CISC 868 Fall 2011
Week 2

September 19, 2011

The Art Gallery Theorem

Please see the presentation slides outlining this gem of a theorem. This material is taken from Chapter 1. of Joseph O'Rourke's other text book on Computational Geometry [1].

Triangulating a Polygon

Two algorithms were presented, one from our text book and the other is taken from Chapter 2. O'Rourke's book [1].

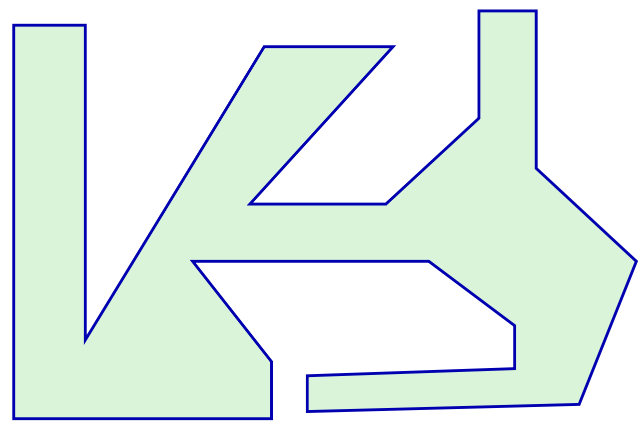
References

[1] Joseph O'Rourke. *Computational Geometry in C second edition* Cambridge University Press, 1998.

Art Gallery Theorem

The floor plan of an art gallery modeled as a simple polygon with n vertices. How many guards needed to see the whole room?

Each guard is stationed at a fixed point, has 360^o vision, and cannot see through the walls.



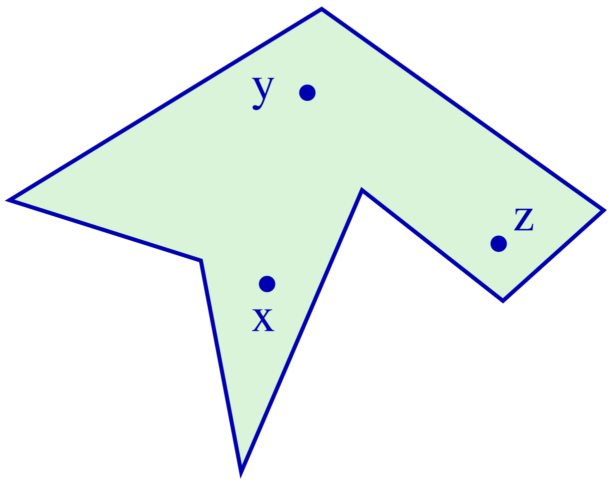
Story: Problem posed to Vasek Chvatal by Victor Klee at a math conference in 1973. Chvatal solved it quickly with a complicated proof, which has since been simplified significantly using triangulation.

Art Gallery Theorem

* These slides copied from Tom Fevens of Concordia U.

Formulation

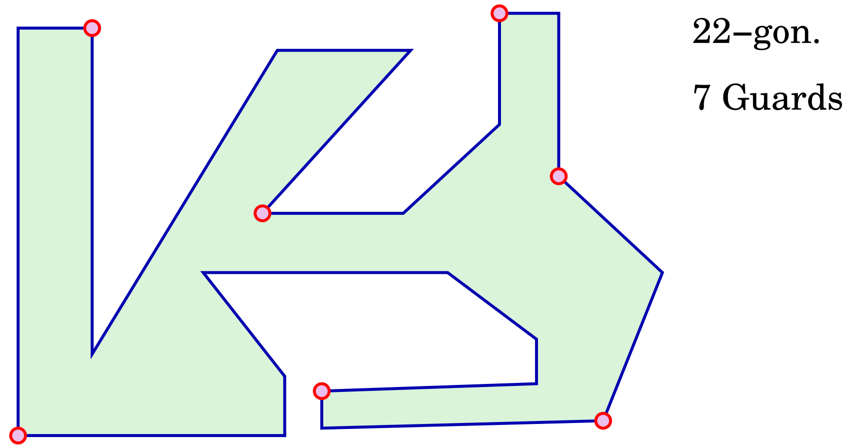
- Visibility: p, q visible if $pq \in P$.
- y is visible from x and z. But x and z not visible to each other.



- $g(P) =$ min. number of guards to see P
- $g(n) = \max_{|P|=n} g(P)$
- Art Gallery Theorem asks for bounds on function $g(n)$: what is the smallest $g(n)$ that always works for any n-gon?

Art Gallery Theorem

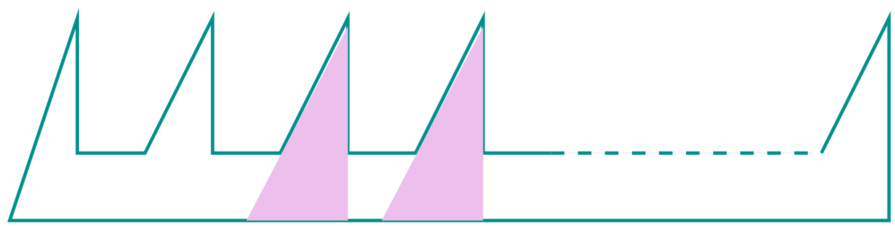
Art Gallery Theorem



Theorem:

$$g(n) = \lfloor n/3 \rfloor$$

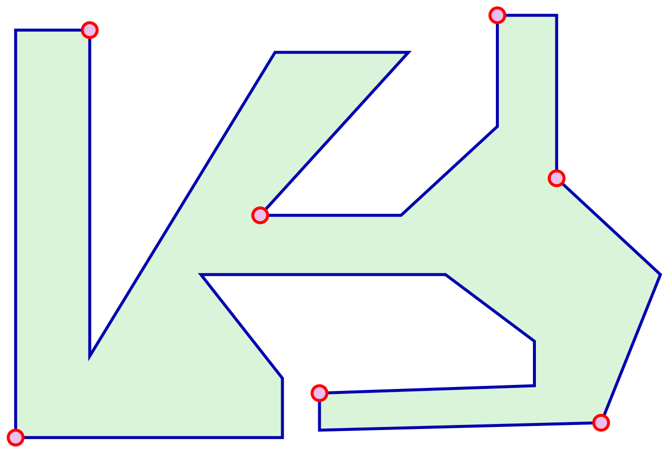
1. Every n-gon can be guarded with $\lfloor n/3 \rfloor$ vertex guards (sufficient).
2. Some n-gons require at least $\lfloor n/3 \rfloor$ (arbitrary) guards.



Necessity Construction

Art Gallery Theorem

Art Gallery Theorem



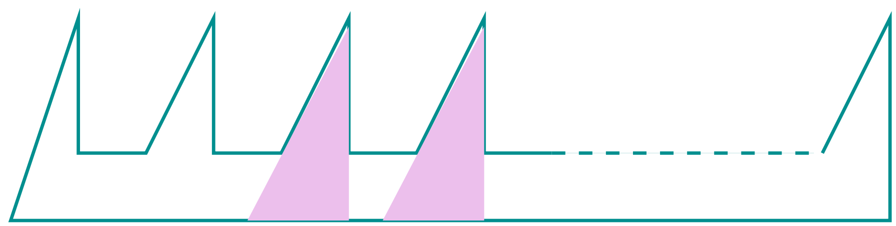
22–gon.

7 Guards

Theorem:

$$g(n) = \lfloor n/3 \rfloor$$

1. Every n-gon can be guarded with $\lfloor n/3 \rfloor$ vertex guards (sufficient).
2. Some n-gons require at least $\lfloor n/3 \rfloor$ (arbitrary) guards.



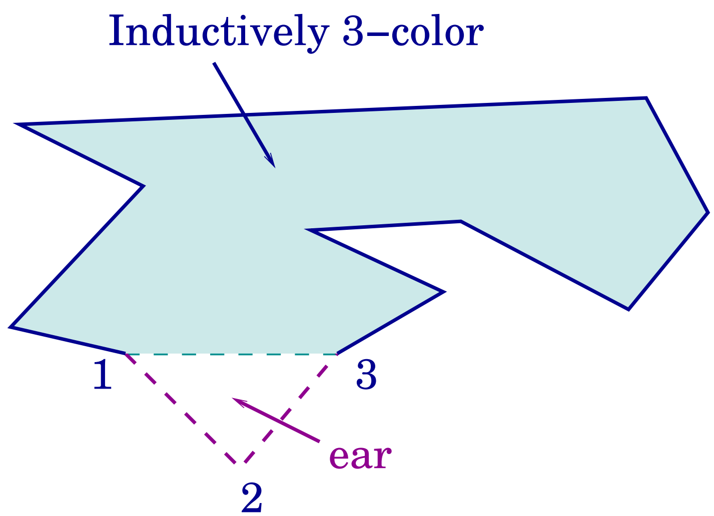
Necessity Construction

Art Gallery Theorem

Fisk's Proof

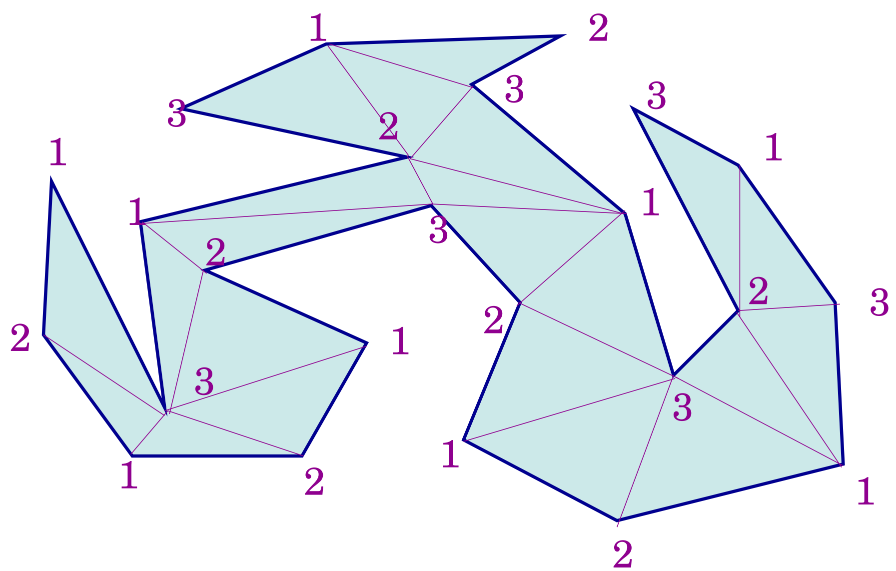
Lemma: Triangulation graph can be 3-colored.

- P plus triangulation is a planar graph.
- 3-coloring means vertices can be labeled 1,2, or 3 so that no edge or diagonal has both endpoints with same label.
- Proof by Induction:
  1. Remove an ear.
  2. Inductively 3-color the rest.
  3. Put ear back, coloring new vertex with the label not used by the boundary diagonal.



Art Gallery Theorem

Proof



- Triangulate P. 3-color it.
- Least frequent color appears at most $\lfloor n/3 \rfloor$ times.
- Place guards at this color positions—a triangle has all 3 colors, so seen by a guard.
- In example: Colors 1, 2, 3 appear 9, 8 and 7 times, resp. So, color 3 works.

Art Gallery Theorem

Algorithm Triangulate Polygon I.

Input: A polygon $P = (p_1, p_2, \ldots, p_n)$ represented by its vertices in counter-clockwise around its boundary

Output: $T(P) = (d_1, d_2, \ldots, d_{n-3})$ a triangulation of P represented by its diagonals.

1. Find a diagonal, d, of P, partitioning P into two polygons P_1 and P_2.

2. Recursively triangulate P_1 and P_2.
   $T(P) = T(P_1) \cup T(P_2) \cup \{d\}$.

Worst case complexity

1. The computational complexity of finding a diagonal is $O(n)$.

2. In the worst case one of the two polygons resulting from the partition is a triangle. So we get the recurrence:

$f(n) = f(n-1) + cn$ where c is a positive constant. Therefore $f(n) = O(n^2)$

Algorithm Triangulate Polygon II.

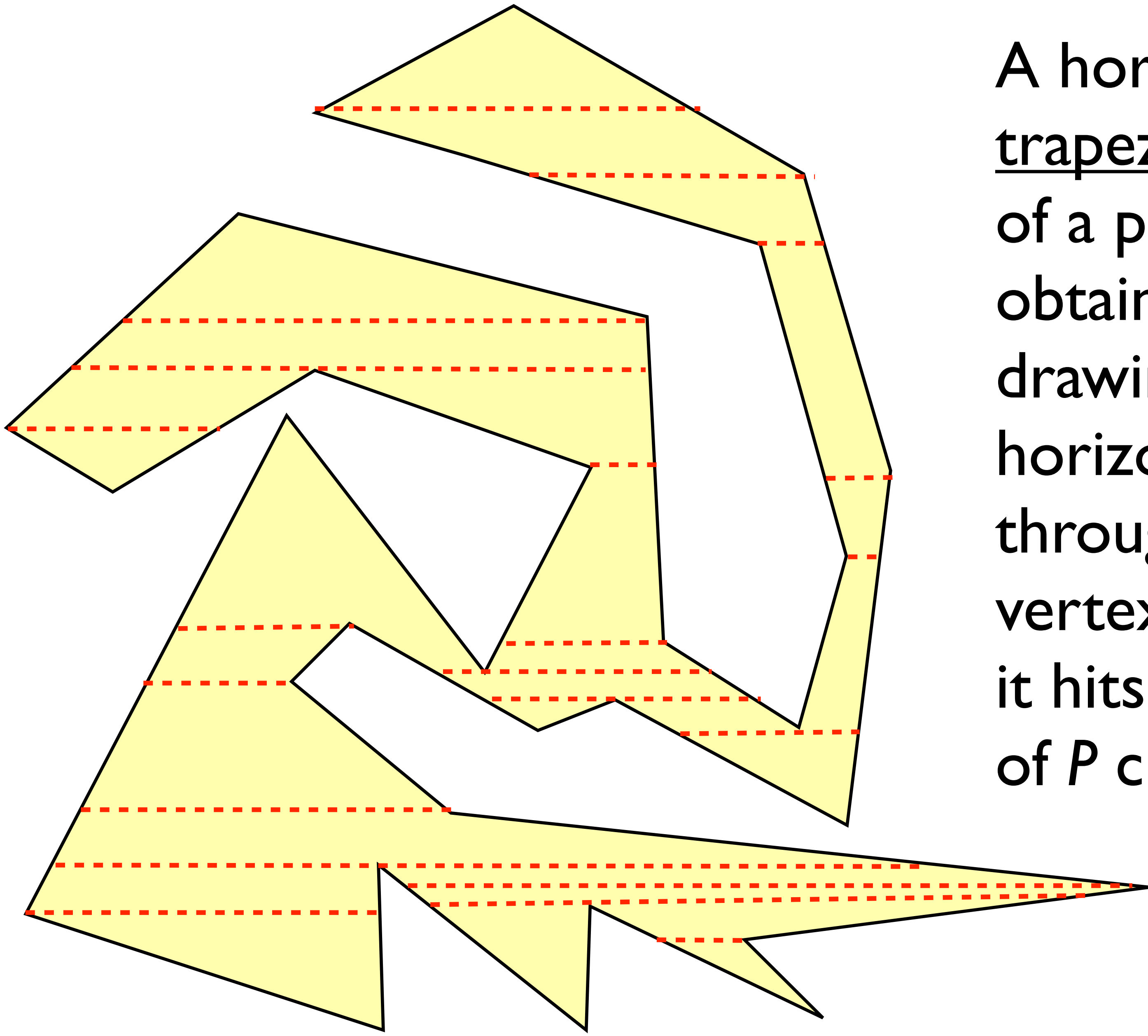
Input: A polygon $P = (p_1, p_2, ..., p_n)$ represented by its vertices in counter-clockwise around its boundary

Output: $T(P) = (d_1, d_2, ..., d_{n-3})$ a triangulation of P represented by its diagonals.

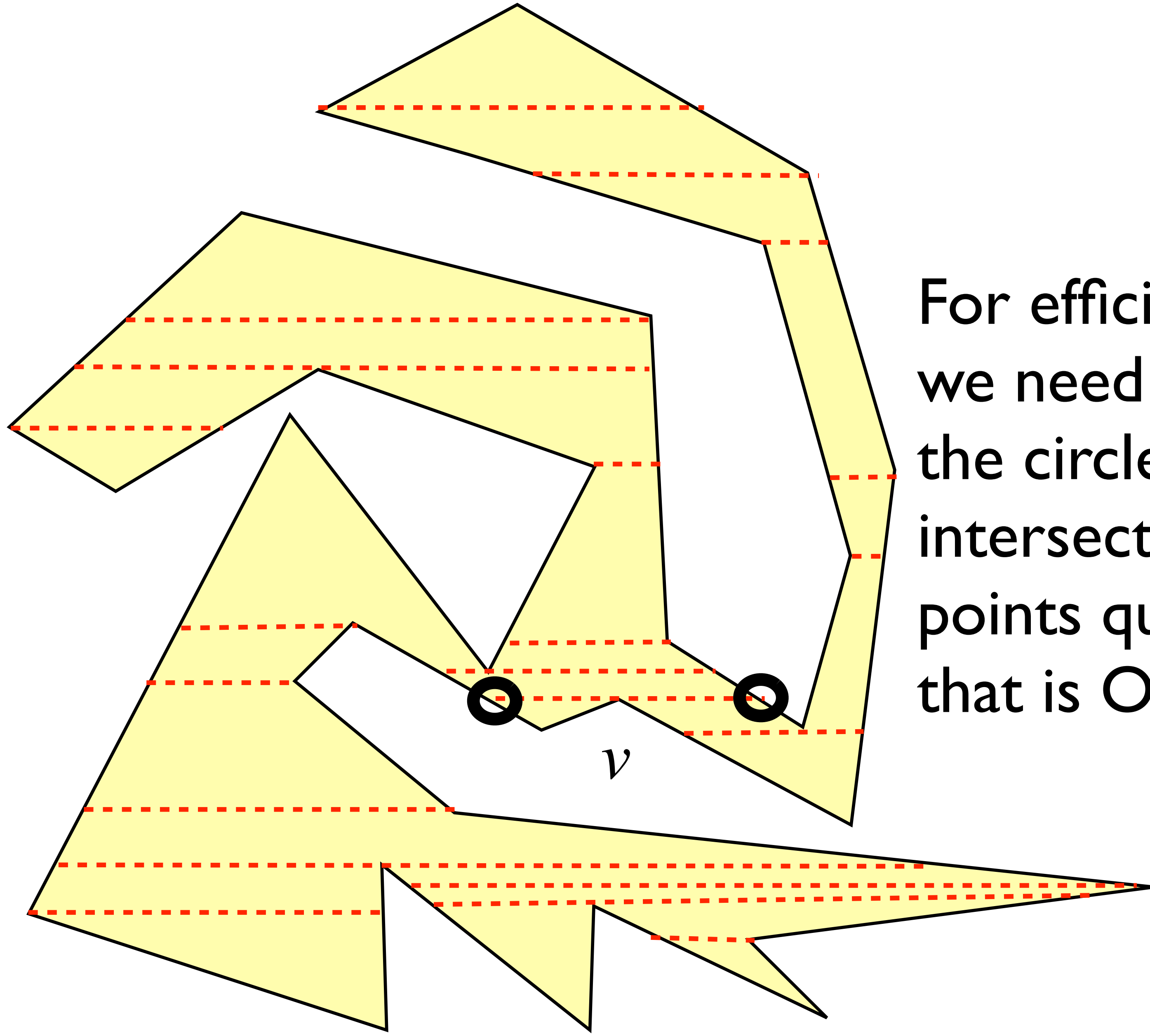
1. Partition P into trapezoids.

2. Add trapezoid diagonals whenever possible yielding a partition of P into triangles and uni-monotone polygons.

3. Triangulate the uni-monotone polygons arising from the partition in step 2.

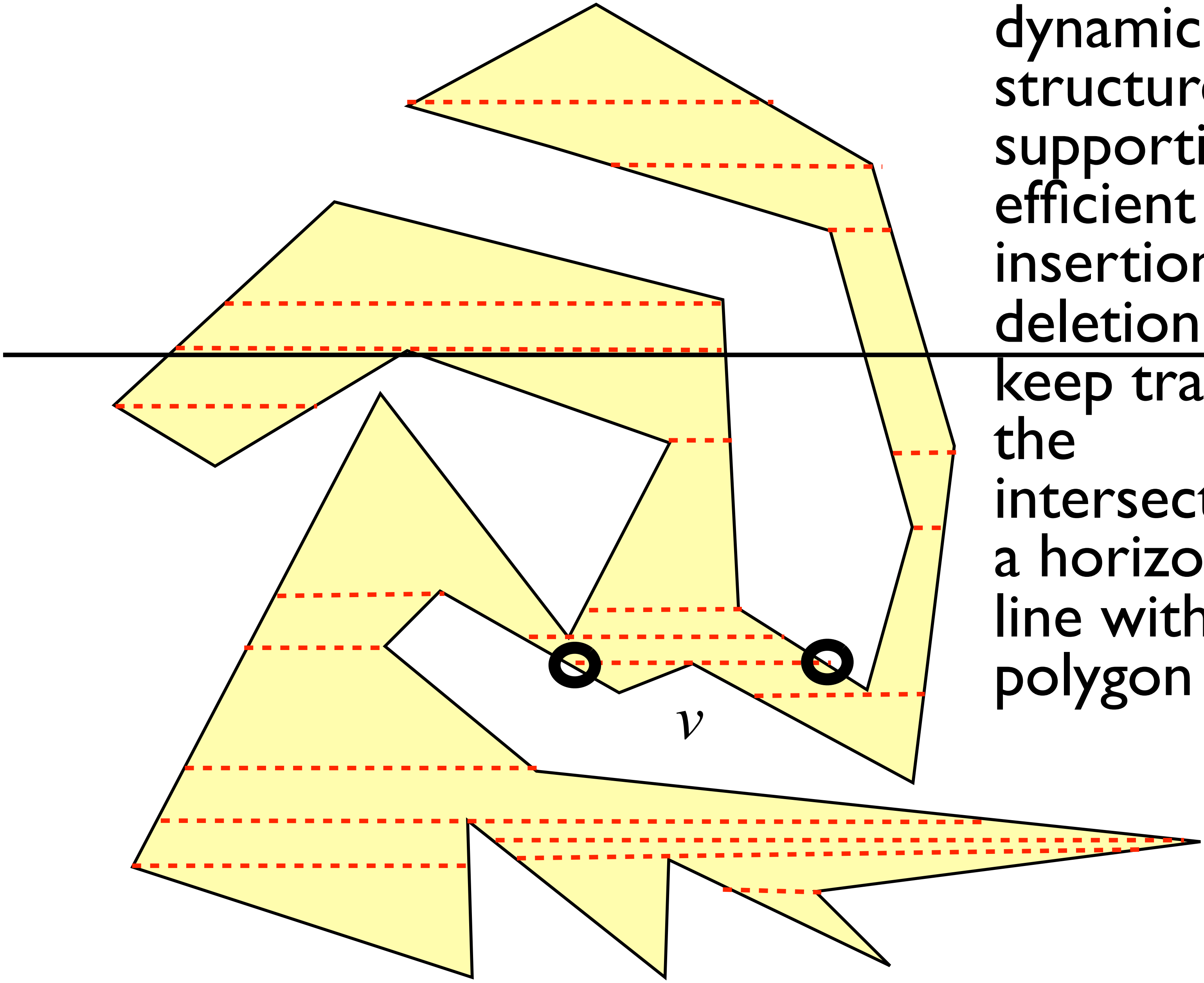


A horizontal <u>trapezoidilization</u> of a polygon *P* is obtained by drawing a horizontal line through every vertex v of P until it hits the edge(s) of *P* closest to *v*.

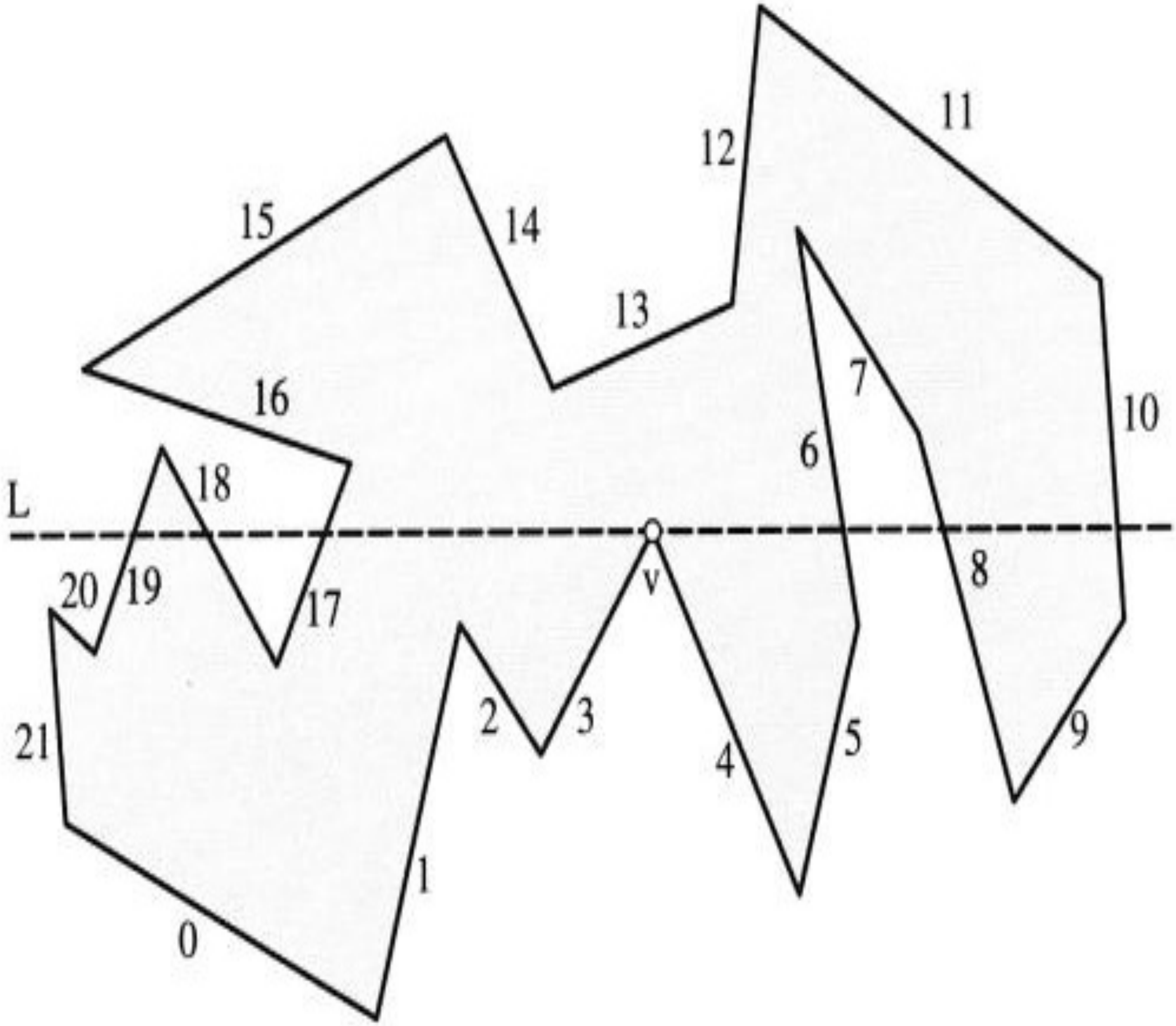


For efficiency we need to find the circled intersection points quickly, that is O(log n)

Sorting the vertices of P (from top to bottom) and maintaining a dynamic structure allows us to find every horizontal trapezoid edge in total time complexity $O(n \log n)$.



Maintain a dynamic data structure supporting efficient insertion and deletion to keep track of the intersection of a horizontal line with the polygon



*This figure is copied from Joseph O'Rourke Computational Geometry in C.

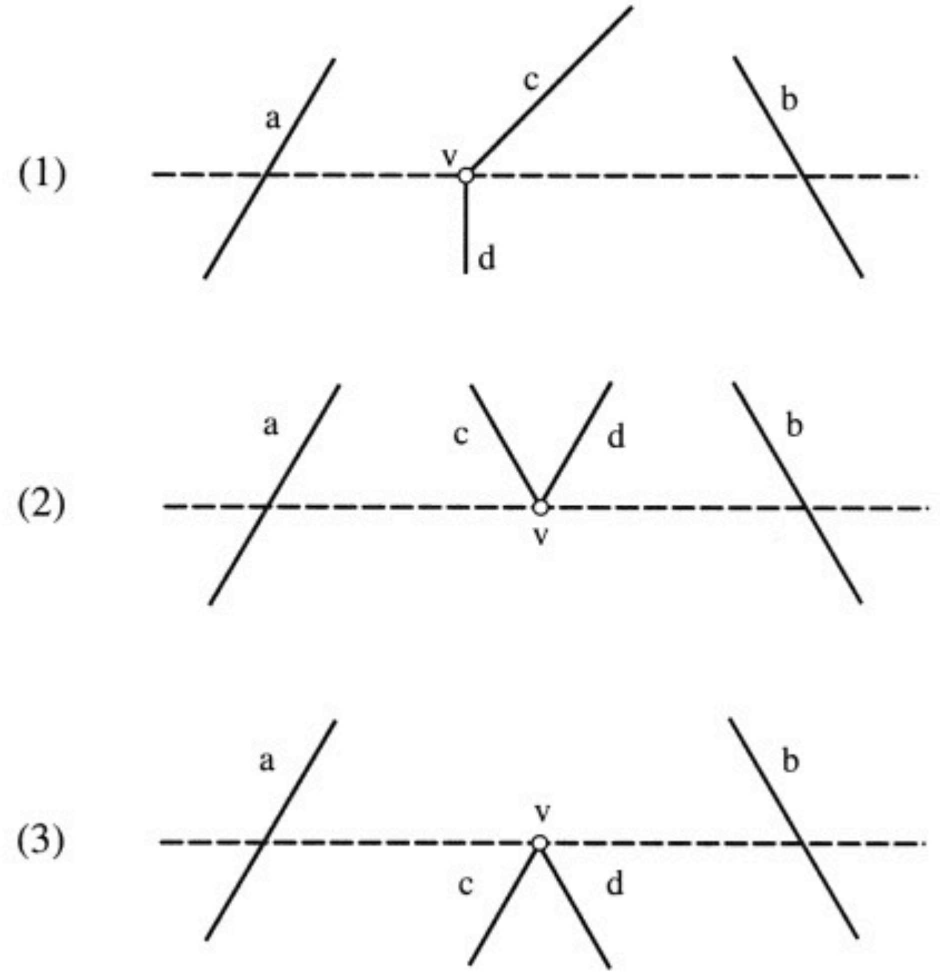
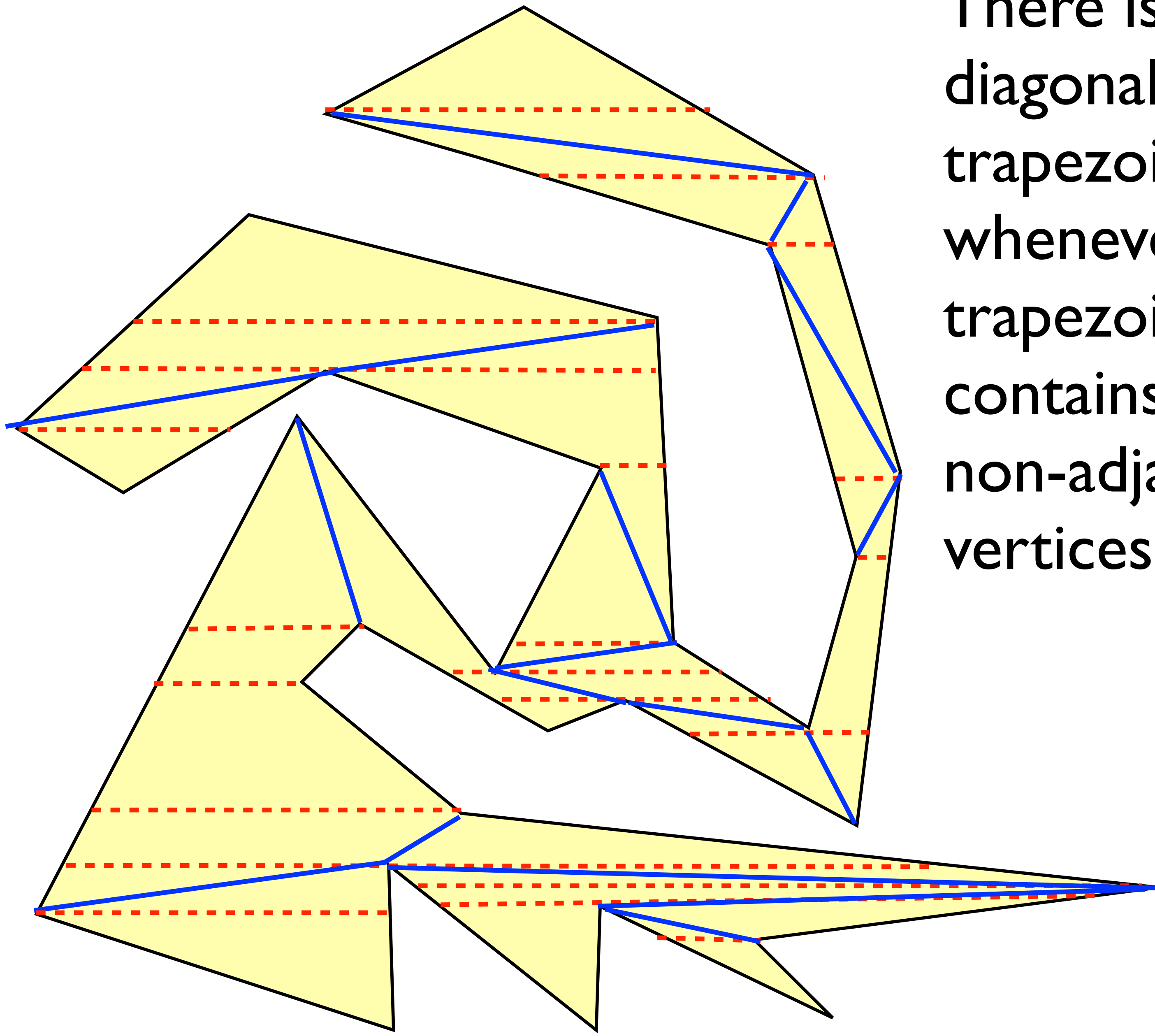


FIGURE 2.5 Sweep line events: (1) replace c by d; (2) delete c and d; (3) insert c and d.

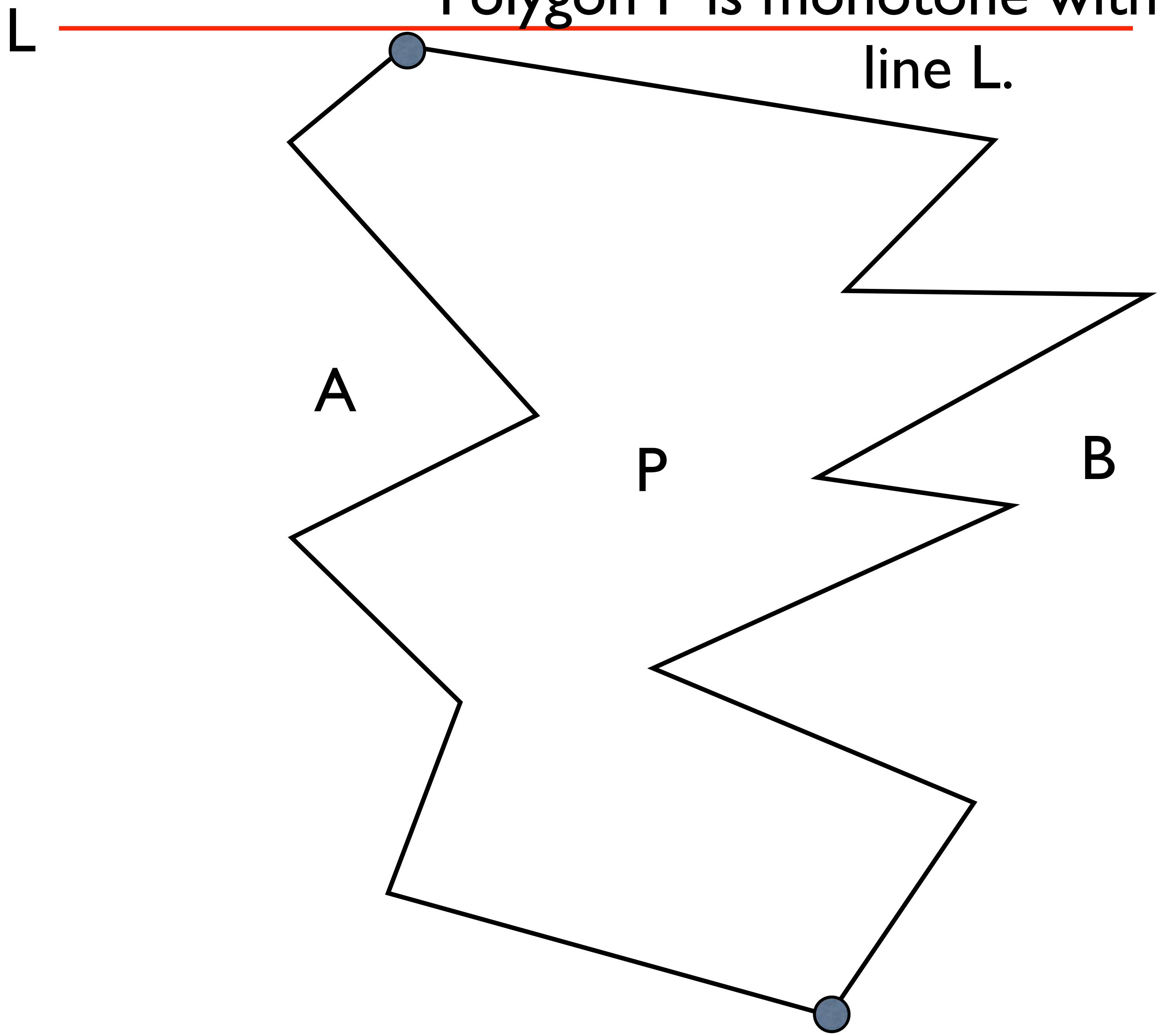
* This figure is copied from Joseph O'Rourke Computational Geometry in C.

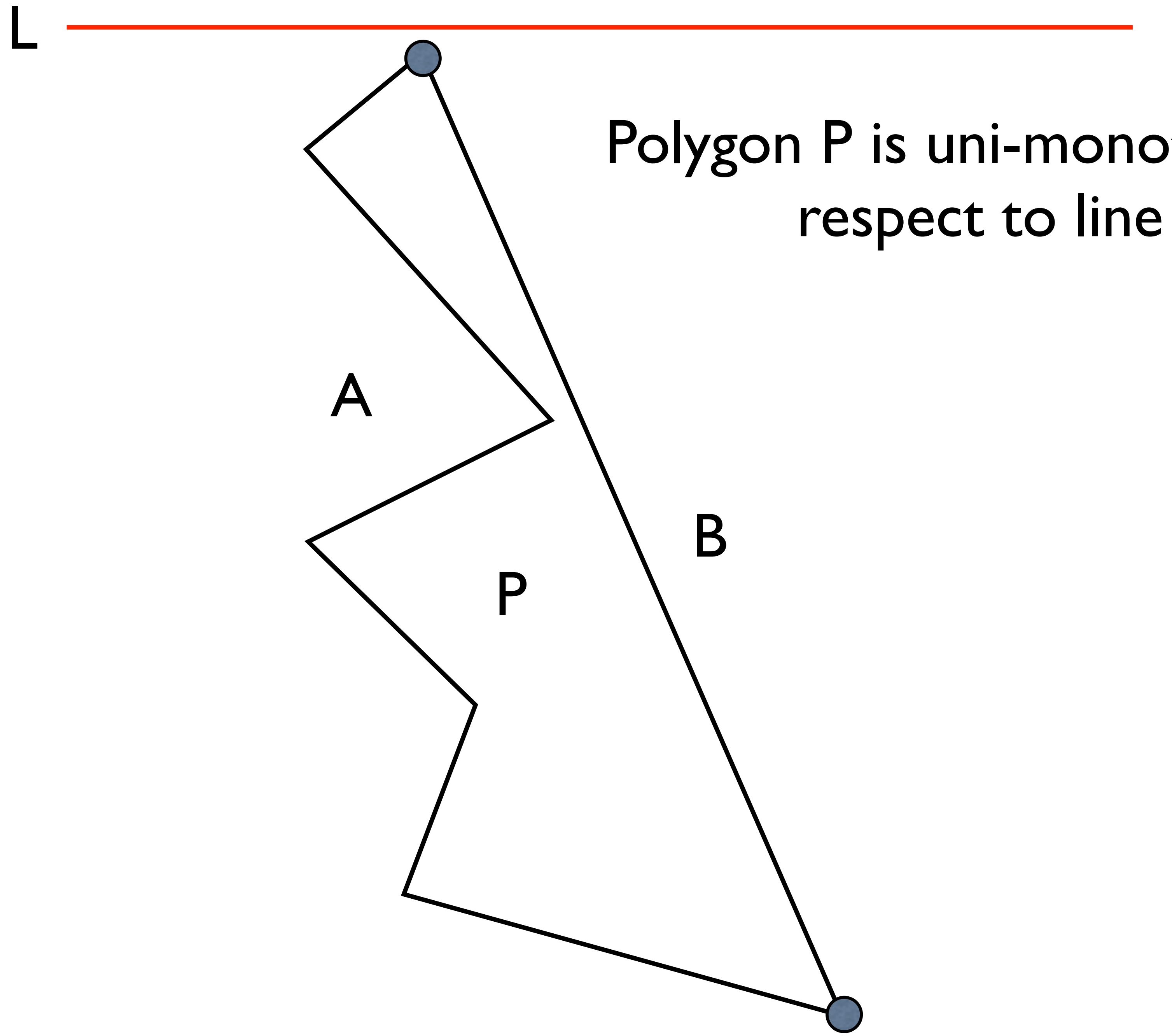


There is a diagonal in a trapezoid whenever the trapezoid contains two non-adjacent vertices of P.

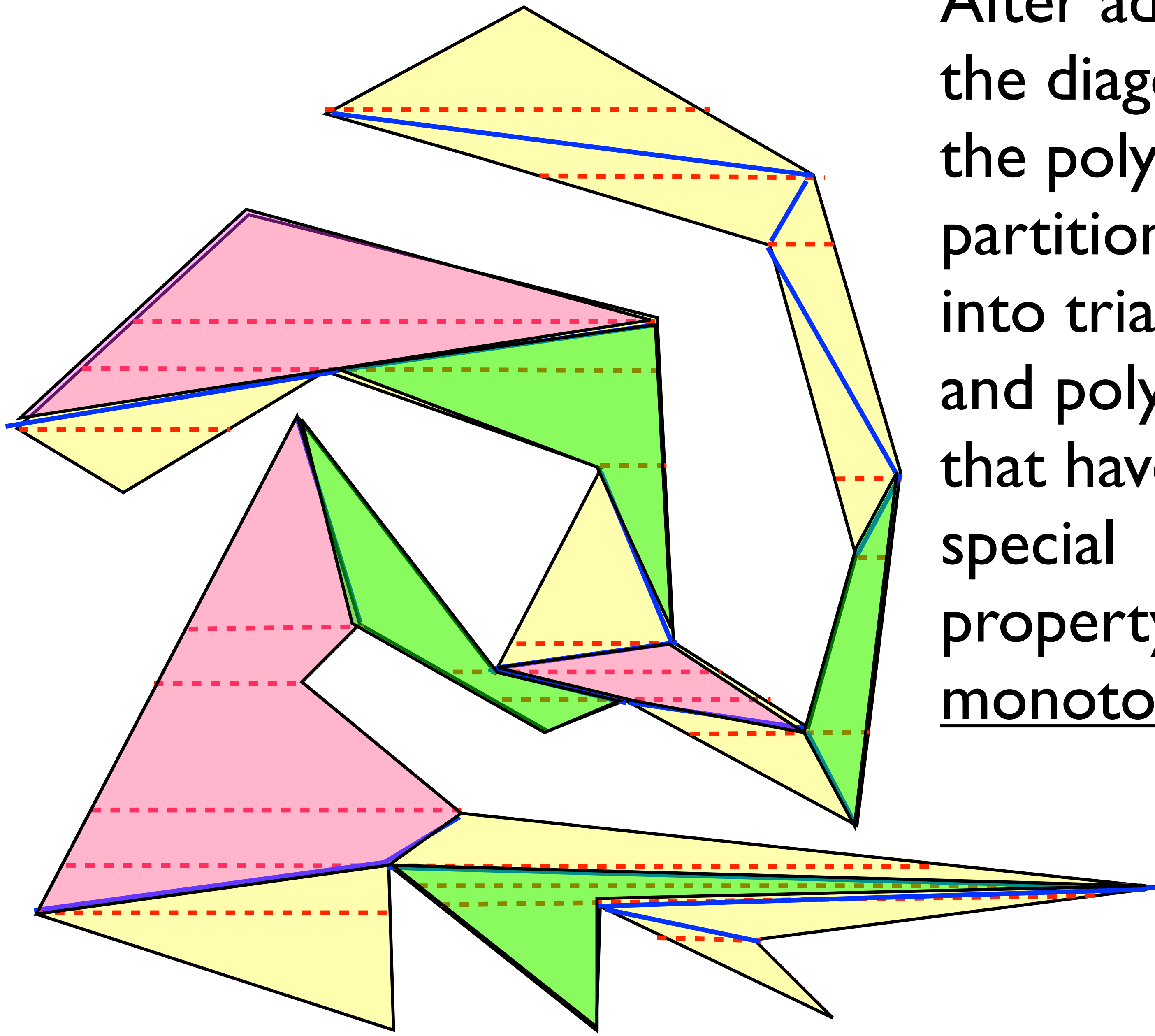
- A <u>polygonal chain C is monotone</u> to a line L if every line parallel to L intersects C in at most one point.
- A <u>polygon P is monotone</u> with respect to a line L if the boundary of P can be split into two polygonal chains, A and B such that each chain is monotone with respect to L.
- A polygon P is uni-monotone with respect to a line L if it is monotone and either chain A or chain B is a single edge.

Polygon P is monotone with respect to line L.

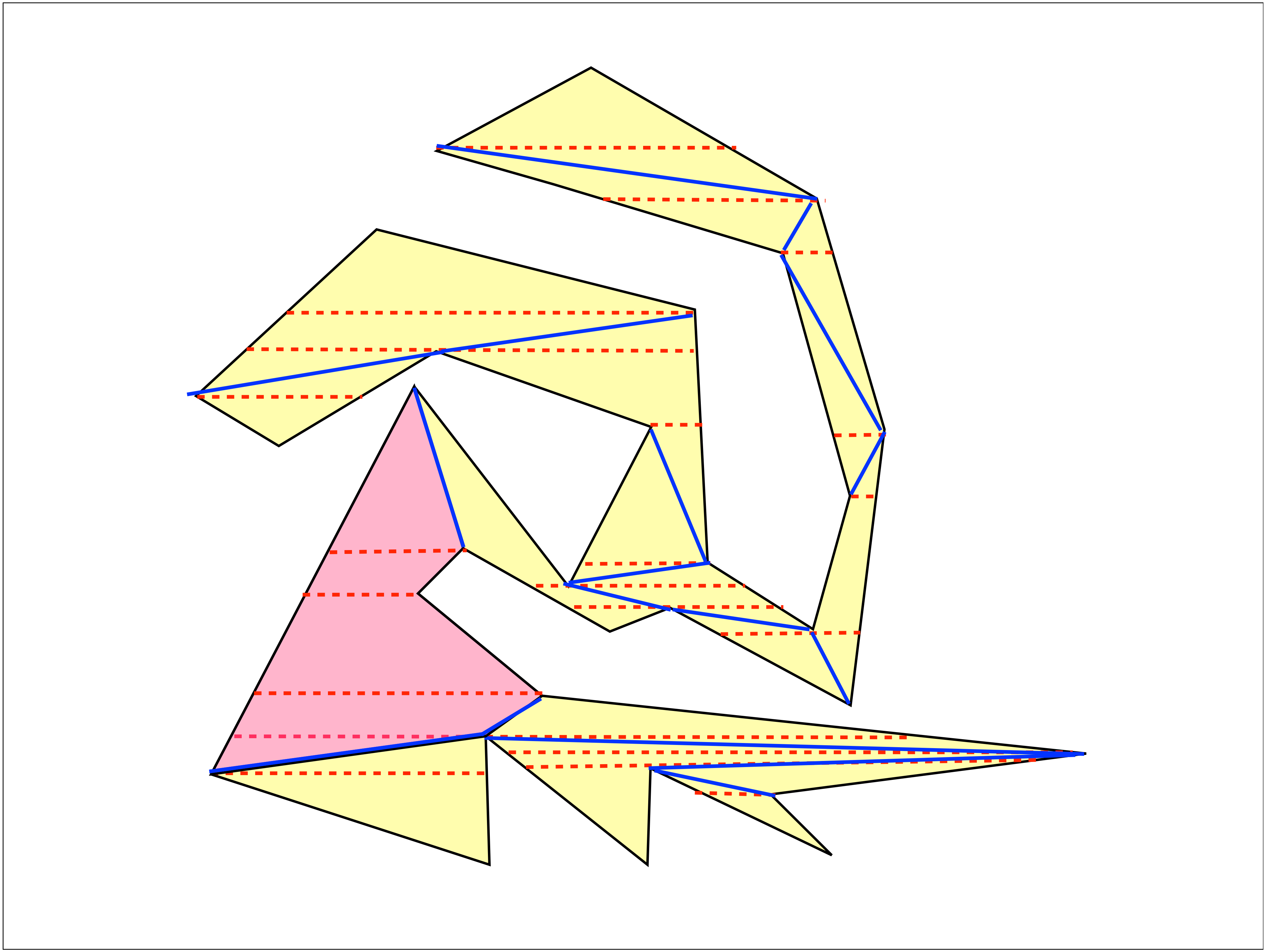


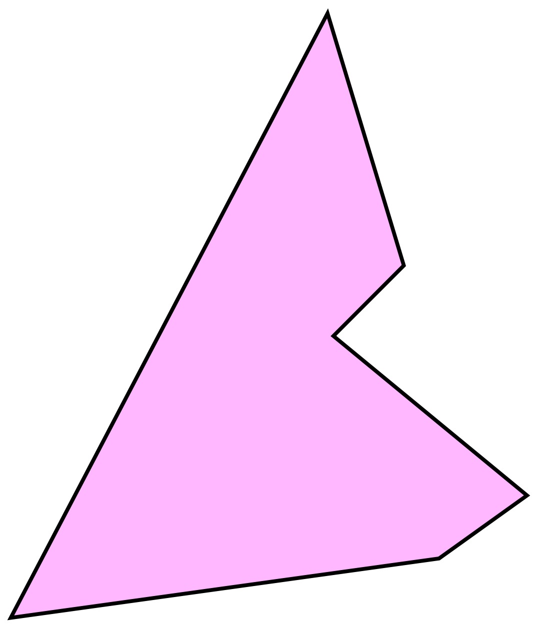


Polygon P is uni-monotone with respect to line L.



After adding the diagonals the polygon is partitioned into triangles and polygons that have the special property <u>uni-monotone</u>.

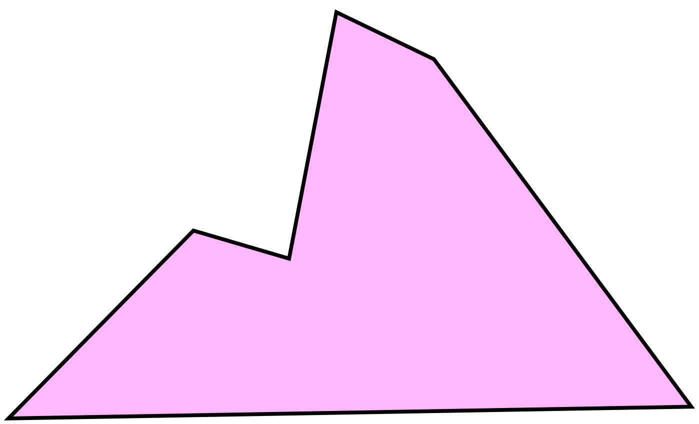




A uni-monotone polygon.

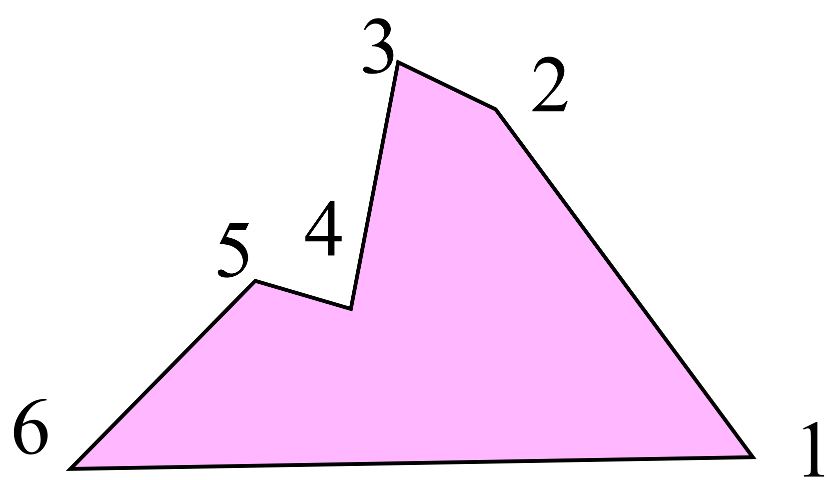
A uni-monotone polygon.

a.k.a "monotone mountain"



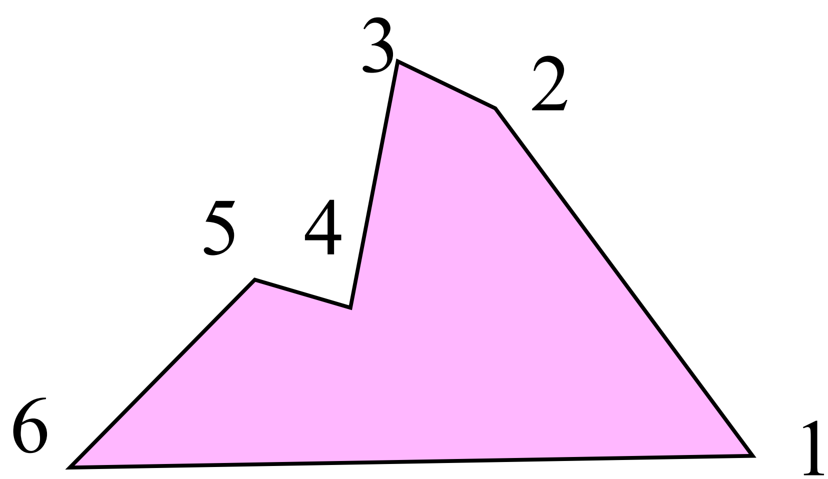
Triangulating a uni-monotone polygon

Label bottom right vertex of P 1 and then sequentially so that the bottom left vertex is labelled n.



Triangulating a uni-monotone polygon

Label bottom right vertex of *P* 1 and then sequentially the bottom left vertex is labelled n. We refer to vertex 1 and n as the <u>base vertices.</u>



Initialize a list of non-base convex vertices of P.

while | list | > 1
  for convex vertex *k* remove Δ(k-1)k(k+1)
  output diagonal (k-1)(k+1)
end while