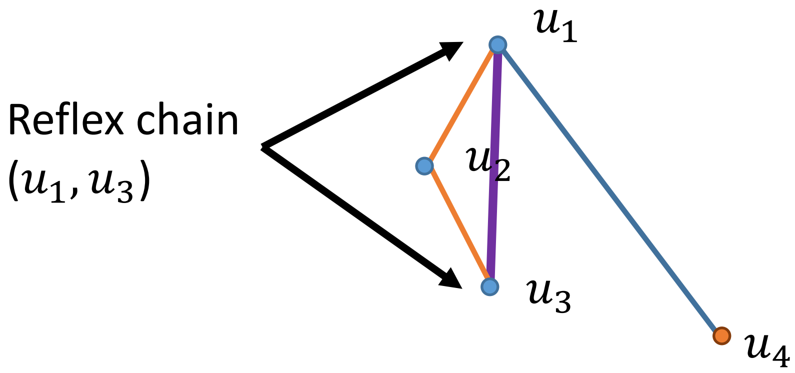



# TRIANGULATION OF MONOTONE POLYGON

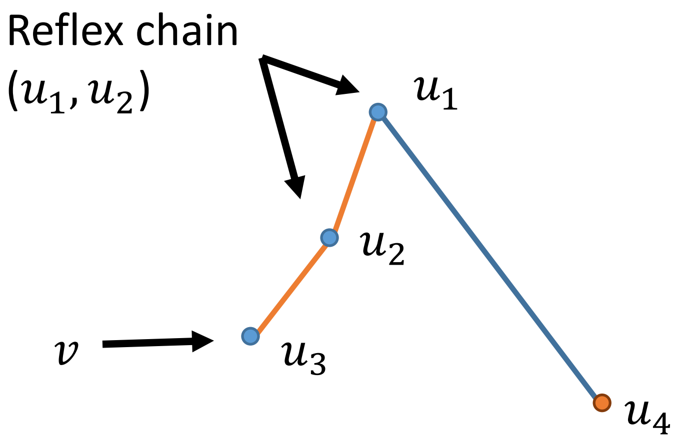
- CASE 2: V IS ADJACENT TO BOTTOM OF REFLEX CHAIN AND V+ IS STRICTLY CONVEX
  - Draw diagonal from v to second vertex from bottom of chain
  - Remove bottom of chain
  - Add v to chain, advance v

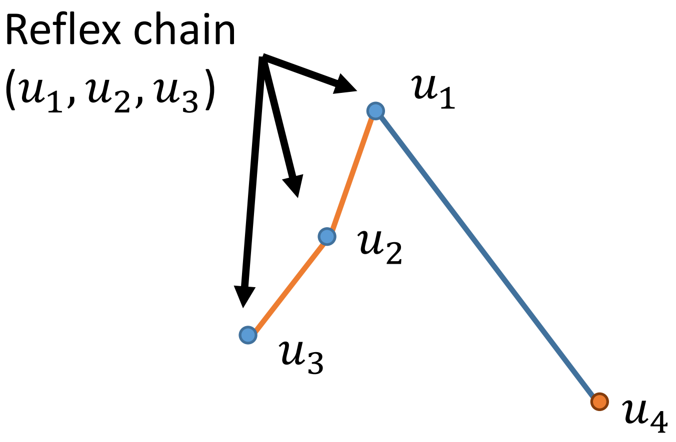


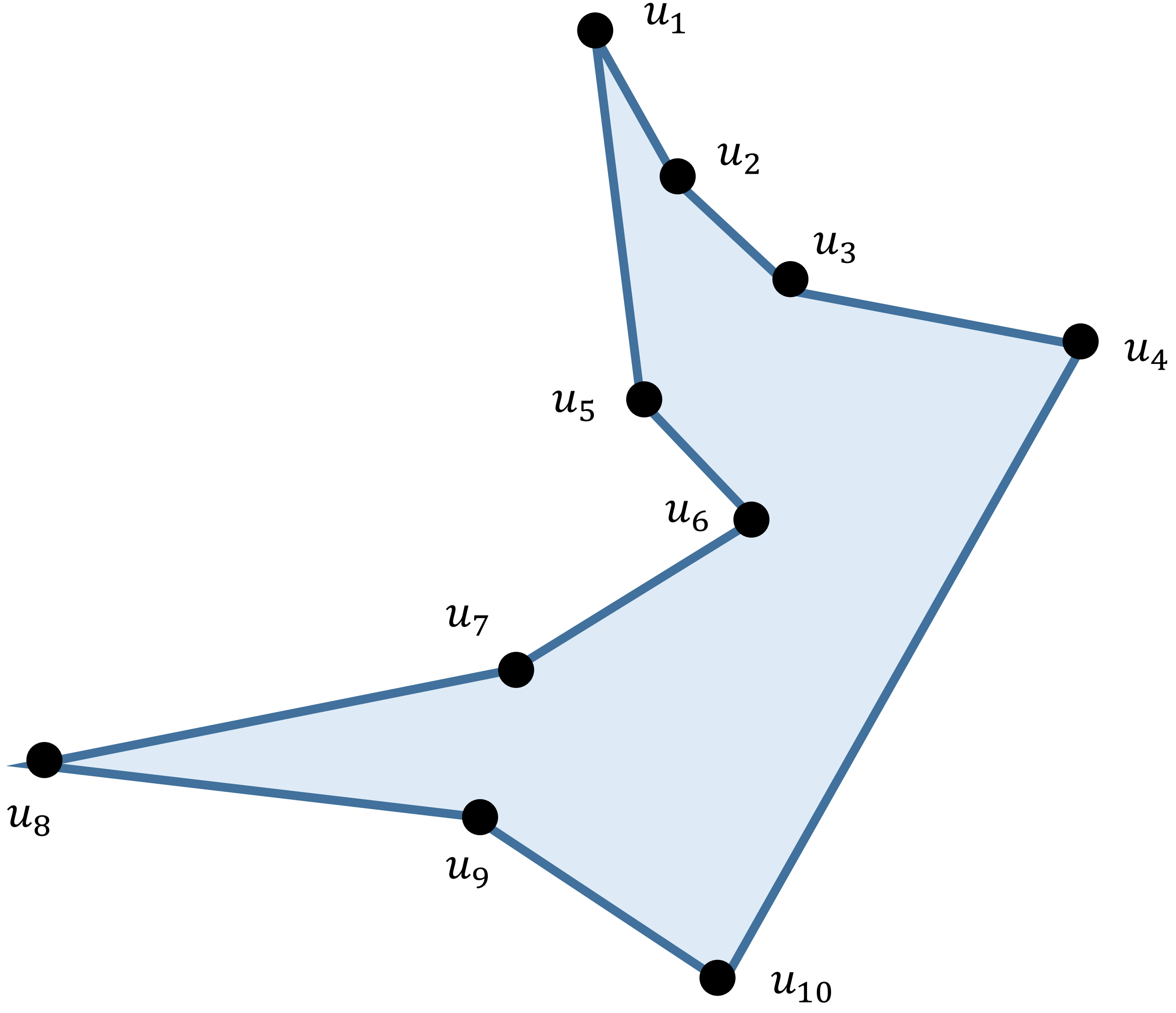
# Triangulation of monotone polygon

- Case 3: v is adjacent to bottom of reflex chain AND v+ is reflex or flat
    - Add v to bottom of reflex chain, advance v



# Triangulation of monotone polygon Example



# TRIANGULATION OF MONOTONE POLYGON EXAMPLE

Case 1: v is on chain opposite reflex chain

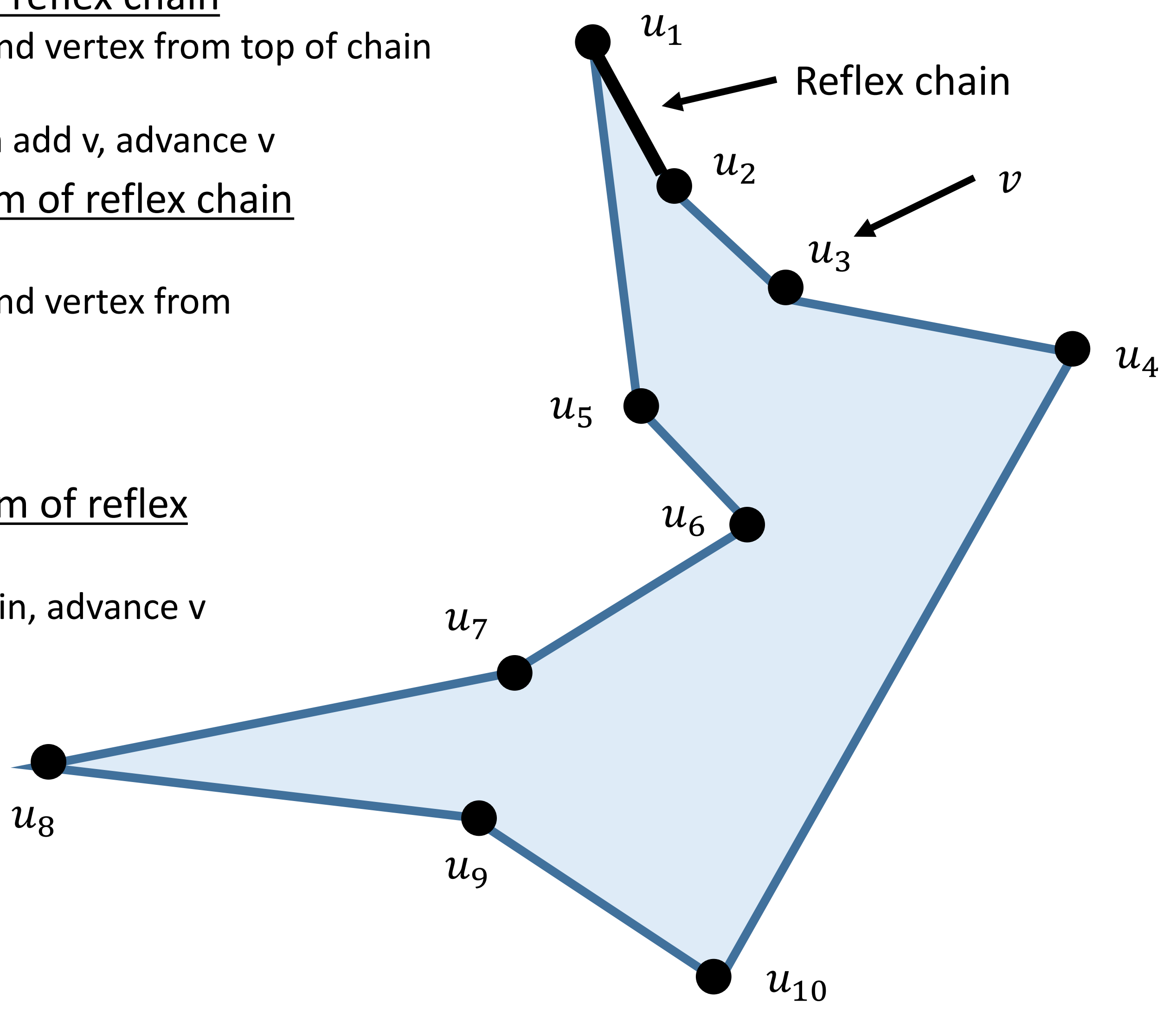
- Draw diagonal from v to second vertex from top of chain
- Remove top of chain
- If chain has one element then add v, advance v

Case 2: v is adjacent to bottom of reflex chain AND v+ is strictly convex

- Draw diagonal from v to second vertex from bottom of chain
- Remove bottom of chain
- Add v to chain, advance v

Case 3: v is adjacent to bottom of reflex chain AND v+ is reflex or flat

- Add v to bottom of reflex chain, advance v




Case 3

# TRIANGULATION OF MONOTONE POLYGON EXAMPLE

Case 1: v is on chain opposite reflex chain

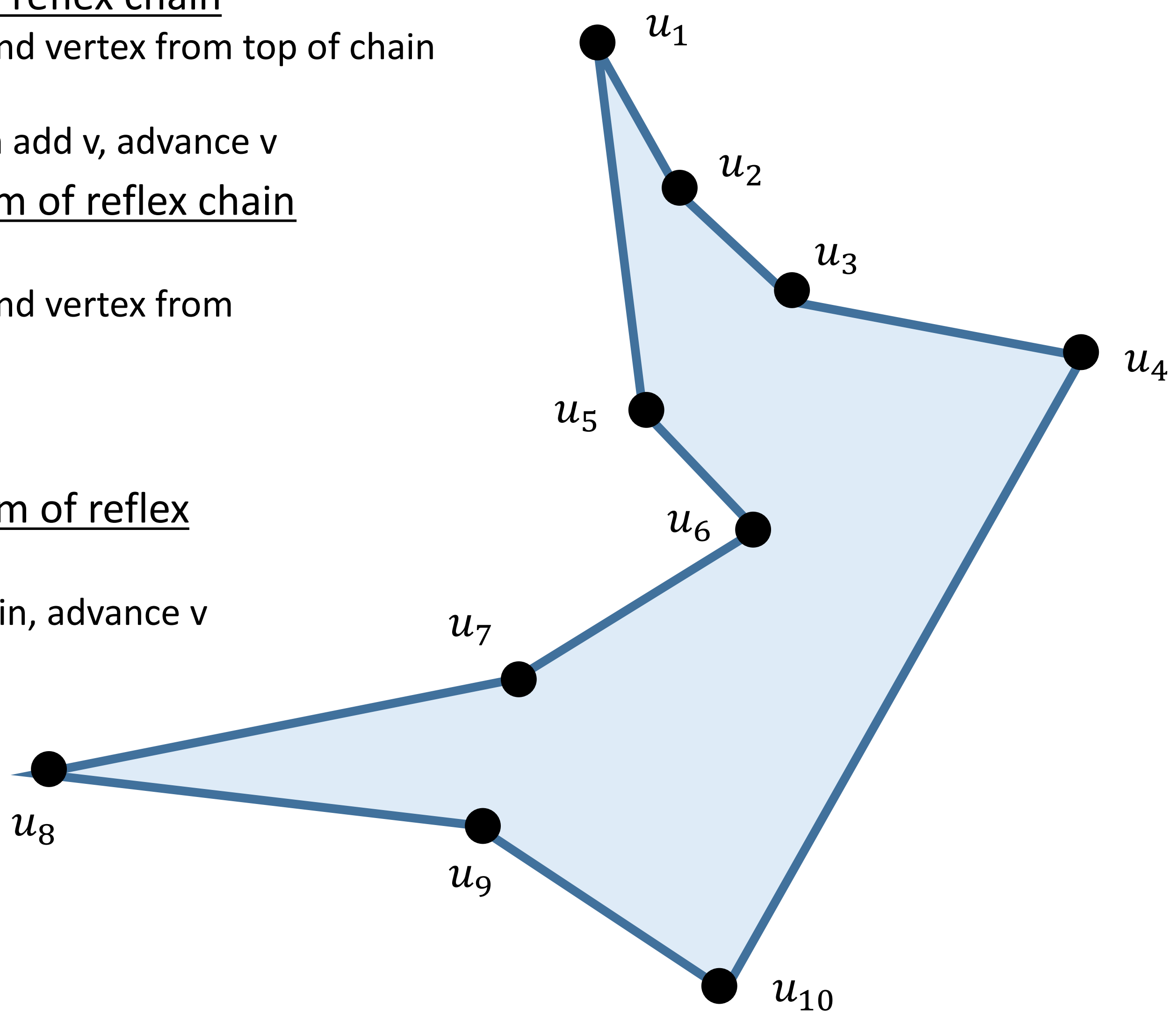
- Draw diagonal from v to second vertex from top of chain
- Remove top of chain
- If chain has one element then add v, advance v

Case 2: v is adjacent to bottom of reflex chain AND v+ is strictly convex

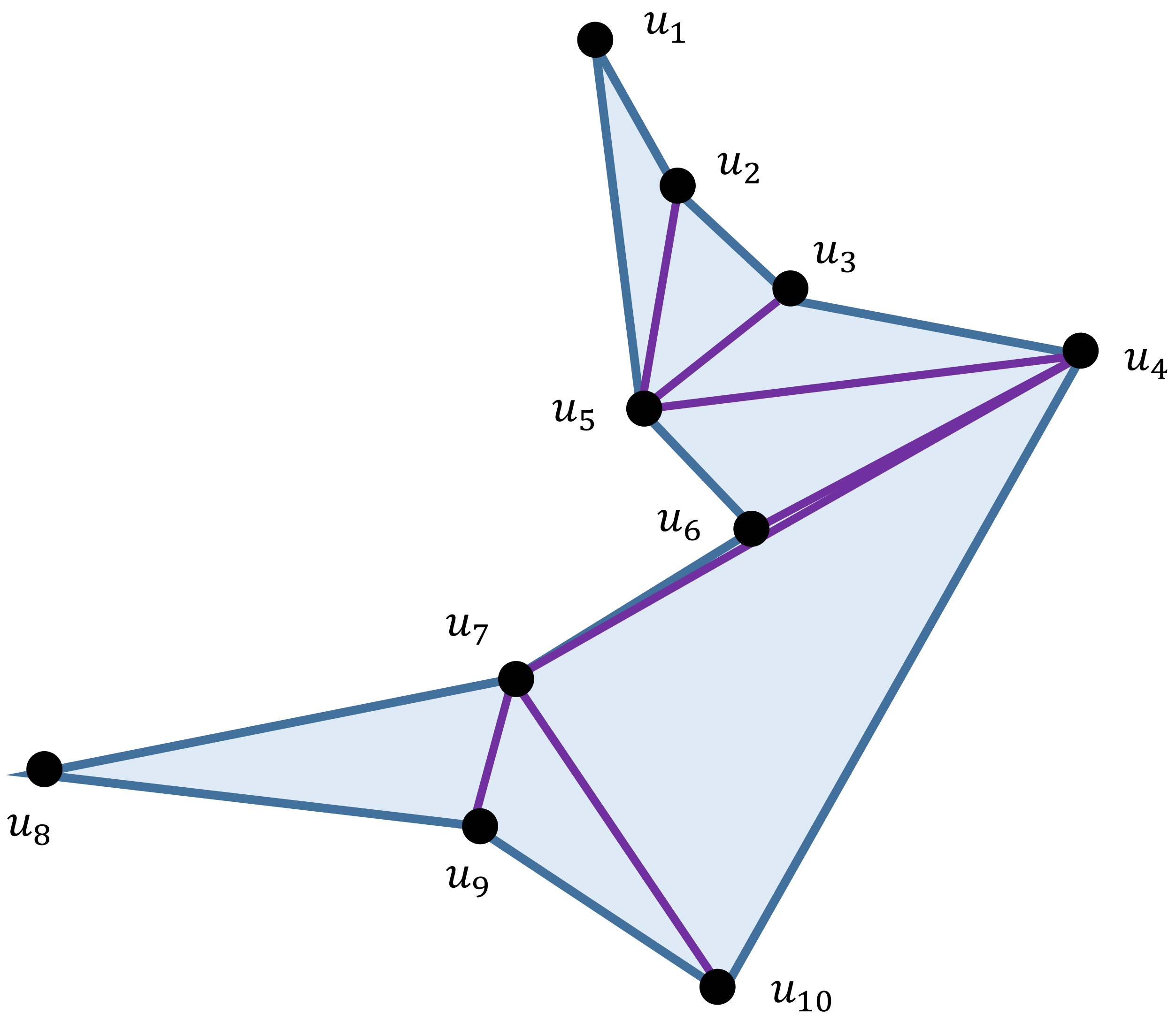
- Draw diagonal from v to second vertex from bottom of chain
- Remove bottom of chain
- Add v to chain, advance v

Case 3: v is adjacent to bottom of reflex chain AND v+ is reflex or flat

- Add v to bottom of reflex chain, advance v



# Triangulation of monotone polygon Example



## ANALYSIS OF TRIANGULATING A MONOTONE POLYGON

- THE INITIAL SORT (MERGE) REQUIRES O(N) TIME.
- TRIANGULATION VISITS EACH OF THE N VERTICES AND PLACES ON THE STACK EXACTLY ONCE, EXCEPT WHEN THE WHILE FAILS IN CASE (II).
  - This happens at most once per vertex, so that time can be charged to the current vertex.

- $\Rightarrow$ THE ALGORITHM REQUIRES O(N) TIME TO TRIANGULATE A MONOTONE POLYGON, WHERE N IS THE NUMBER OF VERTICES OF THE POLYGON.

# PUTTING IT ALL TOGETHER

- ANALYSIS OF TRIANGULATION OF SIMPLE POLYGON
  - Sort vertices by y coordinates: O(N log N)
  - Perform plane sweep to construct trapezoids: O(N log N)
  - Partition into monotone polygons: O(N)
  - Triangulate each monotone polygon: O(N)
- OVERALL: O(N LOG N)
  - N is the number of vertices of the polygon

# Summary

- Triangulation by Ear Removal improves the Diagonal-Based triangulation from $O(n^4)$ to $O(n^2)$
- Partition a simple polygon into monotone parts and triangulation of monotone polygons requires $O(n \log n)$ time complexity

## Description of the processing triangulation of y-monotone polygons

- The algorithm processes one vertex at a time in order of decreasing y coordinate, creating diagonals of polygon P.
    - The sweep line moves from top to down and stops at each vertex of polygon P

- Each diagonal bounds a triangle, and leaves a polygon with one less side still to be triangulated

# DESCRIPTION OF THE PROCESSING TRIANGULATION OF Y-MONOTONE POLYGONS

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

# Towards Linear-Time Triangulation

| Year | Complexity | Authors |
|---|---|---|
| 1911 | $O(n^2)$ | Lennes |
| 1978 | $O(n \log n)$ | Garey et al. |
| 1983 | $O(n \log r)$, r reflex | Hertel & Melhorn |
| 1984 | $O(n \log s)$, s sinuosity | Chazelle & Incerpi |
| 1986 | $O(n \log \log n)$ | Tarjan & Van Wyk |
| 1988 | $O(n + nt_0)$, $t_0$ int. triangles | Toussaint |
| 1989 | $O(n \log^* n)$, randomized | Clarkson, Tarjan & Van Wyk |
| 1990 | $O(n \log^* n)$ bounded integer coordinates | Kirkpatrick, Klawe, Tarjan |
| 1990 | $O(n)$ | Chazelle |
| 1991 | $O(n \log^* n)$, randomized | Seidel |

# LINEAR-TIME TRIANGULATION

- CHAZELLE'S ALGORITHM (HIGH-LEVEL SKETCH)
  - Computes visibility map
  - Algorithm is like MergeSort (divide-and-conquer)
    - Partition polygon of n vertices into n/2 vertex chains
    - Merge visibility maps of subchains to get one for chain
  - Improve this by dividing process into 2 phases:
    1. Coarse approximations of visibility maps for linear-time merge
    2. Refine coarse map into detailed map in linear time

# Convex Partitioning

- Polygon partitioning is an important preprocessing step for many geometric algorithms
- Partitioning a polygon means completely dividing the interior into nonoverlapping pieces.
- Covering a polygon means that our decomposition is permitted to contain mutually overlapping pieces.
- An issue associated with polygon decomposition is whether we are allowed to add Steiner vertices (either by splitting edges or adding interior points) or whether we are restricted to adding chords between two existing vertices.

# Convex Partitioning

- Competing Goals:
  - minimize number of convex pieces
  - minimize partitioning time
- Add (Steiner) points or just use diagonals and not add points?

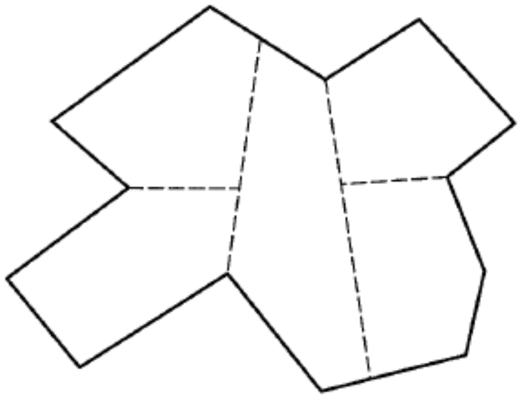
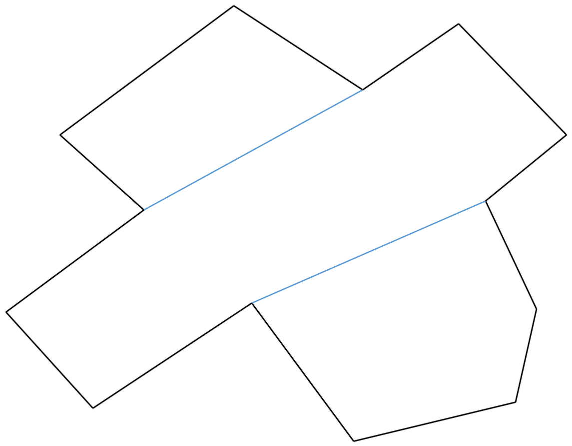


**FIGURE 2.10** $r+1$ convex pieces: $r=4$; 5 pieces.

Adding segments with Steiner points.
$r$ = number of reflex vertices



Adding only diagonals.

# Convex Partitioning

- Theorem (Chazelle): Let F be the fewest number of convex pieces into which a polygon may be partitioned. For a polygon of r reflex vertices:

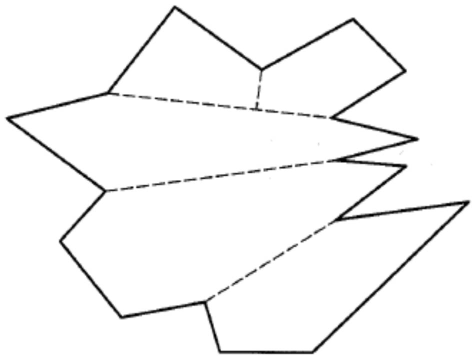


FIGURE 2.11 $\lceil r/2 \rceil + 1$ convex pieces: $r = 7$; 5 pieces.

$$\left\lceil \frac{r}{2} \right\rceil + 1 \le \Phi \le r + 1$$

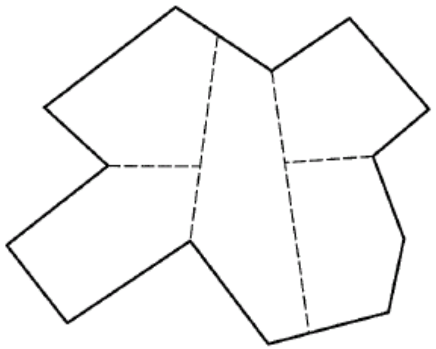


FIGURE 2.10 $r + 1$ convex pieces: $r = 4$; 5 pieces.

<u>Lower bound</u>:
Must eliminate all reflex vertices.
Single segment resolves at most 2 reflex angles.

<u>Upper bound</u>:
Bisect each reflex angle.

# Convex Partitioning

- Hertel-Mehlhorn's Algorithm (1983):
  - Start with any triangulation.
  - Iteratively remove nonessential diagonals
    - Essential diagonal d are those that creates non-convex piece
  - Can be done in O(n) time with the use of appropriate data structures
  - The only issue is how far from the optimum might it be

# Algorithms for Optimal Convex Partitioning of a Polygon

- Optimal convex partition using diagonals
  - Greene (1983): $O(n^4)$ time with dynamic programming
  - Keil (1985): $O(n^3 \log n)$ time with dynamic programming
- Optimal convex partition using arbitrary segments
  - Chazelle (1980) : $O(n^3)$ time
- Approximate convex partition removing inessential diagonals
  - Hertel/ Mehlhorn: $O(n)$ time after triangulation
- Approximate convex partition using sweep-line
  - Greene (1983): $O(n \lg n)$, starts with y-monotone partition

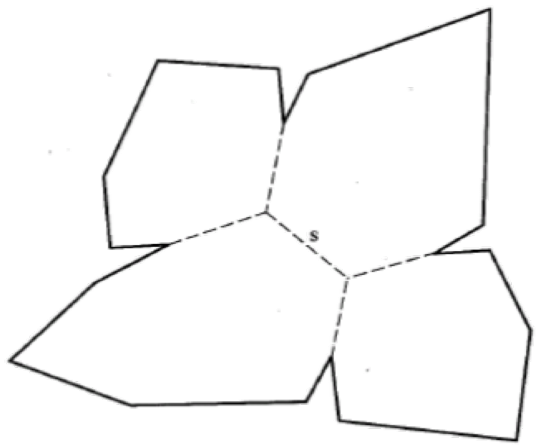



**FIGURE 2.13** An optimal convex partition. Segment $s$ does not touch $\partial P$.

# Convex Polygon Partitioning

- Y-Monotone partition:
  - de Berg et al.: $O(n \lg n)$ time (see earlier slides)
- Optimal convex partition using diagonals
  - Greene (1983): $O(n^4)$ time with dynamic programming
- Approximate convex partition removing inessential diagonals
  - Hertel/ Melhorn: $O(n)$ time after triangulation
- Approximate convex partition using sweep-line
  - Greene (1983): $O(n \lg n)$, starts with y-monotone partition