TRIANGULATING POLYGONS

Vera Sacristán

Computational Geometry
Facultat d’Informàtica de Barcelona
Universitat Politècnica de Catalunya
A **polygon triangulation** is the decomposition of a polygon into triangles. This is done by inserting internal diagonals.

An **internal diagonal** is any segment...

- connecting two vertices of the polygon and
- completely enclosed in the polygon.
A **polygon triangulation** is the decomposition of a polygon into triangles. This is done by inserting internal diagonals.

An **internal diagonal** is any segment...

- connecting two vertices of the polygon and
- completely enclosed in the polygon.
A **polygon triangulation** is the decomposition of a polygon into triangles. This is done by inserting internal diagonals.

An **internal diagonal** is any segment...

- connecting two vertices of the polygon and
- completely enclosed in the polygon.
A **polygon triangulation** is the decomposition of a polygon into triangles. This is done by inserting internal diagonals.

An **internal diagonal** is any segment...

- connecting two vertices of the polygon and
- completely enclosed in the polygon.
A **polygon triangulation** is the decomposition of a polygon into triangles. This is done by inserting internal diagonals.

An **internal diagonal** is any segment...

- connecting two vertices of the polygon and
- completely enclosed in the polygon.
A **polygon triangulation** is the decomposition of a polygon into triangles. This is done by inserting internal diagonals.

An **internal diagonal** is any segment...

- connecting two vertices of the polygon and
- completely enclosed in the polygon.
A **polygon triangulation** is the decomposition of a polygon into triangles. This is done by inserting internal diagonals.

An **internal diagonal** is any segment...

- connecting two vertices of the polygon and
- completely enclosed in the polygon.
A polygon triangulation is the decomposition of a polygon into triangles. This is done by inserting internal diagonals.

An internal diagonal is any segment...

- connecting two vertices of the polygon and
- completely enclosed in the polygon.

An application example:
A **polygon triangulation** is the decomposition of a polygon into triangles. This is done by inserting internal diagonals.

An **internal diagonal** is any segment…

- connecting two vertices of the polygon and
- completely enclosed in the polygon.

**An application example:**

![Diagram of a polygon and its triangulation]
A **polygon triangulation** is the decomposition of a polygon into triangles. This is done by inserting internal diagonals.

An **internal diagonal** is any segment...

- connecting two vertices of the polygon and
- completely enclosed in the polygon.

**An application example:**

![Diagram of a polygon triangulation example]
A **polygon triangulation** is the decomposition of a polygon into triangles. This is done by inserting internal diagonals.

An **internal diagonal** is any segment...

- connecting two vertices of the polygon and
- completely enclosed in the polygon.

**An application example:**
A **polygon triangulation** is the decomposition of a polygon into triangles. This is done by inserting internal diagonals.

An **internal diagonal** is any segment...

- connecting two vertices of the polygon and
- completely enclosed in the polygon.

**An application example:**
A *polygon triangulation* is the decomposition of a polygon into triangles. This is done by inserting internal diagonals.

An *internal diagonal* is any segment...

- connecting two vertices of the polygon and
- completely enclosed in the polygon.

An application example:
Index

1. Every polygon can be triangulated
2. Properties of polygon triangulations
3. Algorithms
   (a) Triangulating by inserting diagonals
   (b) Triangulating by subtracting ears
   (c) Triangulating convex polygons
   (d) Triangulating monotone polygons
   (e) Monotone partitioning
Every polygon admits a triangulation
Every polygon admits a triangulation

Lemma 1. Every polygon has at least one convex vertex (actually, at least three).
Every polygon admits a triangulation

Lemma 1. Every polygon has at least one convex vertex (actually, at least three).

The vertex $p$ with minimum $x$-coordinate (and, if there are more than one, the one with minimum $y$-coordinate) is convex.
Every polygon admits a triangulation

Lemma 1. Every polygon has at least one convex vertex (actually, at least three).

The vertex $p$ with minimum $x$-coordinate (and, if there are more than one, the one with minimum $y$-coordinate) is convex.
Every polygon admits a triangulation

Lemma 1. Every polygon has at least one convex vertex (actually, at least three).

The vertex $p$ with minimum $x$-coordinate (and, if there are more than one, the one with minimum $y$-coordinate) is convex.

So is the vertex $q$ with maximum $x$-coordinate (which is different of the one with minimum one).
Every polygon admits a triangulation

Lemma 1. Every polygon has at least one convex vertex (actually, at least three).

The vertex $p$ with minimum $x$-coordinate (and, if there are more than one, the one with minimum $y$-coordinate) is convex.

So is the vertex $q$ with maximum $x$-coordinate (which is different of the one with minimum one).

Finally, there is at least one vertex which is extreme in the direction orthogonal to $pq$ and does not coincide with any of the above. This third vertex $r$ is necessarily convex.
Every polygon admits a triangulation

Lemma 1. Every polygon has at least one convex vertex (actually, at least three).

Lemma 2. Every $n$-gon with $n \geq 4$ has at least one internal diagonal.
Every polygon admits a triangulation

Lemma 1. Every polygon has at least one convex vertex (actually, at least three).

Lemma 2. Every $n$-gon with $n \geq 4$ has at least one internal diagonal.

Let $v_i$ be a convex vertex.
Then, either $v_{i-1}v_{i+1}$ is an internal diagonal...
Every polygon admits a triangulation

**Lemma 1.** Every polygon has at least one convex vertex (actually, at least three).

**Lemma 2.** Every \( n \)-gon with \( n \geq 4 \) has at least one internal diagonal.

Let \( v_i \) be a convex vertex.
Then, either \( v_{i-1}v_{i+1} \) is an internal diagonal...
or there exists a vertex of the polygon lying in the triangle \( v_{i-1}v_iv_{i+1} \).
Every polygon admits a triangulation

Lemma 1. Every polygon has at least one convex vertex (actually, at least three).

Lemma 2. Every $n$-gon with $n \geq 4$ has at least one internal diagonal.

Let $v_i$ be a convex vertex.
Then, either $v_{i-1}v_{i+1}$ is an internal diagonal...
or there exists a vertex of the polygon lying in the triangle $v_{i-1}v_iv_{i+1}$.

In this case, among all the vertices lying in the triangle, let $v_j$ be the farthest one from the segment $v_{i-1}v_{i+1}$. Then $v_iv_j$ is an internal diagonal (it can not be intersected by any edge of the polygon).
Every polygon admits a triangulation

**Lemma 1.** Every polygon has at least one convex vertex (actually, at least three).

**Lemma 2.** Every $n$-gon with $n \geq 4$ has at least one internal diagonal.

Let $v_i$ be a convex vertex.

Then, either $v_{i-1}v_{i+1}$ is an internal diagonal... or there exists a vertex of the polygon lying in the triangle $v_{i-1}v_iv_{i+1}$.

In this case, among all the vertices lying in the triangle, let $v_j$ be the farthest one from the segment $v_{i-1}v_{i+1}$. Then $v_iv_j$ is an internal diagonal (it can not be intersected by any edge of the polygon).
Every polygon admits a triangulation

**Lemma 1.** Every polygon has at least one convex vertex (actually, at least three).

**Lemma 2.** Every $n$-gon with $n \geq 4$ has at least one internal diagonal.

**Corollary.** Every polygon can be triangulated. (By induction.)
Properties of the triangulations of polygons
Properties of the triangulations of polygons

Let $P$ be a simple $n$-gon.

**Property 1.** Every triangulation of $P$ has $n - 3$ diagonals.
Properties of the triangulations of polygons

Let $P$ be a simple $n$-gon.

**Property 1.** Every triangulation of $P$ has $n - 3$ diagonals.

Proof by induction.

Base case: When $n = 3$, the number of diagonals is $d = 0 = n - 3$.

Inductive step: Consider a diagonal of a triangulation $T$ of $P$, decomposing $P$ into two subpolygons: a $(k + 1)$-gon $P_1$ and an $(n - k + 1)$-gon $P_2$. By inductive hypothesis, the number of diagonals of the triangulations induced by $T$ in $P_1$ and $P_2$ are:

\[
d_1 = k + 1 - 3, \\
d_2 = n - k + 1 - 3,
\]

therefore, $d = d_1 + d_2 + 1 = k + 1 - 3 + n - k + 1 - 3 + 1 = n - 3$. 

\[\text{Diagram: } P_1 \text{ and } P_2\]
Properties of the triangulations of polygons

Let $P$ be a simple $n$-gon.

**Property 1.** Every triangulation of $P$ has $n - 3$ diagonals.

**Property 2.** Every triangulation of $P$ has $n - 2$ triangles.
Properties of the triangulations of polygons

Let $P$ be a simple $n$-gon.

**Property 1.** Every triangulation of $P$ has $n - 3$ diagonals.

**Property 2.** Every triangulation of $P$ has $n - 2$ triangles.

Again, the proof is by induction.

Base case: When $n = 3$, the number of triangles is $t = 1 = n - 2$.

Inductive step: With the same conditions of the previous proof,

\[
\begin{align*}
t_1 &= k + 1 - 2, \\
t_2 &= n - k + 1 - 2,
\end{align*}
\]

hence,

\[
t = t_1 + t_2 = k + 1 - 2 + n - k + 1 - 2 = n - 2.
\]
Properties of the triangulations of polygons

Let $P$ be a simple $n$-gon.

**Property 1.** Every triangulation of $P$ has $n - 3$ diagonals.

**Property 2.** Every triangulation of $P$ has $n - 2$ triangles.

**Property 3.** The dual graph of any triangulation of $P$ is a tree.
Properties of the triangulations of polygons

Let $P$ be a simple $n$-gon.

**Property 1.** Every triangulation of $P$ has $n - 3$ diagonals.

**Property 2.** Every triangulation of $P$ has $n - 2$ triangles.

**Property 3.** The dual graph of any triangulation of $P$ is a tree.

Given a triangulation of $P$, its dual graph has one vertex for each triangle, and one edge connecting two vertices whenever their corresponding triangles are adjacent. We want to prove that this graph is connected and acyclic.

The graph is trivially connected.

About the acyclicity: Notice that each edge of the dual graph "separates" the two endpoints of the internal diagonal of $P$ shared by the two adjacent triangles. If the graph had a cycle, it would enclose the endpoint(s) of the diagonals intersected by the cycle and, therefore, it would enclose points belonging to the boundary of the polygon, contradicting the hypothesis that $P$ is simple and without holes.
Properties of the triangulations of polygons

Let $P$ be a simple $n$-gon.

**Property 1.** Every triangulation of $P$ has $n - 3$ diagonals.

**Property 2.** Every triangulation of $P$ has $n - 2$ triangles.

**Property 3.** The dual graph of any triangulation of $P$ is a tree.

**Corollary.** Every $n$-gon with $n \geq 4$ has at least two non-adjacent ears.
ALGORITHMS FOR POLYGON TRIANGULATION
Tringulating a polygon by subtracting ears
Tringulating a polygon by subtracting ears

**Input:** \( v_1, \ldots, v_n \), sorted list of the vertices of a simple polygon \( P \).

**Output:** List of internal diagonals of \( P \), \( v_i v_j \), determining a triangulation of \( P \).
**TRIANGULATING POLYGONS**

**Tringulating a polygon by subtracting ears**

**Input:** $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon $P$.

**Output:** List of internal diagonals of $P$, $v_i v_j$, determining a triangulation of $P$.

**Procedure**

1. Sequentially explore the vertices until you find an ear
2. Cut it out
3. Proceed recursively
Tringulating a polygon by subtracting ears

Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon $P$.

Output: List of internal diagonals of $P$, $v_i v_j$, determining a triangulation of $P$.

Procedure
1. Sequentially explore the vertices until you find an ear
2. Cut it out
3. Proceed recursively

Running time
Trigulating a polygon by subtracting ears

**Input:** \( v_1, \ldots, v_n \), sorted list of the vertices of a simple polygon \( P \).

**Output:** List of internal diagonals of \( P \), \( v_i v_j \), determining a triangulation of \( P \).

**Procedure**
1. Sequentially explore the vertices until you find an ear
2. Cut it out
3. Proceed recursively

**Running time**
Detecting whether a vertex is convex: \( O(1) \).
Trigulating a polygon by subtracting ears

Input: \( v_1, \ldots, v_n \), sorted list of the vertices of a simple polygon \( P \).

Output: List of internal diagonals of \( P \), \( v_i v_j \), determining a triangulation of \( P \).

Procedure
1. Sequentially explore the vertices until you find an ear
2. Cut it out
3. Proceed recursively

Running time
Detecting whether a vertex is convex: \( O(1) \).
Detecting whether a convex vertex is an ear: \( O(n) \).
Trigulating a polygon by subtracting ears

Input: \( v_1, \ldots, v_n \), sorted list of the vertices of a simple polygon \( P \).

Output: List of internal diagonals of \( P \), \( v_iv_j \), determining a triangulation of \( P \).

Procedure

1. Sequentially explore the vertices until you find an ear
2. Cut it out
3. Proceed recursively

Running time

Detecting whether a vertex is convex: \( O(1) \).
Detecting whether a convex vertex is an ear: \( O(n) \).
Finding an ear: \( O(n^2) \).
Triangulating a polygon by subtracting ears

Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon $P$.

Output: List of internal diagonals of $P$, $v_i v_j$, determining a triangulation of $P$.

Procedure
1. Sequentially explore the vertices until you find an ear
2. Cut it out
3. Proceed recursively

Running time
Detecting whether a vertex is convex: $O(1)$.
Detecting whether a convex vertex is an ear: $O(n)$.
Finding an ear: $O(n^2)$.

Overall running time:

$$T(n) = O(n^2) + O((n - 1)^2) + O((n - 2)^2) + \cdots + O(1) = \Theta(n^3).$$
Tringulating a polygon by subtracting ears

**Input:** $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon $P$.

**Output:** List of internal diagonals of $P$, $v_i v_j$, determining a triangulation of $P$.

**Improved procedure**

**Initialization**
1. Detect all convex vertices
2. Detect all ears

**Next step**
1. Crop an ear
2. Update the information of the convex vertices
3. Update the information of the ears
Trigulating a polygon by subtracting ears

**Input:** $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon $P$.

**Output:** List of internal diagonals of $P$, $v_i v_j$, determining a triangulation of $P$.

**Improved procedure**

**Initialization**
1. Detect all convex vertices
2. Detect all ears

**Running time**

<table>
<thead>
<tr>
<th></th>
<th>$O(n)$</th>
<th>$O(n^2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Initialization</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Only once</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Next step**
1. Crop an ear
2. Update the information of the convex vertices
3. Update the information of the ears

<table>
<thead>
<tr>
<th></th>
<th>$O(1)$</th>
<th>$O(n)$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Next step</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$O(n)$ times</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Tringulating a polygon by subtracting ears

**Input:** $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon $P$.

**Output:** List of internal diagonals of $P$, $v_i v_j$, determining a triangulation of $P$.

**Improved procedure**

**Initialization**

1. Detect all convex vertices
2. Detect all ears

**Next step**

1. Crop an ear
2. Update the information of the convex vertices
3. Update the information of the ears

**Running time**

<table>
<thead>
<tr>
<th></th>
<th>Running time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialization</td>
<td>$O(n)$</td>
</tr>
<tr>
<td></td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>Next step</td>
<td></td>
</tr>
<tr>
<td>1. Crop an ear</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>2. Update the information of the convex vertices</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>3. Update the information of the ears</td>
<td>$O(n)$</td>
</tr>
</tbody>
</table>

**Running time:** $O(n^2)$
TRIANGULATING POLYGONS

Tringulating a polygon by inserting diagonals

Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon $P$.

Output: List of internal diagonals of $P$, $v_i v_j$, determining a triangulation of $P$. 
Tringulating a polygon by inserting diagonals

**Input:** $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon $P$.

**Output:** List of internal diagonals of $P$, $v_i v_j$, determining a triangulation of $P$.

**Procedure:**
1. Find an internal diagonal
2. Decompose the polygon into two subpolygons
3. Proceed recursively
Tringulating a polygon by inserting diagonals

**Input:** $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon $P$.

**Output:** List of internal diagonals of $P$, $v_iv_j$, determining a triangulation of $P$.

**Procedure:**
1. Find an internal diagonal
2. Decompose the polygon into two subpolygons
3. Proceed recursively

**Test.** How to decide whether a given segment $v_iv_j$ is an internal diagonal?
Tringulating a polygon by inserting diagonals

Input: \( v_1, \ldots, v_n \), sorted list of the vertices of a simple polygon \( P \).

Output: List of internal diagonals of \( P \), \( v_i v_j \), determining a triangulation of \( P \).

Procedure:
1. Find an internal diagonal
2. Decompose the polygon into two subpolygons
3. Proceed recursively

Test. How to decide whether a given segment \( v_i v_j \) is an internal diagonal?
   Is it a diagonal?
Tringulating a polygon by inserting diagonals

**Input:** \( v_1, \ldots, v_n \), sorted list of the vertices of a simple polygon \( P \).

**Output:** List of internal diagonals of \( P \), \( v_i v_j \), determining a triangulation of \( P \).

**Procedure:**
1. Find an internal diagonal
2. Decompose the polygon into two subpolygons
3. Proceed recursively

**Test.** How to decide whether a given segment \( v_i v_j \) is an internal diagonal?

Is it a diagonal?

Check \( v_i v_j \) against all segments \( v_k v_{k+1} \) for intersection.
Tringulating a polygon by inserting diagonals

**Input:** $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon $P$.

**Output:** List of internal diagonals of $P$, $v_i v_j$, determining a triangulation of $P$.

**Procedure:**
1. Find an internal diagonal
2. Decompose the polygon into two subpolygons
3. Proceed recursively

**Test.** How to decide whether a given segment $v_i v_j$ is an internal diagonal?

- Is it a diagonal?
- Check $v_i v_j$ against all segments $v_k v_{k+1}$ for intersection.
- Is it internal?
**TRIANGULATING POLYGONS**

**Tringulating a polygon by inserting diagonals**

**Input:** $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon $P$.

**Output:** List of internal diagonals of $P$, $v_iv_j$, determining a triangulation of $P$.

**Procedure:**
1. Find an internal diagonal
2. Decompose the polygon into two subpolygons
3. Proceed recursively

**Test.** How to decide whether a given segment $v_iv_j$ is an internal diagonal?

Is it a diagonal?

Check $v_iv_j$ against all segments $v_kv_{k+1}$ for intersection.

Is it internal?

If $v_i$ is convex, the oriented line $\overrightarrow{v_iv_j}$ should leave $v_{i-1}$ to its left and $v_{i+1}$ to its right.
Tringulating a polygon by inserting diagonals

**Input:** $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon $P$.

**Output:** List of internal diagonals of $P$, $v_iv_j$, determining a triangulation of $P$.

**Procedure:**
1. Find an internal diagonal
2. Decompose the polygon into two subpolygons
3. Proceed recursively

**Test.** How to decide whether a given segment $v_iv_j$ is an internal diagonal?

Is it a diagonal?
Check $v_iv_j$ against all segments $v_kv_{k+1}$ for intersection.

Is it internal?
If $v_i$ is convex, the oriented line $\overrightarrow{v_iv_j}$ should leave $v_{i-1}$ to its left and $v_{i+1}$ to its right.

If $v_i$ is reflex, the oriented line $\overrightarrow{v_iv_j}$ should not leave $v_{i-1}$ to its right and $v_{i+1}$ to its left.
**TRIANGULATING POLYGONS**

**Tringulating a polygon by inserting diagonals**

**Input:** $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon $P$.

**Output:** List of internal diagonals of $P$, $v_iv_j$, determining a triangulation of $P$.

**Procedure:**
1. Find an internal diagonal
2. Decompose the polygon into two subpolygons
3. Proceed recursively

**Test.** How to decide whether a given segment $v_iv_j$ is an internal diagonal?

- Is it a diagonal?
  - Check $v_iv_j$ against all segments $v_kv_{k+1}$ for intersection.

- Is it internal?
  - If $v_i$ is convex, the oriented line $\overrightarrow{v_iv_j}$ should leave $v_{i-1}$ to its left and $v_{i+1}$ to its right.
  - If $v_i$ is reflex, the oriented line $\overrightarrow{v_iv_j}$ should not leave $v_{i-1}$ to its right and $v_{i+1}$ to its left.
**TRIANGULATING POLYGONS**

**Tringulating a polygon by inserting diagonals**

**Input:** $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon $P$.

**Output:** List of internal diagonals of $P$, $v_i v_j$, determining a triangulation of $P$.

**Procedure:**
1. Find an internal diagonal
2. Decompose the polygon into two subpolygons
3. Proceed recursively

**Test.** How to decide whether a given segment $v_i v_j$ is an internal diagonal?

Is it a diagonal?

Check $v_i v_j$ against all segments $v_k v_{k+1}$ for intersection.

Is it internal?

If $v_i$ is convex, the oriented line $\overrightarrow{v_i v_j}$ should leave $v_{i-1}$ to its left and $v_{i+1}$ to its right.

If $v_i$ is reflex, the oriented line $\overrightarrow{v_i v_j}$ should not leave $v_{i-1}$ to its right and $v_{i+1}$ to its left.

**Running time** $O(n)$
TRIANGULATING POLYGONS

**Tringulating a polygon by inserting diagonals**

**Input:** \( v_1, \ldots, v_n \), sorted list of the vertices of a simple polygon \( P \).

**Output:** List of internal diagonals of \( P \), \( v_iv_j \), determining a triangulation of \( P \).

**Procedure:**
1. Find an internal diagonal
2. Decompose the polygon into two subpolygons
3. Proceed recursively

**Test.** How to decide whether a given segment \( v_iv_j \) is an internal diagonal? \( O(n) \)

**Search.** How to find an internal diagonal?
Tringulating a polygon by inserting diagonals

**Input:** $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon $P$.

**Output:** List of internal diagonals of $P$, $v_iv_j$, determining a triangulation of $P$.

**Procedure:**
1. Find an internal diagonal
2. Decompose the polygon into two subpolygons
3. Proceed recursively

**Running time** $O(n)$

**Test.** How to decide whether a given segment $v_iv_j$ is an internal diagonal?

**Search.** How to find an internal diagonal?

*Brute-force solution:*
TRIANGULATING POLYGONS

Tringulating a polygon by inserting diagonals

Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon $P$.

Output: List of internal diagonals of $P$, $v_i v_j$, determining a triangulation of $P$.

Procedure:
1. Find an internal diagonal
2. Decompose the polygon into two subpolygons
3. Proceed recursively

Test. How to decide whether a given segment $v_i v_j$ is an internal diagonal? $O(n)$

Search. How to find an internal diagonal?

Brute-force solution:
Apply the test to each candidate segment.
Triangulating a polygon by inserting diagonals

**Input:** $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon $P$.

**Output:** List of internal diagonals of $P$, $v_iv_j$, determining a triangulation of $P$.

**Procedure:**
1. Find an internal diagonal
2. Decompose the polygon into two subpolygons
3. Proceed recursively

**Running time**

**Test.** How to decide whether a given segment $v_iv_j$ is an internal diagonal? $O(n)$

**Search.** How to find an internal diagonal?

*Brute-force solution:*
Apply the test to each candidate segment. $O(n^3)$
Tringulating a polygon by inserting diagonals

**Input:** $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon $P$.

**Output:** List of internal diagonals of $P$, $v_iv_j$, determining a triangulation of $P$.

**Procedure:**
1. Find an internal diagonal
2. Decompose the polygon into two subpolygons
3. Proceed recursively

**Test.** How to decide whether a given segment $v_iv_j$ is an internal diagonal? $O(n)$

**Search.** How to find an internal diagonal?

**Brute-force solution:**
Apply the test to each candidate segment. $O(n^3)$

Testing each candidate takes $O(n)$ time, and there are $\binom{n}{2}$ of them.
Tringulating a polygon by inserting diagonals

**Input:** $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon $P$.

**Output:** List of internal diagonals of $P$, $v_iv_j$, determining a triangulation of $P$.

**Procedure:**
1. Find an internal diagonal
2. Decompose the polygon into two subpolygons
3. Proceed recursively

**Running time**

**Test.** How to decide whether a given segment $v_iv_j$ is an internal diagonal? $O(n)$

**Search.** How to find an internal diagonal?

*Brute-force solution:*

Apply the test to each candidate segment.

*Applying previous results:*

1. Find a convex vertex, $v_i$.
2. Detect whether $v_i-1v_i+1$ is an internal diagonal.
3. If so, report it.
   
   Else, find the farthest $v_k$ from the segment $v_i-1v_i+1$, lying in the triangle $v_i-1v_iv_i+1$. 

$O(n^3)$
Triangulating a polygon by inserting diagonals

**Input:** $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon $P$.

**Output:** List of internal diagonals of $P$, $v_iv_j$, determining a triangulation of $P$.

**Procedure:**
1. Find an internal diagonal
2. Decompose the polygon into two subpolygons
3. Proceed recursively

**Test.** How to decide whether a given segment $v_iv_j$ is an internal diagonal? $O(n)$

**Search.** How to find an internal diagonal?

*Brute-force solution:*

Apply the test to each candidate segment.

*Applying previous results:*

1. Find a convex vertex, $v_i$.
2. Detect whether $v_{i-1}v_{i+1}$ is an internal diagonal.
3. If so, report it.
   Else, find the farthest $v_k$ from the segment $v_{i-1}v_{i+1}$, lying in the triangle $v_{i-1}v_i v_{i+1}$.

**Running time**

$O(n^3)$
Tringulating a polygon by inserting diagonals

**Input:** $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon $P$.

**Output:** List of internal diagonals of $P$, $v_iv_j$, determining a triangulation of $P$.

**Procedure:**
1. Find an internal diagonal
2. Decompose the polygon into two subpolygons
3. Proceed recursively

**Test.** How to decide whether a given segment $v_iv_j$ is an internal diagonal? $O(n)$

**Search.** How to find an internal diagonal? $O(n)$
TRIANGULATING POLYGONS

Tringulating a polygon by inserting diagonals

Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon $P$.

Output: List of internal diagonals of $P$, $v_iv_j$, determining a triangulation of $P$.

Procedure:
1. Find an internal diagonal
2. Decompose the polygon into two subpolygons
3. Proceed recursively

Test. How to decide whether a given segment $v_iv_j$ is an internal diagonal? $O(n)$

Search. How to find an internal diagonal? $O(n)$

Partition. How to partition the polygon into two subpolygons?
Tringulating a polygon by inserting diagonals

Input: \(v_1, \ldots, v_n\), sorted list of the vertices of a simple polygon \(P\).

Output: List of internal diagonals of \(P\), \(v_iv_j\), determining a triangulation of \(P\).

Procedure:
1. Find an internal diagonal
2. Decompose the polygon into two subpolygons
3. Proceed recursively

Running time

Test. How to decide whether a given segment \(v_iv_j\) is an internal diagonal? \(O(n)\)

Search. How to find an internal diagonal? \(O(n)\)

Partition. How to partition the polygon into two subpolygons?

   From the diagonal found, create the sorted list of the vertices of the two subpolygons.
Tringulating a polygon by inserting diagonals

Input: \( v_1, \ldots, v_n \), sorted list of the vertices of a simple polygon \( P \).

Output: List of internal diagonals of \( P, v_i v_j \), determining a triangulation of \( P \).

Procedure:
1. Find an internal diagonal
2. Decompose the polygon into two subpolygons
3. Proceed recursively

Running time

Test. How to decide whether a given segment \( v_i v_j \) is an internal diagonal? \( O(n) \)

Search. How to find an internal diagonal? \( O(n) \)

Partition. How to partition the polygon into two subpolygons?

From the diagonal found, create the sorted list of the vertices of the two subpolygons.
**Triangulating a polygon by inserting diagonals**

**Input:** $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon $P$.

**Output:** List of internal diagonals of $P$, $v_i v_j$, determining a triangulation of $P$.

**Procedure:**
1. Find an internal diagonal
2. Decompose the polygon into two subpolygons
3. Proceed recursively

**Test.** How to decide whether a given segment $v_i v_j$ is an internal diagonal? $O(n)$

**Search.** How to find an internal diagonal? $O(n)$

**Partition.** How to partition the polygon into two subpolygons? $O(n)$

**Total running time of the algorithm:** $O(n^2)$

It finds $n - 3$ diagonals and each one is found in $O(n)$ time.
Is it possible to triangulate a polygon more efficiently?
Is it possible to triangulate a polygon more efficiently?

Triangulating a convex polygon
Is it possible to triangulate a polygon more efficiently?

Triangulating a convex polygon

Trivially done in $O(n)$ time.
Is it possible to triangulate a polygon more efficiently?

Triangulating a convex polygon

Trivially done in $O(n)$ time.
Is it possible to triangulate a polygon more efficiently?

Triangulating a convex polygon
Trivially done in $O(n)$ time.
Is it possible to triangulate a polygon more efficiently?

Triangulating a convex polygon
Trivially done in $O(n)$ time.
Is it possible to triangulate a polygon more efficiently?

Triangulating a convex polygon
Trivially done in $O(n)$ time.
Is it possible to triangulate a polygon more efficiently?

**Triangulating a convex polygon**

Trivially done in $O(n)$ time.
Is it possible to triangulate a polygon more efficiently?

Triangulating a convex polygon

Trivially done in $O(n)$ time.
Is it possible to triangulate a polygon more efficiently?

Triangulating a convex polygon
Trivially done in $O(n)$ time.
Is it possible to triangulate a polygon more efficiently?

Triangulating a convex polygon
Trivially done in $O(n)$ time.
Is it possible to triangulate a polygon more efficiently?

Triangulating a convex polygon

Trivially done in $O(n)$ time.
Is it possible to triangulate a polygon more efficiently?

Triangulating a convex polygon

Trivially done in $O(n)$ time.
Is it possible to triangulate a polygon more efficiently?

Triangulating a convex polygon
Trivially done in $O(n)$ time.
Is it possible to triangulate a polygon more efficiently?

Triangulating a convex polygon

Trivially done in $O(n)$ time.
Is it possible to triangulate a polygon more efficiently?

Triangulating a convex polygon
Trivially done in $O(n)$ time.
Is it possible to triangulate a polygon more efficiently?

Triangulating a convex polygon
Trivially done in $O(n)$ time.

Triangulating a star-shaped polygon
Can be done in $O(n)$ time. Posed as problem.
Is it possible to triangulate a polygon more efficiently?

**Triangulating a convex polygon**
Trivially done in $O(n)$ time.

**Triangulating a star-shaped polygon**
Can be done in $O(n)$ time. Posed as problem.

**Triangulating a monotone polygon**
It can also be done in $O(n)$ time. In the following we will see how.
Monotone polygon

A polygon $P$ is called **monotone** with respect to a direction $r$ if, for every line $r'$ orthogonal to $r$, the intersection $P \cap r'$ is connected (i.e., it is a segment, a point or the empty set).
Monotone polygon

A polygon $P$ is called monotone with respect to a direction $r$ if, for every line $r'$ orthogonal to $r$, the intersection $P \cap r'$ is connected (i.e., it is a segment, a point or the empty set).
Monotone polygon

A polygon $P$ is called monotone with respect to a direction $r$ if, for every line $r'$ orthogonal to $r$, the intersection $P \cap r'$ is connected (i.e., it is a segment, a point or the empty set).
Monotone polygon

A polygon $P$ is called monotone with respect to a direction $r$ if, for every line $r'$ orthogonal to $r$, the intersection $P \cap r'$ is connected (i.e., it is a segment, a point or the empty set).
Monotone polygon

A polygon $P$ is called **monotone** with respect to a direction $r$ if, for every line $r'$ orthogonal to $r$, the intersection $P \cap r'$ is connected (i.e., it is a segment, a point or the empty set).

Local characterization

A polygon is $y$-monotone if and only if it does not have any cusp.
Monotone polygon

A polygon $P$ is called monotone with respect to a direction $r$ if, for every line $r'$ orthogonal to $r$, the intersection $P \cap r'$ is connected (i.e., it is a segment, a point or the empty set).

Local characterization

A polygon is $y$-monotone if and only if it does not have any cusp.
Monotone polygon

A polygon $P$ is called monotone with respect to a direction $r$ if, for every line $r'$ orthogonal to $r$, the intersection $P \cap r'$ is connected (i.e., it is a segment, a point or the empty set).

Local characterization

A polygon is $y$-monotone if and only if it does not have any cusp.

A cusp is a reflex vertex $v$ of the polygon such that its two incident edges both lie to the same side of the horizontal line through $v$. 
Monotone polygon

A polygon $P$ is called **monotone** with respect to a direction $r$ if, for every line $r'$ orthogonal to $r$, the intersection $P \cap r'$ is connected (i.e., it is a segment, a point or the empty set).

Local characterization

A polygon is $y$-monotone if and only if it does not have any cusp.

A **cusp** is a reflex vertex $v$ of the polygon such that its two incident edges both lie to the same side of the horizontal line through $v$.

**Proof:**

If the polygon has a local maximum cusp $v$, an infinitesimal downwards translation of the horizontal line through $v$ would intersect the polygon in at least two connected components.
Monotone polygon

A polygon $P$ is called **monotone** with respect to a direction $r$ if, for every line $r'$ orthogonal to $r$, the intersection $P \cap r'$ is connected (i.e., it is a segment, a point or the empty set).

Local characterization

A polygon is $y$-monotone if and only if it does not have any cusp.

A **cusp** is a reflex vertex $v$ of the polygon such that its two incident edges both lie to the same side of the horizontal line through $v$.

Proof:

If the polygon has a local maximum cusp $v$, an infinitesimal downwards translation of the horizontal line through $v$ would intersect the polygon in at least two connected components.

If the polygon is not monotone, let $r$ be a horizontal line intersecting the polygon in two or more connected components. Consider two consecutive components, with facing endpoints $p$ and $q$ as in the figure. The polygon boundary needs to connect $p$ and $q$. No matter whether it goes through $r^+$ or $r^-$, it will have a cusp.
Monotone polygon

A polygon $P$ is called **monotone** with respect to a direction $r$ if, for every line $r'$ orthogonal to $r$, the intersection $P \cap r'$ is connected (i.e., it is a segment, a point or the empty set).

**Local characterization**

A polygon is $y$-monotone if and only if it does not have any cusp.

A **cusp** is a reflex vertex $v$ of the polygon such that its two incident edges both lie to the same side of the horizontal line through $v$.

**Proof:**

If the polygon has a local maximum cusp $v$, an infinitesimal downwards translation of the horizontal line through $v$ would intersect the polygon in at least two connected components.

If the polygon is not monotone, let $r$ be a horizontal line intersecting the polygon in two or more connected components. Consider two consecutive components, with facing endpoints $p$ and $q$ as in the figure. The polygon boundary needs to connect $p$ and $q$. No matter whether it goes through $r^+$ or $r^-$, it will have a cusp.
Monotone polygon

A polygon $P$ is called monotonically monotone with respect to a direction $r$ if, for every line $r'$ orthogonal to $r$, the intersection $P \cap r'$ is connected (i.e., it is a segment, a point or the empty set).

Local characterization

A polygon is $y$-monotone if and only if it does not have any cusp.

A cusp is a reflex vertex $v$ of the polygon such that its two incident edges both lie to the same side of the horizontal line through $v$.

Proof:

If the polygon has a local maximum cusp $v$, an infinitesimal downwards translation of the horizontal line through $v$ would intersect the polygon in at least two connected components.

If the polygon is not monotone, let $r$ be a horizontal line intersecting the polygon in two or more connected components. Consider two consecutive components, with facing endpoints $p$ and $q$ as in the figure. The polygon boundary needs to connect $p$ and $q$. No matter whether it goes through $r^+$ or $r^-$, it will have a cusp.
Monotone polygon

A polygon $P$ is called monotone with respect to a direction $r$ if, for every line $r'$ orthogonal to $r$, the intersection $P \cap r'$ is connected (i.e., it is a segment, a point or the empty set).

Local characterization

A polygon is $y$-monotone if and only if it does not have any cusp.

A cusp is a reflex vertex $v$ of the polygon such that its two incident edges both lie to the same side of the horizontal line through $v$. 
Monotone polygon

A polygon $P$ is called monotone with respect to a direction $r$ if, for every line $r'$ orthogonal to $r$, the intersection $P \cap r'$ is connected (i.e., it is a segment, a point or the empty set).

Local characterization

A polygon is $y$-monotone if and only if it does not have any cusp.

A cusp is a reflex vertex $v$ of the polygon such that its two incident edges both lie to the same side of the horizontal line through $v$.

Corollary

If a polygon is $y$-monotone, then it can be decomposed into two $y$-monotone non intersecting chains sharing their endpoints.
Triangulating a monotone polygon
Triangulating a monotone polygon

The vertices of the polygon $P$ are processed by decreasing order of their $y$-coordinate.
Triangulating a monotone polygon

The vertices of the polygon $P$ are processed by decreasing order of their $y$-coordinate.

During the process a queue $Q$ is used to store the vertices that have already been visited but are still needed in order to generate the triangulation. Characteristics of $Q$:

- The topmost vertex in $Q$, is a convex vertex of the subpolygon $P'$ still to be triangulated.
- All the remaining vertices in $Q$ are reflex.
- All the vertices in $Q$ belong to the same monotone chain of $P'$. 
Triangulating a monotone polygon

The vertices of the polygon \( P \) are processed by decreasing order of their \( y \)-coordinate.

During the process a queue \( Q \) is used to store the vertices that have already been visited but are still needed in order to generate the triangulation. Characteristics of \( Q \):
- The topmost vertex in \( Q \), is a convex vertex of the subpolygon \( P' \) still to be triangulated.
- All the remaining vertices in \( Q \) are reflex.
- All the vertices in \( Q \) belong to the same monotone chain of \( P' \).

Processing a vertex \( v_i \):
Triangulating a monotone polygon

The vertices of the polygon $P$ are processed by decreasing order of their $y$-coordinate.

During the process a queue $Q$ is used to store the vertices that have already been visited but are still needed in order to generate the triangulation. Characteristics of $Q$:
- The topmost vertex in $Q$, is a convex vertex of the subpolygon $P'$ still to be triangulated.
- All the remaining vertices in $Q$ are reflex.
- All the vertices in $Q$ belong to the same monotone chain of $P'$.

Processing a vertex $v_i$:

- If $v_i$ belongs to the opposite chain, report the diagonals connecting $v_i$ to every vertex of $Q$ and delete them all from $Q$, except the last one. Add $v_i$ to $Q$. 
Triangulating a monotone polygon

The vertices of the polygon \( P \) are processed by decreasing order of their \( y \)-coordinate.

During the process a queue \( Q \) is used to store the vertices that have already been visited but are still needed in order to generate the triangulation. Characteristics of \( Q \):

- The topmost vertex in \( Q \), is a convex vertex of the subpolygon \( P' \) still to be triangulated.
- All the remaining vertices in \( Q \) are reflex.
- All the vertices in \( Q \) belong to the same monotone chain of \( P' \).

Processing a vertex \( v_i \):

- If \( v_i \) belongs to the opposite chain, report the diagonals connecting \( v_i \) to every vertex of \( Q \) and delete them all from \( Q \), except the last one. Add \( v_i \) to \( Q \).
Triangulating a monotone polygon

The vertices of the polygon $P$ are processed by decreasing order of their $y$-coordinate.

During the process a queue $Q$ is used to store the vertices that have already been visited but are still needed in order to generate the triangulation. Characteristics of $Q$:
- The topmost vertex in $Q$, is a convex vertex of the subpolygon $P'$ still to be triangulated.
- All the remaining vertices in $Q$ are reflex.
- All the vertices in $Q$ belong to the same monotone chain of $P'$.

Processing a vertex $v_i$:

- If $v_i$ belongs to the opposite chain, report the diagonals connecting $v_i$ to every vertex of $Q$ and delete them all from $Q$, except the last one. Add $v_i$ to $Q$.
- If $v_i$ belongs to the same chain and produces a reflex turn, add $v_i$ to $Q$. 

Triangulating a monotone polygon

The vertices of the polygon \( P \) are processed by decreasing order of their \( y \)-coordinate.

During the process a queue \( Q \) is used to store the vertices that have already been visited but are still needed in order to generate the triangulation. Characteristics of \( Q \):
- The topmost vertex in \( Q \), is a convex vertex of the subpolygon \( P' \) still to be triangulated.
- All the remaining vertices in \( Q \) are reflex.
- All the vertices in \( Q \) belong to the same monotone chain of \( P' \).

Processing a vertex \( v_i \):

- If \( v_i \) belongs to the opposite chain, report the diagonals connecting \( v_i \) to every vertex of \( Q \) and delete them all from \( Q \), except the last one. Add \( v_i \) to \( Q \).
- If \( v_i \) belongs to the same chain and produces a reflex turn, add \( v_i \) to \( Q \).
- If \( v_i \) belongs to the same chain and produces a convex turn, report the diagonal connecting \( v_i \) to the penultimate element of \( Q \), delete the last element of \( Q \) and process \( v_i \) again.


Triangulating a monotone polygon

The vertices of the polygon \( P \) are processed by decreasing order of their \( y \)-coordinate.

During the process a queue \( Q \) is used to store the vertices that have already been visited but are still needed in order to generate the triangulation. Characteristics of \( Q \):

- The topmost vertex in \( Q \), is a convex vertex of the subpolygon \( P' \) still to be triangulated.
- All the remaining vertices in \( Q \) are reflex.
- All the vertices in \( Q \) belong to the same monotone chain of \( P' \).

Processing a vertex \( v_i \):

- If \( v_i \) belongs to the opposite chain, report the diagonals connecting \( v_i \) to every vertex of \( Q \) and delete them all from \( Q \), except the last one. Add \( v_i \) to \( Q \).

- If \( v_i \) belongs to the same chain and produces a reflex turn, add \( v_i \) to \( Q \).

- If \( v_i \) belongs to the same chain and produces a convex turn, report the diagonal connecting \( v_i \) to the penultimate element of \( Q \), delete the last element of \( Q \) and process \( v_i \) again.
Triangulating a monotone polygon
Triangulating a monotone polygon

Start

Queue state
1, 2
Triangulating a monotone polygon

Actual vertex: 3

Queue state: 1, 2, 3

Add
Triangulating a monotone polygon

Actual vertex: 4

Add

Queue state: 1, 2, 3, 4
Triangulating a monotone polygon

Actual vertex: 5

Add

Queue state:
1, 2, 3, 4, 5
Triangulating a monotone polygon

Actual vertex: 6

Add

Queue state:
1, 2, 3, 4, 5, 6
Triangulating a monotone polygon

Actual vertex: 7

Opposite chain

Queue state:
6, 7
Triangulating a monotone polygon

Actual vertex: 8

Ear

Queue state: 6, 8
Triangulating a monotone polygon

Actual vertex: 9

Ear

Queue state: 6, 9
Triangulating a monotone polygon

Actual vertex: 10

Add

Queue state:
6, 9, 10
Triangulating a monotone polygon

Actual vertex: 11

Add

Queue state:
6, 9, 10, 11
Triangulating a monotone polygon

Actual vertex: 12

Add

Queue state: 6, 9, 10, 11, 12
Triangulating a monotone polygon

Actual vertex: 13

Ear

Queue state: 6, 9, 10, 13
Actual vertex: 14

Ear

Queue state: 6, 9, 10, 14
Triangulating a monotone polygon

Actual vertex: 15

Opposite chain

Queue state:
14, 15
Triangulating a monotone polygon

Actual vertex: 16

Opposite chain

Queue state:
15, 16
Triangulating a monotone polygon

Actual vertex: 17

Ear

Queue state: 15, 17
Triangulating a monotone polygon

Actual vertex: 18

Ear

Queue state: 15, 18
Triangulating a monotone polygon

Actual vertex: 19

Opposite chain

Queue state:
18, 19
Triangulating a monotone polygon
Triangulating a monotone polygon

Running time: \( O(n) \)

Each vertex is removed from the queue \( Q \) in \( O(1) \) time.
Running time for triangulating a polygon:

- $O(n^2)$ by subtracting ears
- $O(n^2)$ by inserting diagonals

If the polygon is convex:

- $O(n)$ trivially

If the polygon is monotone:

- $O(n)$ scanning the monotone chains in order
Running time for triangulating a polygon:

- $O(n^2)$ by subtracting ears
- $O(n^2)$ by inserting diagonals

If the polygon is convex:

- $O(n)$ trivially

If the polygon is monotone:

- $O(n)$ scanning the monotone chains in order

**Is it possible to make it faster?**
Running time for triangulating a polygon:

- $O(n^2)$ by subtracting ears
- $O(n^2)$ by inserting diagonals

If the polygon is convex:

- $O(n)$ trivially

If the polygon is monotone:

- $O(n)$ scanning the monotone chains in order

Is it possible to be more efficient? Yes!

1. Decompose the polygon into monotone subpolygons
2. Triangulate the monotone subpolygons
Monotone partition
Monotone partition

In order to create a monotone partition of a polygon, all cusps need to be “broken” by internal diagonals.
Monotone partition

In order to create a monotone partition of a polygon, all cusps need to be “broken” by internal diagonals.
Monotone partition

In order to create a monotone partition of a polygon, all cusps need to be “broken” by internal diagonals.
Monotone partition

In order to create a monotone partition of a polygon, all cusps need to be “broken” by internal diagonals.
Monotone partition

In order to create a monotone partition of a polygon, all cusps need to be “broken” by internal diagonals.

This can be done starting from a trapezoidal decomposition of the polygon.
Monotone partition

In order to create a monotone partition of a polygon, all cusps need to be “broken” by internal diagonals.

This can be done starting from a trapezoidal decomposition of the polygon.
Monotone partition

In order to create a monotone partition of a polygon, all cusps need to be “broken” by internal diagonals.

This can be done starting from a trapezoidal decomposition of the polygon.

Connect each cusp with the opposite vertex in its trapezoid (the upper trapezoid, if the cusp is a local maximum, the lower one, if it is a local minimum).
Monotone partition

In order to create a monotone partition of a polygon, all cusps need to be “broken” by internal diagonals.

This can be done starting from a trapezoidal decomposition of the polygon.

Connect each cusp with the opposite vertex in its trapezoid (the upper trapezoid, if the cusp is a local maximum, the lower one, if it is a local minimum).
Monotone partition

In order to create a monotone partition of a polygon, all cusps need to be “broken” by internal diagonals.

This can be done starting from a trapezoidal decomposition of the polygon.

Connect each cusp with the opposite vertex in its trapezoid (the upper trapezoid, if the cusp is a local maximum, the lower one, if it is a local minimum).
Monotone partition

In order to create a monotone partition of a polygon, all cusps need to be “broken” by internal diagonals.

This can be done starting from a trapezoidal decomposition of the polygon.

Connect each cusp with the opposite vertex in its trapezoid (the upper trapezoid, if the cusp is a local maximum, the lower one, if it is a local minimum).
Monotone partition

In order to create a monotone partition of a polygon, all cusps need to be “broken” by internal diagonals.

This can be done starting from a trapezoidal decomposition of the polygon.

Connect each cusp with the opposite vertex in its trapezoid (the upper trapezoid, if the cusp is a local maximum, the lower one, if it is a local minimum).
Monotone partition

In order to create a monotone partition of a polygon, all cusps need to be “broken” by internal diagonals.

This can be done starting from a trapezoidal decomposition of the polygon.

Connect each cusp with the opposite vertex in its trapezoid (the upper trapezoid, if the cusp is a local maximum, the lower one, if it is a local minimum).

This gives rise to a correct algorithm:

- The diagonals do not intersect, because they belong to different trapezoids.
- The polygon ends up decomposed into monotone subpolygons.

Sweep line algorithm
Monotone partition

Sweep line algorithm
Monotone partition

Sweep line algorithm
A straight line (horizontal, in this case) scans the object (the polygon) and allows detecting and constructing the desired elements (cusps, trapezoids, diagonals), leaving the problem solved behind it. The sweeping process is discretized.
Monotone partition

Sweep line algorithm

A straight line (horizontal, in this case) scans the object (the polygon) and allows detecting and constructing the desired elements (cusps, trapezoids, diagonals), leaving the problem solved behind it. The sweeping process is discretized.

Essential elements of a sweep line algorithm:

- Events queue
  
  Priority queue keeping the information of the algorithm stops. In our problem, the events will be the vertices of the polygon, sorted by their $y$-coordinate, and no events will be generated throughout the sweeping advance.

- Sweep line
  
  Data structure storing the information of the portion of the object intersected by the sweeping line. It gets updated at each event. In our problem, it will contain the information of the edges of the polygon intersected by the line, sorted by abscissa, as well as the list of active trapezoids and their upper vertex.
Monotone partition

Updating the sweep line:

- Passing vertex: replace
- Local minimum cusp: delete
- Local maximum cusp: insert
- Initial vertex: insert
- Final vertex: delete
Monotone partition

Updating the sweep line:

- Passing vertex: replace
- Local minimum cusp: delete
- Local maximum cusp: insert
- Initial vertex: insert
- Final vertex: delete
Monotone partition

Updating the sweep line:

- Passing vertex: replace
- Local minimum cusp: delete
- Local maximum cusp: insert
- Initial vertex: insert
- Final vertex: delete
Monotone partition

Updating the sweep line:

- Passing vertex: replace
- Local minimum cusp: delete
- Local maximum cusp: insert
- Initial vertex: insert
- Final vertex: delete
Monotone partition

Updating the sweep line:

- Passing vertex: replace
- Local minimum cusp: delete
- Local maximum cusp: insert
- Initial vertex: insert
- Final vertex: delete
Monotone partition

Updating the sweep line:

- Passing vertex: replace
- Local minimum cusp: delete
- Local maximum cusp: insert
- Initial vertex: insert
- Final vertex: delete

\[ e_{i-1} e_i e_{i+1} \]

\[ e_m e_i e_n \]
Monotone partition

Updating the sweep line:

- Passing vertex: replace
- Local minimum cusp: delete
- Local maximum cusp: insert
- Initial vertex: insert
- Final vertex: delete
Monotone partition

Updating the sweep line:

- Passing vertex: replace
- Local minimum cusp: delete
- Local maximum cusp: insert
- Initial vertex: insert
- Final vertex: delete

In addition, eventually update the information of the upper vertex of the starting trapezoid.
Monotone partition

Updating the sweep line:

- Passing vertex: replace
- Local minimum cusp: delete
- Local maximum cusp: insert
- Initial vertex: insert
- Final vertex: delete

In addition, eventually update the information of the upper vertex of the starting trapezoid.
Monotone partition

Updating the sweep line:

- Passing vertex: replace
- Local minimum cusp: delete
- Local maximum cusp: insert
- Initial vertex: insert
- Final vertex: delete

In addition, eventually update the information of the upper vertex of the starting trapezoid.
Monotone partition

Updating the sweep line:

- Passing vertex: replace
- Local minimum cusp: delete
- Local maximum cusp: insert
- Initial vertex: insert
- Final vertex: delete

In addition, eventually update the information of the upper vertex of the starting trapezoid.
Monotone partition

Updating the sweep line:

- Passing vertex: replace
- Local minimum cusp: delete
- Local maximum cusp: insert
- Initial vertex: insert
- Final vertex: delete

In addition, eventually update the information of the upper vertex of the starting trapezoid.
Monotone partition

Updating the sweep line:

- Passing vertex: replace
- Local minimum cusp: delete
- Local maximum cusp: insert
- Initial vertex: insert
- Final vertex: delete

In addition, eventually update the information of the upper vertex of the starting trapezoid.
Monotone partition

As for diagonals:

- Passing vertex: nothing
- Local minimum cusp: wait until it can be connect to the final vertex of the starting trapezoid
- Local maximum cusp: connect it with the initial vertex of the ending trapezoid
- Initial vertex: nothing
- Final vertex: nothing
As for diagonals:

- Passing vertex: nothing
- Local minimum cusp: wait until it can be connect to the final vertex of the starting trapezoid
- Local maximum cusp: connect it with the initial vertex of the ending trapezoid
- Initial vertex: nothing
- Final vertex: nothing
Monotone partition

As for diagonals:

- Passing vertex: nothing
- Local minimum cusp: wait until it can be connect to the final vertex of the starting trapezoid
- Local maximum cusp: connect it with the initial vertex of the ending trapezoid
- Initial vertex: nothing
- Final vertex: nothing
Monotone partition

As for diagonals:

- Passing vertex: nothing
- Local minimum cusp: wait until it can be connect to the final vertex of the starting trapezoid
- Local maximum cusp: connect it with the initial vertex of the ending trapezoid
- Initial vertex: nothing
- Final vertex: nothing
Monotone partition

As for diagonals:

- Passing vertex: nothing
- Local minimum cusp: wait until it can be connect to the final vertex of the starting trapezoid
- Local maximum cusp: connect it with the initial vertex of the ending trapezoid
- Initial vertex: nothing
- Final vertex: nothing
Monotone partition

As for diagonals:

- Passing vertex: nothing
- Local minimum cusp: wait until it can be connect to the final vertex of the starting trapezoid
- Local maximum cusp: connect it with the initial vertex of the ending trapezoid
- Initial vertex: nothing
- Final vertex: nothing
Monotone partition

vertex
1

sweep line

$e_1, e_19$

Computational Geometry, Facultat d'Informàtica de Barcelona, UPC
Monotone partition

vertex
18

sweep line

\[
\begin{align*}
\text{v1} & \quad e1, e19 \\
\text{v18} & \quad e18, e17
\end{align*}
\]
Monotone partition

vertex
3

sweep line
\[ v_{18} \]
\[ e_1, e_{19}, e_{18}, e_{17} \]
\[ e_2, e_3 \]
Monotone partition

vertex
3

sweep line
\[ e_1, e_{19}, e_{18}, e_{17} \]
\[ e_2, e_3 \]

Diagonal \( v_1 - v_3 \)
Monotone partition

vertex
19

sweep line

\[ v_3 \quad \not\in \quad v_8 \quad \not\in \quad e_1, e_2, e_3, e_9, e_8, e_{17} \]
Monotone partition

vertex
2

sweep line
\[ v_3 \quad v_{19} \]
\[ \overline{v_3 \quad v_{19}} \]
\[ \overline{v_3 \quad v_{19}} \]
\[ \overline{e_3 \quad e_{17}} \]
Monotone partition

vertex
17

sweep line
\[ e_3, e_7 \]
\[ e_9 \]
\[ e_16 \]
Monotone partition

vertex
17

sweep line
e3, e16

Diagonal v19 − v17
Monotone partition

vertex
15

sweep line

\[ \overline{v_{17}} \]
\[ e_3, e_{16} \]

\[ \overline{v_{15}} \]
\[ e_{15}, e_{14} \]
Monotone partition

vertex
16

sweep line
\[ v^7, v^5 \]
\[ e_3, e_6, e_7, e_{14} \]
Monotone partition

vertex
5

sweep line

\[ v_{16} \]
\[ e_{3}, e_{14} \]

\[ v_{5} \]
\[ e_{5}, e_{4} \]
Monotone partition

vertex
6

sweep line
\[ v_{16}, e_{4}, e_{3}, e_{14} \]
\[ e_{6} \]
Monotone partition

vertex
7

sweep line

\( e_7 \)
\( v_{16} \)
\( e_6, e_4, e_3, e_{14} \)
Monotone partition

vertex
14

sweep line

\[
\begin{align*}
v7 & \quad vX6 \\
& \quad e7, e4, e3, eX4
\end{align*}
\]

\[e13\]
Monotone partition

vertex
14

sweep line
\[
\begin{align*}
    v7 & \quad v6 \\
    e7, e4, e3, e4 \\
\end{align*}
\]

Diagonal \( v16 - v14 \)
Monotone partition

vertex
4

sweep line
$e_7, e_4, e_3, e_{13}$
Monotone partition

vertex 9

sweep line

\[ \overline{e_7, e_{13}} \]

\[ \overline{e_8, e_9} \]
Monotone partition

vertex
9

sweep line

Diagonal $v_4 - v_9$
Monotone partition

vertex
13

sweep line

$v_9 \not\in e_7, e_8, e_9, e_{13}$

$e_{12}$
Monotone partition

vertex
8

sweep line

\[ v_9, v_{13}, e_9, e_{12} \]
TRIANGULATING POLYGONS

Monotone partition

vertex
11

sweep line
\( e_{9}, e_{12} \)
\( e_{10}, e_{11} \)
Monotone partition

vertex

11

sweep line

Diagonal $v_{13} - v_{11}$
Monotone partition

vertex
12

sweep line
\[ v_{11}, v_{11}, e_{9}, e_{10}, e_{2}, e_{2} \]
Monotone partition

vertex
10

sweep line

$v_{11}$

$\not{x_9}, \not{x_{10}}$
Monotone partition
Monotone partition
Monotone partition

Running time:

- Sorting the vertices in the event queue: $O(n \log n)$ time.
- On each event, replace, insert or delete in $O(\log n)$ time.
- There are $n$ events.

The algorithm runs in $O(n \log n)$ time.
Summarizing

Running time of polygon triangulation:

- $O(n^2)$ by subtracting ears
- $O(n^2)$ by inserting diagonals
- $O(n \log n)$ by:
  1. Decomposing the polygon into monotone subpolygons in $O(n \log n)$ time
  2. Triangulating each monotone subpolygon in $O(n)$ time
Summarizing

Running time of polygon triangulation:

- $O(n^2)$ by subtracting ears
- $O(n^2)$ by inserting diagonals
- $O(n \log n)$ by:
  1. Decomposing the polygon into monotone subpolygons in $O(n \log n)$ time
  2. Triangulating each monotone subpolygon in $O(n)$ time

Is it possible to triangulate a polygon in $o(n \log n)$ time?
Summarizing

Running time of polygon triangulation:

- \(O(n^2)\) by substracting ears
- \(O(n^2)\) by inserting diagonals
- \(O(n \log n)\) by:
  1. Decomposing the polygon into monotone subpolygons in \(O(n \log n)\) time
  2. Triangulating each monotone subpolygon in \(O(n)\) time

Is it possible to triangulate a polygon in \(o(n \log n)\) time?

Yes.
There exists an algorithm to triangulate an \(n\)-gon in \(O(n)\) time, but it is too complicated and, in practice, it is not used.
STORING THE POLYGON TRIANGULATION
Possible options, advantages and disadvantages
Storing the polygon triangulation

Possible options, advantages and disadvantages

Storing the list of all the diagonals of the triangulation
Possible options, advantages and disadvantages

Storing the list of all the diagonals of the triangulation

**Advantage:** small memory usage.

**Disadvantage:** it suffices to draw the triangulation, but it does not contain the proximity information. For example, finding the triangles incident to a given diagonal, or finding the neighbors of a given triangle are expensive computations.
Possible options, advantages and disadvantages

Storing the list of all the diagonals of the triangulation

**Advantage:** small memory usage.

**Disadvantage:** it suffices to draw the triangulation, but it does not contain the proximity information. For example, finding the triangles incident to a given diagonal, or finding the neighbors of a given triangle are expensive computations.

For each triangle, storing the sorted list of its vertices and edges, as well as the sorted list of its neighbors.
Possible options, advantages and disadvantages

Storing the list of all the diagonals of the triangulation

**Advantage:** small memory usage.

**Disadvantage:** it suffices to draw the triangulation, but it does not contain the proximity information. For example, finding the triangles incident to a given diagonal, or finding the neighbors of a given triangle are expensive computations.

For each triangle, storing the sorted list of its vertices and edges, as well as the sorted list of its neighbors.

**Advantage:** allows to quickly recover neighborhood information.

**Disadvantage:** the stored data is redundant and it uses more space than required.
Storing the polygon triangulation

Possible options, advantages and disadvantages

Storing the list of all the diagonals of the triangulation

**Advantage:** small memory usage.

**Disadvantage:** it suffices to draw the triangulation, but it does not contain the proximity information. For example, finding the triangles incident to a given diagonal, or finding the neighbors of a given triangle are expensive computations.

For each triangle, storing the sorted list of its vertices and edges, as well as the sorted list of its neighbors.

**Advantage:** allows to quickly recover neighborhood information.

**Disadvantage:** the stored data is redundant and it uses more space than required.

The data structure which is most frequently used to store a triangulation is the DCEL (doubly connected edge list).

The DCEL is also used to store plane partitions, polyhedra, meshes, etc.
Storing the polygon triangulation

DCEL
Storing the polygon triangulation

DCEL
Storing the polygon triangulation

DCEL

Computational Geometry, Facultat d'Informàtica de Barcelona, UPC
Storing the polygon triangulation

DCEL
Storing the polygon triangulation

DCEL

Computational Geometry, Facultat d'Informàtica de Barcelona, UPC
Storing the polygon triangulation

DCEL

Computational Geometry, Facultat d'Informàtica de Barcelona, UPC
Storing the polygon triangulation

DCEL

\[ e \]
Storing the polygon triangulation

DCEL

$e$

$v_B$

$v_E$
Storing the polygon triangulation

DCEL

\[ v_B \quad e \quad v_E \]

\[ f_L \quad f_R \]
Storing the polygon triangulation

DCEL

Computational Geometry, Facultat d'Informàtica de Barcelona, UPC
Storing the polygon triangulation

DCEL

Computational Geometry, Facultat d'Informàtica de Barcelona, UPC
Storing the polygon triangulation

DCEL

Computational Geometry, Facultat d'Informàtica de Barcelona, UPC
Storing the polygon triangulation

Table of faces

<table>
<thead>
<tr>
<th>f</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>11</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>$\infty$</td>
<td>9</td>
</tr>
</tbody>
</table>
Storing the polygon triangulation

Table of vertices

<table>
<thead>
<tr>
<th>v</th>
<th>x</th>
<th>y</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$x_1$</td>
<td>$y_1$</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>$x_2$</td>
<td>$y_2$</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>$x_3$</td>
<td>$y_3$</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>$x_4$</td>
<td>$y_4$</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>$x_5$</td>
<td>$y_5$</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>$x_6$</td>
<td>$y_6$</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>$x_7$</td>
<td>$y_7$</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>$x_8$</td>
<td>$y_8$</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>$x_9$</td>
<td>$y_9$</td>
<td>9</td>
</tr>
</tbody>
</table>
Storing the polygon triangulation

DCEL

<table>
<thead>
<tr>
<th>e</th>
<th>$v_B$</th>
<th>$v_E$</th>
<th>$f_L$</th>
<th>$f_R$</th>
<th>$e_P$</th>
<th>$e_N$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>$\infty$</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>7</td>
<td>$\infty$</td>
<td>13</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>3</td>
<td>$\infty$</td>
<td>7</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>$\infty$</td>
<td>15</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>6</td>
<td>2</td>
<td>$\infty$</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>7</td>
<td>6</td>
<td>$\infty$</td>
<td>15</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>8</td>
<td>4</td>
<td>$\infty$</td>
<td>11</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>9</td>
<td>4</td>
<td>$\infty$</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>1</td>
<td>5</td>
<td>$\infty$</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>7</td>
<td>3</td>
<td>6</td>
<td>13</td>
<td>6</td>
</tr>
<tr>
<td>11</td>
<td>9</td>
<td>7</td>
<td>4</td>
<td>1</td>
<td>8</td>
<td>14</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>9</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>7</td>
<td>14</td>
<td>3</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>7</td>
<td>1</td>
<td>3</td>
<td>12</td>
<td>10</td>
</tr>
<tr>
<td>15</td>
<td>4</td>
<td>6</td>
<td>6</td>
<td>2</td>
<td>10</td>
<td>5</td>
</tr>
</tbody>
</table>
Storing the polygon triangulation

Storage space

- For each face:
  1 pointer
- For each vertex:
  2 coordinates + 1 pointer
- For each edge:
  6 pointers

In total, the storage space is $O(n)$. 
There are other DCEL variants, as for example:
There are other DCEL variants, as for example:
Storing the polygon triangulation

DCEL

There are other DCEL variants, as for example:

\[ e \rightarrow v_B, f_R, e_N, e' \]
\[ e' \rightarrow v_B, f_R, e_N, e \]
Storing the polygon triangulation

DCEL

There are other DCEL variants, as for example:

\[ e \rightarrow v_B, f_R, e_N, e' \]
\[ e' \rightarrow v_B, f_R, e_N, e \]
There are other DCEL variants, as for example:

\[ e \rightarrow v_B, f_R, e_N, e' \]
\[ e' \rightarrow v_B, f_R, e_N, e \]
Storing the polygon triangulation

There are other DCEL variants, as for example:

\[ e \rightarrow v_B, f_R, e_N, e' \]
\[ e' \rightarrow v_B, f_R, e_N, e \]
Storing the polygon triangulation

<table>
<thead>
<tr>
<th>$e$</th>
<th>$v_B$</th>
<th>$v_E$</th>
<th>$f_L$</th>
<th>$f_R$</th>
<th>$e_P$</th>
<th>$e_N$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>$\infty$</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>7</td>
<td>$\infty$</td>
<td>13</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>3</td>
<td>$\infty$</td>
<td>7</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>$\infty$</td>
<td>15</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>6</td>
<td>2</td>
<td>$\infty$</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>7</td>
<td>6</td>
<td>$\infty$</td>
<td>15</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>8</td>
<td>4</td>
<td>$\infty$</td>
<td>11</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>9</td>
<td>4</td>
<td>$\infty$</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>1</td>
<td>5</td>
<td>$\infty$</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>7</td>
<td>3</td>
<td>6</td>
<td>13</td>
<td>6</td>
</tr>
<tr>
<td>11</td>
<td>9</td>
<td>7</td>
<td>4</td>
<td>1</td>
<td>8</td>
<td>14</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>9</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>7</td>
<td>14</td>
<td>3</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>7</td>
<td>1</td>
<td>3</td>
<td>12</td>
<td>10</td>
</tr>
<tr>
<td>15</td>
<td>4</td>
<td>6</td>
<td>6</td>
<td>2</td>
<td>10</td>
<td>5</td>
</tr>
</tbody>
</table>

DCEL
Storing the polygon triangulation

<table>
<thead>
<tr>
<th>$e$</th>
<th>$v_B$</th>
<th>$f_R$</th>
<th>$e_N$</th>
<th>$e'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>$\infty$</td>
<td>2</td>
<td>1'</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>$\infty$</td>
<td>3</td>
<td>2'</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>7</td>
<td>2</td>
<td>3'</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>$\infty$</td>
<td>5</td>
<td>4'</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>$\infty$</td>
<td>6</td>
<td>5'</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>$\infty$</td>
<td>7</td>
<td>6'</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>$\infty$</td>
<td>8</td>
<td>7'</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>$\infty$</td>
<td>9</td>
<td>8'</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>$\infty$</td>
<td>1</td>
<td>9'</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>6</td>
<td>6</td>
<td>10'</td>
</tr>
<tr>
<td>11</td>
<td>9</td>
<td>1</td>
<td>14</td>
<td>11'</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>1</td>
<td>11</td>
<td>12'</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
<td>7</td>
<td>3</td>
<td>13'</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>3</td>
<td>10</td>
<td>14'</td>
</tr>
<tr>
<td>15</td>
<td>4</td>
<td>2</td>
<td>5</td>
<td>15'</td>
</tr>
<tr>
<td>1'</td>
<td>2</td>
<td>5</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>2'</td>
<td>3</td>
<td>7</td>
<td>13</td>
<td>2</td>
</tr>
<tr>
<td>3'</td>
<td>3</td>
<td>$\infty$</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>4'</td>
<td>5</td>
<td>2</td>
<td>15</td>
<td>4</td>
</tr>
<tr>
<td>5'</td>
<td>6</td>
<td>2</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>6'</td>
<td>7</td>
<td>6</td>
<td>15</td>
<td>6</td>
</tr>
<tr>
<td>7'</td>
<td>8</td>
<td>4</td>
<td>11</td>
<td>7</td>
</tr>
<tr>
<td>8'</td>
<td>9</td>
<td>4</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>9'</td>
<td>1</td>
<td>5</td>
<td>12</td>
<td>9</td>
</tr>
<tr>
<td>10'</td>
<td>7</td>
<td>3</td>
<td>13</td>
<td>10</td>
</tr>
<tr>
<td>11'</td>
<td>7</td>
<td>4</td>
<td>8</td>
<td>11</td>
</tr>
<tr>
<td>12'</td>
<td>9</td>
<td>5</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>13'</td>
<td>4</td>
<td>3</td>
<td>14</td>
<td>13</td>
</tr>
<tr>
<td>14'</td>
<td>7</td>
<td>1</td>
<td>12</td>
<td>14</td>
</tr>
<tr>
<td>15'</td>
<td>6</td>
<td>6</td>
<td>10</td>
<td>15</td>
</tr>
</tbody>
</table>
Storing the polygon triangulation

How to build the DCEL
Storing the polygon triangulation

How to build the DCEL

Algorithm 1: substracting ears
How to build the DCEL

Algorithm 1: substracting ears
How to build the DCEL

Algorithm 1: substracting ears

Initialize
Storing the polygon triangulation

How to build the DCEL

Algorithm 1: substracting ears

Initialize

Table of vertices

<table>
<thead>
<tr>
<th>v</th>
<th>x</th>
<th>y</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$x_1$</td>
<td>$y_1$</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>$x_2$</td>
<td>$y_2$</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>$x_3$</td>
<td>$y_3$</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>$x_4$</td>
<td>$y_4$</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>$x_5$</td>
<td>$y_5$</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>$x_6$</td>
<td>$y_6$</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>$x_7$</td>
<td>$y_7$</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>$x_8$</td>
<td>$y_8$</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>$x_9$</td>
<td>$y_9$</td>
<td>9</td>
</tr>
</tbody>
</table>
Storing the polygon triangulation

How to build the DCEL

Algorithm 1: substracting ears

Initialize

Table of faces

<table>
<thead>
<tr>
<th>f</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>∞</td>
<td>9</td>
</tr>
</tbody>
</table>
Storing the polygon triangulation

How to build the DCEL

Algorithm 1: substracting ears

Initialize

DCEL

<table>
<thead>
<tr>
<th>e</th>
<th>v_B</th>
<th>v_E</th>
<th>f_L</th>
<th>f_R</th>
<th>e_P</th>
<th>e_N</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>∞</td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>∞</td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>4</td>
<td>∞</td>
<td></td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>5</td>
<td>∞</td>
<td></td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>6</td>
<td>∞</td>
<td></td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>7</td>
<td>∞</td>
<td></td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>8</td>
<td>∞</td>
<td></td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>9</td>
<td>∞</td>
<td></td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>1</td>
<td>∞</td>
<td></td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Storing the polygon triangulation

How to build the DCEL

Algorithm 1: substracting ears

Initialize

<table>
<thead>
<tr>
<th>$e$</th>
<th>$v_B$</th>
<th>$v_E$</th>
<th>$f_L$</th>
<th>$f_R$</th>
<th>$e_P$</th>
<th>$e_N$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>$\infty$</td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>$\infty$</td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>3</td>
<td>$\infty$</td>
<td></td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>5</td>
<td>$\infty$</td>
<td></td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>6</td>
<td>$\infty$</td>
<td></td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>7</td>
<td>$\infty$</td>
<td></td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>8</td>
<td>$\infty$</td>
<td></td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>9</td>
<td>$\infty$</td>
<td></td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>1</td>
<td>$\infty$</td>
<td></td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
How to build the DCEL

Algorithm 1: substracting ears

Advance
How to build the DCEL

Algorithm 1: substracting ears

Advance
How to build the DCEL

Algorithm 1: substracting ears

Advance

Table of faces

<table>
<thead>
<tr>
<th>f</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>9</td>
</tr>
</tbody>
</table>
Storing the polygon triangulation

How to build the DCEL

Algorithm 1: subtracting ears

Advance

Table of faces

<table>
<thead>
<tr>
<th>f</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>9</td>
</tr>
</tbody>
</table>

DCEL

<table>
<thead>
<tr>
<th>e</th>
<th>v_B</th>
<th>v_E</th>
<th>f_L</th>
<th>f_R</th>
<th>e_P</th>
<th>e_N</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>∞</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>1</td>
<td>5</td>
<td>∞</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>9</td>
<td>5</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Storing the polygon triangulation

How to build the DCEL

Algorithm 2: inserting diagonals
How to build the DCEL

Algorithm 2: inserting diagonals
Storing the polygon triangulation

How to build the DCEL

Algorithm 2: inserting diagonals

Initialize
Storing the polygon triangulation

How to build the DCEL

Algorithm 2: inserting diagonals

Initialize

Table of vertices

<table>
<thead>
<tr>
<th>v</th>
<th>x</th>
<th>y</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$x_1$</td>
<td>$y_1$</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>$x_2$</td>
<td>$y_2$</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>$x_3$</td>
<td>$y_3$</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>$x_4$</td>
<td>$y_4$</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>$x_5$</td>
<td>$y_5$</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>$x_6$</td>
<td>$y_6$</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>$x_7$</td>
<td>$y_7$</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>$x_8$</td>
<td>$y_8$</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>$x_9$</td>
<td>$y_9$</td>
<td>9</td>
</tr>
</tbody>
</table>
Storing the polygon triangulation

How to build the DCEL

Algorithm 2: inserting diagonals

Initialize

Table of faces

<table>
<thead>
<tr>
<th>$f$</th>
<th>$e$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\infty$</td>
<td>9</td>
</tr>
</tbody>
</table>
How to build the DCEL

Algorithm 2: inserting diagonals

Base step
Storing the polygon triangulation

How to build the DCEL

Algorithm 2: inserting diagonals

Base step

Table of faces

<table>
<thead>
<tr>
<th>f</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>10</td>
</tr>
</tbody>
</table>
Storing the polygon triangulation

How to build the DCEL

Algorithm 2: inserting diagonals

Base step

Table of faces

<table>
<thead>
<tr>
<th>f</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>10</td>
</tr>
</tbody>
</table>

DCEL

<table>
<thead>
<tr>
<th>e</th>
<th>v_B</th>
<th>v_E</th>
<th>f_L</th>
<th>f_R</th>
<th>e_P</th>
<th>e_N</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>6</td>
<td>7</td>
<td>6</td>
<td>∞</td>
<td>15</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>7</td>
<td>4</td>
<td>6</td>
<td>∞</td>
<td>6</td>
<td>15</td>
</tr>
<tr>
<td>15</td>
<td>4</td>
<td>6</td>
<td>6</td>
<td>∞</td>
<td>10</td>
<td>6</td>
</tr>
</tbody>
</table>
Storing the polygon triangulation

How to build the DCEL

Algorithm 2: inserting diagonals

Merge step
Storing the polygon triangulation

How to build the DCEL

Algorithm 2: inserting diagonals

Merge step

DCEL 1

<table>
<thead>
<tr>
<th>$e$</th>
<th>$v_B$</th>
<th>$v_E$</th>
<th>$f_L$</th>
<th>$f_R$</th>
<th>$e_P$</th>
<th>$e_N$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>$\infty$</td>
<td>9</td>
<td>14</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>7</td>
<td>1</td>
<td>$\infty$</td>
<td>12</td>
<td>7</td>
</tr>
</tbody>
</table>

DCEL 2

<table>
<thead>
<tr>
<th>$e$</th>
<th>$v_B$</th>
<th>$v_E$</th>
<th>$f_L$</th>
<th>$f_R$</th>
<th>$e_P$</th>
<th>$e_N$</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>6</td>
<td>7</td>
<td>6</td>
<td>$\infty$</td>
<td>15</td>
<td>14</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>7</td>
<td>$\infty$</td>
<td>3</td>
<td>2</td>
<td>10</td>
</tr>
</tbody>
</table>
How to build the DCEL

Algorithm 2: inserting diagonals

Merge step

DCEL 1

<table>
<thead>
<tr>
<th>e</th>
<th>v₁</th>
<th>v₂</th>
<th>v₃</th>
<th>v₄</th>
<th>v₅</th>
<th>v₆</th>
<th>v₇</th>
<th>v₈</th>
<th>v₉</th>
<th>v₁₀</th>
<th>v₁₁</th>
<th>v₁₂</th>
<th>v₁₃</th>
<th>v₁₄</th>
<th>v₁₅</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>∞</td>
<td>9</td>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>7</td>
<td>1</td>
<td></td>
<td>12</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

DCEL 2

<table>
<thead>
<tr>
<th>e</th>
<th>v₁</th>
<th>v₂</th>
<th>v₃</th>
<th>v₄</th>
<th>v₅</th>
<th>v₆</th>
<th>v₇</th>
<th>v₈</th>
<th>v₉</th>
<th>v₁₀</th>
<th>v₁₁</th>
<th>v₁₂</th>
<th>v₁₃</th>
<th>v₁₄</th>
<th>v₁₅</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>6</td>
<td>7</td>
<td>6</td>
<td>∞</td>
<td>15</td>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>7</td>
<td>1</td>
<td></td>
<td>3</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Storing the polygon triangulation

How to build the DCEL

Algorithm 2: inserting diagonals

Merge step

DCEL 1

<table>
<thead>
<tr>
<th>e</th>
<th>v_B</th>
<th>v_E</th>
<th>f_L</th>
<th>f_R</th>
<th>e_P</th>
<th>e_N</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>∞</td>
<td>9</td>
<td>14</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>7</td>
<td>1</td>
<td>∞</td>
<td>12</td>
<td>7</td>
</tr>
</tbody>
</table>

DCEL 2

<table>
<thead>
<tr>
<th>e</th>
<th>v_B</th>
<th>v_E</th>
<th>f_L</th>
<th>f_R</th>
<th>e_P</th>
<th>e_N</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>6</td>
<td>7</td>
<td>6</td>
<td>∞</td>
<td>15</td>
<td>14</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>7</td>
<td>3</td>
<td>∞</td>
<td>2</td>
<td>10</td>
</tr>
</tbody>
</table>

Merged DCEL

<table>
<thead>
<tr>
<th>e</th>
<th>v_B</th>
<th>v_E</th>
<th>f_L</th>
<th>f_R</th>
<th>e_P</th>
<th>e_N</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>∞</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>7</td>
<td>6</td>
<td>∞</td>
<td>15</td>
<td>7</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>7</td>
<td>1</td>
<td>3</td>
<td>12</td>
<td>10</td>
</tr>
</tbody>
</table>
Storing the polygon triangulation

How to build the DCEL

Algorithm 3:
Storing the polygon triangulation

How to build the DCEL

Algorithm 3:

1. Decompose into monotone polygons
2. Triangulate monotone pieces
How to build the DCEL

Algorithm 3:

1. Decompose into monotone polygons
2. Triangulate monotone pieces

Computing the DCEL is done by combining previous strategies:

- Separating ears for triangulating each monotone subpolygon.
- Merging DCELs for putting together the monotone pieces.
WHAT HAPPENS IN 3D?
WHAT HAPPENS IN 3D?

A polyhedron that can be tetrahedralized:
WHAT HAPPENS IN 3D?

A polyhedron that cannot be tetrahedralized:
A cube can be decomposed into 6 tetrahedra...
A cube can be decomposed into 6 tetrahedra... but also into 5!
TRIANGULATING POLYGONS

TO LEARN MORE


TRIANGULATING POLYGONS

TO LEARN MORE


A NICE APPLICATION

The art gallery theorem