Triangulating a Monotone Polygon

1. Problem Statement
   - triangulating y-monotone polygons
2. An Incremental Triangulation Algorithm
   - walking left and right boundary chains
   - pseudo code for the algorithm
3. Linear Time
   - cost analysis

MCS 481 Lecture 8
Computational Geometry
Jan Verschelde, 4 February 2019

Triangulating a Monotone Polygon

1. Problem Statement
   - triangulating y-monotone polygons

2. An Incremental Triangulation Algorithm
   - walking left and right boundary chains
   - pseudo code for the algorithm

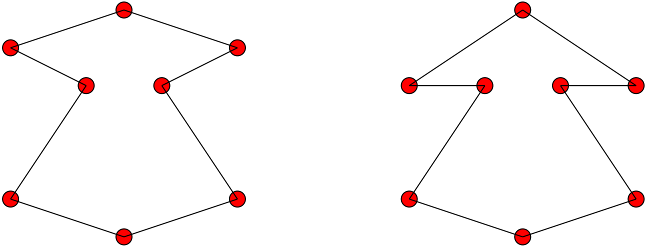
3. Linear Time
   - cost analysis

y-monotone polygons

Definition (y-monotone polygon)

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

A *strictly* y-monotone polygon has no horizontal edges.

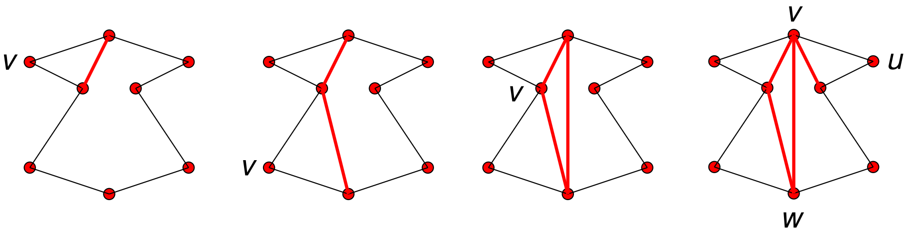


We will assume our polygons are strictly y-monotone.

adding diagonals

We can triangulate by recursively adding diagonals:

1. Take the highest leftmost vertex v.
2. Try first to connect the neighbors u and w.
3. If u and w cannot be connected, connect v to the vertex farthest from the edge (u, w) inside the triangle spanned by u, v, and w.



Do you see the next, last step?

asymptotic cost analysis

For a y-monotone polygon with n vertices,
computing the next diagonal can take n steps,
resulting in a $O(n)$ cost per step.

While we may optimistically

- hope that every diagonal cuts the polygon in two equal halves;
- in the worst case (the normal case in an asymptotic analysis), every diagonal may leave a polygon with $n - 1$ vertices,

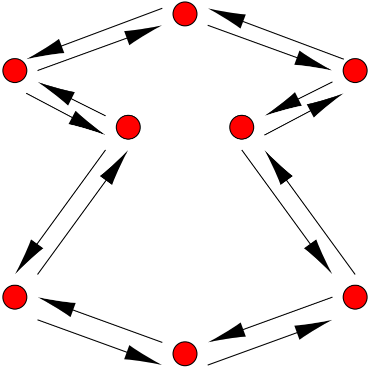
and thus a total cost of $O(n^2)$.

We will derive an $O(n)$ algorithm.

Sorting n points takes already $O(n \log(n))$ time,
so we assume the vertices of the y-monotone polygon P are sorted.

specification of the input and output

The polygon P is given as a doubly connected edge list $\mathcal{D}$.
The $\mathcal{D}$ below stores 8 vertex records, 16 half edge records, and 2 face records.



The triangulation is also stored in a doubly connected edge list.

Triangulating a Monotone Polygon

1. Problem Statement
   - triangulating y-monotone polygons

2. An Incremental Triangulation Algorithm
   - walking left and right boundary chains
   - pseudo code for the algorithm

3. Linear Time
   - cost analysis

walking left and right boundary chains

For a strictly y-monotone polygon P,

- we start at the highest leftmost vertex,
- take vertices from the left or right boundary chain, and
- construct diagonals whenever possible.

We need to ensure all added diagonals are in P.

Definition (convex and reflex vertices)

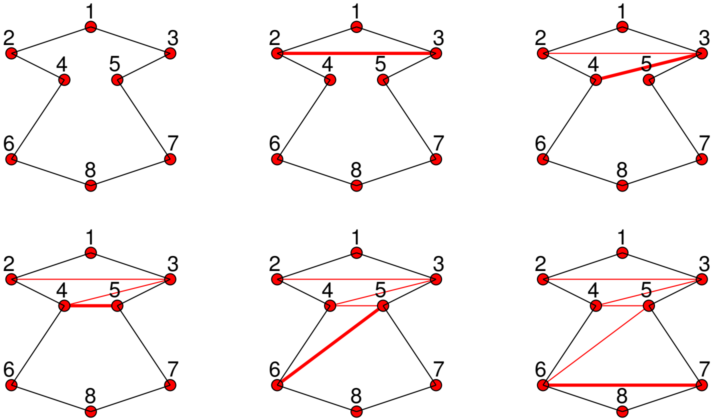
A vertex is *convex* if its inner angle is less than π.
A vertex that is not convex is *a reflex vertex*.

The highest leftmost vertex is a convex vertex.

If P is a convex polygon, then all vertices are convex and adding diagonals from the highest vertex v to all other vertices, those not adjacent to v, will give a triangulation in $O(n)$ time.

splitting off triangles

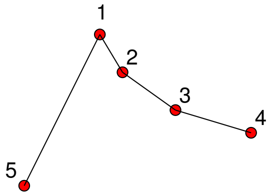
Invariant of the algorithm: the highest leftmost vertex is convex.



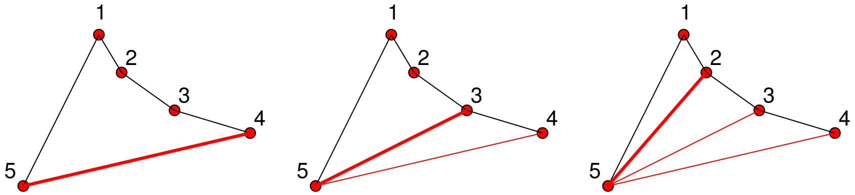
The two reflex vertices 4 and 5 cause no problems because the next vertex is on the opposite chain.

sequence of reflex vertices on the same chain

Consider the sequence of reflex vertices 2, 3, and 4:

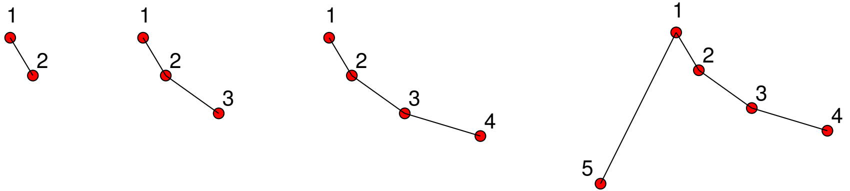


We can add diagonals only when we get to the 5-th vertex:

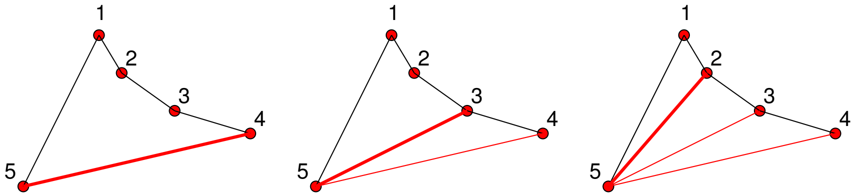


a stack stores reflex vertices on the same chain

Reflex vertices 2, 3, and 4 are stored on a stack:



At the 5-th vertex, we pop 4, 3, and 2, and add diagonals:



Triangulating a Monotone Polygon

1. Problem Statement
   - triangulating y-monotone polygons

2. An Incremental Triangulation Algorithm
   - walking left and right boundary chains
   - pseudo code for the algorithm

3. Linear Time
   - cost analysis

pseudo code – the initialization and loop

Algorithm TRIANGULATEMONOTONEPOLYGON(P)

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

1. Merge vertices of the left and right chains in $[u_1, u_2, \ldots, u_n]$, sorted on their y-coordinate, leftmost breaks ties, in descending order.
2. Initialize the stack S, push u_1 and u_2 onto S.
3. For j from 3 to $n - 1$ do
4. process vertex u_j.

The statement "process vertex u_j" is explained in the next two slides.

processing vertices on opposite chains

For j from 3 to $n - 1$ do
    if u_j and Top(S) are on opposite chains then
        for all $u \in S \setminus$ Bottom(S) do
            $u =$ pop(S)
            insert diagonal (u_j, u) into $\mathcal{D}$
        $u =$ pop(S)
        push(S, u_{j-1}); push(S, u_j)
    else ...

The popping of all vertices and the removal of Bottom(S) corresponds to triangles splitting off.

Exercise 1: Explain why the diagonals (u_j, u) are inside P. In your proof, take into account that P is y-monotone and the processing order of the vertices.

processing vertices on the same chain

else

$u_\ell = \text{pop}(S)$

$u = u_\ell$

while the diagonal $(u_j, u) \in P$ do

insert (u_j, u) into $\mathcal{D}$

$u = \text{pop}(S)$

push(S, u_ℓ); push(S, u_j);

Add diagonal from u_n to all $u \in S$ except for Top(S) and Bottom(S).

Exercise 2: Using your solution to Exercise 1 as a Lemma, prove the correctness of Algorithm TRIANGULATEMONOTONEPOLYGON.

Triangulating a Monotone Polygon

1. Problem Statement
   - triangulating y-monotone polygons

2. An Incremental Triangulation Algorithm
   - walking left and right boundary chains
   - pseudo code for the algorithm

3. Linear Time
   - cost analysis

cost analysis of the initialization

The initialization of Algorithm TRIANGULATEMONOTONEPOLYGON:

- Locating the highest leftmost vertex in a doubly connected edge list takes $O(n)$ time.

  1. Merge vertices of the left and right chains in $[u_1, u_2, \ldots, u_n]$,

  Merging two sorted lists takes $O(n)$,
  where n is the length of the result.

- Then, consider:

  2. Initialize the stack S, push u_1 and u_2 onto S.

  which runs in $O(1)$.

cost analysis of the loop

The loop

3. For j from 3 to $n - 1$ do

is executed $n - 3$ times. However, there are inner loops:

5. for all $u \in S \setminus \text{Bottom}(S)$ do

and

13. while the diagonal $(u_j, u) \in P$ do

For all vertices, one of the two inner loops is executed, leading to a potential $O(n^2)$ running time.

growth of the stack

How large can the stack get?

Let us count the push operations in the loop:

For j from 3 to $n-1$ do
    if u_j and Top(S) are on opposite chains then
        push(S, u_{j-1}); push(S, u_j)
    else
        push(S, u_ℓ); push(S, u_j);

As the loop is executed $n-3$ times, with two push operations per step, starting with 2 vertices after the initialization,
the stack size equals $2 + 2n - 6 = 2n - 4$.

The $2n-4$ is an upper bound on the number of pop operations in the inner loops. Therefore, the number of times the instructions in the inner loops are executed is also bounded by $2n-4$, which is $O(n)$.

linear time

We have proven the following theorem.

Theorem (time to triangulate a monotone polygon)

It takes $O(n)$ time to triangulate a stricly y-monotone polygon given as a doubly connected edge list of n vertices.

Exercise 3: Illustrate on a well chosen example the modifications to Algorithm TriangulateMonotonePolygon to handle y-monotone polygons with horizontal edges.
Show that your modified algorithm runs in linear time.

recommended assignments

We closed the third chapter in the textbook.

Consider the following activities, listed below.

1. Write the solutions to exercises 1, 2, and 3.
2. Consider the exercises 10,13,14 in the textbook.

Homework will be collected on Wednesday 6 February at noon, in class.