TRIANGULATING POLYGONS

Vera Sacristán

Computational Geometry
Facultat d'Informàtica de Barcelona
Universitat Politcnica de Catalunya

TRIANGULATING POLYGONS

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.

- intersección de los Δ's es vacía
- unión de los Δ's es el polígono
- vértices de Δ's = vértices del polígono.

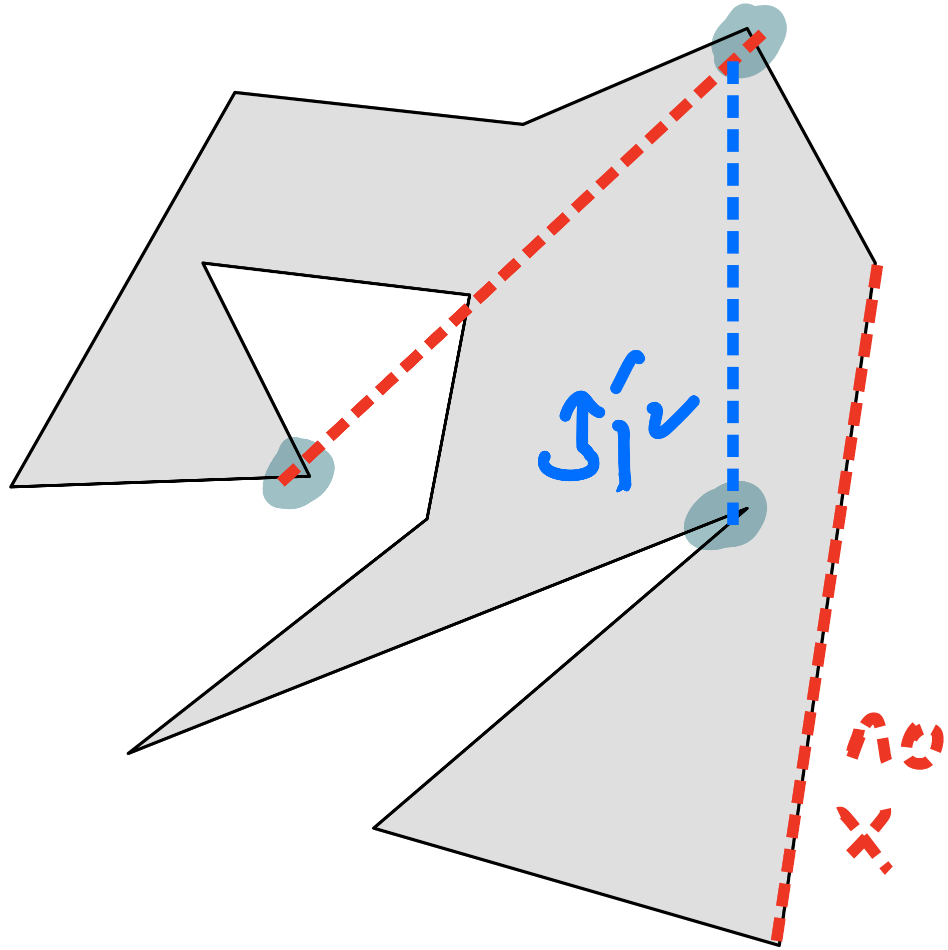
En el interior del polígono es una gráfica plana maximal, sus aristas son segmentos de recta

TRIANGULATING POLYGONS

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.

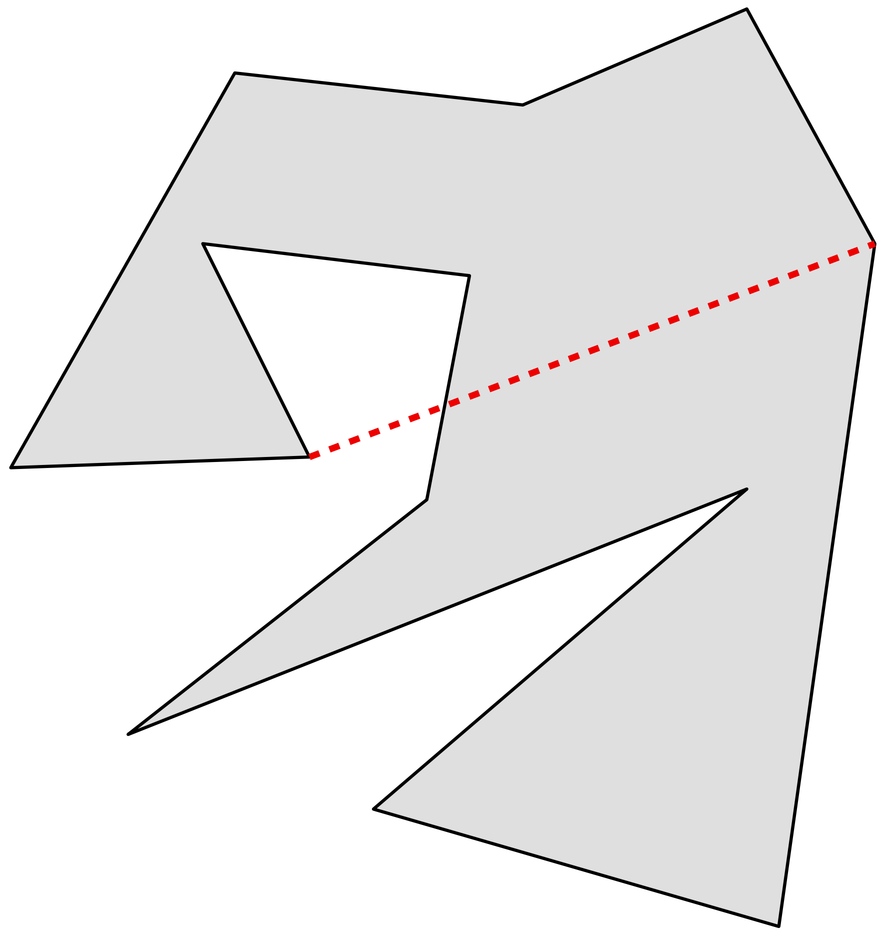


TRIANGULATING POLYGONS

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.

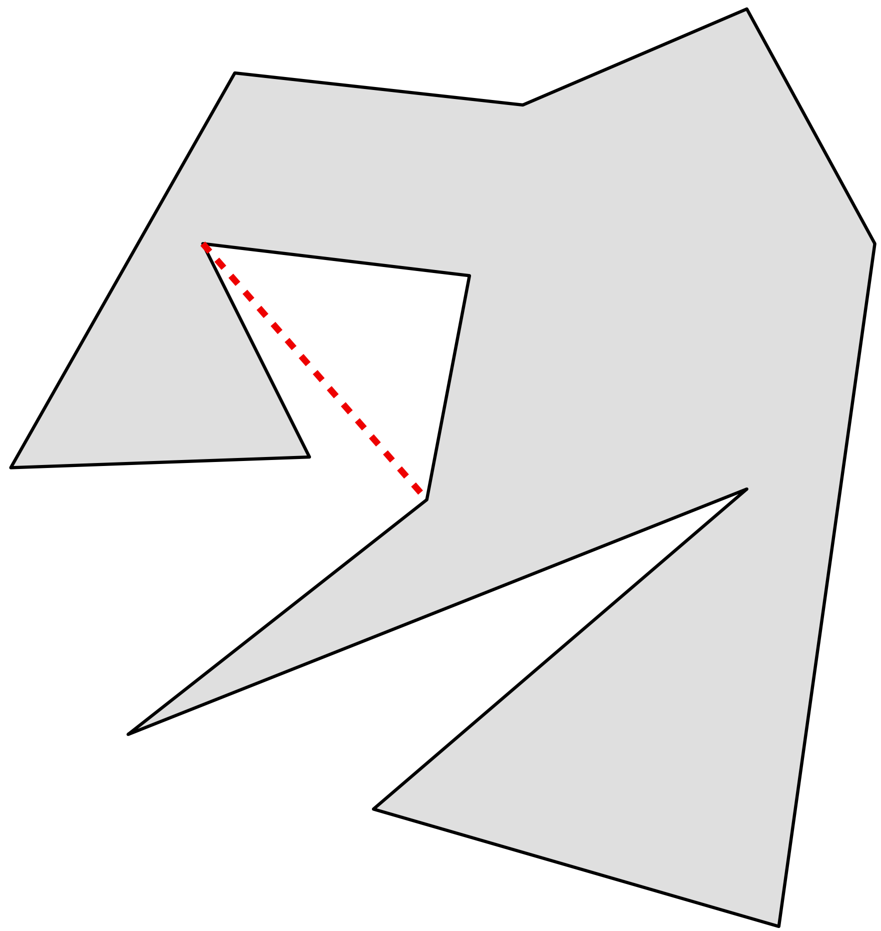


TRIANGULATING POLYGONS

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.



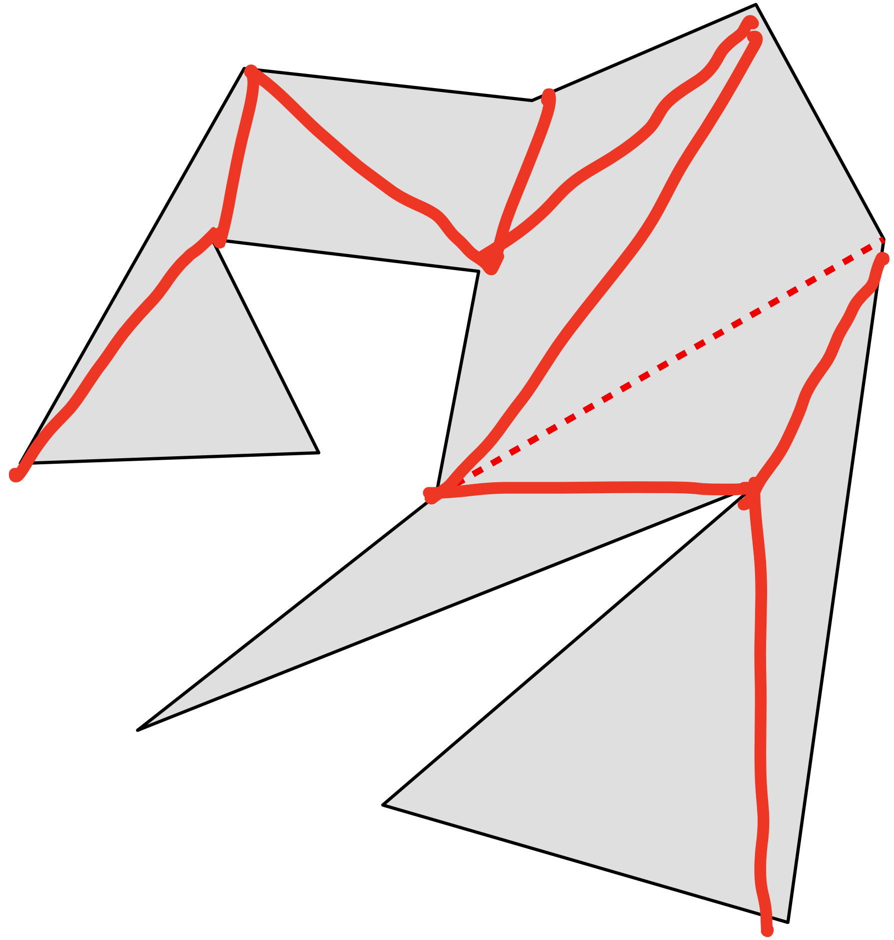
TRIANGULATING POLYGONS

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.

o No hay cruces.

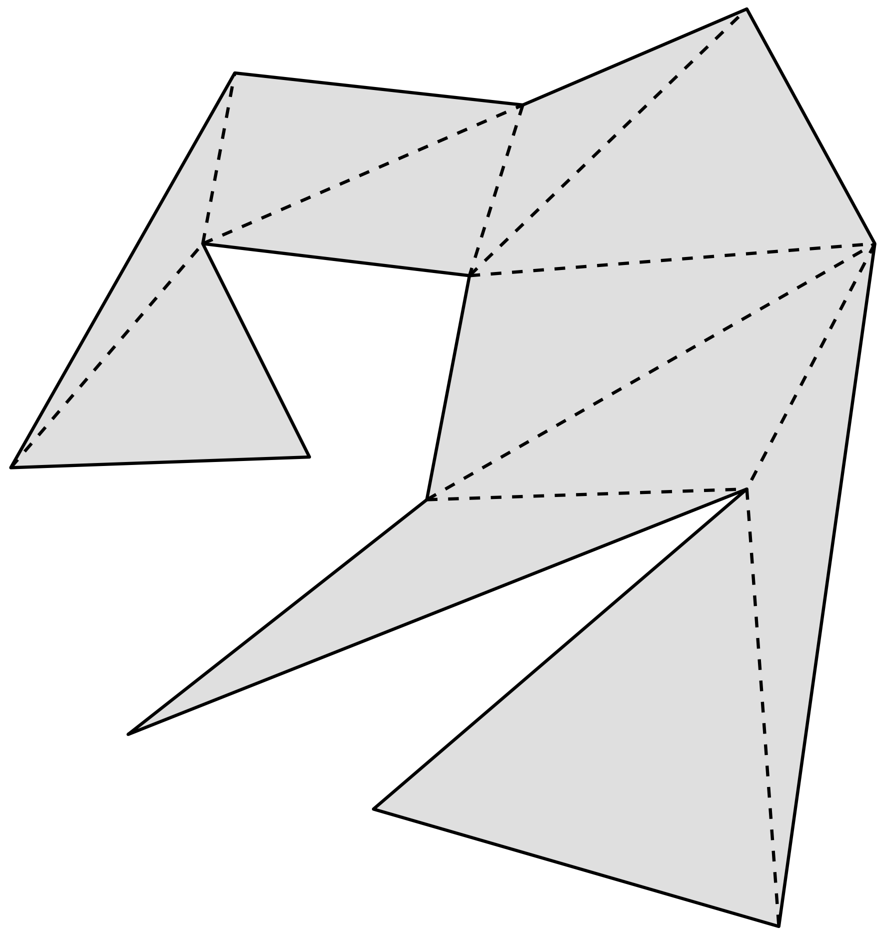


TRIANGULATING POLYGONS

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.



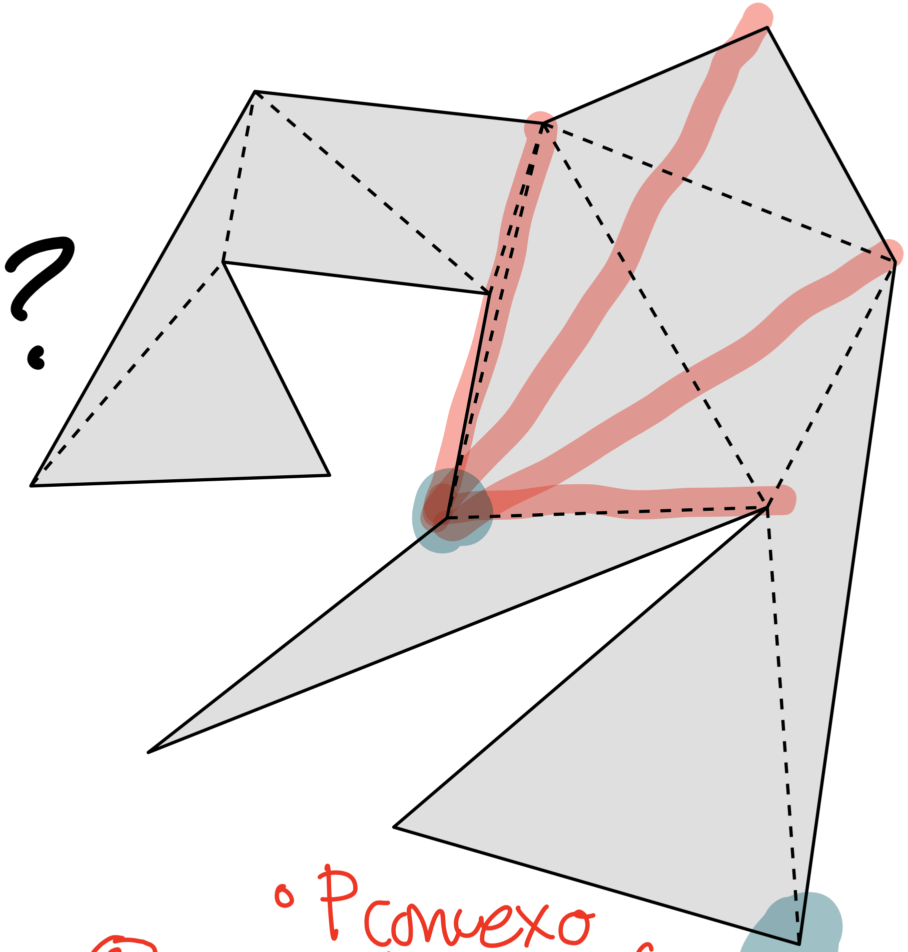
TRIANGULATING POLYGONS

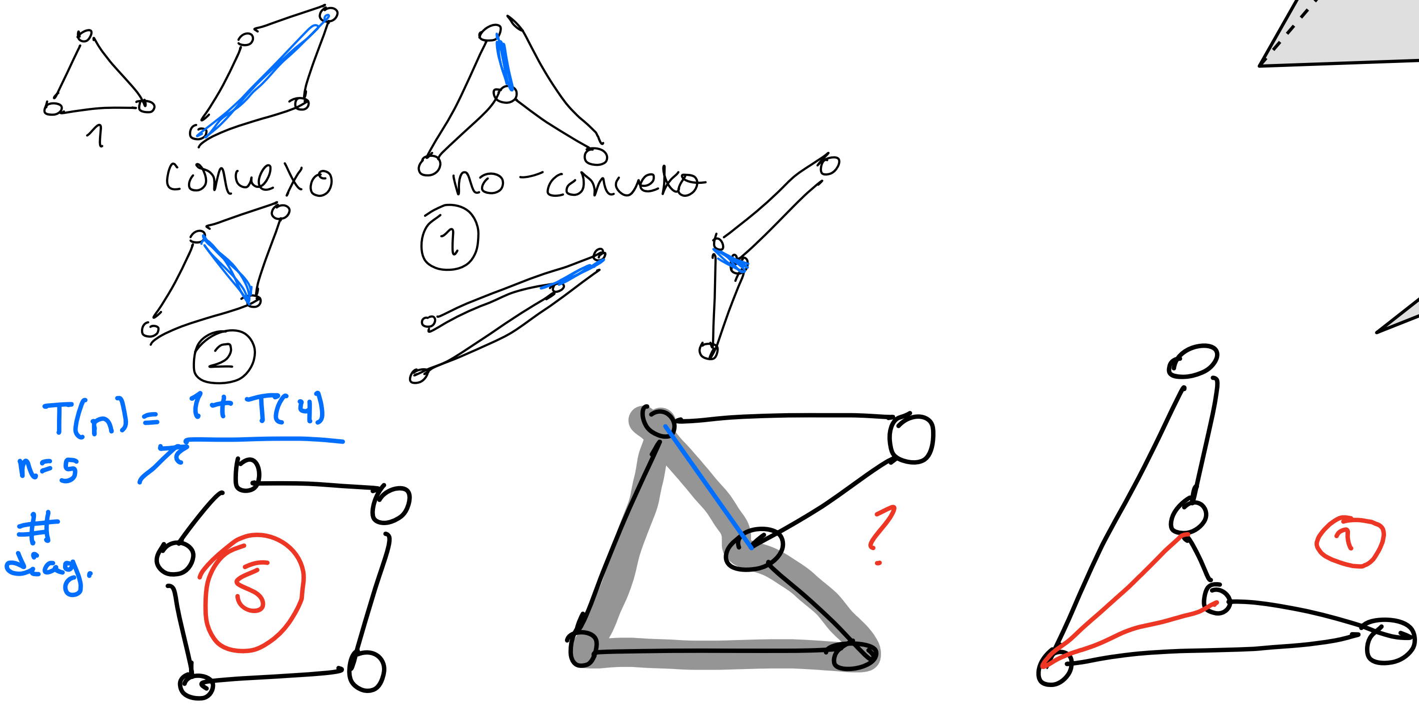
A **polygon triangulation** is the decomposition of a polygon into triangles. This is done by inserting internal diagonals.

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

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

¿Cuántas Δ's tiene P?





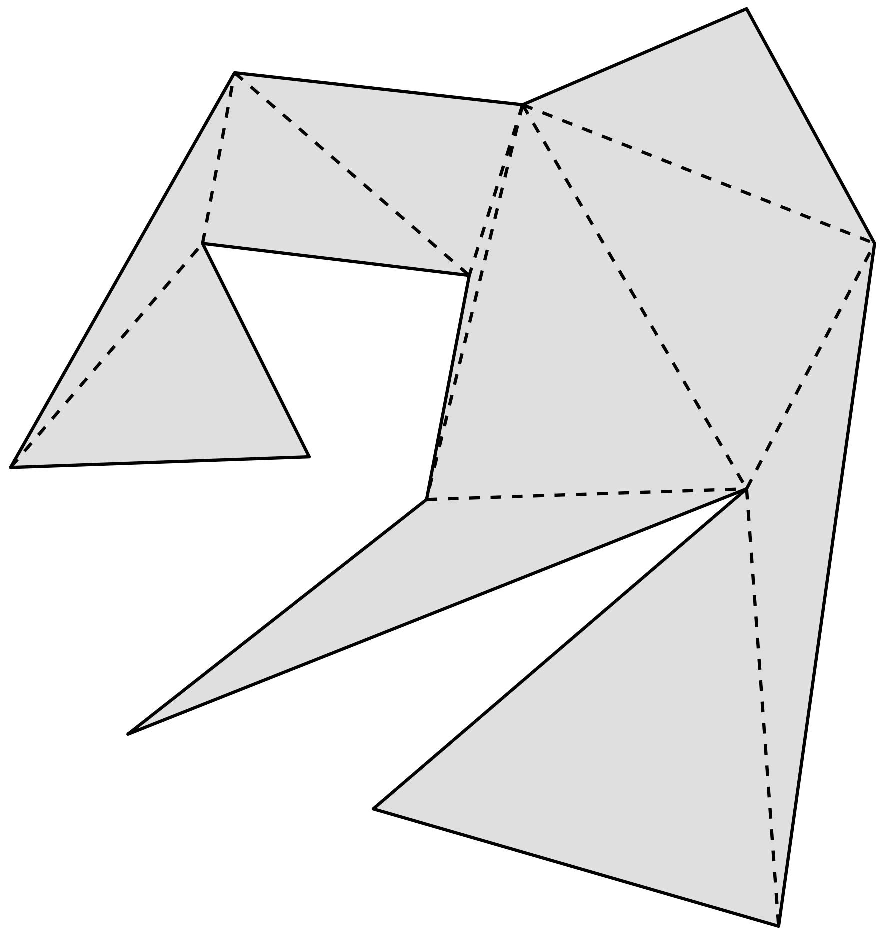
∘ P convexo
#Catalán → exp.

TRIANGULATING POLYGONS

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.

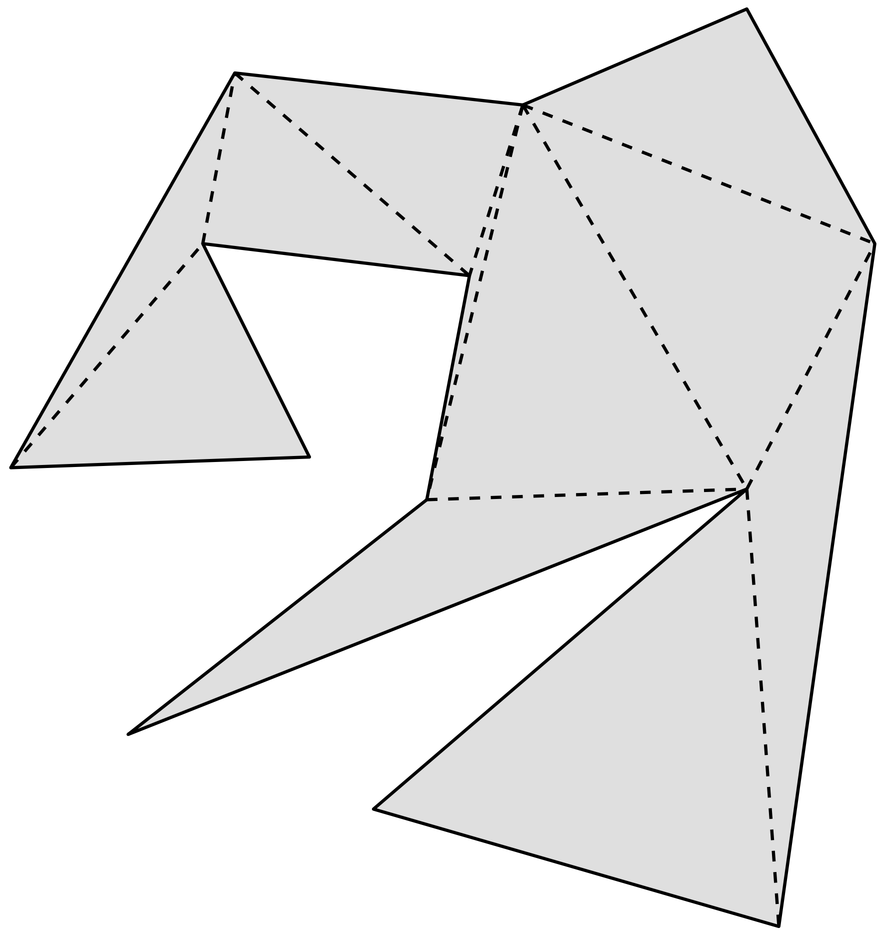


An application example:

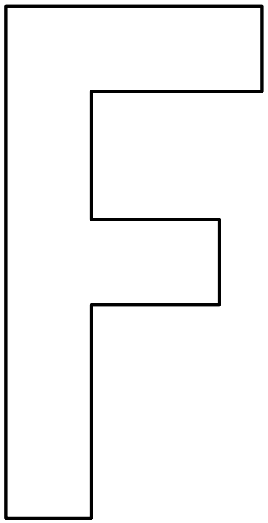
A **polygon triangulation** is the decomposition of a polygon into triangles. This is done by inserting internal diagonals.

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

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



An application example:

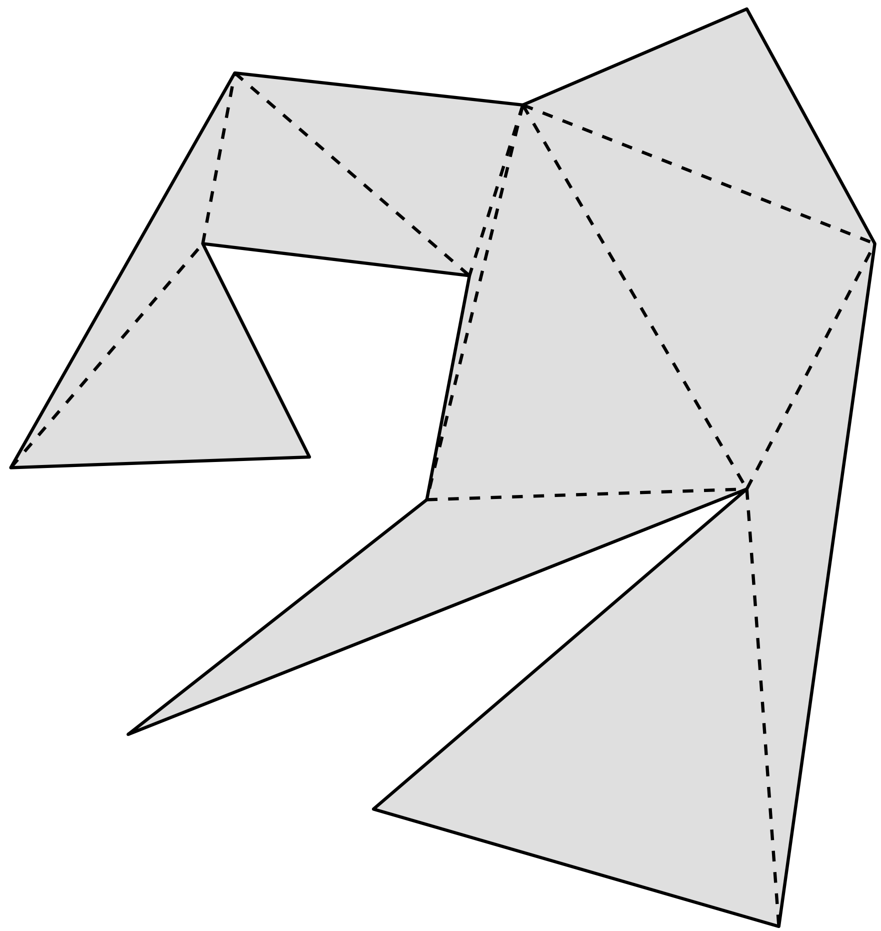


TRIANGULATING POLYGONS

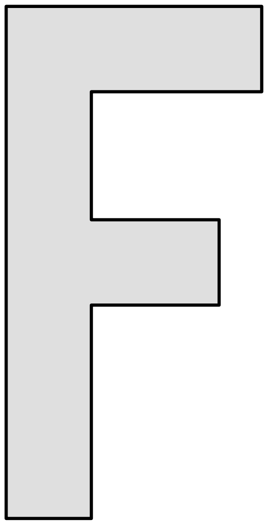
A **polygon triangulation** is the decomposition of a polygon into triangles. This is done by inserting internal diagonals.

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

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



An application example:

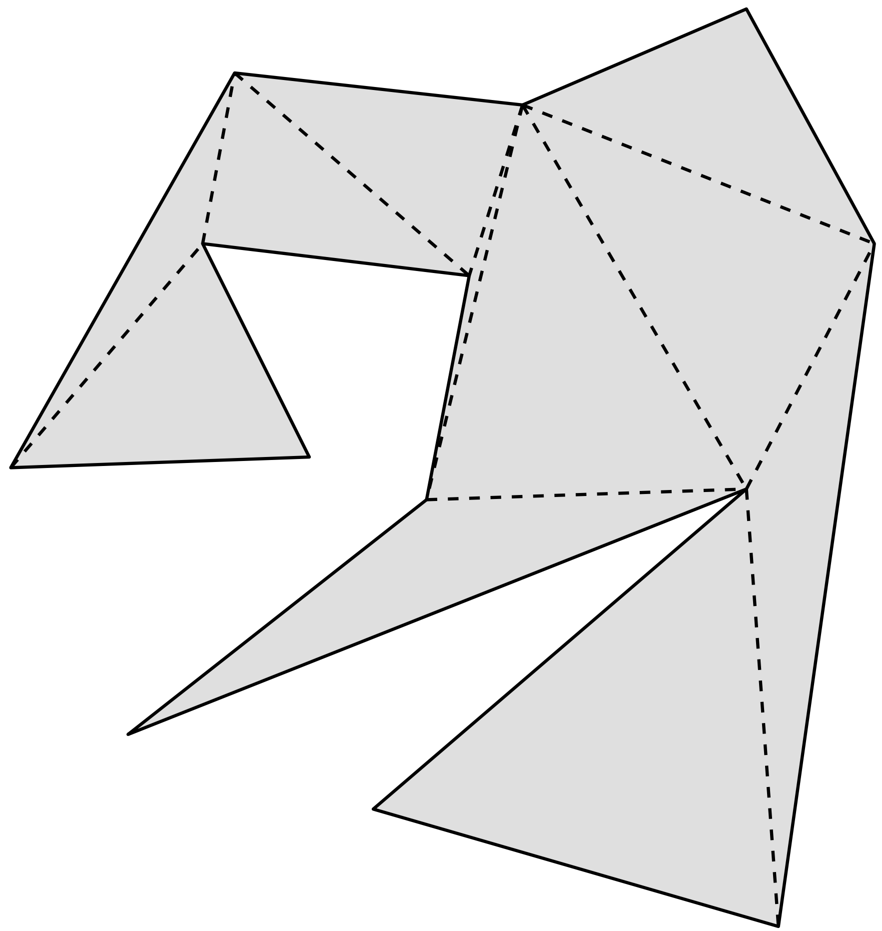


TRIANGULATING POLYGONS

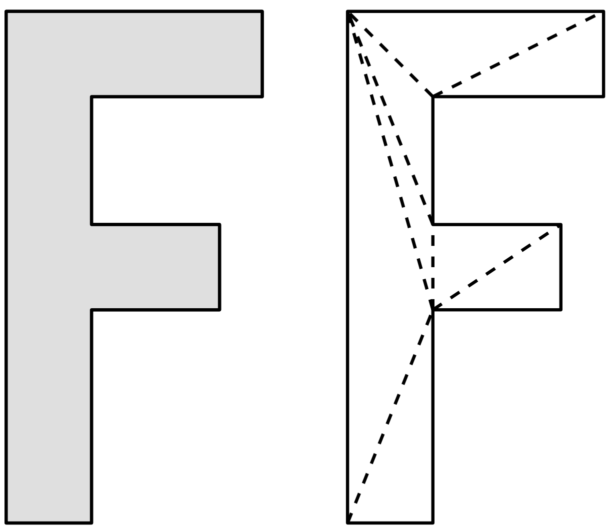
A **polygon triangulation** is the decomposition of a polygon into triangles. This is done by inserting internal diagonals.

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

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



An application example:

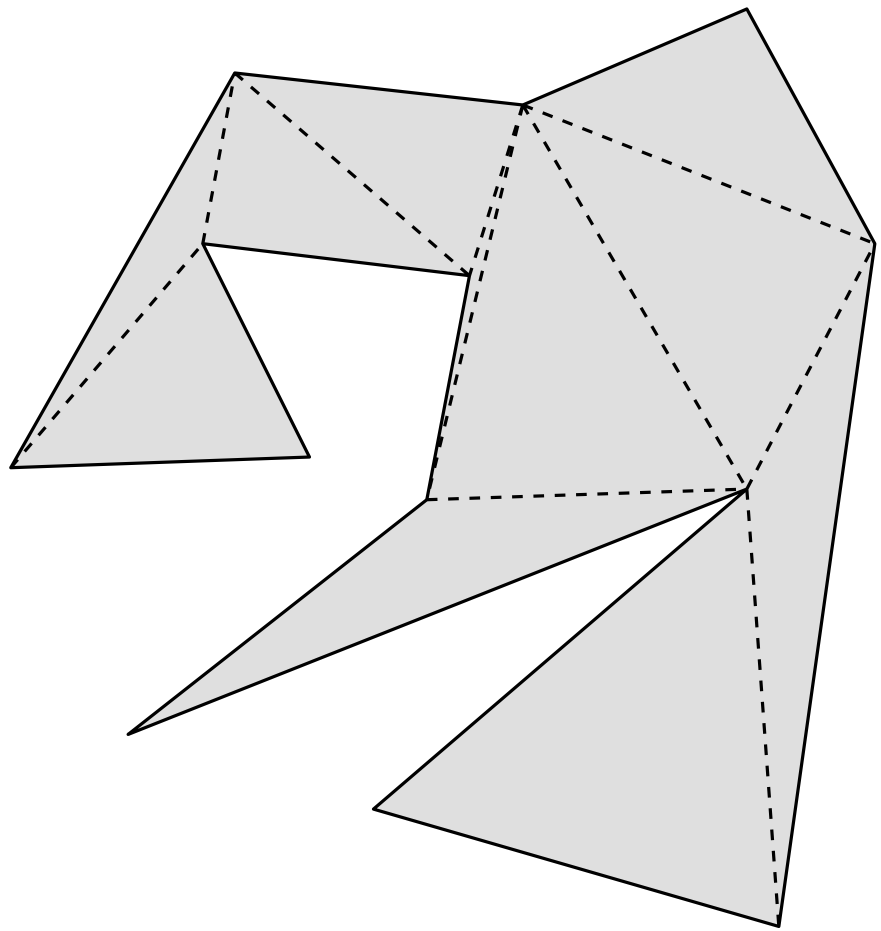


TRIANGULATING POLYGONS

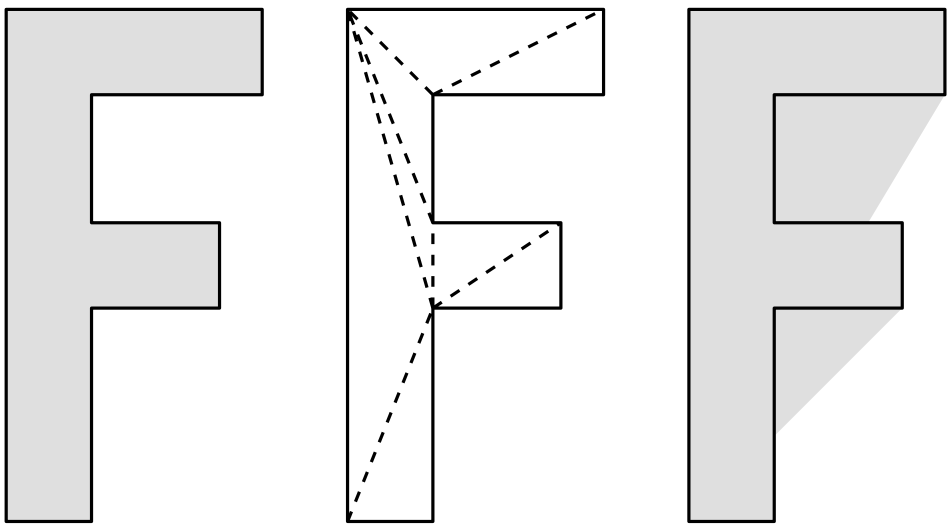
A **polygon triangulation** is the decomposition of a polygon into triangles. This is done by inserting internal diagonals.

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

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



An application example:

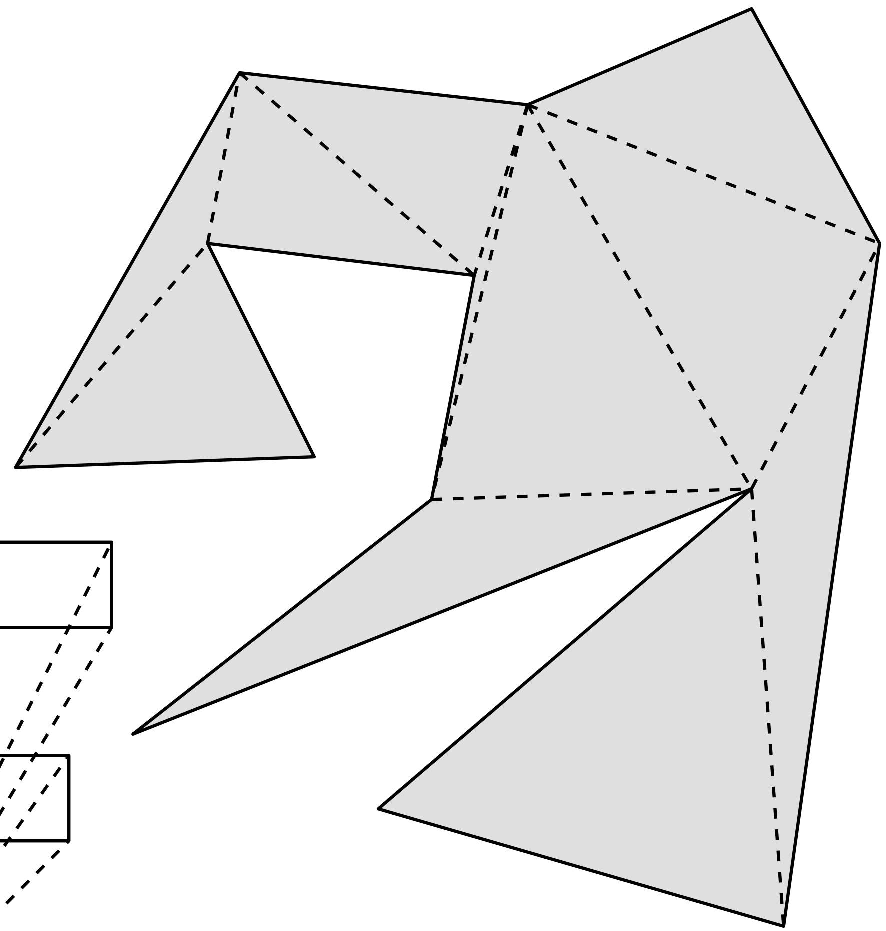


TRIANGULATING POLYGONS

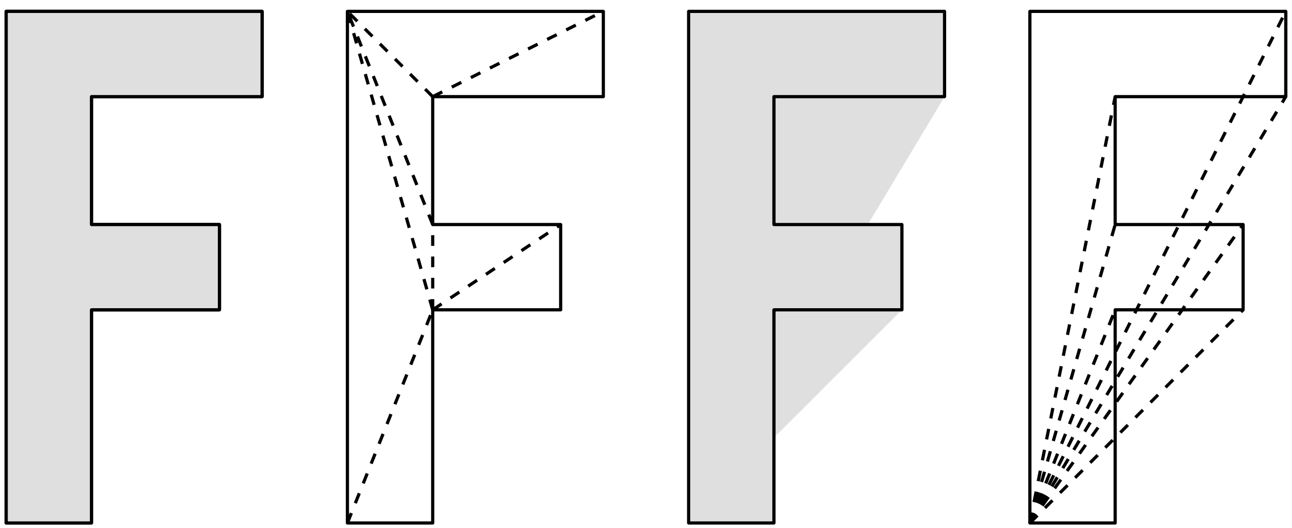
A **polygon triangulation** is the decomposition of a polygon into triangles. This is done by inserting internal diagonals.

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

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

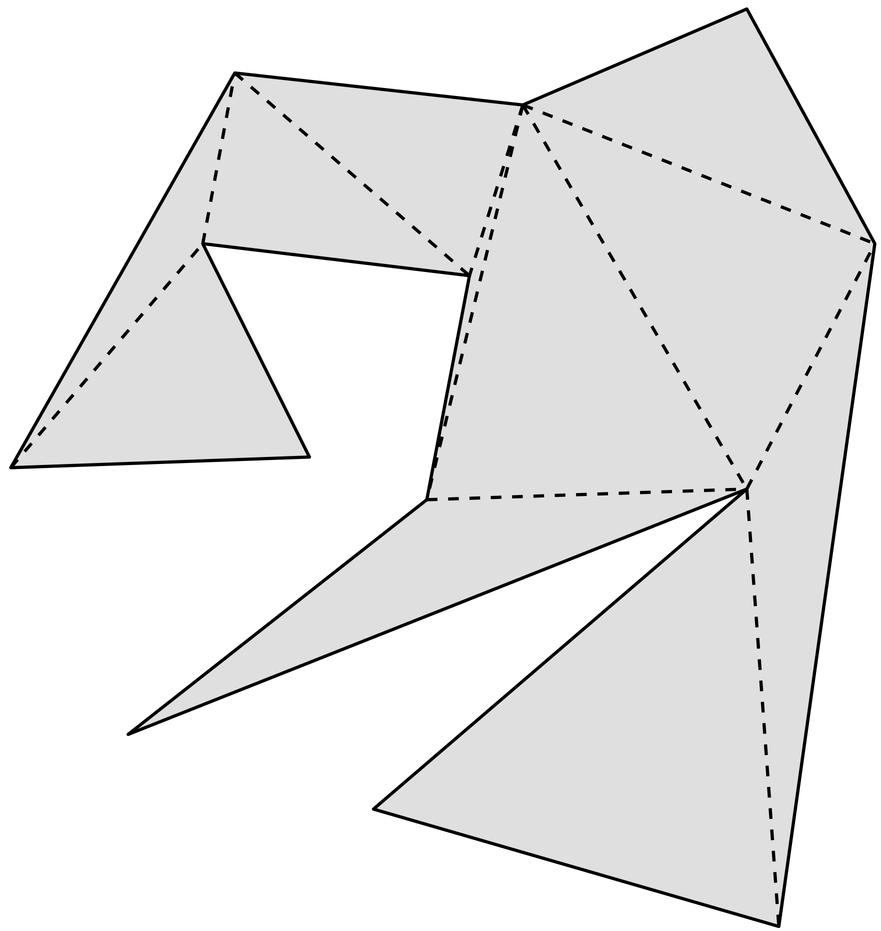


An application example:



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 partinioning

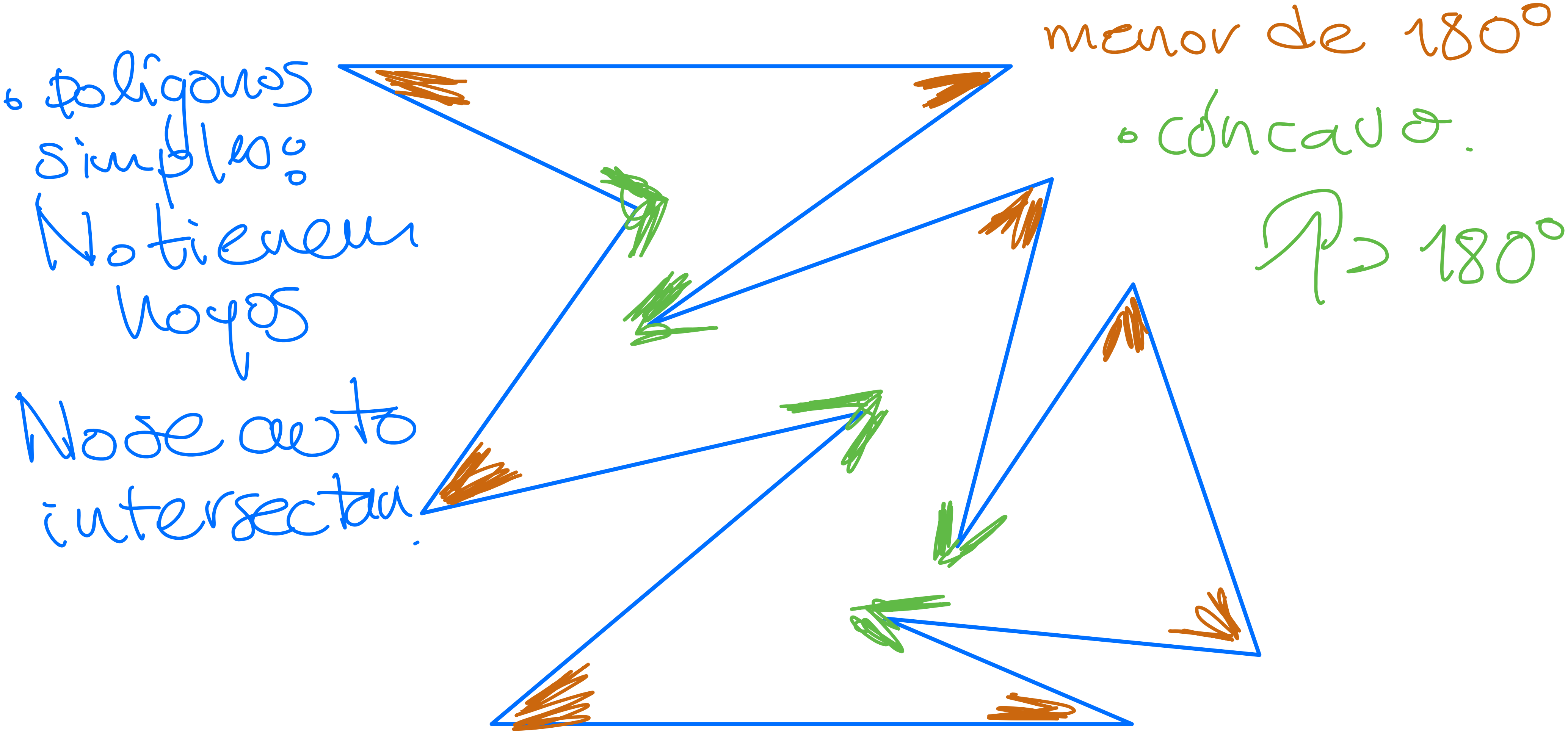


TRIANGULATING POLYGONS

Every polygon admits a triangulation → descomposición en triángulos.

Every polygon admits a triangulation

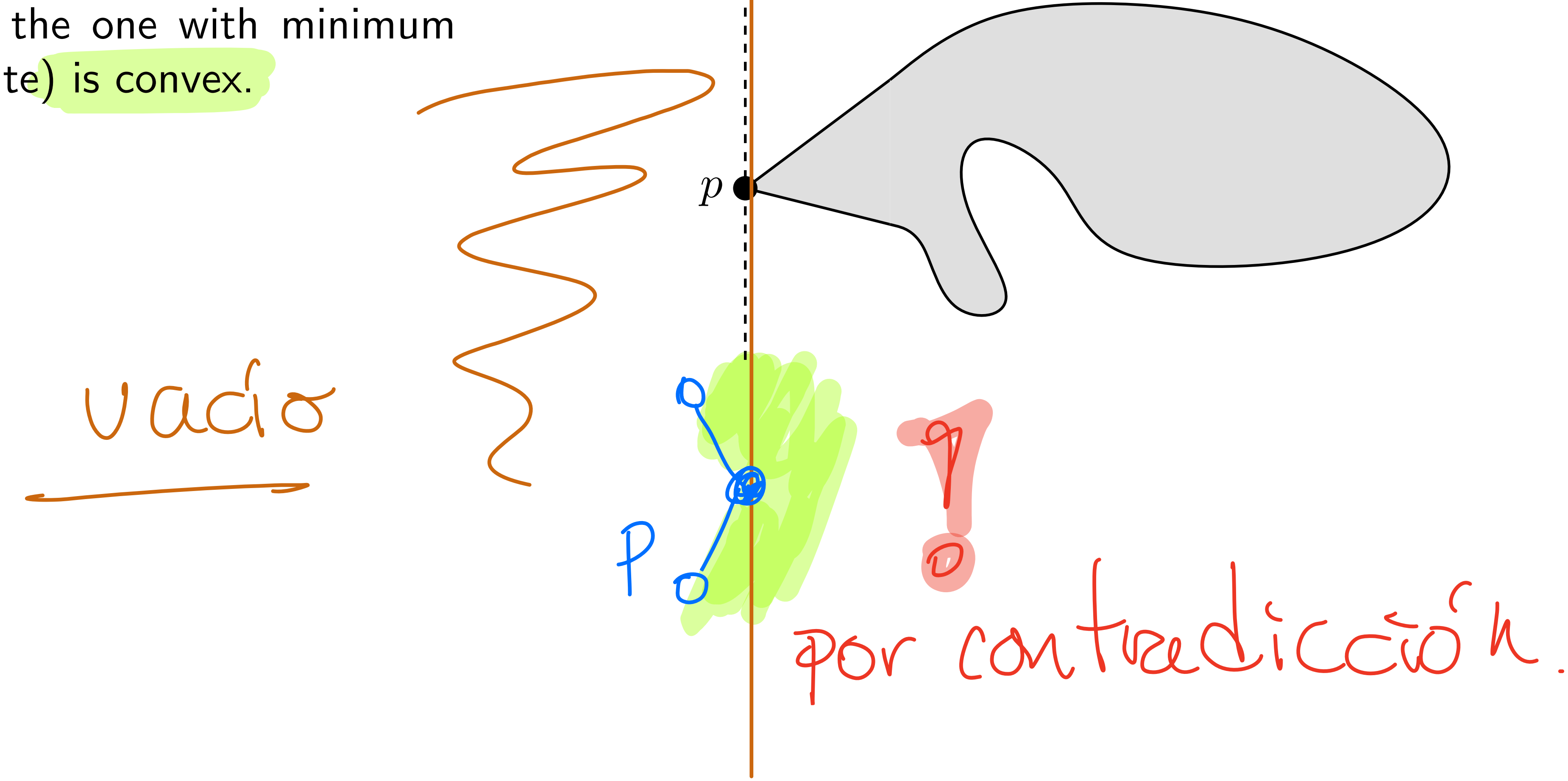
Lemma 1. Every polygon has at least one convex vertex (actually, at least three).



Every polygon admits a triangulation

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

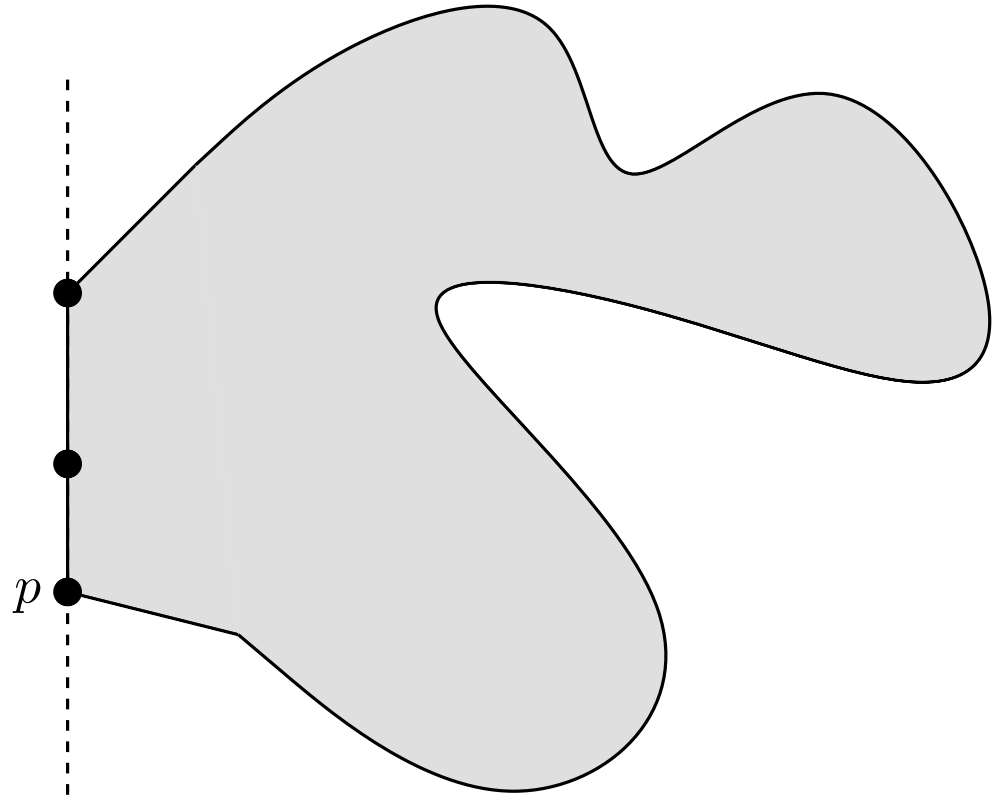
The vertex p with minimum x-coordinate (and, if there are more than one, the one with minimum y-coordinate) is convex.



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.

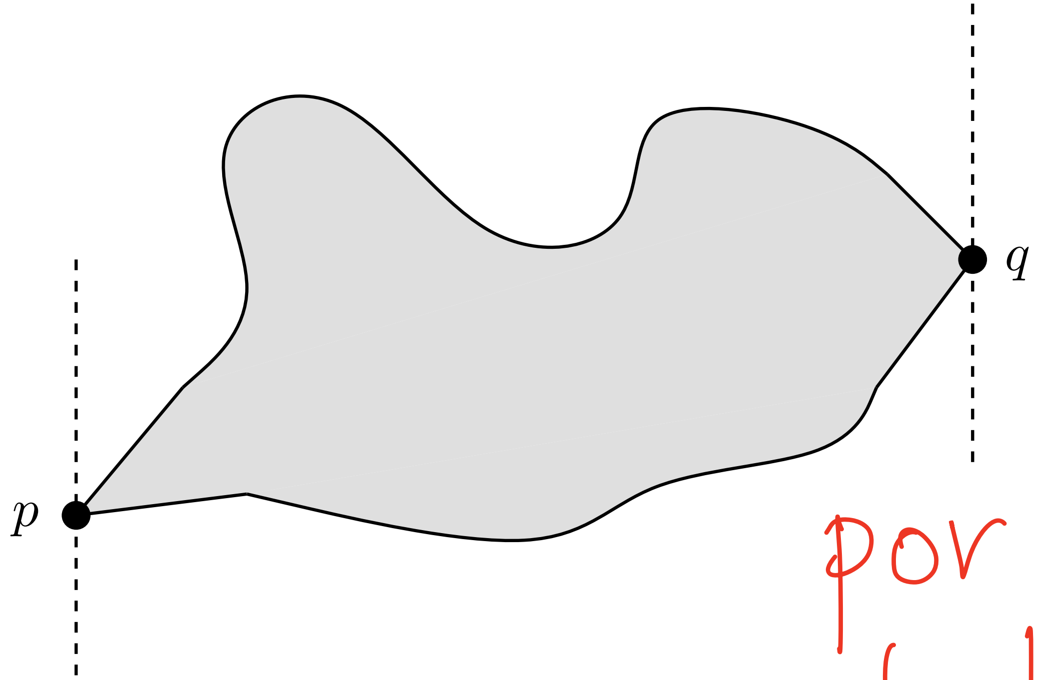


Every polygon admits a triangulation

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).



por contradicción,

TRIANGULATING POLYGONS

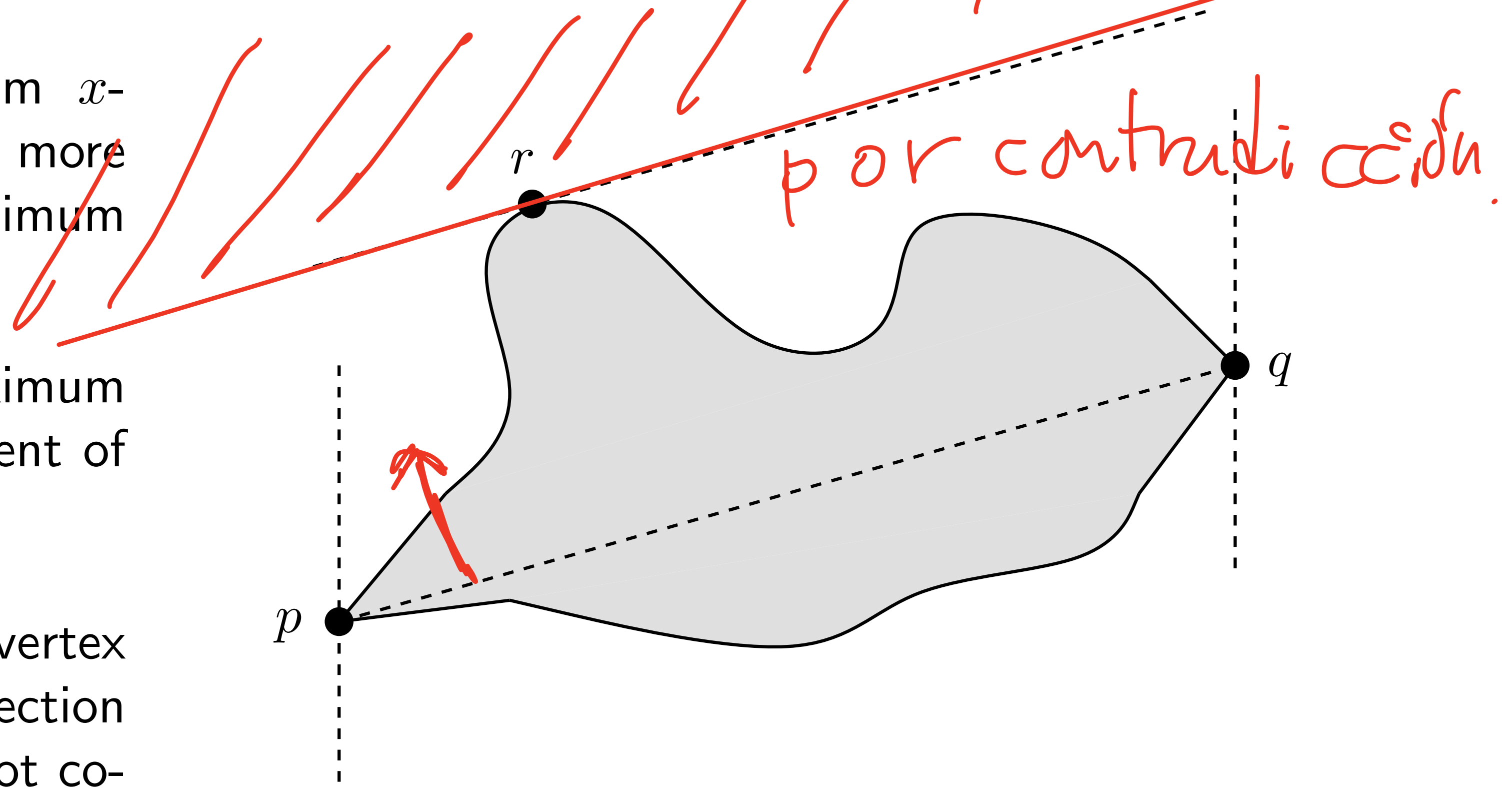
Every polygon admits a triangulation

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.

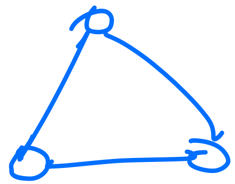


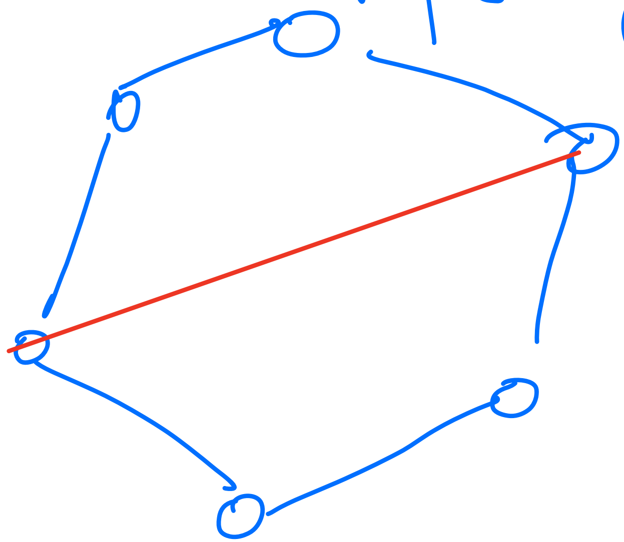
Every polygon admits a triangulation

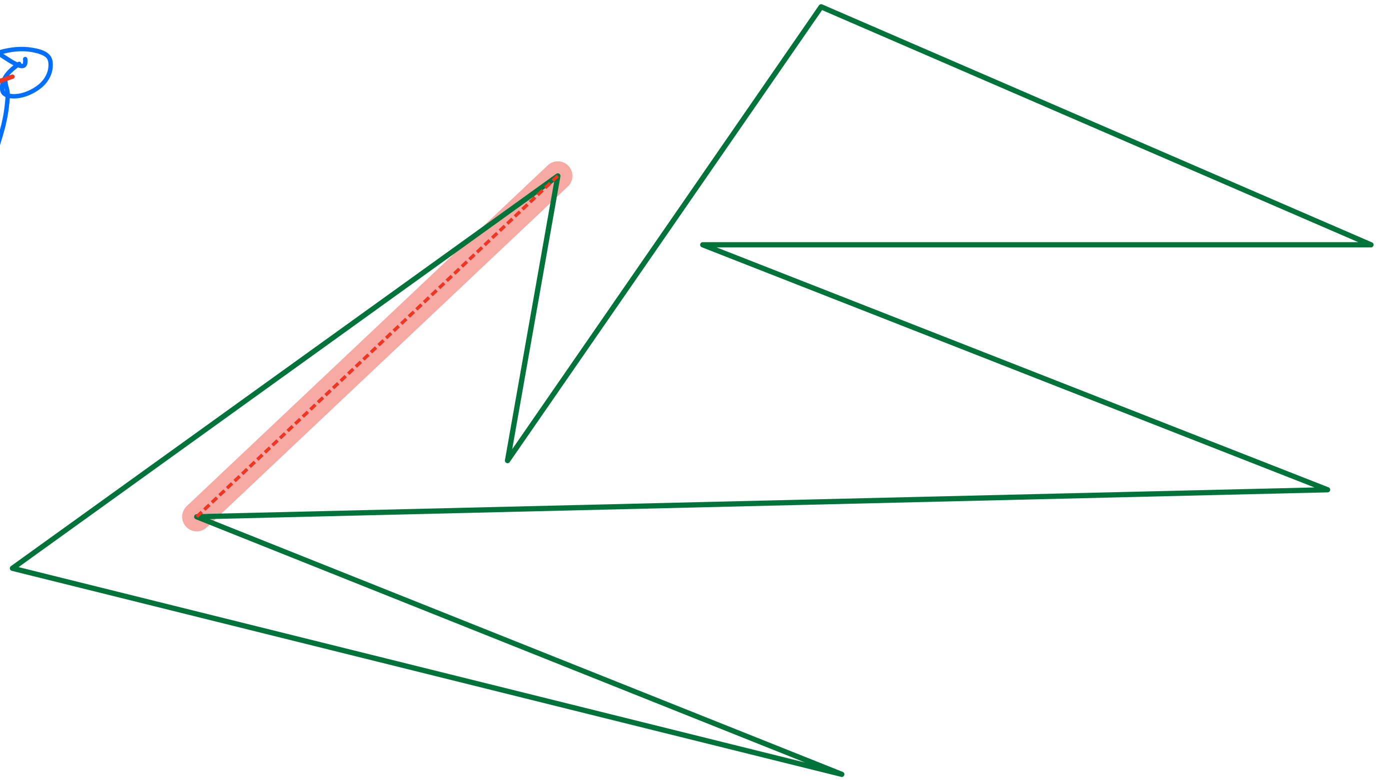
Lemma 1. Every polygon has at least one convex vertex (actually, at least three).

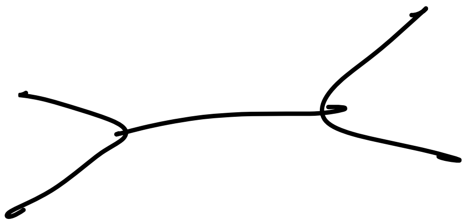
Lemma 2. Every n-gon with $n \geq 4$ has at least one internal diagonal.

#polígono con n vértices.









TRIANGULATING POLYGONS

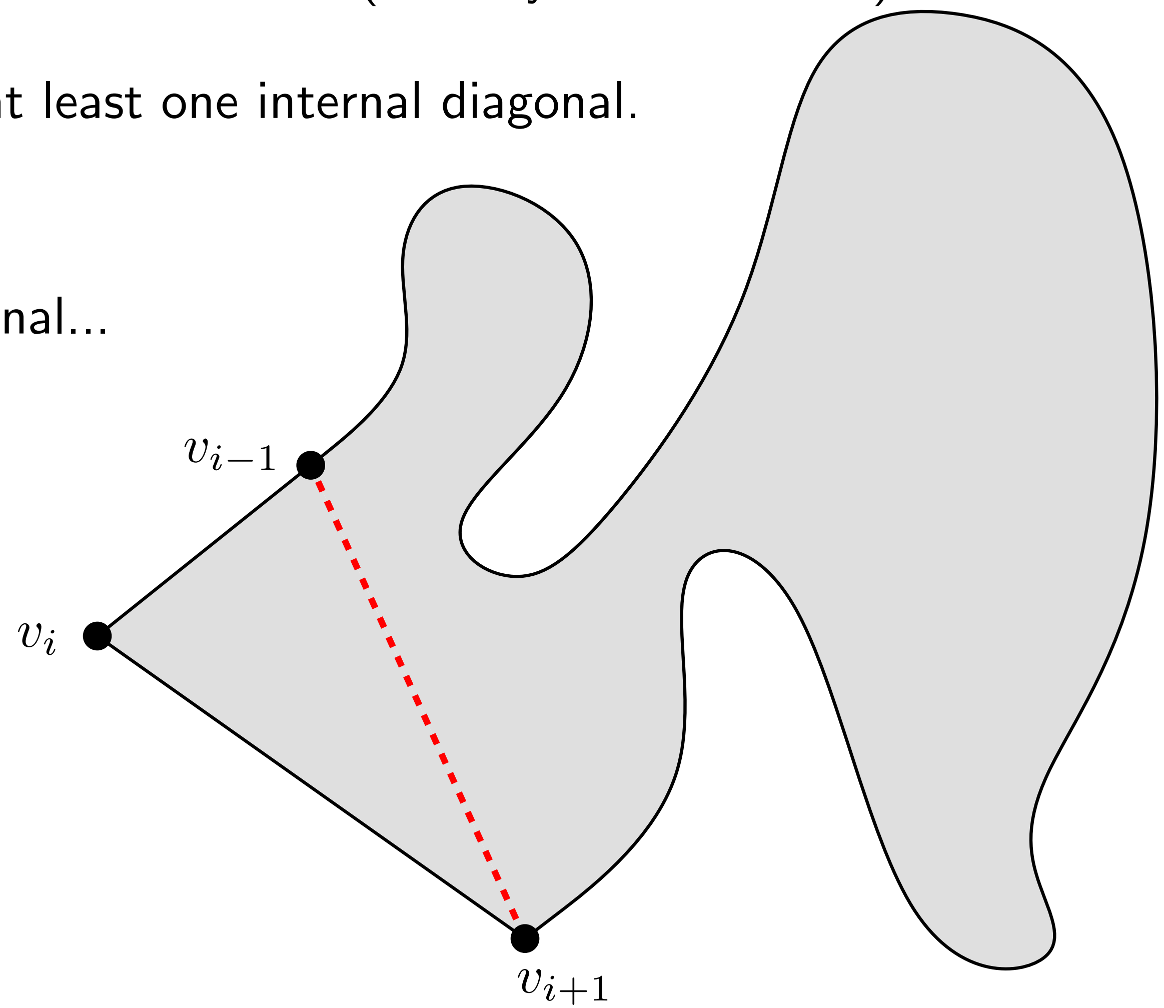
Every polygon admits a triangulation

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

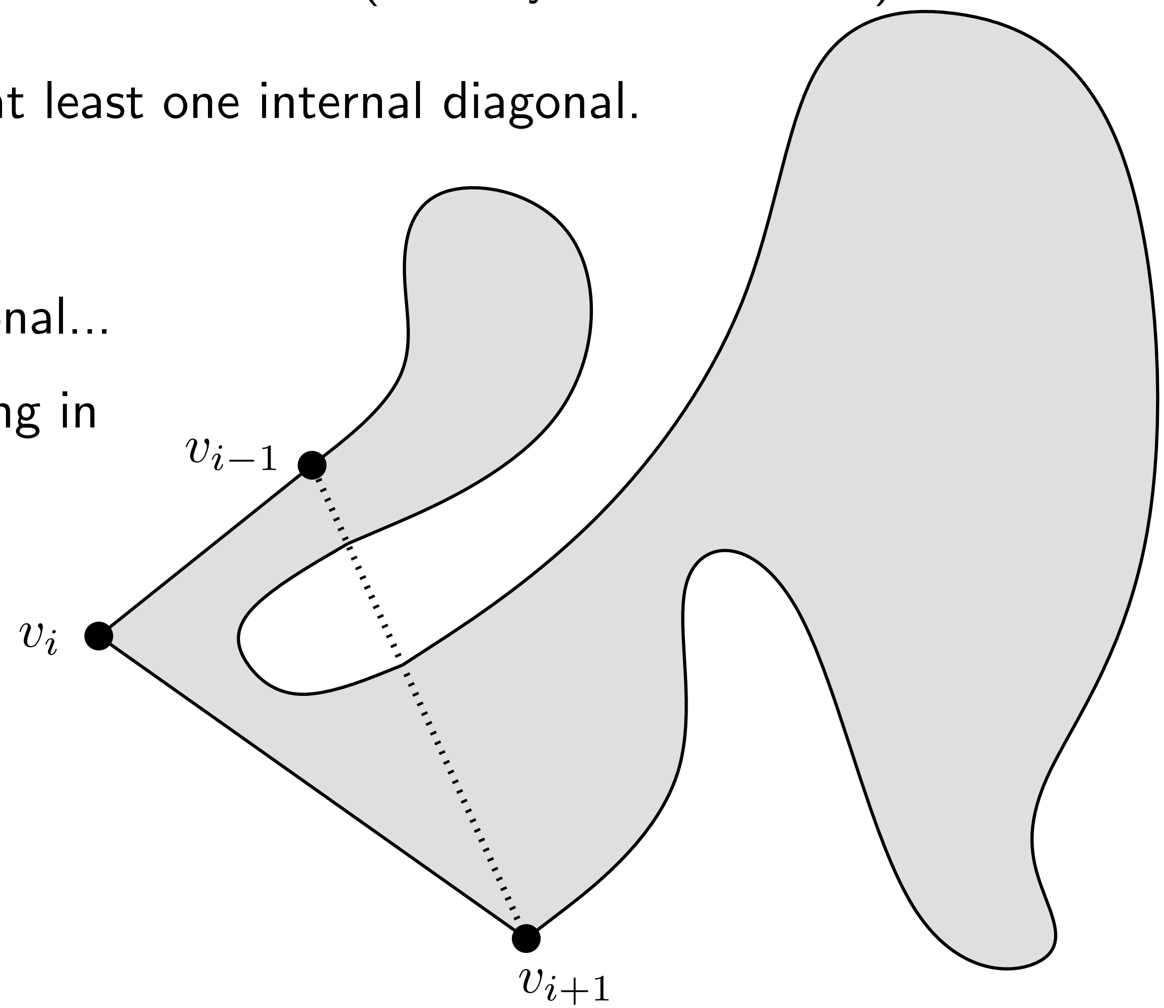
Lemma 1. Every polygon has at least one convex vertex (actually, at least three).

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).

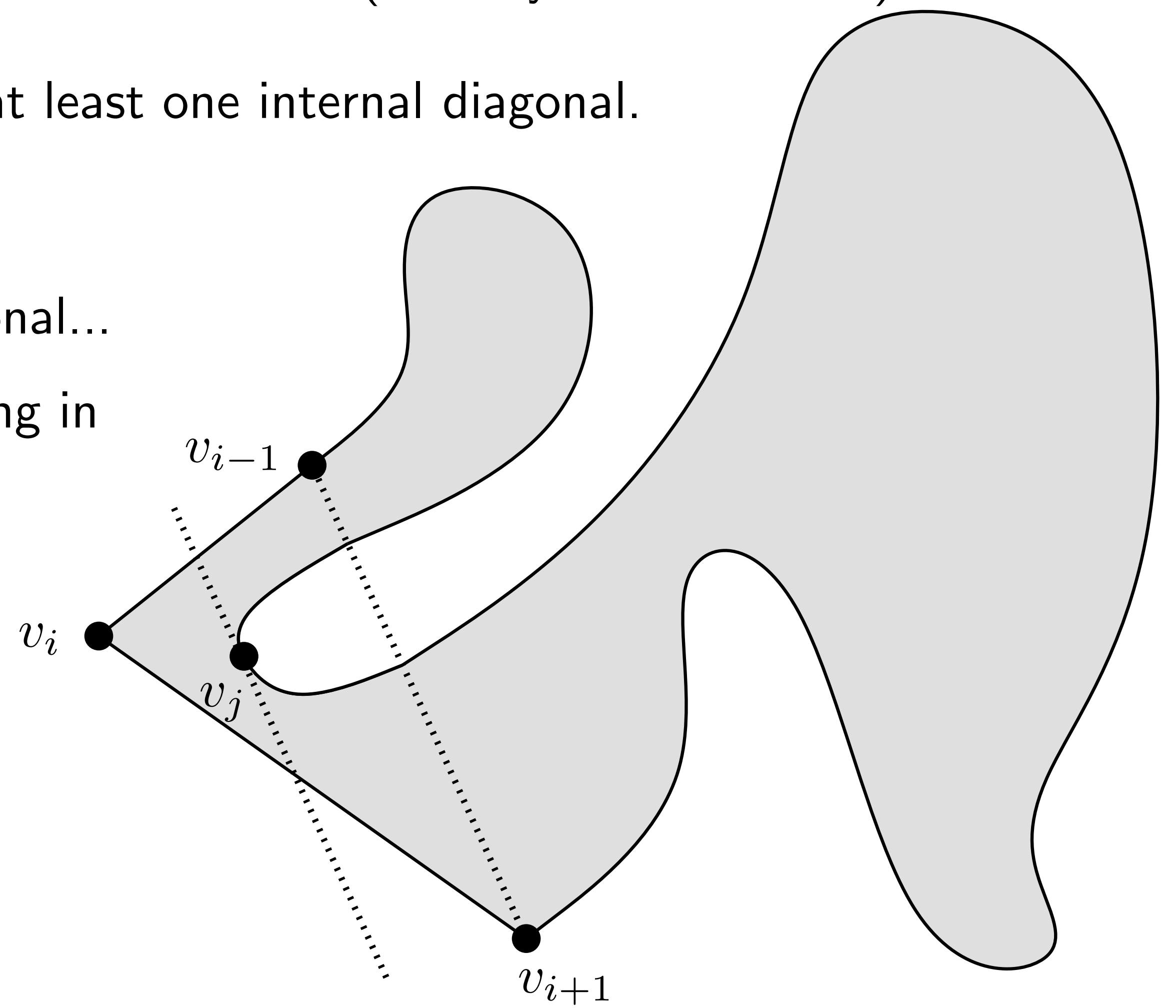
Lemma 2. Every n-gon with $n \geq 4$ has at least one internal diagonal.

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).



TRIANGULATING POLYGONS

Every polygon admits a triangulation

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

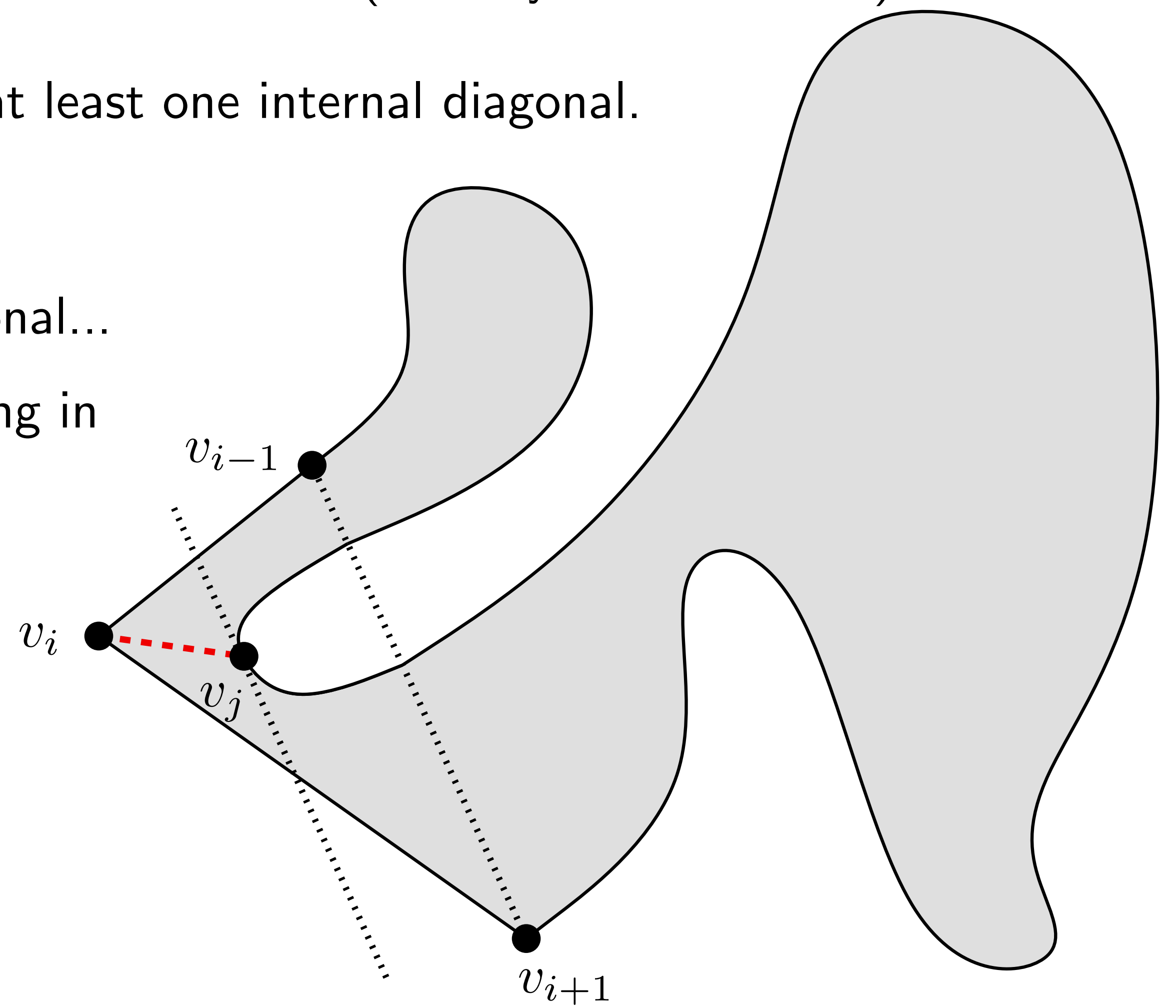
Lemma 2. Every n-gon with $n \geq 4$ has at least one internal diagonal.

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).



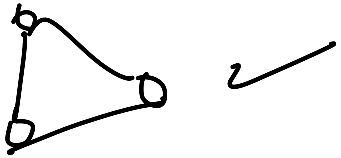
Every polygon admits a triangulation

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.

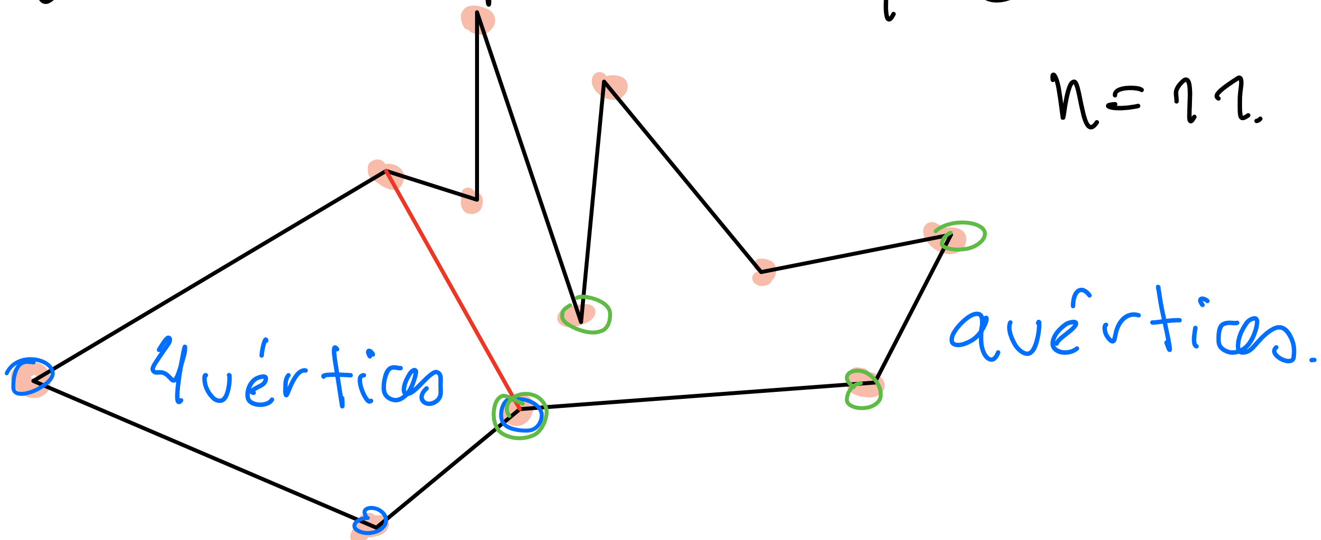
Corollary. Every polygon can be triangulated. (By induction.) → sobre n.

Caso base $n=3$.



Para $n \geq 3$ $P = P_1 \cup P_2$

n=11.



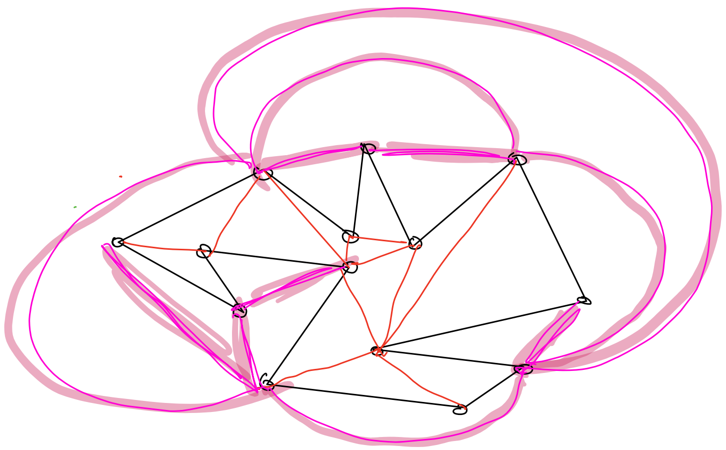
Properties of the triangulations of polygons

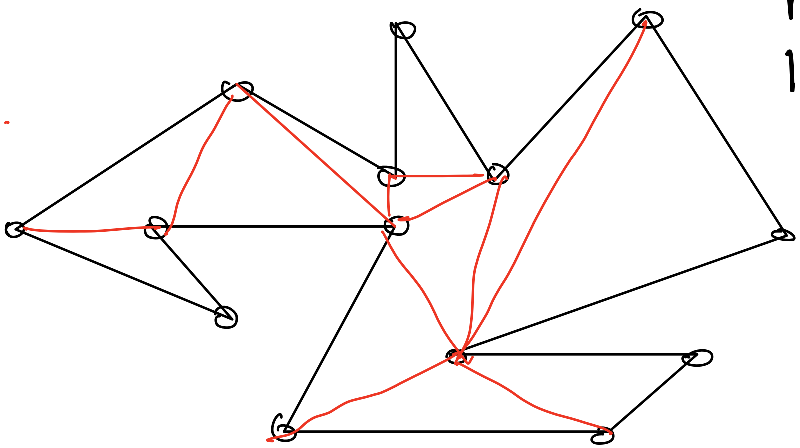
Properties of the triangulations of polygons

Let P be a simple n-gon.

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

$V - E + F = 2$





aristas = 14

n= 14

11 diagonals

12 Δ's.

¿ "identidad de Euler"?

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.

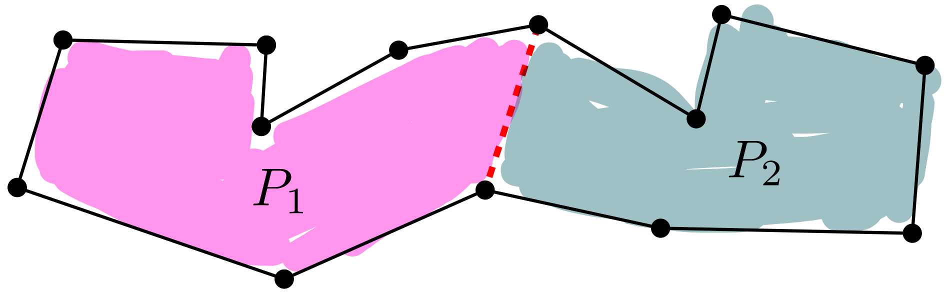
Base case: When $n=3$, the number of diagonals is $d=0=n-3$.

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

k+1 vértices.



n- k+1 vértices.

Properties of the triangulations of polygons

Let P be a simple n-gon.

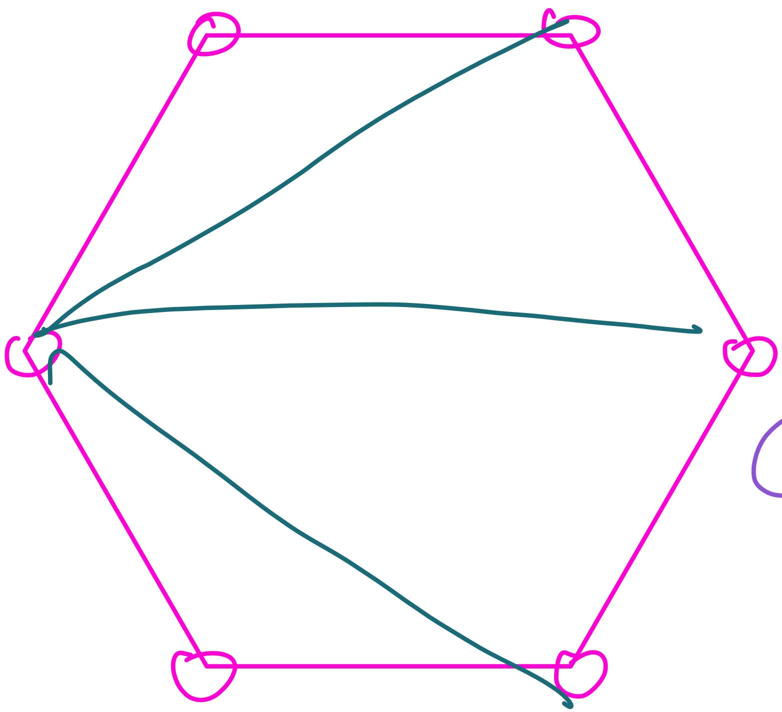
Property 1. Every triangulation of P has $n-3$ diagonals.

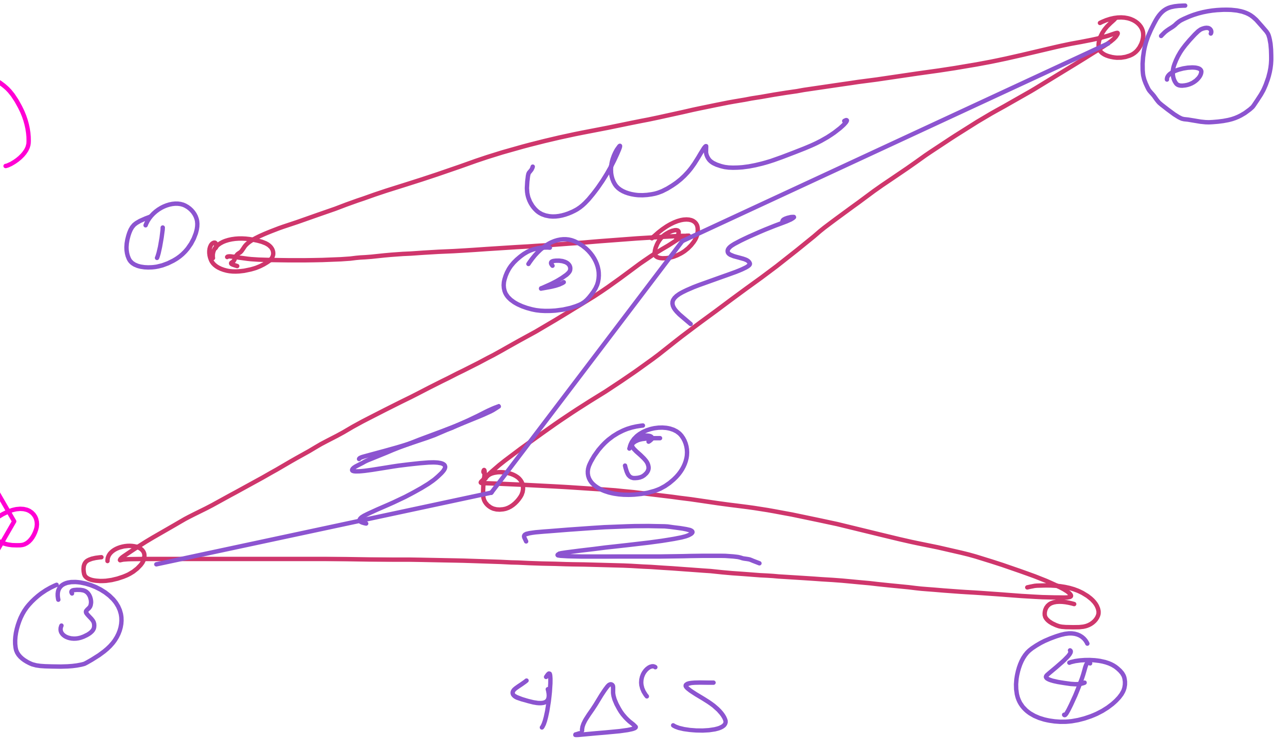
Property 2. Every triangulation of P has $n-2$ triangles.

n=6

n

n+(n-3)





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.

Teorema. (Identidad de Euler).
Para cada gráfica conexa aplanable con V vértices, E aristas y F caras ocurre que:

$$V-E+F=2.$$

Teorema: Si G es una gráfica aplanable con $V\geq 3$ y E aristas, entonces:

$$E\leq 3V-6.$$

Lema. El # de diagonales exteriores en una gráfica plana maximal es $n-3$.

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.

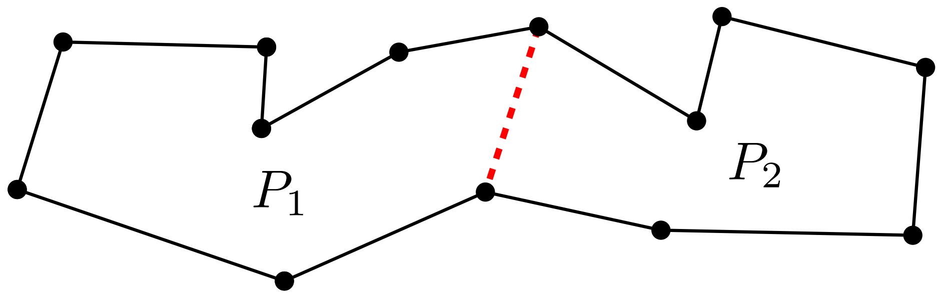
Base case: When $n=3$, the number of triangles is $t=1=n-2$.

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

$$t_1 = k+1-2,$$
$$t_2 = n-k+1-2,$$

hence,

$$t = t_1 + t_2 = k+1-2+n-k+1-2 = n-2.$$



Properties of the triangulations of polygons

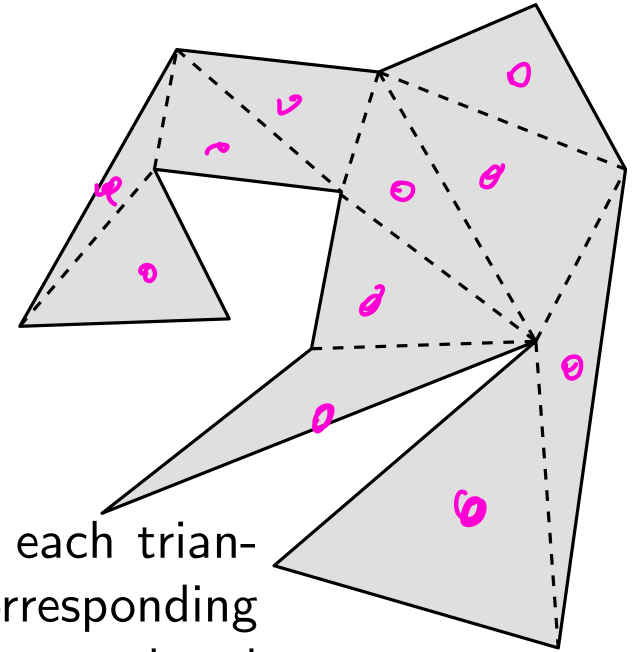
Let P be a simple n-gon.

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.

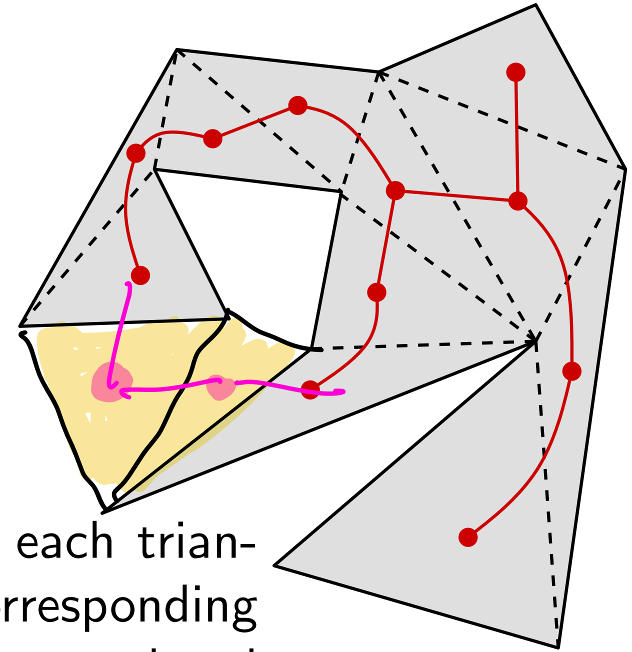
Property 2. Every triangulation of P has $n-2$ triangles.

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.

TRIANGULATING POLYGONS

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.

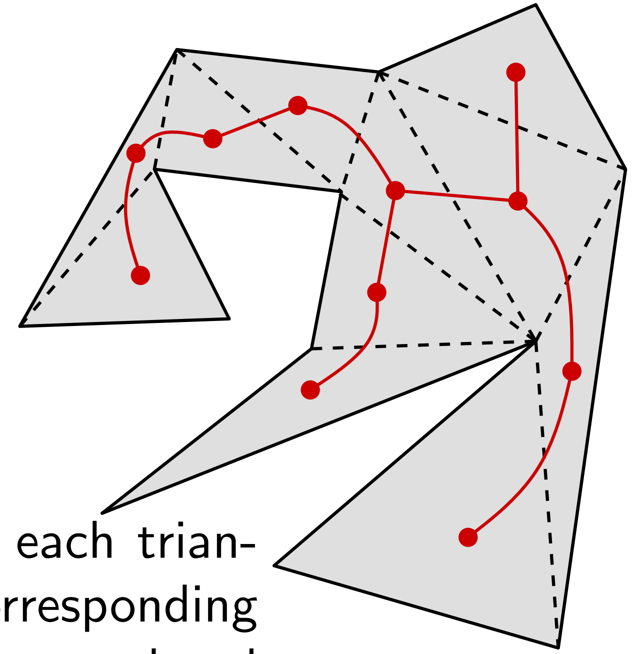
Given a triangulation of P, its dual graph has one vertex for each triangle, and one edge connecting two vertices whenever their corresponding triangles are adjacent. We want to prove that this graph is connected and acyclic.

V = un vértice por cada triangulo
E = { (vi, vj) | el triángulo correspond. a vi y el corresp. a vj tienen una diagonal en común }
|V| = n-2 ; |E| = n-3

① conexa
② acíclica. } para demostrar que es un árbol.

TRIANGULATING POLYGONS

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.

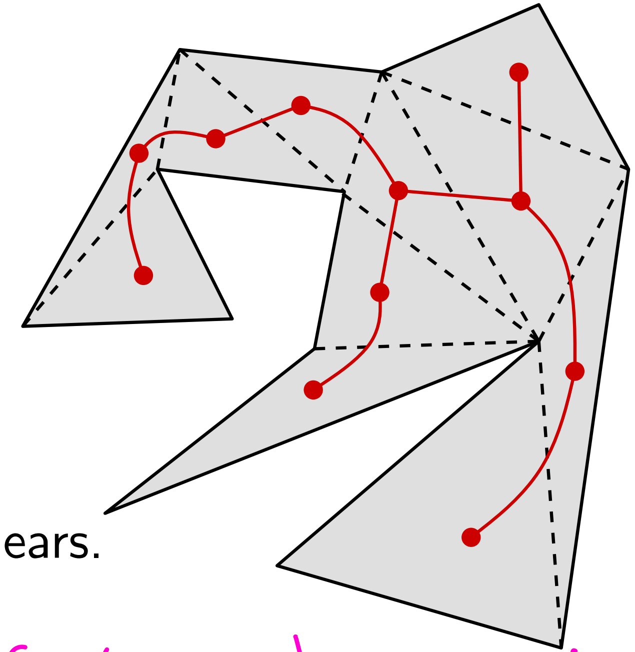
Given a triangulation of P, its dual graph has one vertex for each triangle, and one edge connecting two vertices whenever their corresponding triangles are adjacent. We want to prove that this graph is connected and acyclic.

① conexa → desconectado
② acíclica → hoyos.

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.

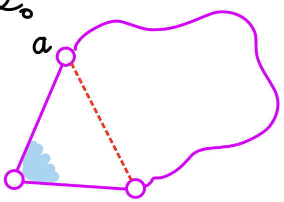
Property 2. Every triangulation of P has $n-2$ triangles.

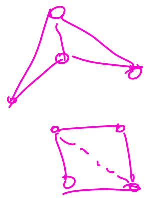
Property 3. The dual graph of any triangulation of P is a tree.

Corollary. Every n-gon with $n \geq 4$ has at least two non-adjacent ears.

? → Revisar.

Oreja: Sean a, b, c tres vértices consecutivos de P. Decimos que a, b, c forman una oreja si $\overline{ac}$ es una diagonal.





vértice de grado 1 = hoja.

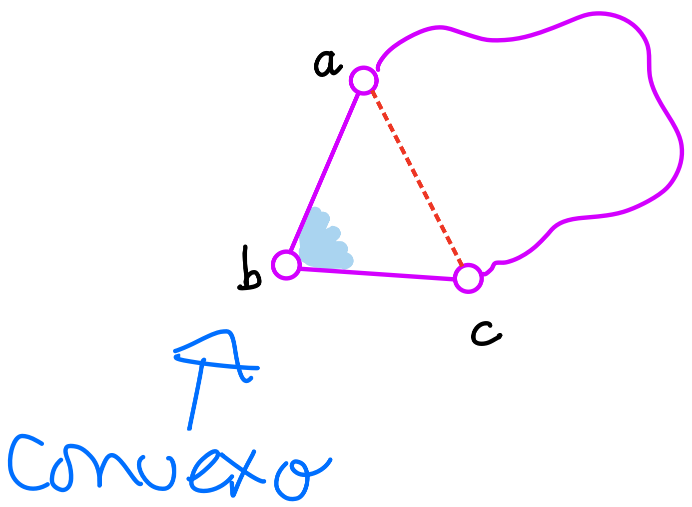
al remover el △ el polígono no se desconecta.
$\overline{ab}$ y $\overline{bc}$ son aristas del polígono

En la gráfica dual corresp. a hojas.

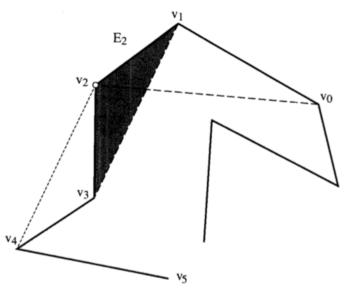
Bastaría con demostrar que todo árbol tiene al menos 2 hojas.

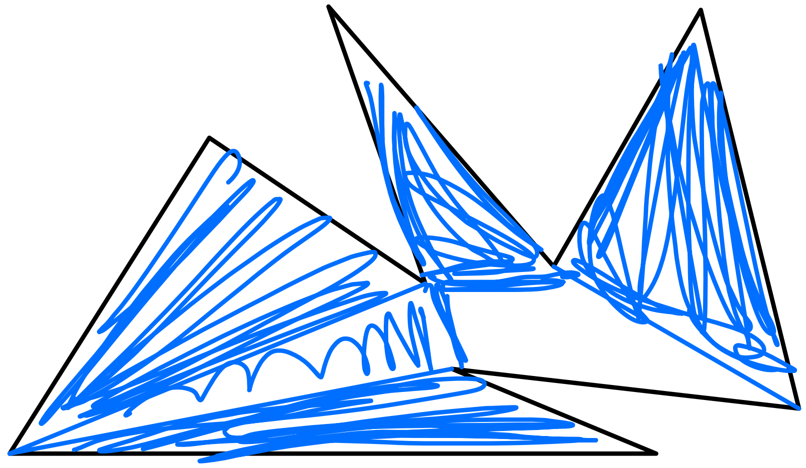
ALGORITHMS FOR POLYGON TRIANGULATION

Tringulating a polygon by subtracting ears



1. no cruza
2. está en el interior
3. Removerla separa el polígono en un Δ y un polígono de n-1 vértices.





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.

Tringulating a polygon by subtracting ears

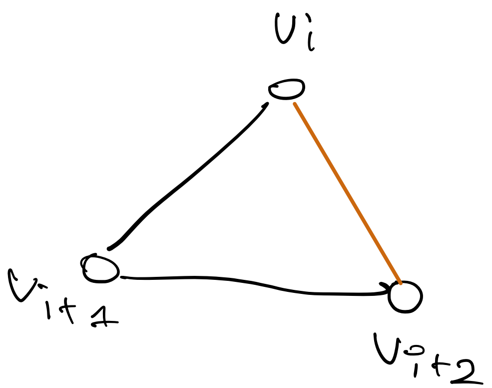
Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

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

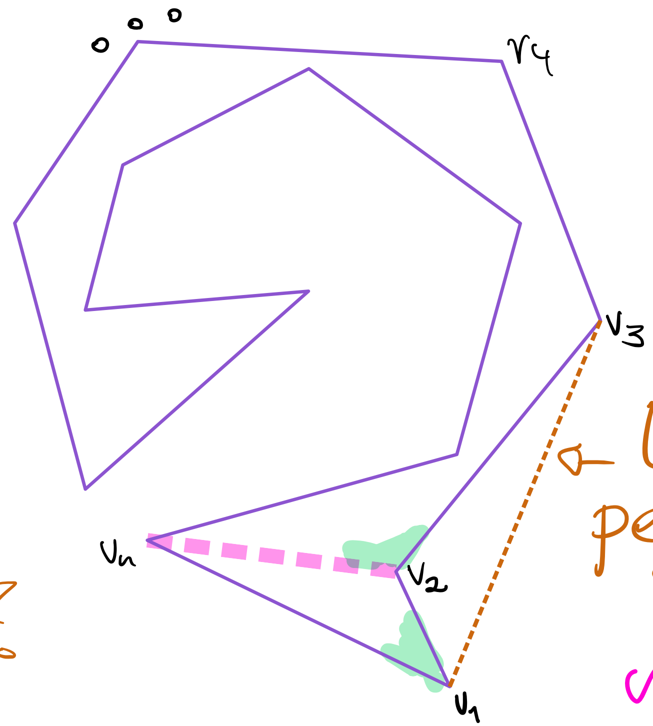
Procedure

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

únicas tercias a verificar.



¿(v_i, v_{i+2}) es diagonal? mod n



• ∄ es convexo O(1)

⊙ O(n) para determinar si es diagonal

↳ Revisar todas las aristas del polígono.

← (v_3, v_1) es diag. pero no es interior.

no ocurre: por la elección de v_i, v_{i-1}, v_{i+1}.

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. Crop it
3. Proceed recursively

Running time

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. Crop it
3. Proceed recursively

Running time

Detecting whether a vertex is convex: $O(1)$.

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_iv_j, determining a triangulation of P.

Procedure

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

Running time

Detecting whether a vertex is convex: $O(1)$.
Detecting whether a convex vertex is an ear: $O(n)$.

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.

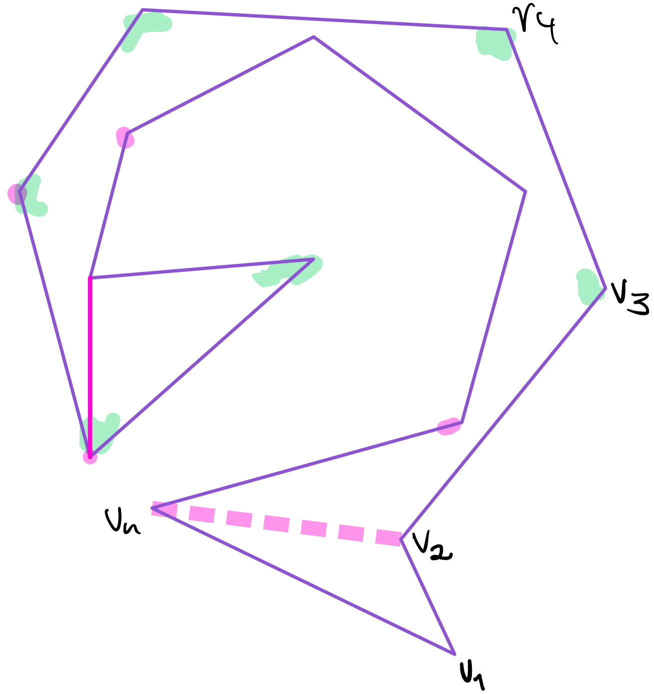
Procedure

1. Sequentially explore the vertices until you find an ear
2. Crop it
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: $O(n^2)$.

una.



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. Crop it
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:

$$T(n) = O(n^2) + O((n-1)^2) + O((n-2)^2) + \cdots + O(1) = O(n^3).$$

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.

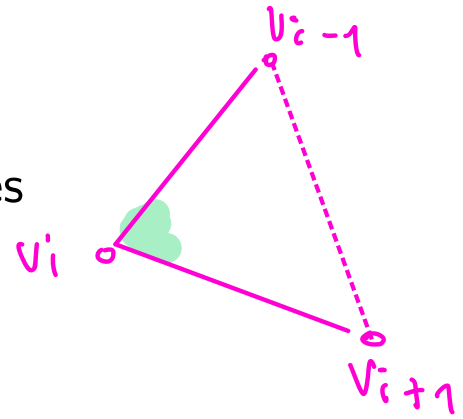
Improved procedure

Initialization

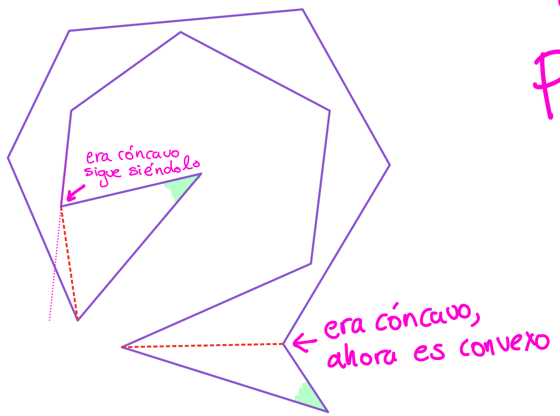
1. Detect all convex vertices
2. Detect all ears

Next step

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



- los únicos ángulos afectados son el ∡ con ápice en v_{i-1} y con ápice en v_{i+1}.
- Si v_{i-1} (v_{i+1}) era convexo sigue siendo convexo.
- Si v_{i-1} (v_{i+1}) era cóncavo podría ser ahora convexo.



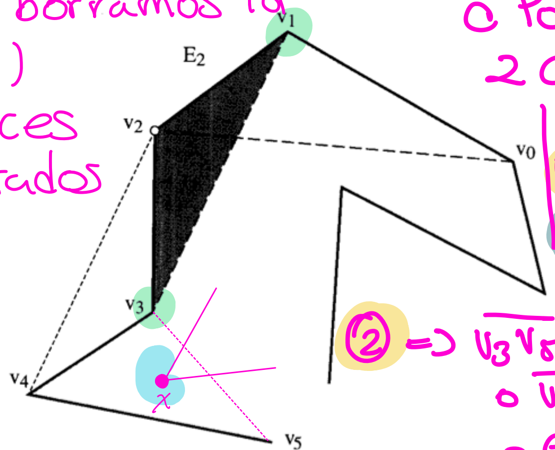
Tringulating a polygon by subtracting ears

Input: $v_1, \dots, v_n$, 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	**Running time**	
Initialization		
1. Detect all convex vertices	$O(n)$	Only once
2. Detect all ears	$O(n^2)$	
Next step		
1. Crop an ear	$O(1)$	
2. Update the information of the convex vertices	$O(1)$	$O(n)$ times
3. Update the information of the ears	$O(n)$	

Supongamos que borramos la oreja $\Delta(v_1, v_2, v_3)$ los únicos ápices de orejas afectados son v_1 y v_3.



¿Podría afectarse v_4?

2 casos:

① era ápice de oreja

② no era ápice

① ⇒ $\overline{v_3 v_5}$ es diagon y esto no cambia

② ⇒ $\overline{v_3 v_5}$ no es diagonal.

- $\overline{v_3 v_5}$ es externa
- está bloqueada: $\Delta(v_3, v_4, v_5)$ no está vacío.

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	**Running time**	
Initialization		
1. Detect all convex vertices	$O(n)$	Only once
2. Detect all ears	$O(n^2)$	
Next step		
1. Crop an ear	$O(1)$	
2. Update the information of the convex vertices	$O(1)$	$O(n)$ times
3. Update the information of the ears	$O(n)$	

```
bool   InCone( tVertex a
{
  tVertex a0,a1;     /* a0

  a1 = a->next;
  a0 = a->prev;

  /* If a is a convex vertex ... */
  if( LeftOn( a->v, a1->
```

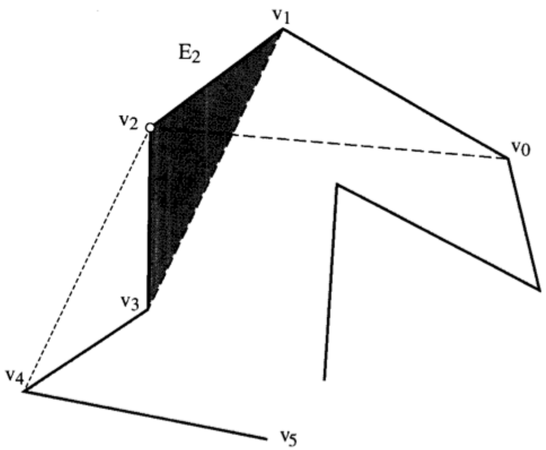
Δ(v_3,v_4,v_5) está bloqueada: Δ(v_3,v_4,v_5) no está vacío.
Sea x el punto en el interior de Δ(v_3,v_4,v_5). x es de hecho un ángulo cóncavo. Por lo tanto, no podría ocurrir que x fuera eliminado al remover la oreja (borramos un ángulo convexo).

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	**Running time**	
Initialization		
1. Detect all convex vertices	$O(n)$	Only once
2. Detect all ears	$O(n^2)$	
Next step		
1. Crop an ear	$O(1)$	
2. Update the information of the convex vertices	$O(1)$	$O(n)$ times
3. Update the information of the ears	$O(n)$	



Algorithm: TRIANGULATION
Initialize the ear tip status of each vertex.
while $n > 3$ do
  Locate an ear tip v_2.
  Output diagonal $v_1 v_3$.
  Delete v_2.
  Update the ear tip status of v_1 and v_3.

Algorithm 1.1 Triangulation algorithm.

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_iv_j, determining a triangulation of P.

Improved procedure	**Running time**	
Initialization		
1. Detect all convex vertices	$O(n)$	Only once
2. Detect all ears	$O(n^2)$	
Next step		
1. Crop an ear	$O(1)$	
2. Update the information of the convex vertices	$O(1)$	$O(n)$ times
3. Update the information of the ears	$O(n)$	

Running time: $O(n^2)$

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.

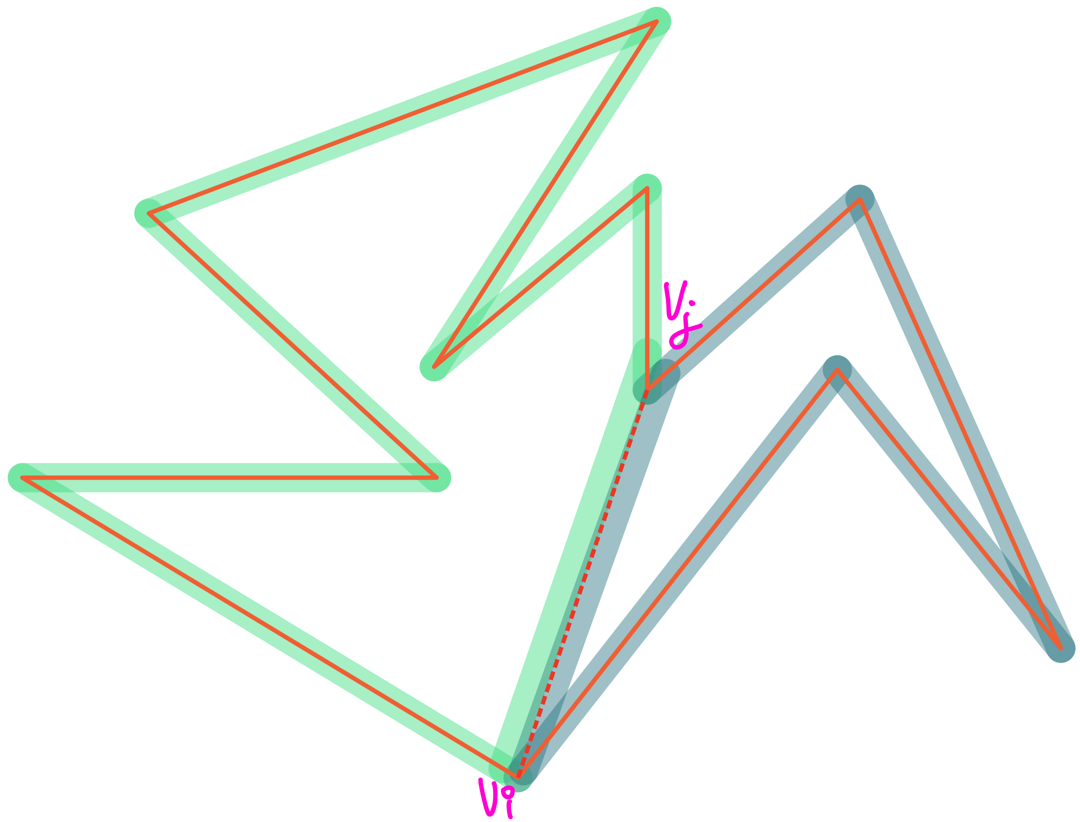
Tringulating a polygon by inserting diagonals

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

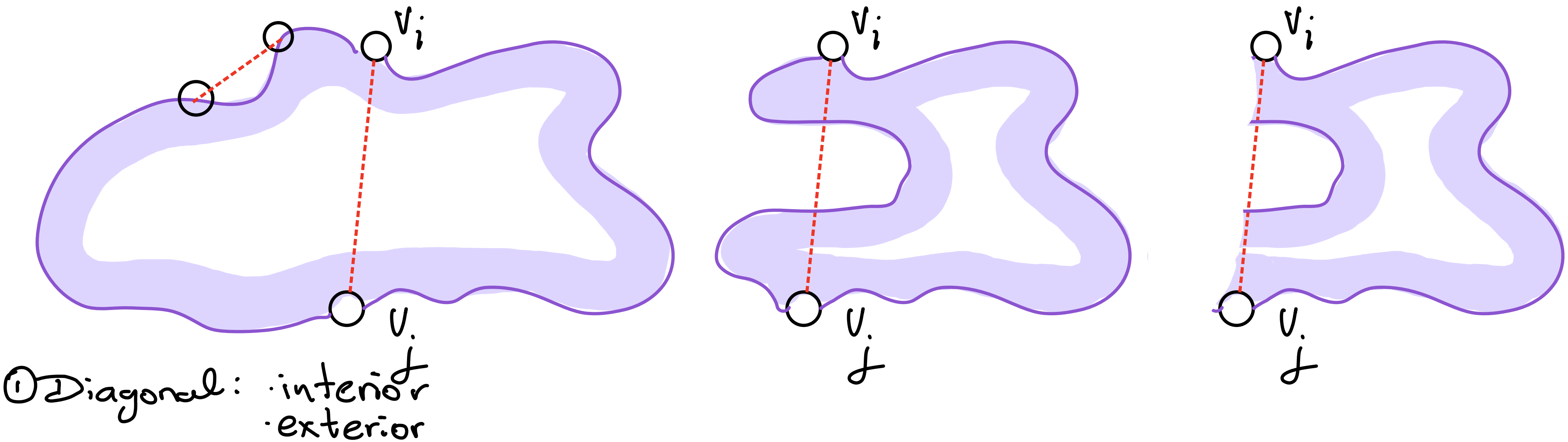
Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

Output: List of internal diagonals of P, $v_i v_j$, 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?



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?

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.

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?

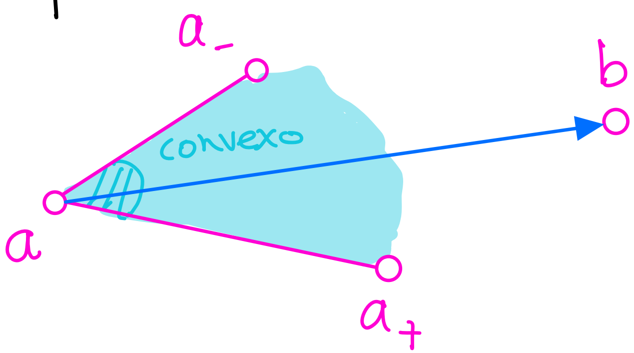
Is it a diagonal?

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

Is it internal?

Noten que en este caso $v_i v_j$ no necesariamente son consecutivos (antes sí).

¿Cómo podemos determinar si es diagonal interior?



si a es convexo, ab es interior si:
a_- está a la izq de ab
a_+ está a la der. de ab

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

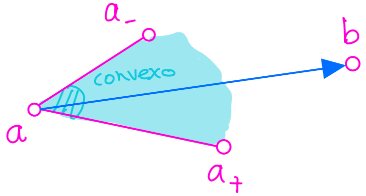
Test. How to decide whether a given segment $v_i v_j$ is an internal diagonal?

Is it a diagonal?

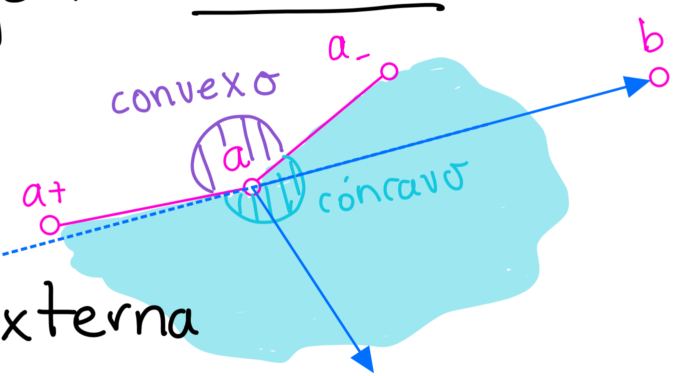
Check $v_i v_j$ against all segments $v_k v_{k+1}$ for intersection.

Is it internal?

¿Cómo podemos determinar si es diagonal interior?



si a es convexo, ab es interior si:
a_ está a la izq de ab
a+ está a la der. de ab



• ab es interna si no es externa

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?

¿Cómo podemos determinar si es diagonal interior?

Lema El segmento $s = v_i v_j$ es una diagonal interior de P iff:

1. $\forall$ aristas e de P que no son adyacentes a v_i ni a v_j, $s \cap e = \emptyset$
2. s es interna a P en una vecindad de v_i y de v_j.

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?

¿Cómo podemos determinar si es diagonal interior?

```
bool   InCone( tVertex a, tVertex b )
{
  tVertex a0,a1;      /* a0,a,a1 are consecutive vertices. */

  a1 = a->next;
  a0 = a->prev;

  /* If a is a convex vertex ... */
  if( LeftOn( a->v, a1->v, a0->v ) )
      return    Left( a->v, b->v, a0->v )
             && Left( b->v, a->v, a1->v );

  /* Else a is reflex: */
      return !(    LeftOn( a->v, b->v, a1->v )
                && LeftOn( b->v, a->v, a0->v ) );
}
```

Code 1.11 InCone.

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

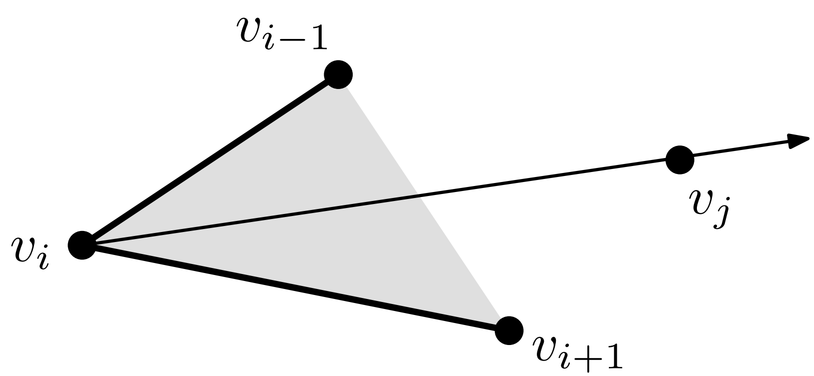
Test. How to decide whether a given segment v_iv_j is an internal diagonal?

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?

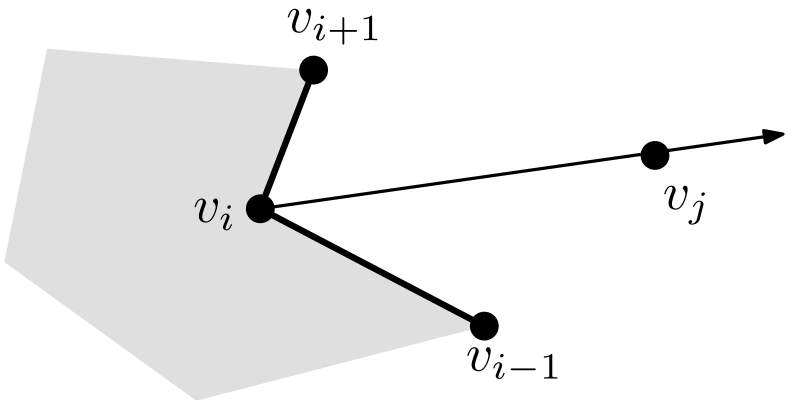
Is it a diagonal?

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.



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?

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.

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

Running time

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

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.

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

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?

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?

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

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?

Brute-force solution:

Apply the test to each candidate segment.

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

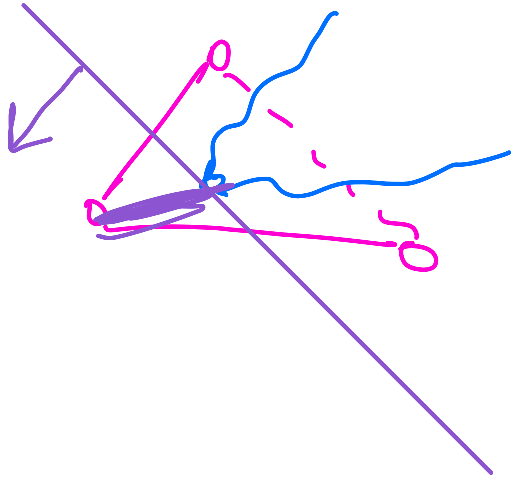
Running time

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. $O(n^3)$



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

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?

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

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 $v_{i-1}v_i v_{i+1}$.

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?

Brute-force solution:

Apply the test to each candidate segment. $O(n^3)$

Applying previous results: $O(n)$

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 $v_{i-1}v_i v_{i+1}$.

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?	$O(n)$

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

	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?	

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

	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.

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?	$O(n)$

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?	$O(n)$

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

It finds $n-3$ diagonals and each one is found in $O(n)$ time.

TRIANGULATING POLYGONS

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?

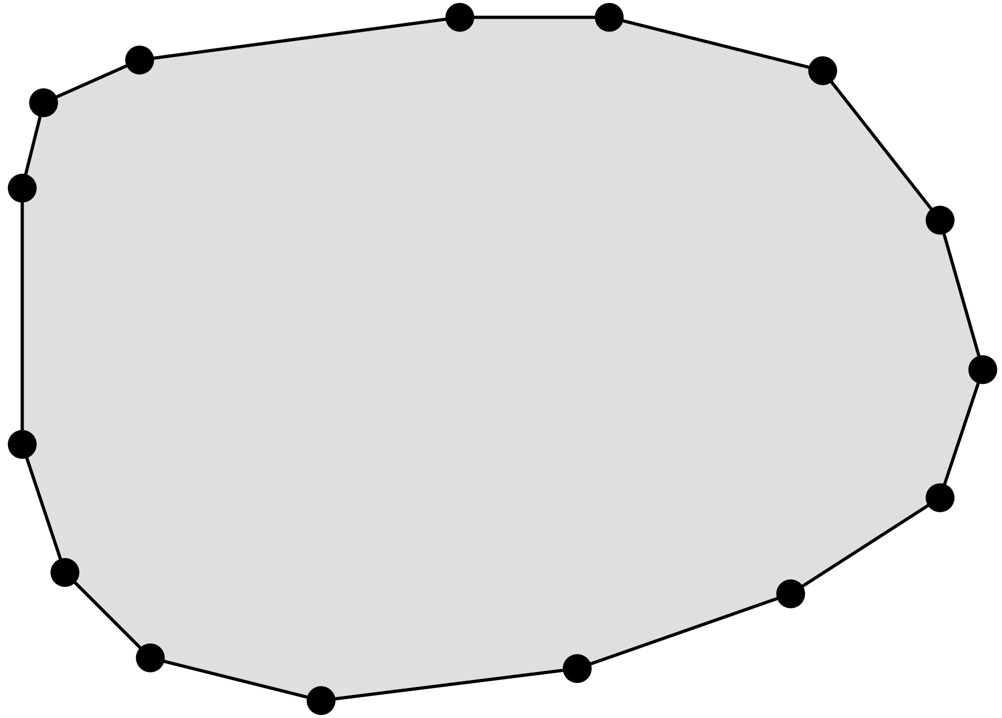
Triangulating a convex polygon

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.

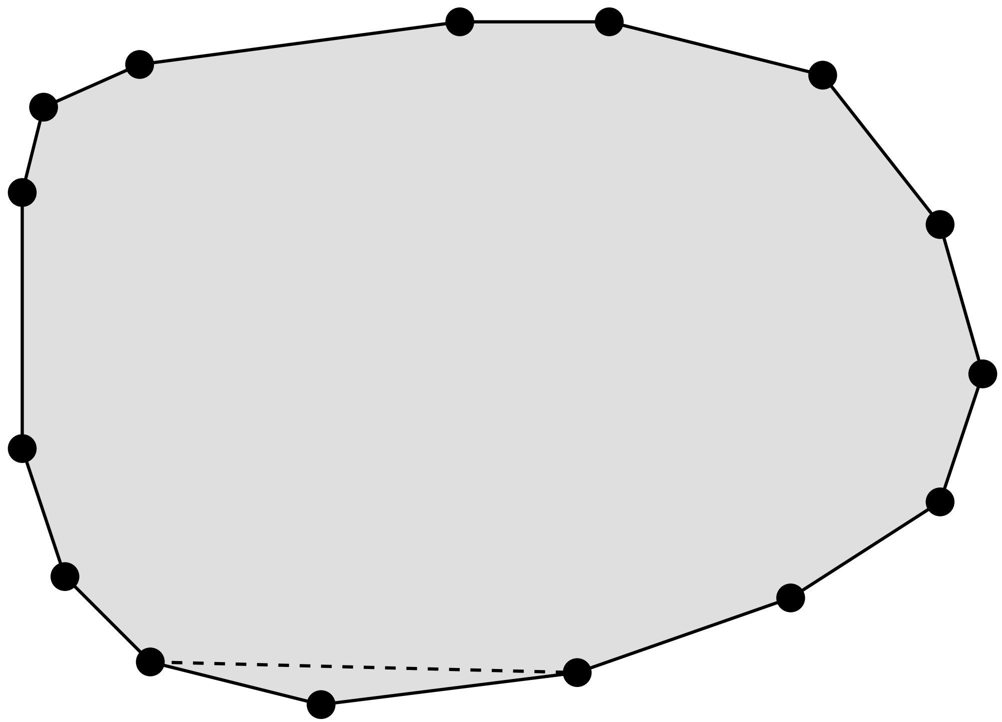


TRIANGULATING POLYGONS

Is it possible to triangulate a polygon more efficiently?

Triangulating a convex polygon

Trivially done in $O(n)$ time.

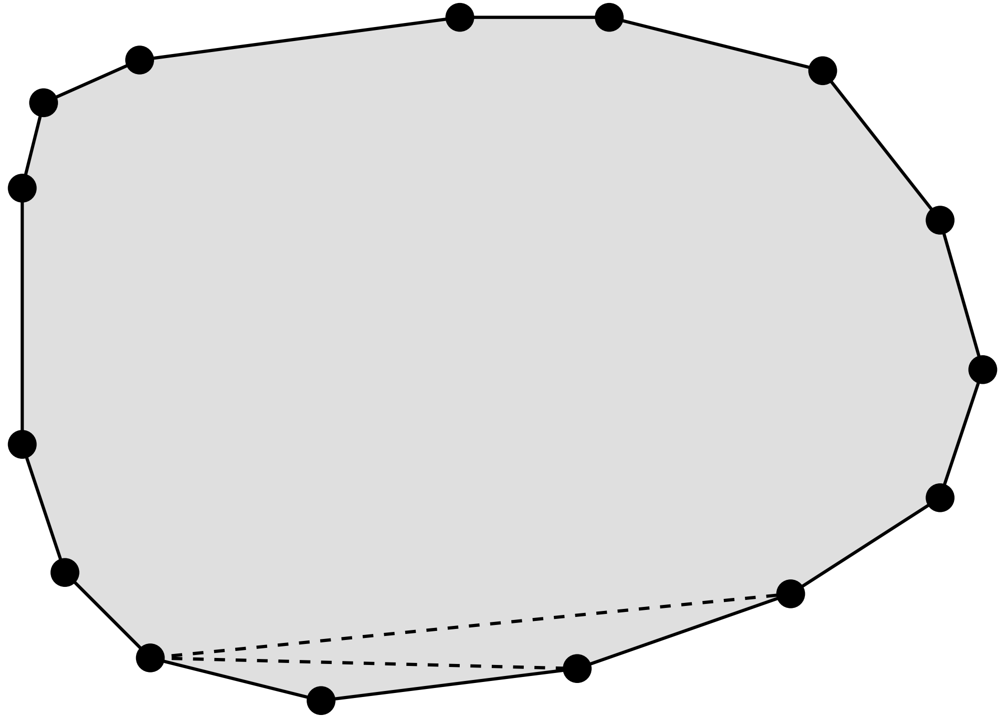


TRIANGULATING POLYGONS

Is it possible to triangulate a polygon more efficiently?

Triangulating a convex polygon

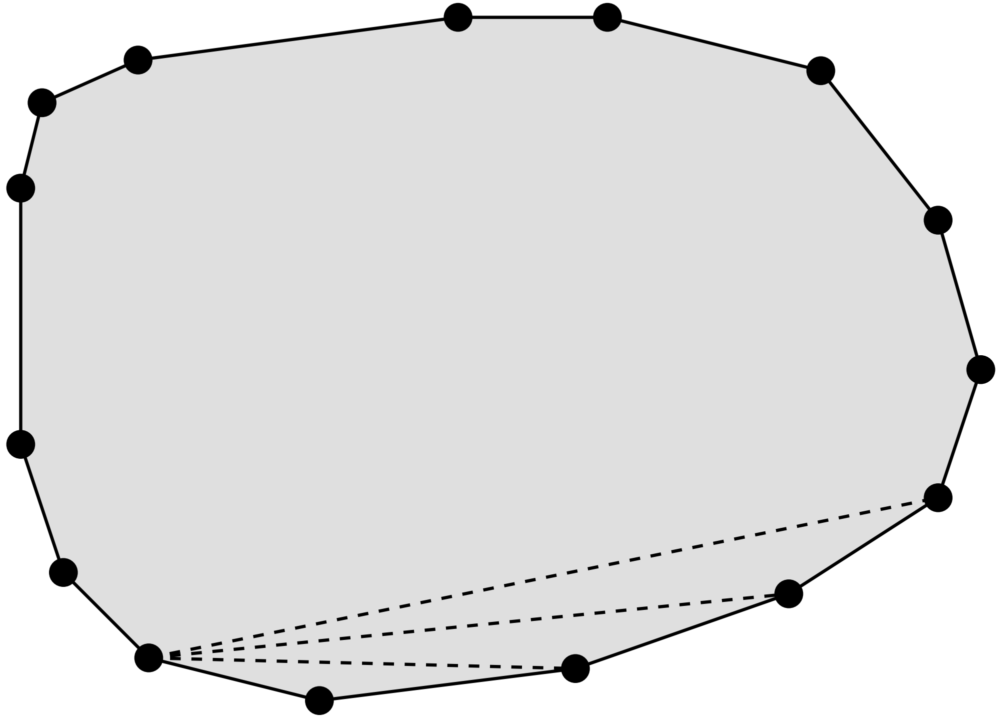
Trivially done in $O(n)$ time.



Is it possible to triangulate a polygon more efficiently?

Triangulating a convex polygon

Trivially done in $O(n)$ time.

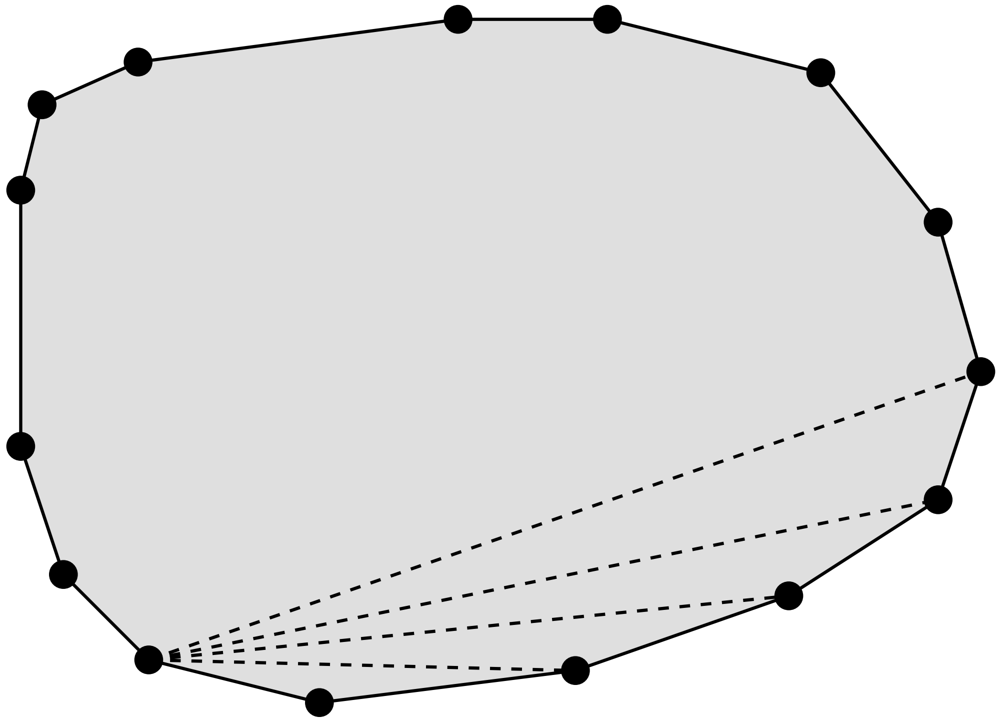


TRIANGULATING POLYGONS

Is it possible to triangulate a polygon more efficiently?

Triangulating a convex polygon

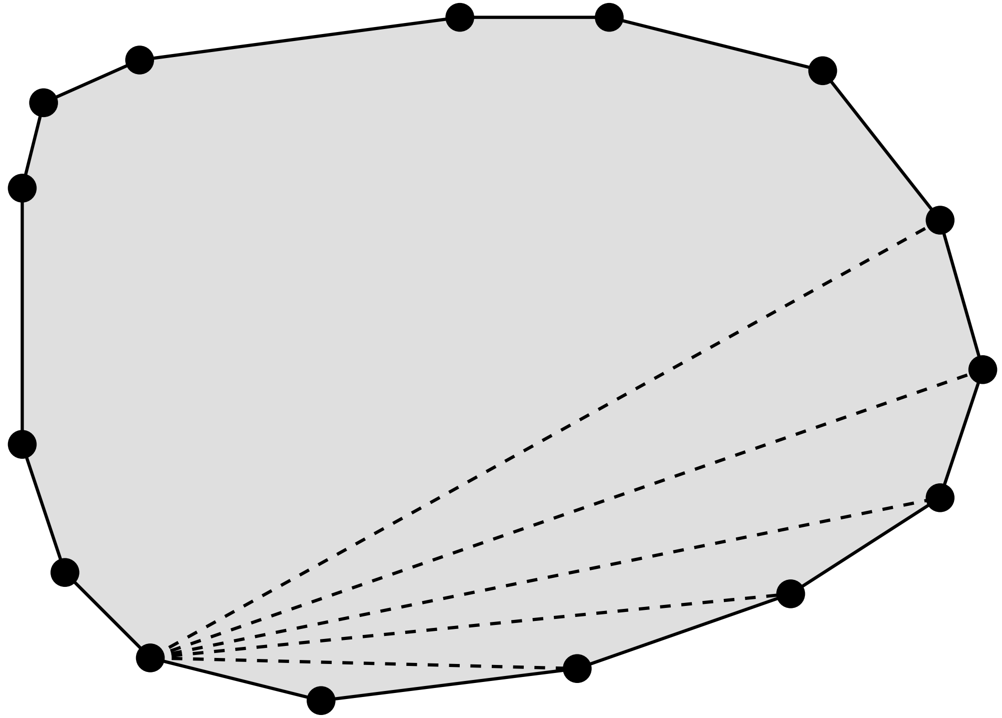
Trivially done in $O(n)$ time.



Is it possible to triangulate a polygon more efficiently?

Triangulating a convex polygon

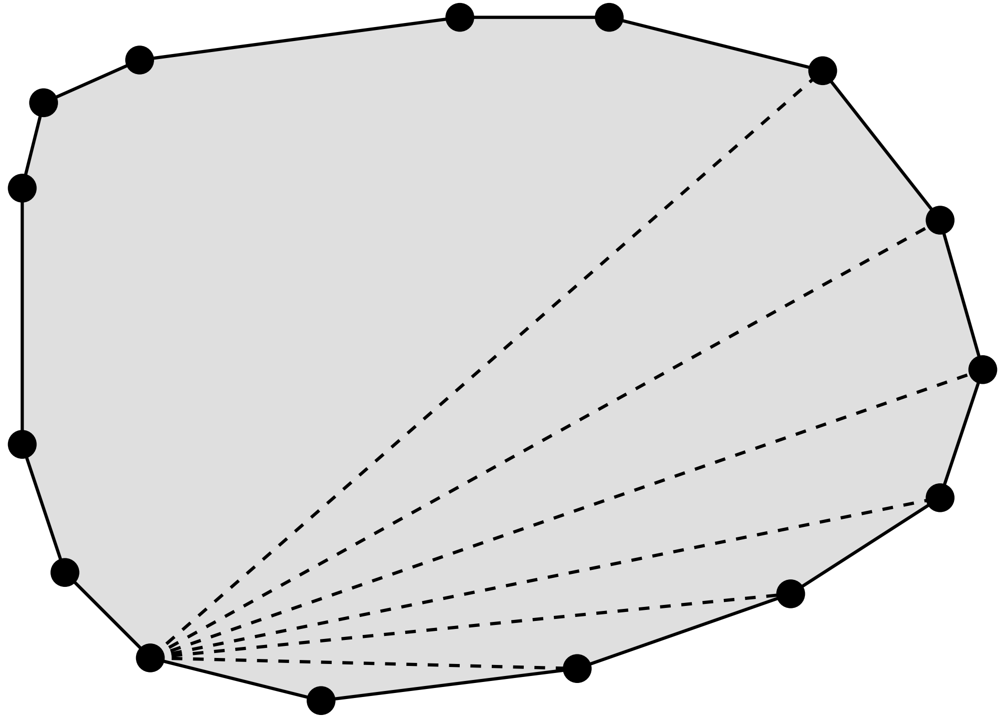
Trivially done in $O(n)$ time.



Is it possible to triangulate a polygon more efficiently?

Triangulating a convex polygon

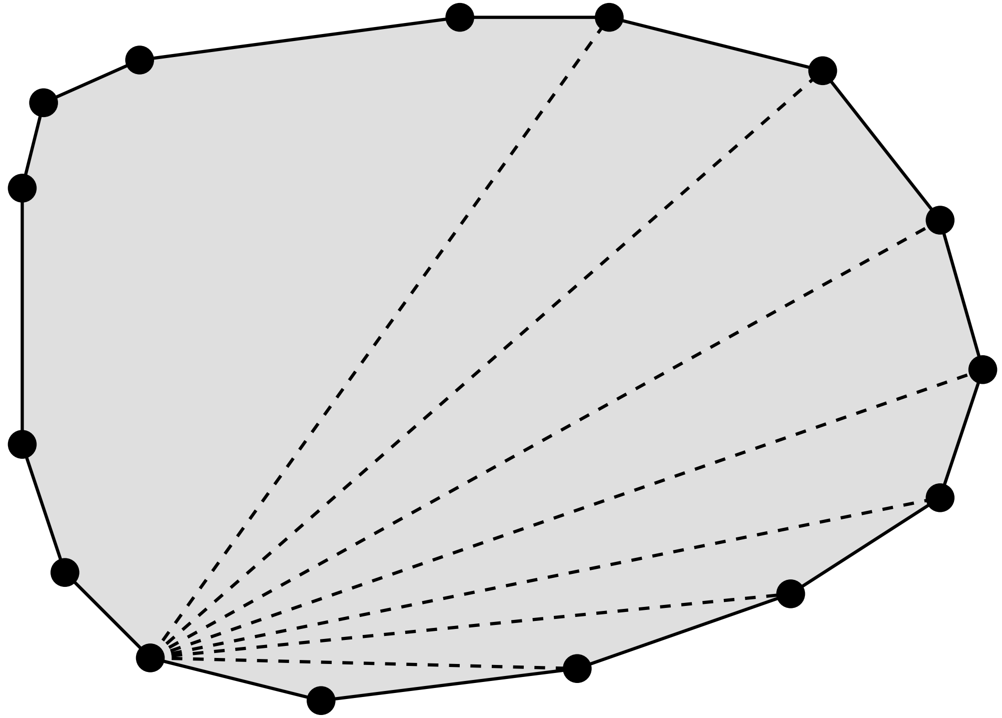
Trivially done in $O(n)$ time.



Is it possible to triangulate a polygon more efficiently?

Triangulating a convex polygon

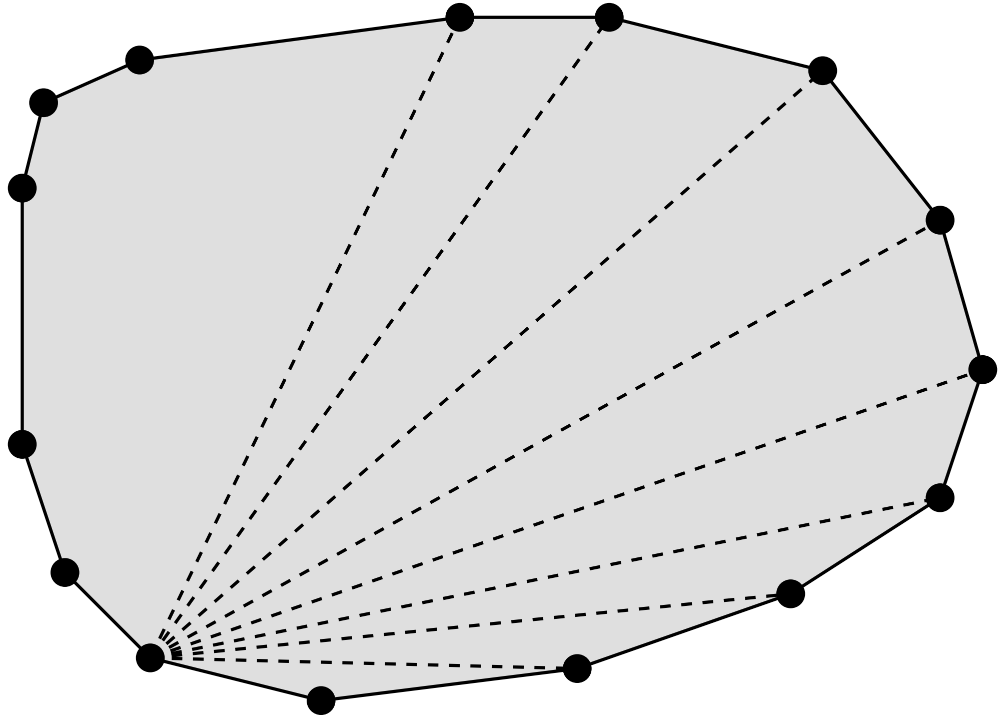
Trivially done in $O(n)$ time.



Is it possible to triangulate a polygon more efficiently?

Triangulating a convex polygon

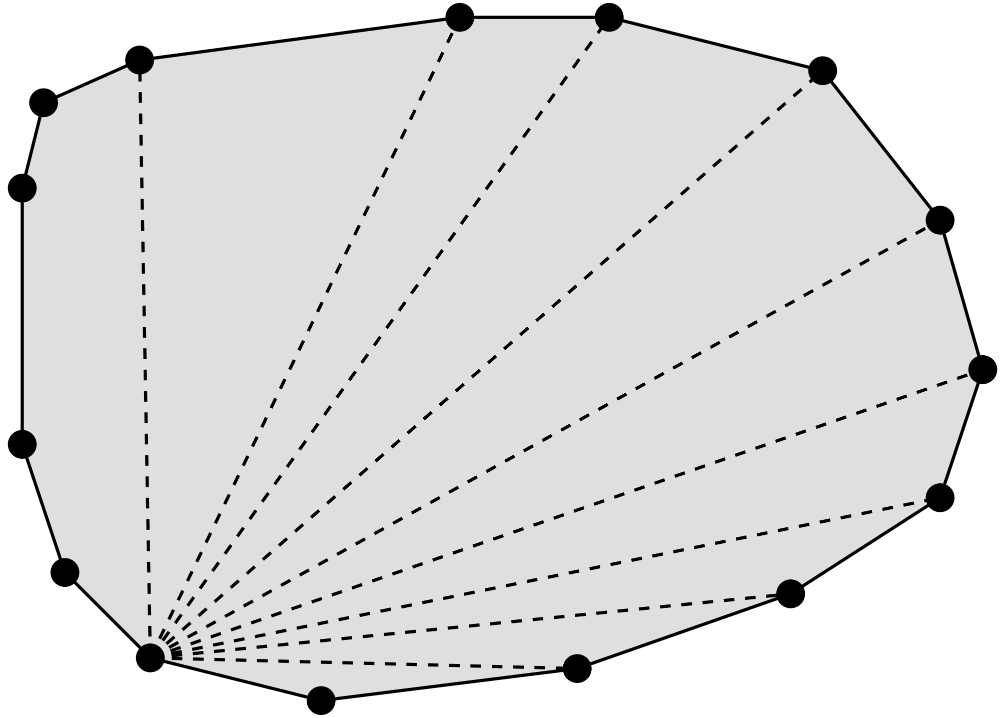
Trivially done in $O(n)$ time.



Is it possible to triangulate a polygon more efficiently?

Triangulating a convex polygon

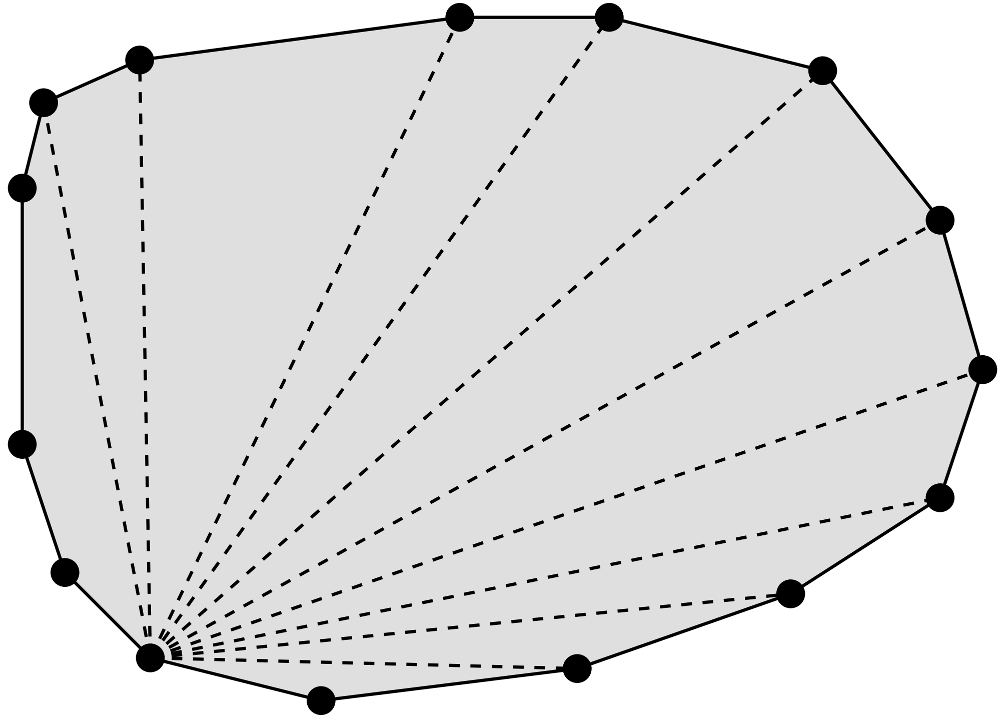
Trivially done in $O(n)$ time.



Is it possible to triangulate a polygon more efficiently?

Triangulating a convex polygon

Trivially done in $O(n)$ time.

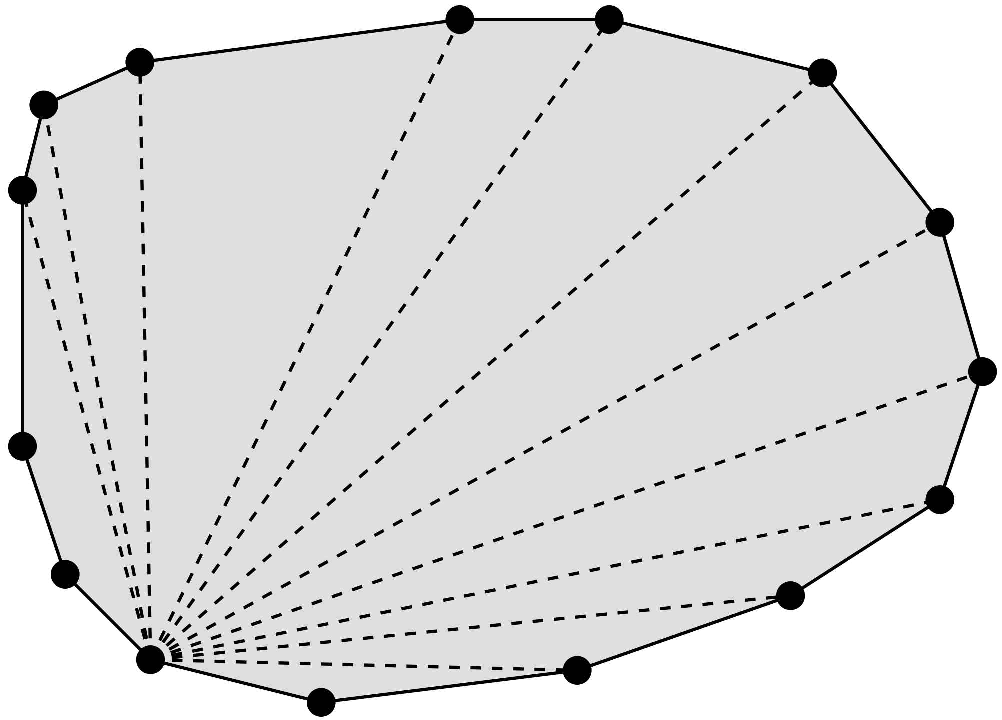


TRIANGULATING POLYGONS

Is it possible to triangulate a polygon more efficiently?

Triangulating a convex polygon

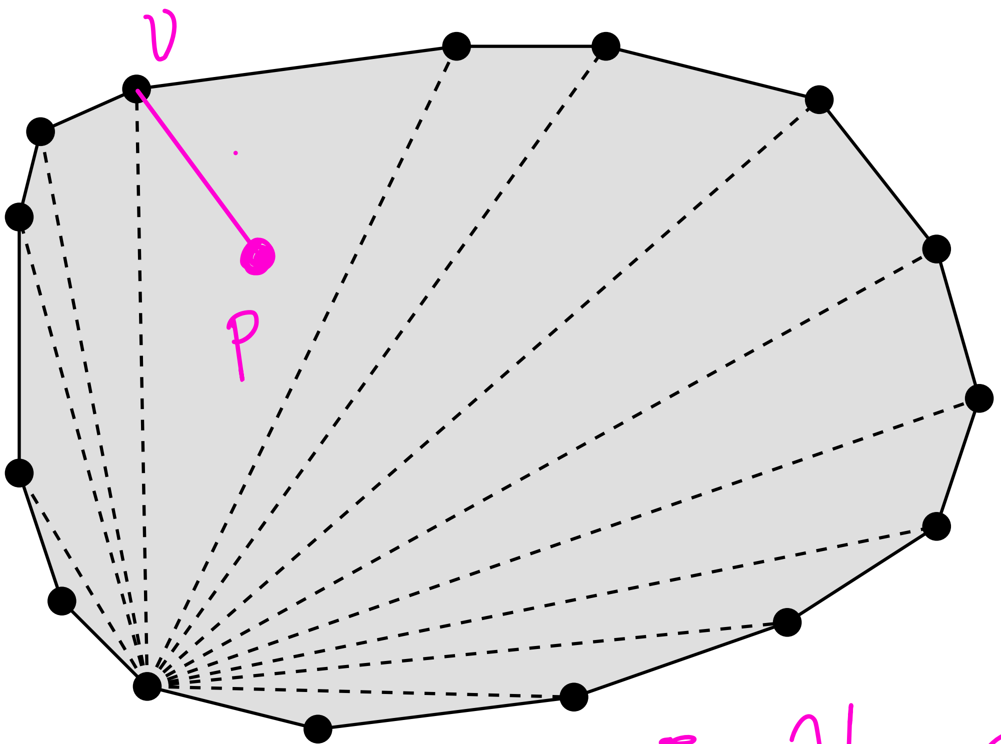
Trivially done in $O(n)$ time.



Is it possible to triangulate a polygon more efficiently?

Triangulating a convex polygon

Trivially done in $O(n)$ time.



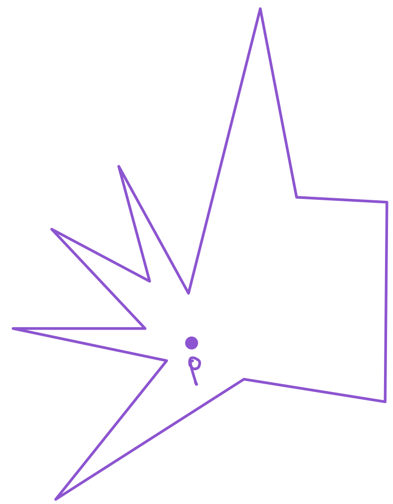
Un polígono es convexo si $\forall p \in P$ ocurre que p ve a todos los vértices de P.

TRIANGULATING POLYGONS

Is it possible to triangulate a polygon more efficiently?

Triangulating a convex polygon $\forall p \in P$

Trivially done in $O(n)$ time.



Triangulating a star-shaped polygon

Can be done in $O(n)$ time. Posed as problem.

Si $\exists p \in P$ que puede ver a todos los vértices del polígono.

- Los polígonos convexos son estrellados.

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

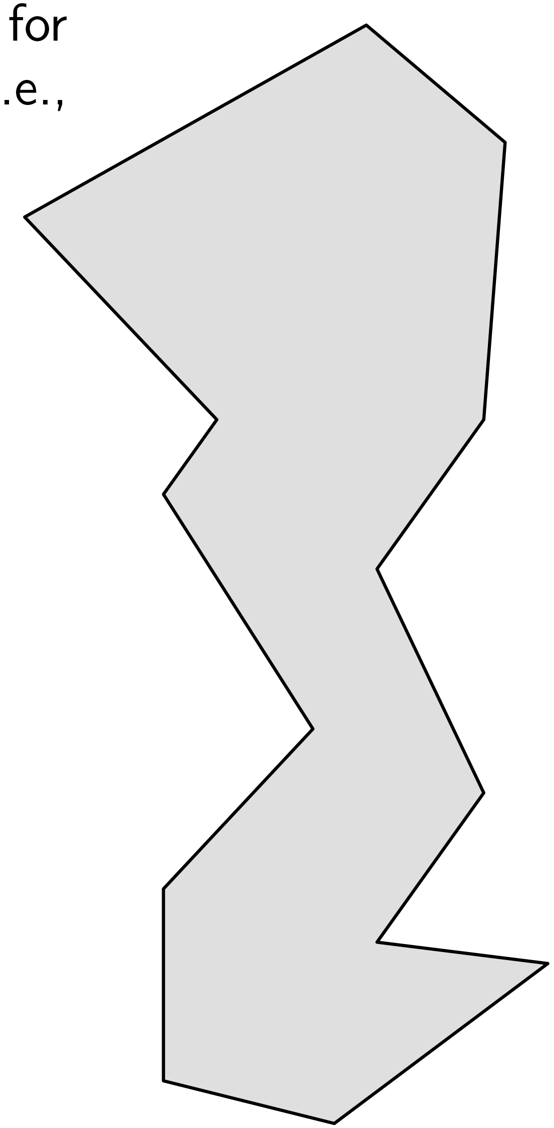
It can also be done in $O(n)$ time. In the following we will see how.

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).

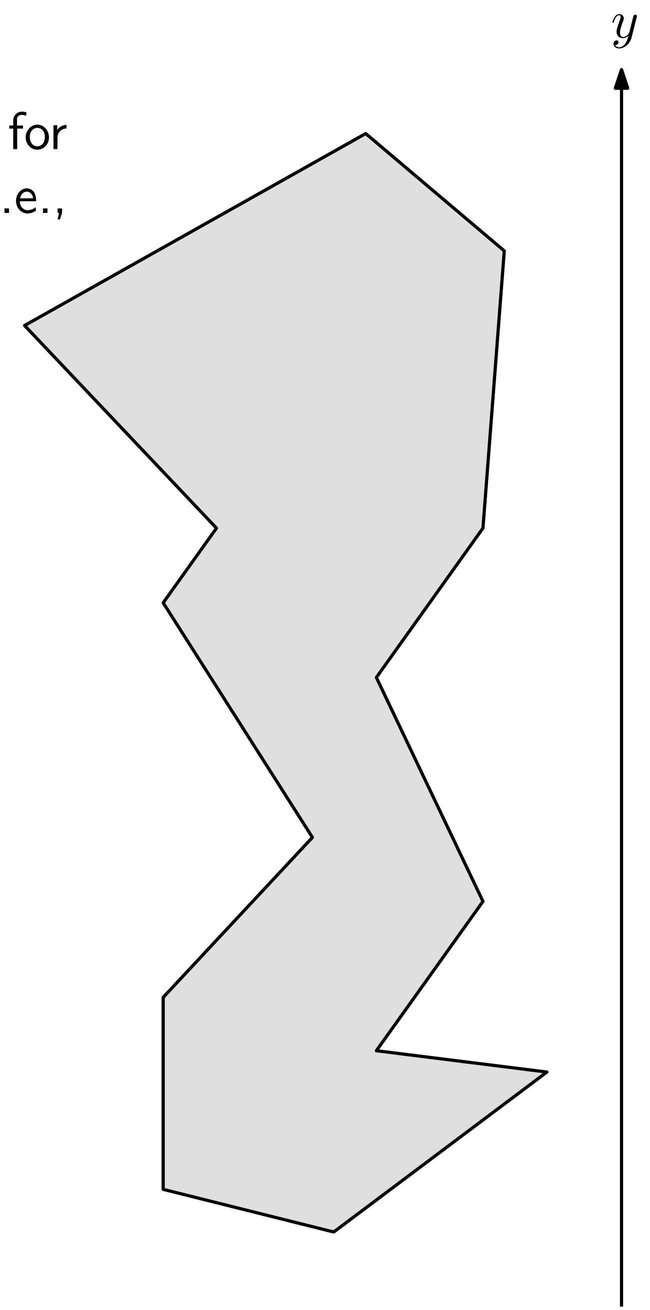
Monotone polygon

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

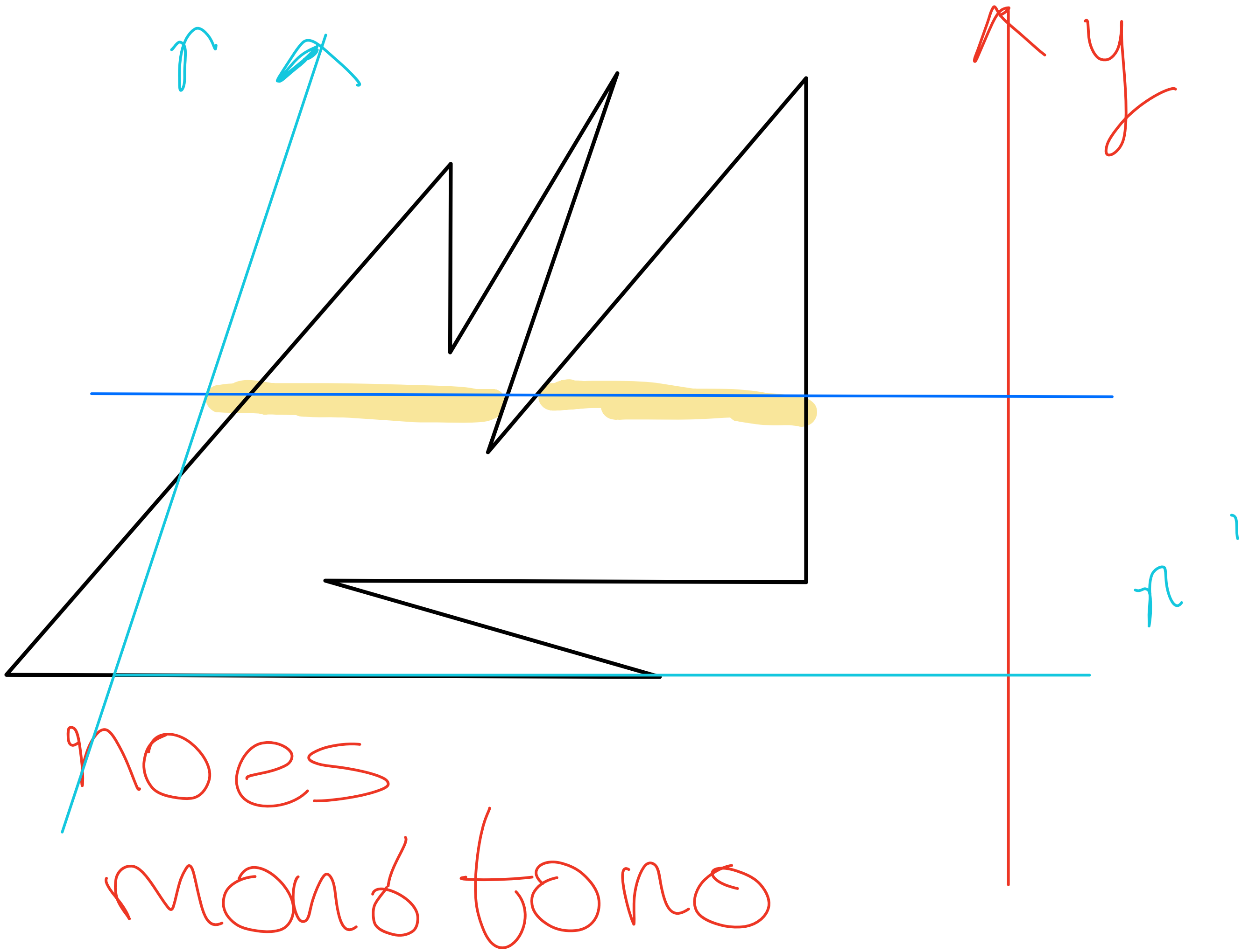
A polygon P is called **monotone** with respect to a direction r if, for every line r' orthogonal to r, the intersection $P \cap r'$ is connected (i.e., it is a segment, a point or the empty set).

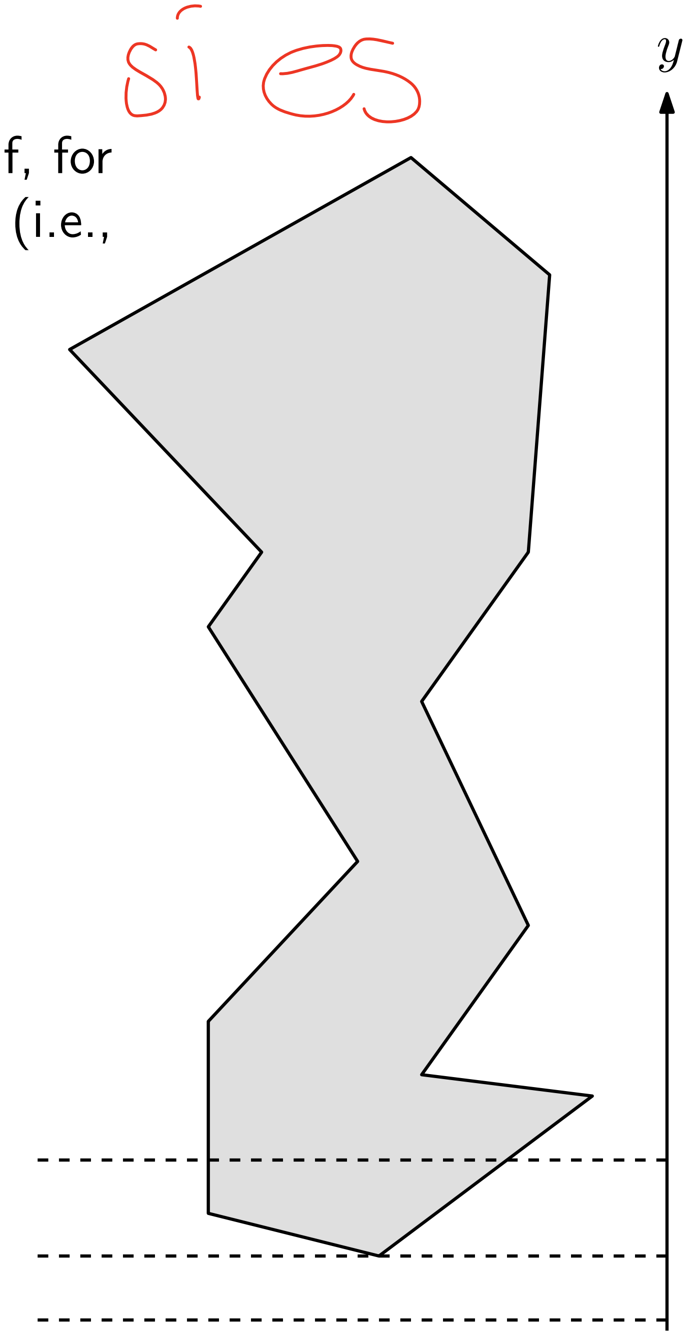


TRIANGULATING POLYGONS

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

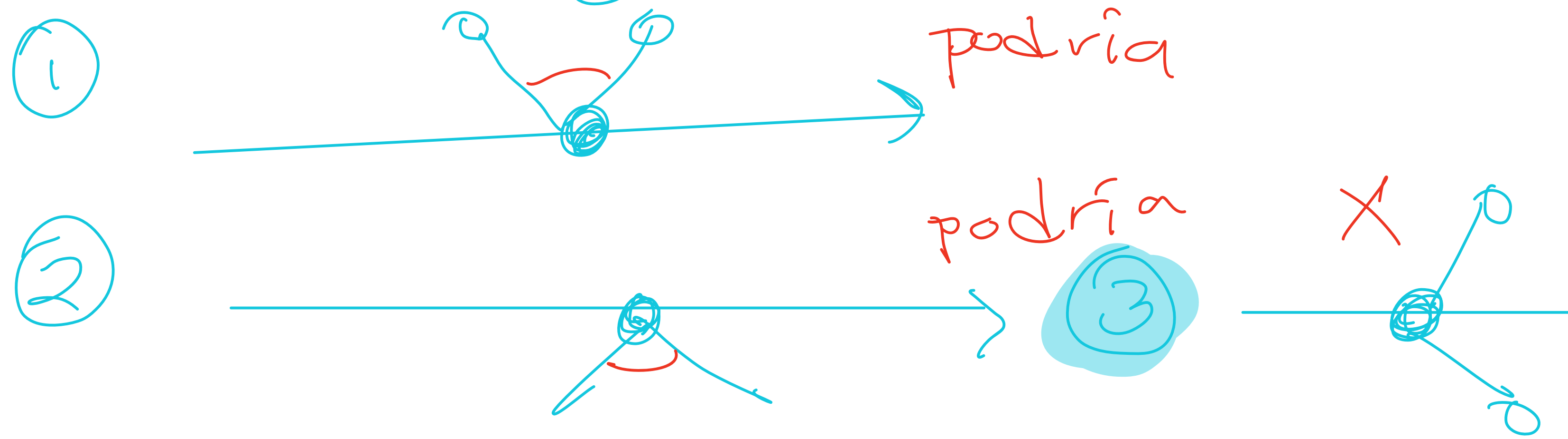
A polygon P is called **monotone** with respect to a direction r if, for every line r' orthogonal to r, the intersection $P \cap r'$ is connected (i.e., it is a segment, a point or the empty set).

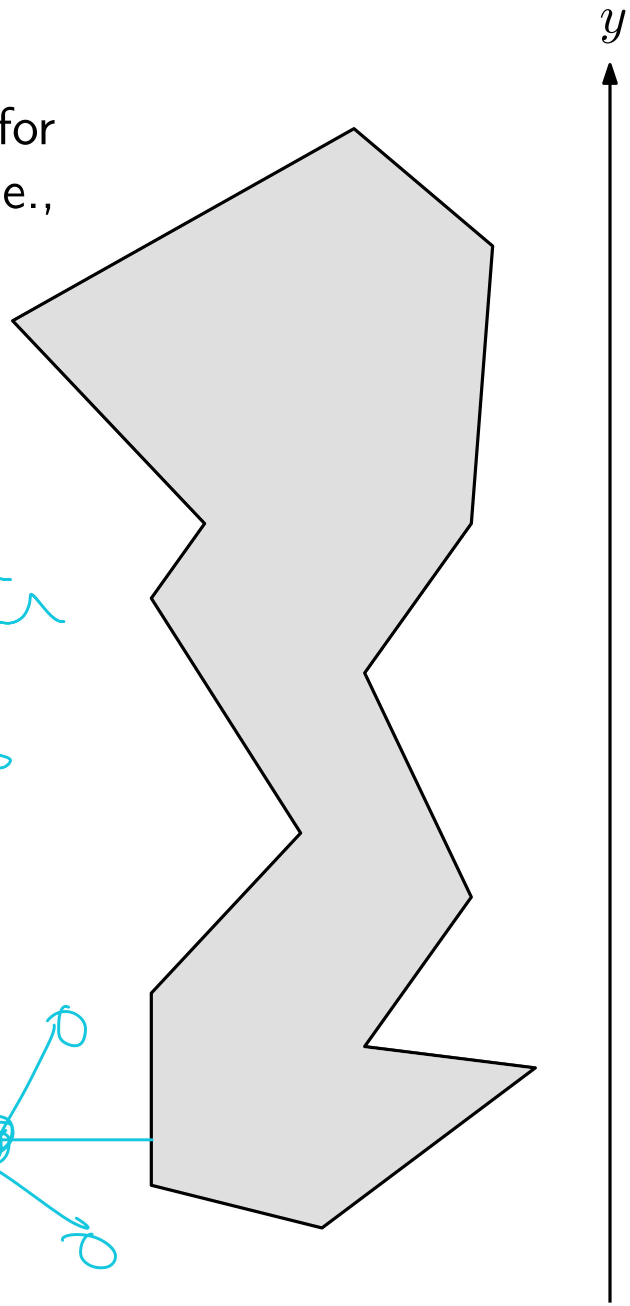
Local characterization

cúspide

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

Dado un vértice y una recta horizontal:

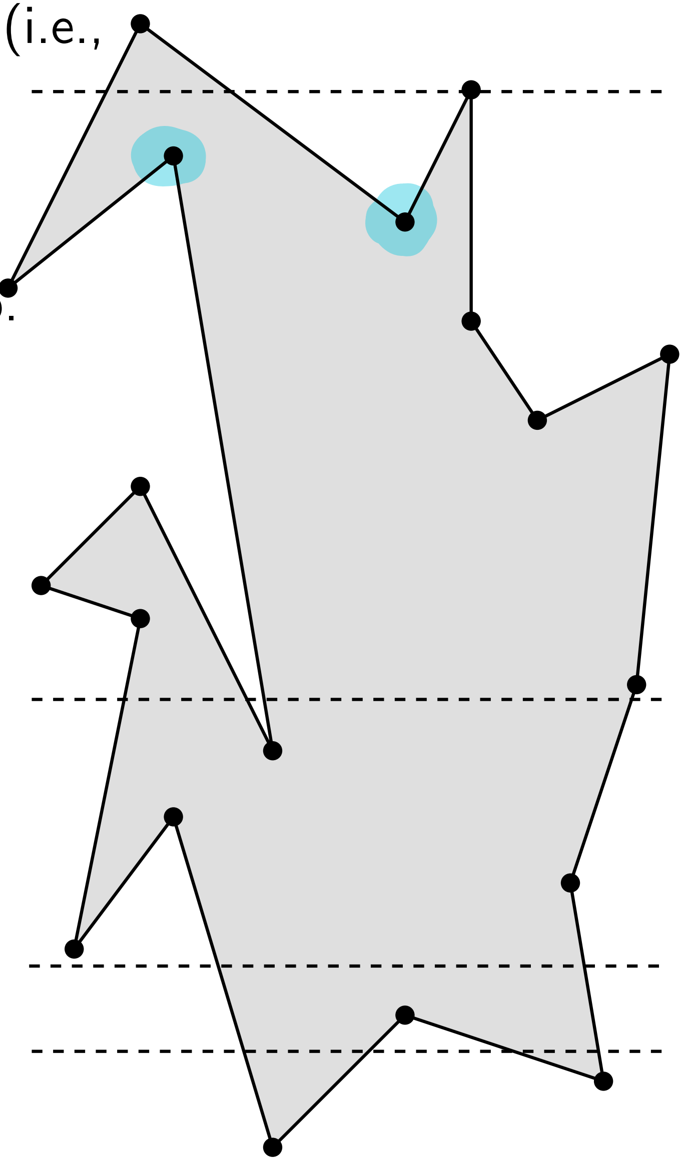




TRIANGULATING POLYGONS

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).

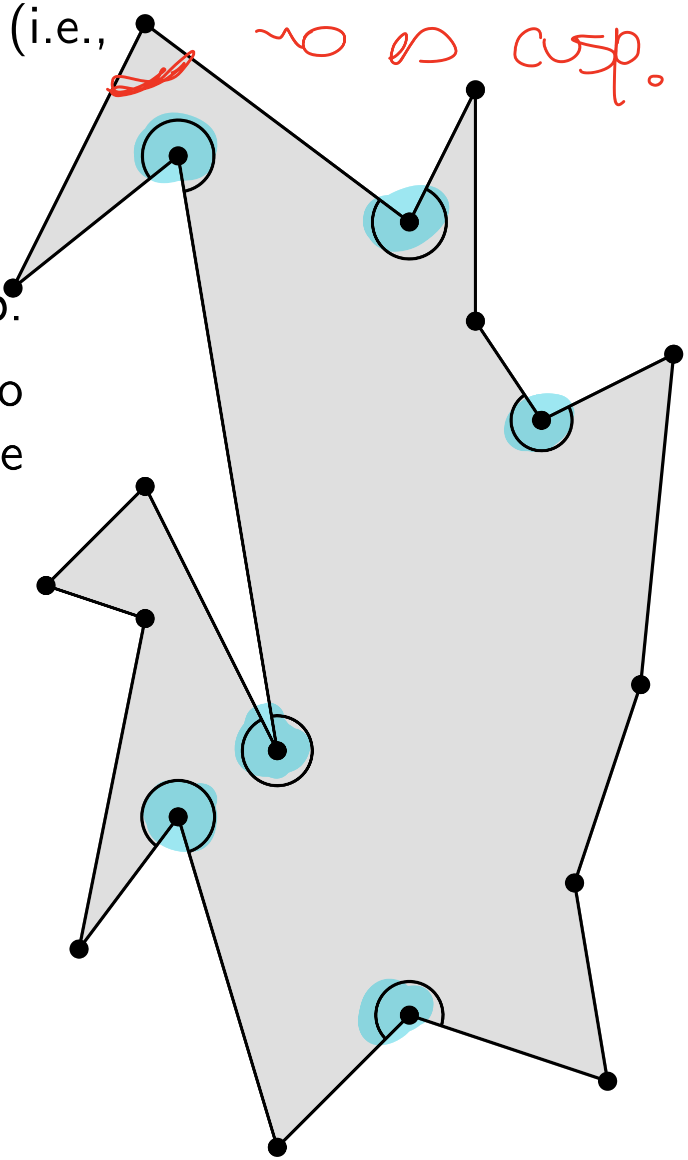
convexo no es cusp.

Local characterization

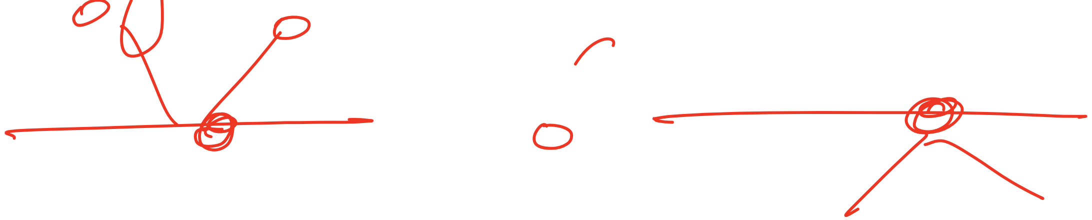
cóncavo

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.

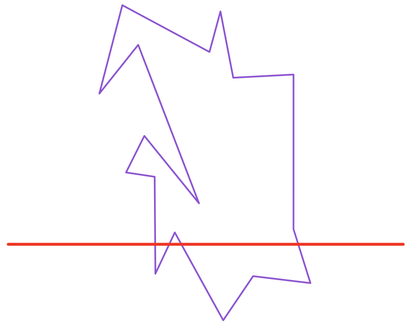


① ángulo es cóncavo

② 

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).



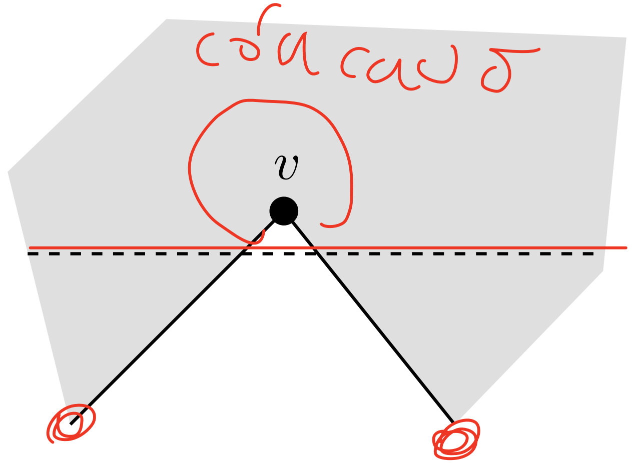
Local characterization

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.



TRIANGULATING POLYGONS

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).

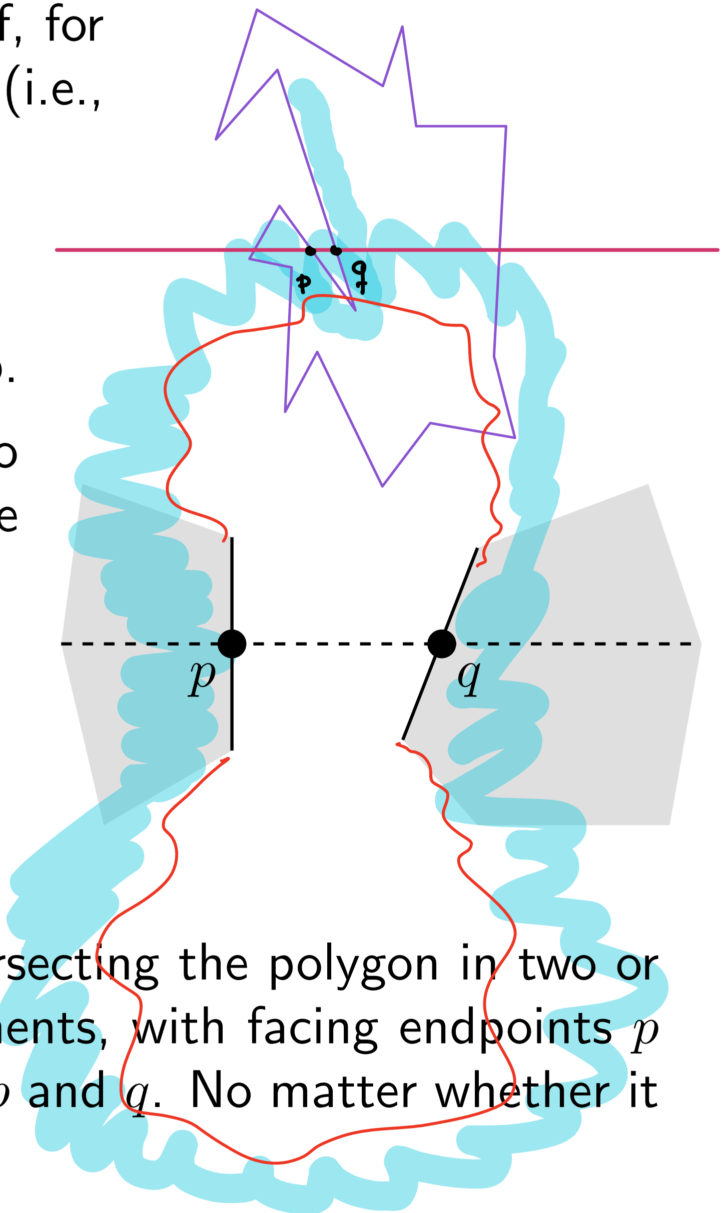
Local characterization

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 y-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 above or below the horizontal line, it will have a cusp.

Monotone polygon

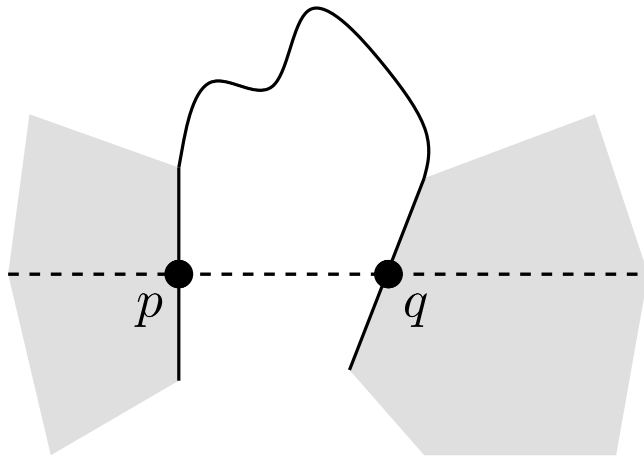
A polygon P is called **monotone** with respect to a direction r if, for every line r' orthogonal to r, the intersection $P \cap r'$ is connected (i.e., it is a segment, a point or the empty set).

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 y-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 above or below the horizontal line, it will have a cusp.

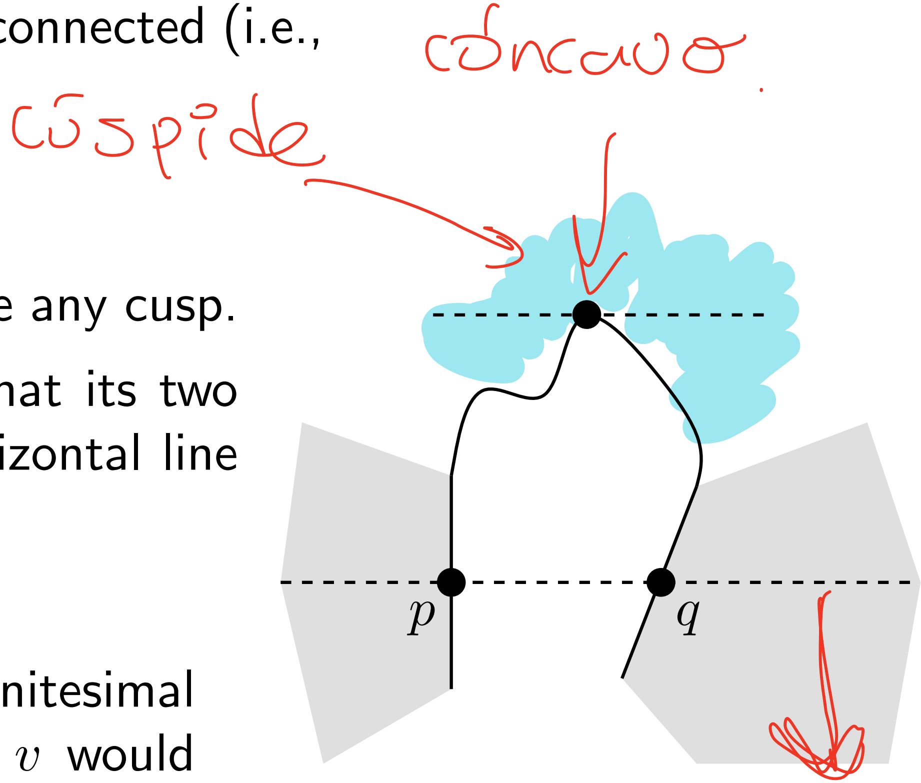
Monotone polygon

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 y-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 above or below the horizontal line, 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.

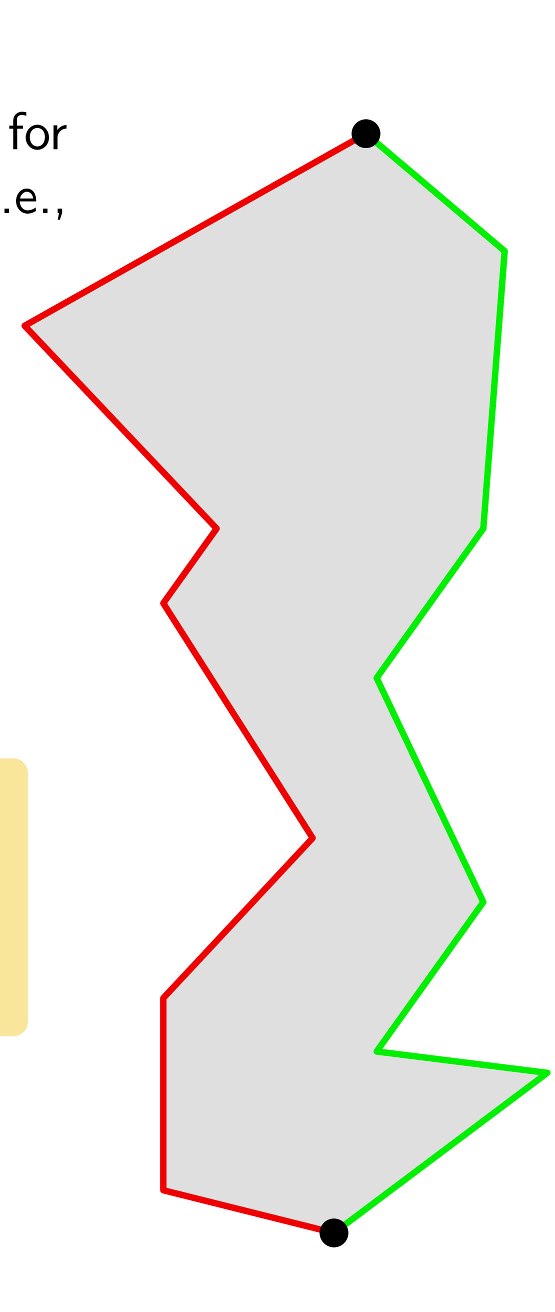
Monotone polygon

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

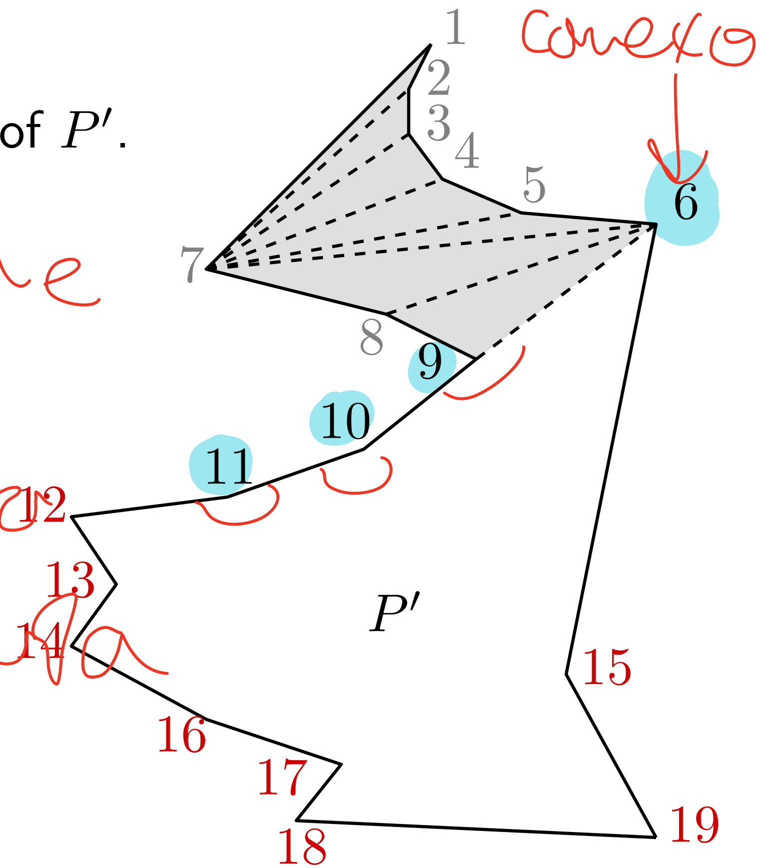
The vertices of the polygon P are processed by decreasing order of their y-coordinate.

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 (i.e., largest y-coordinate) 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'.



Q es una cola que contiene vértices.

Vértices que ya exploré pero que aún no descarto o todavía necesitan diagonales.

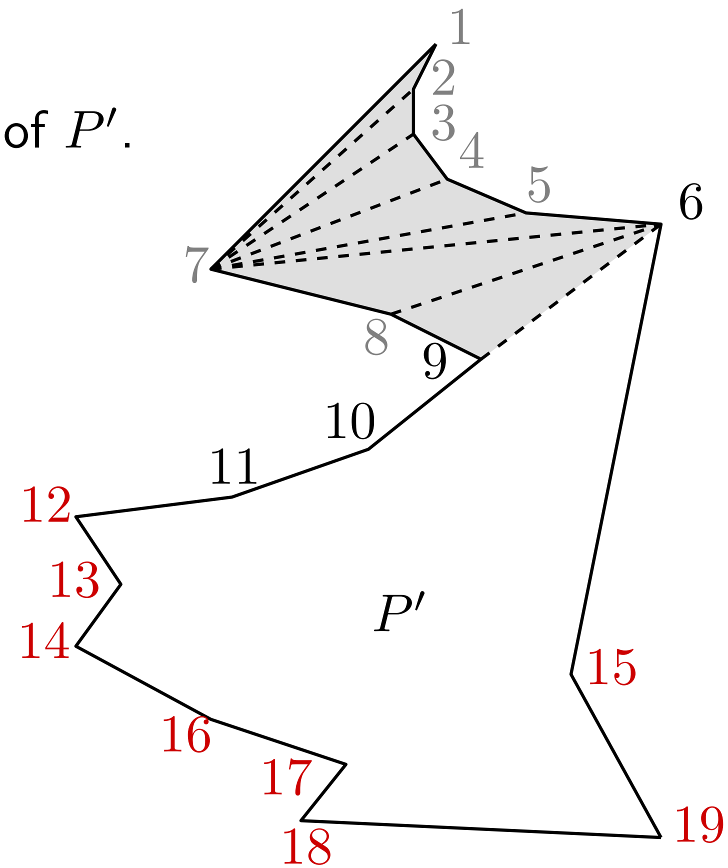
Triangulating a monotone polygon

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 (i.e., largest y-coordinate) 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:



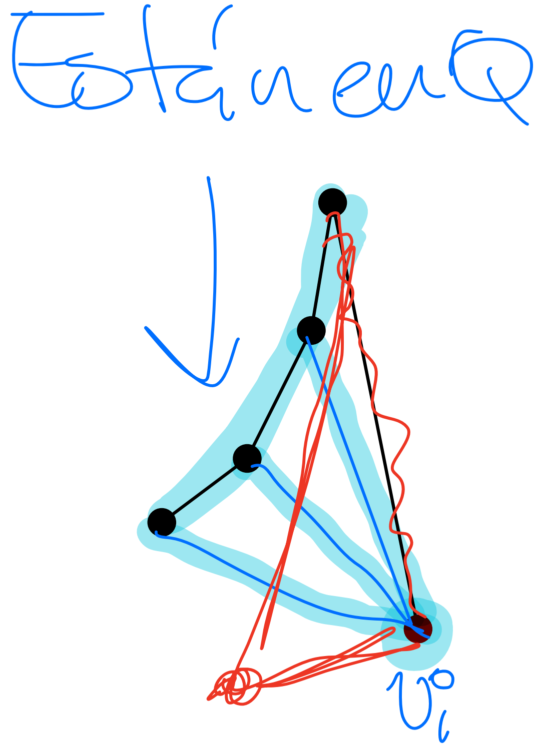
Triangulating a monotone polygon

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 (i.e., largest y-coordinate) 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.

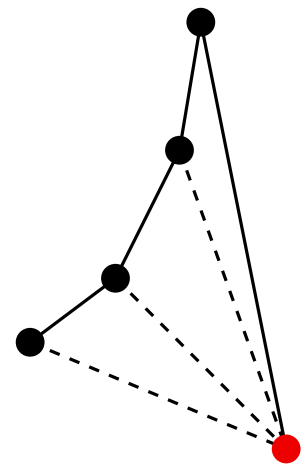
Triangulating a monotone polygon

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 (i.e., largest y-coordinate) 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 POLYGONS

Triangulating a monotone polygon

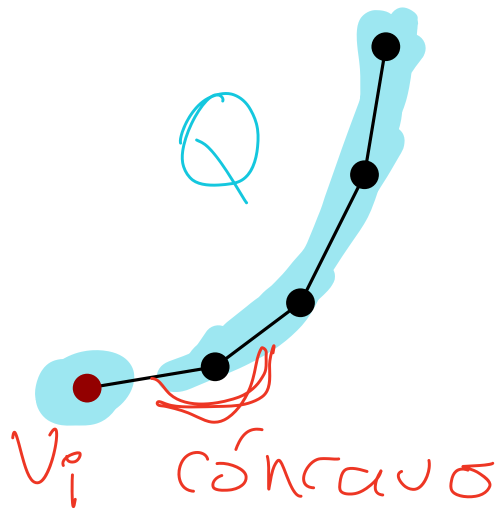
The vertices of the polygon P are processed by decreasing order of their y-coordinate.

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 (i.e., largest y-coordinate) 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 POLYGONS

Triangulating a monotone polygon

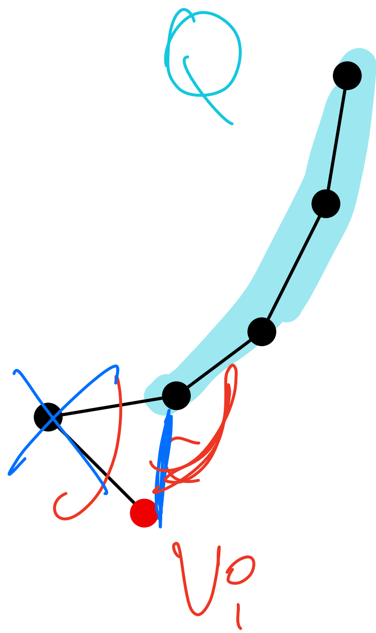
The vertices of the polygon P are processed by decreasing order of their y-coordinate.

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 (i.e., largest y-coordinate) 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.



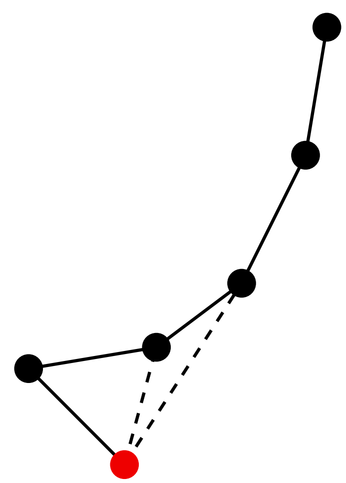
Triangulating a monotone polygon

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 (i.e., largest y-coordinate) 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'.

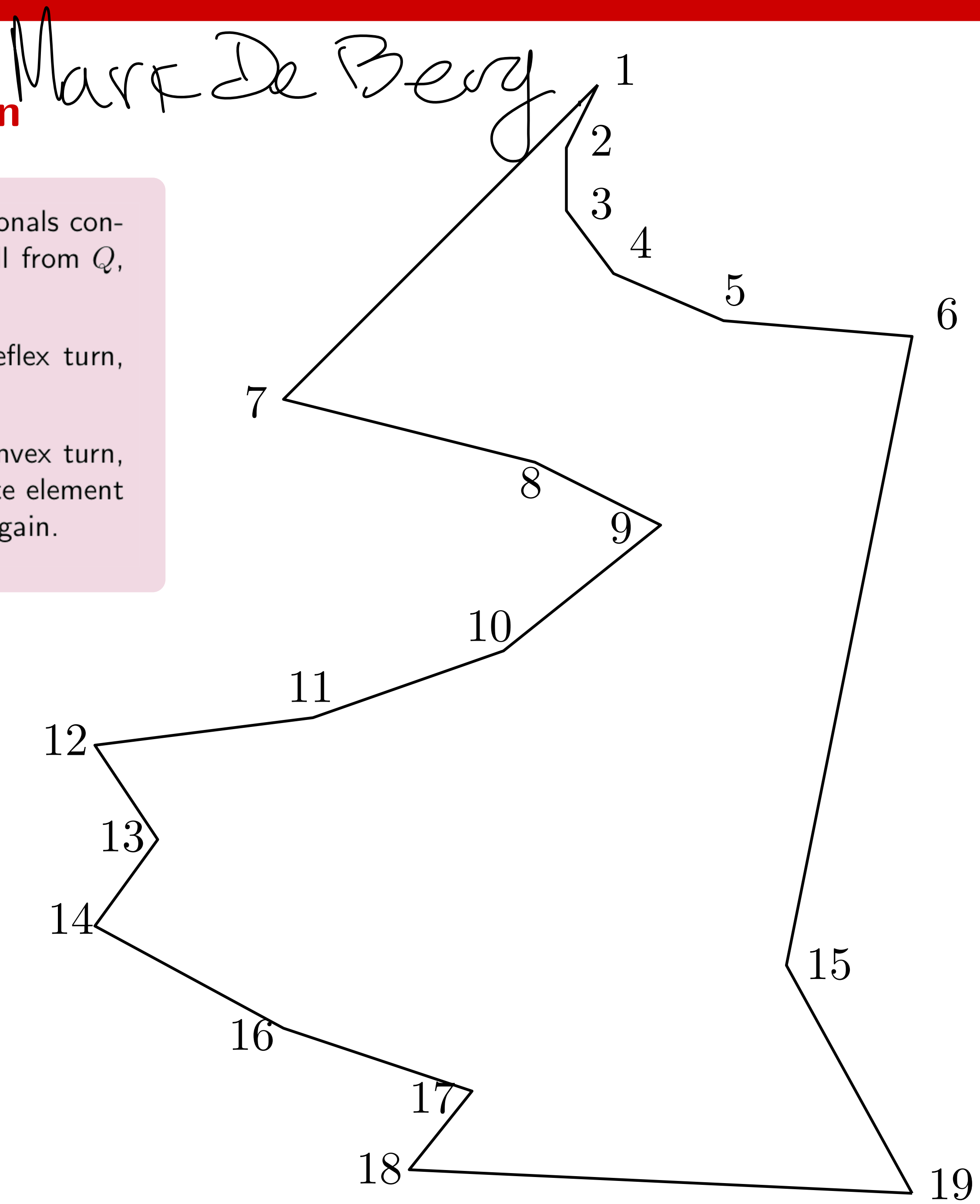
Processing a vertex v_i:

- 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

- 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.



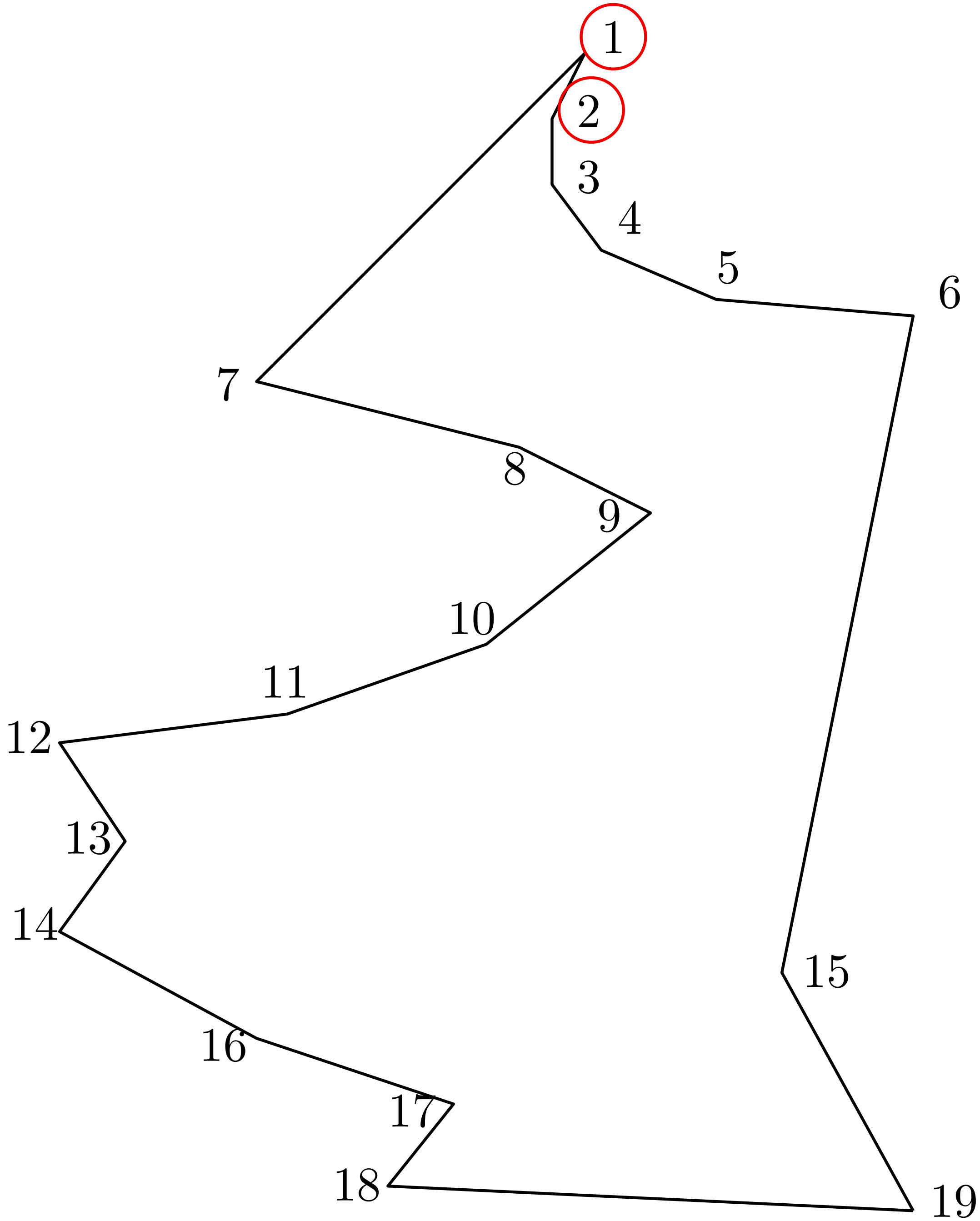
Triangulating a monotone polygon

Start

Queue state

1, 2

- 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.



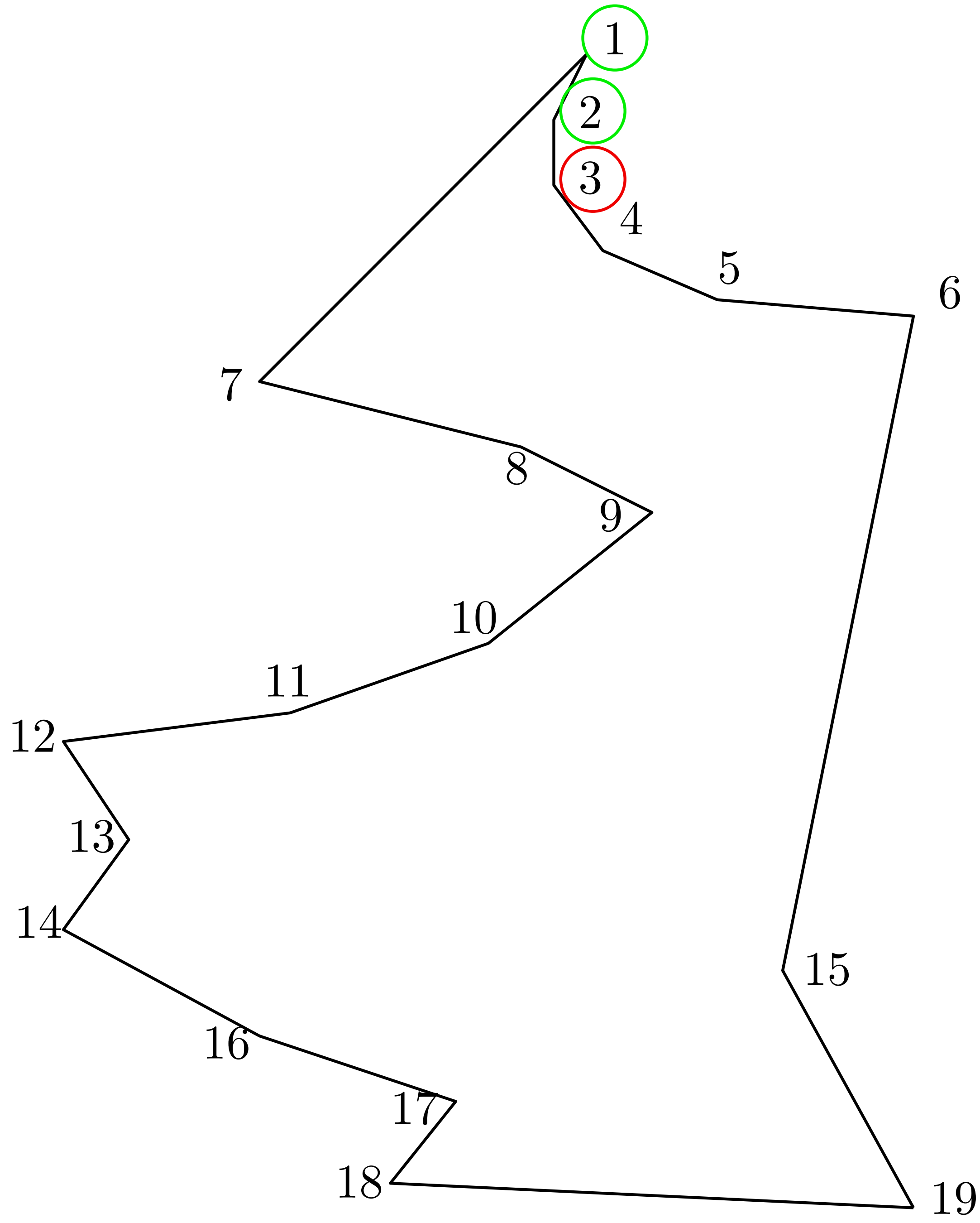
Triangulating a monotone polygon

Current vertex: 3

Add

Queue state:

1, 2, 3



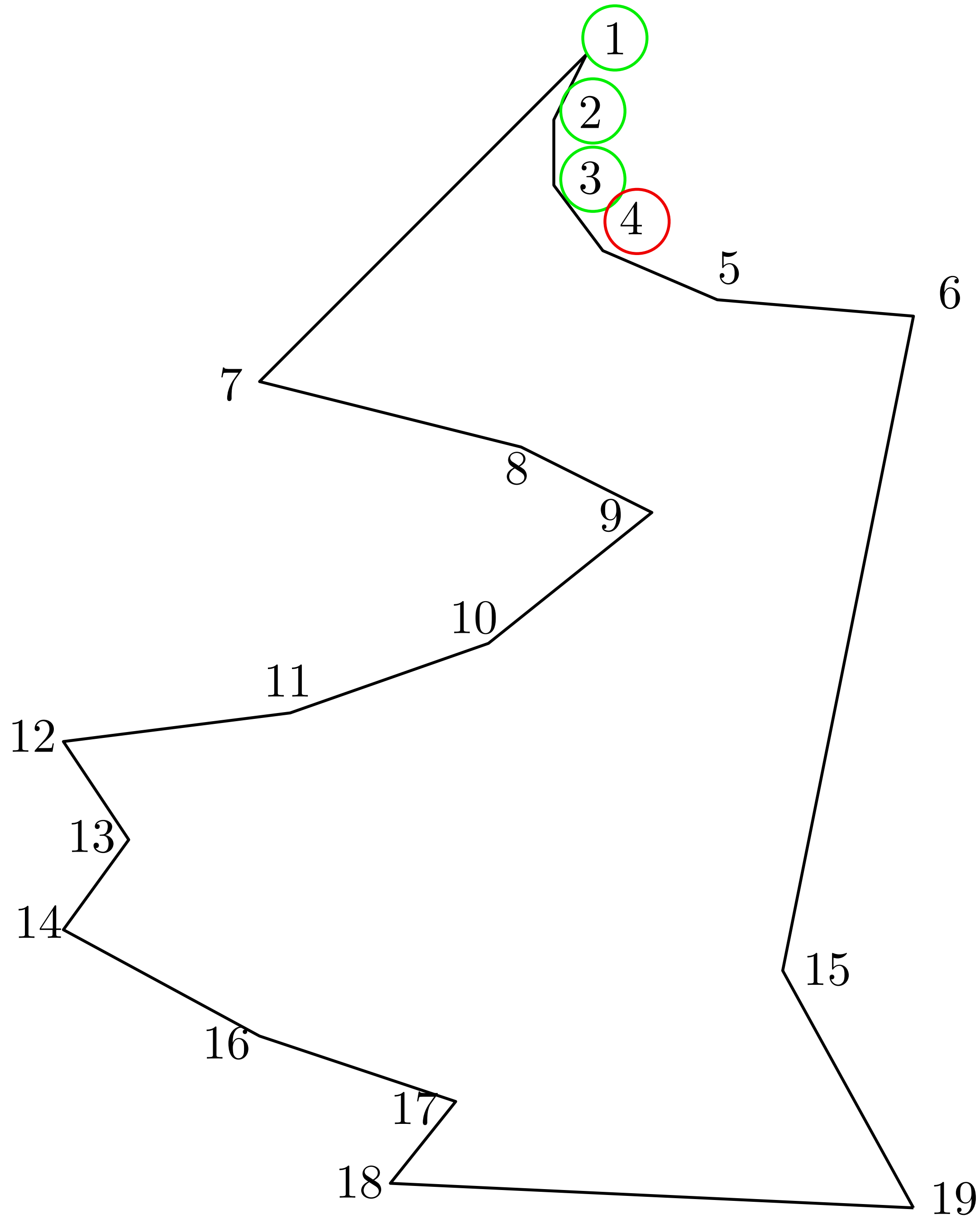
Triangulating a monotone polygon

Current vertex: 4

Add

Queue state:

1, 2, 3, 4



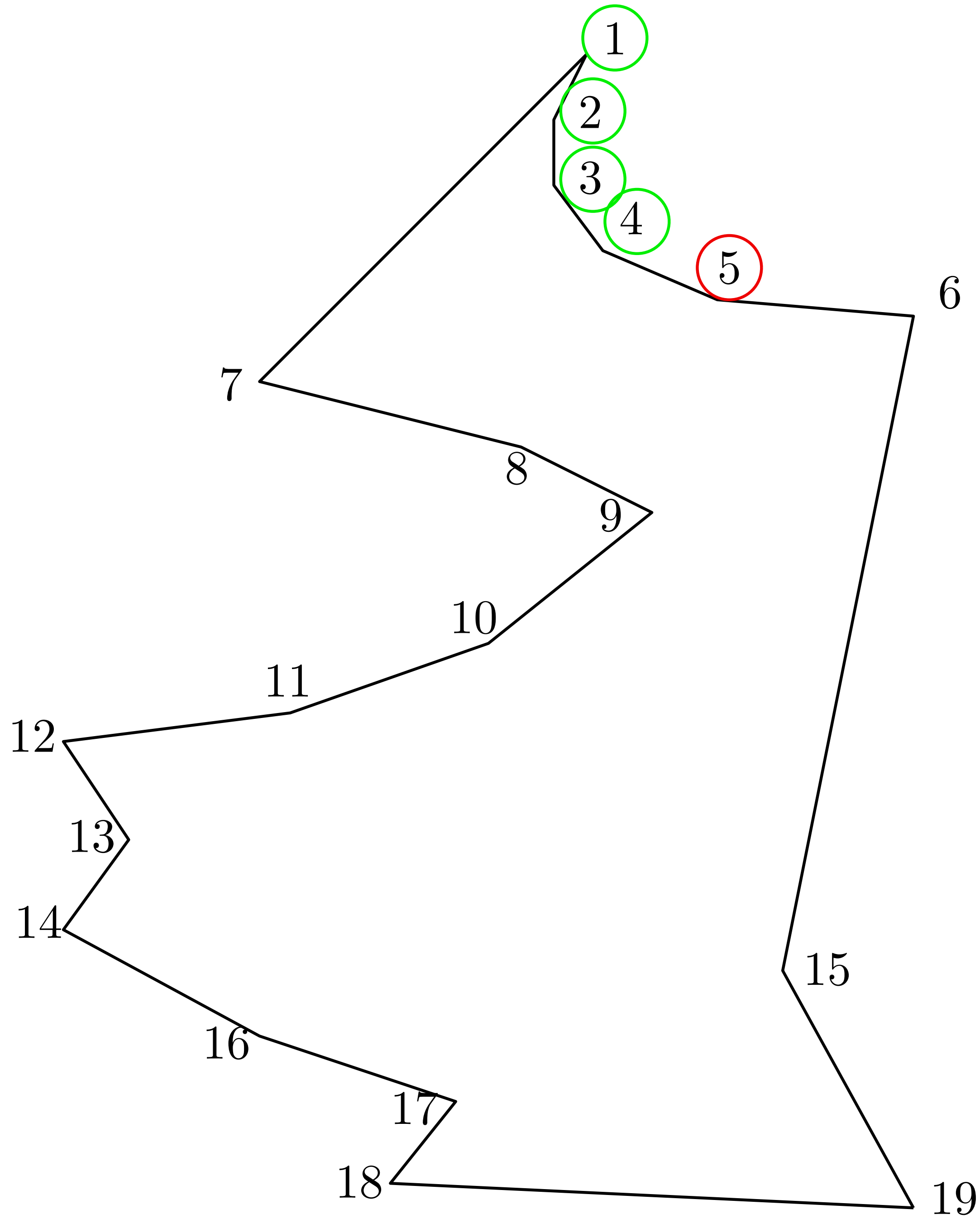
Triangulating a monotone polygon

Current vertex: 5

Add

Queue state:

1, 2, 3, 4, 5



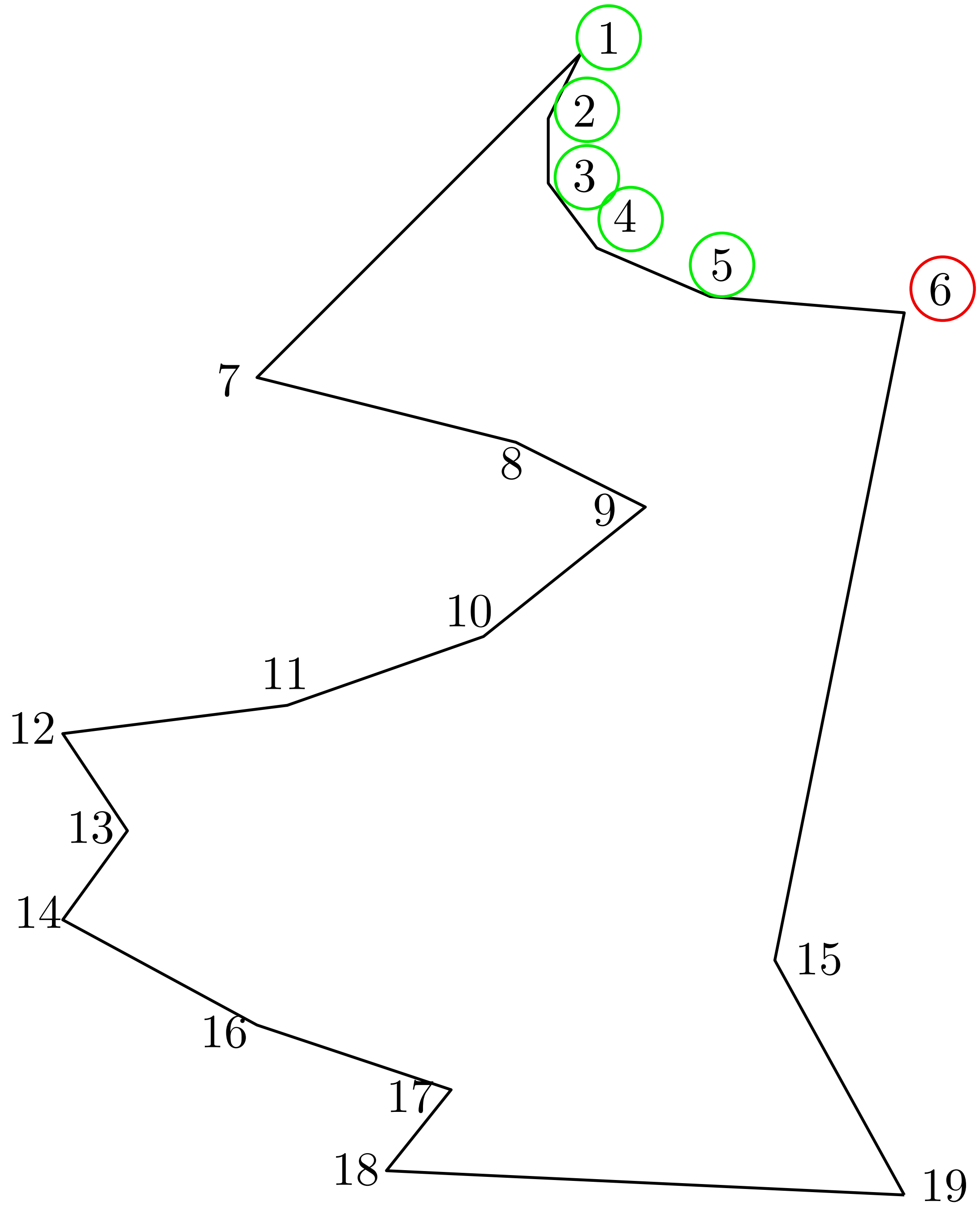
Triangulating a monotone polygon

Current vertex: 6

Add

Queue state:

1, 2, 3, 4, 5, 6



TRIANGULATING POLYGONS

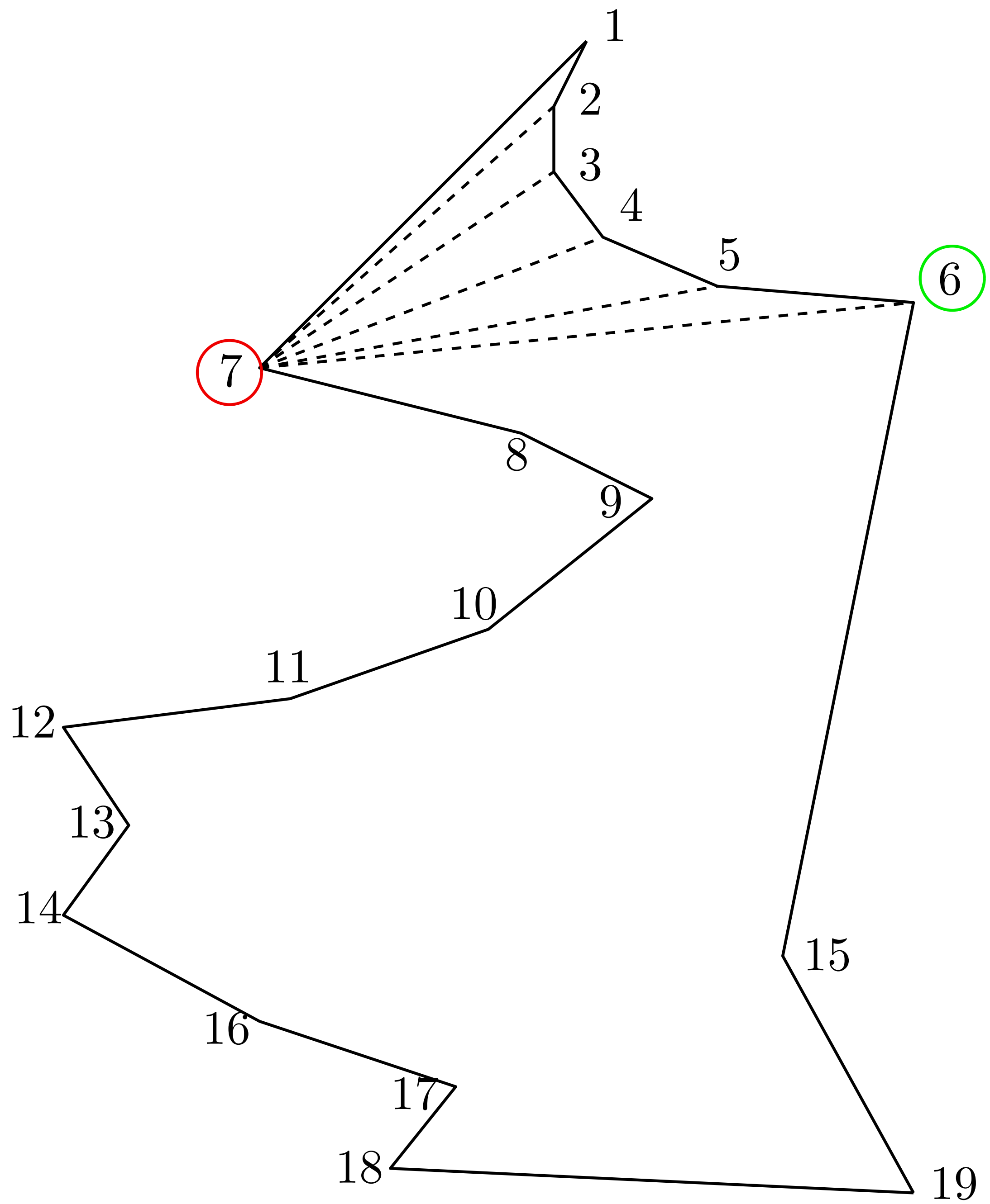
Triangulating a monotone polygon

Current vertex: 7

Opposite chain

Queue state:

6, 7



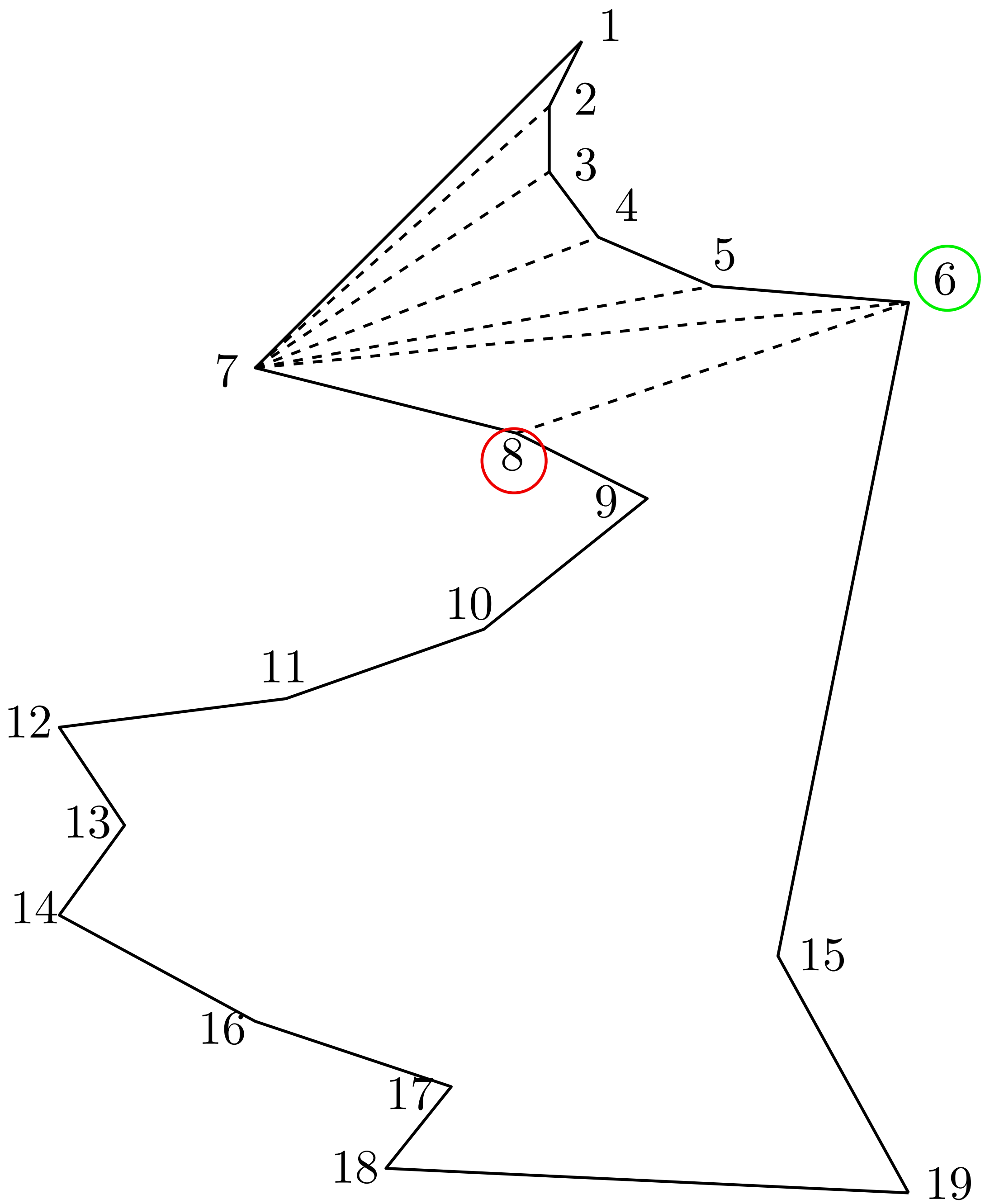
Triangulating a monotone polygon

Current vertex: 8

Ear

Queue state:

6, 8



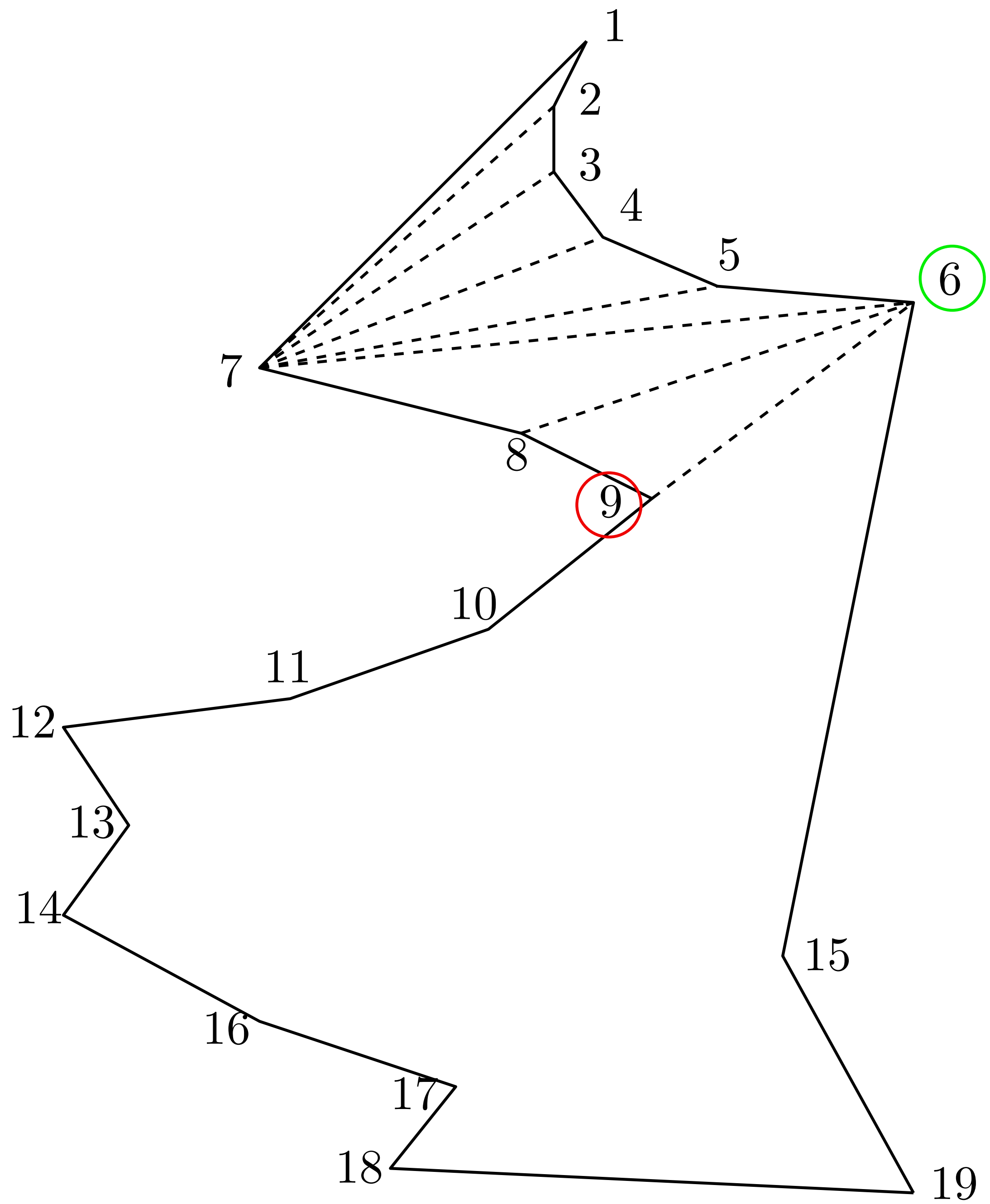
Triangulating a monotone polygon

Current vertex: 9

Ear

Queue state:

6, 9



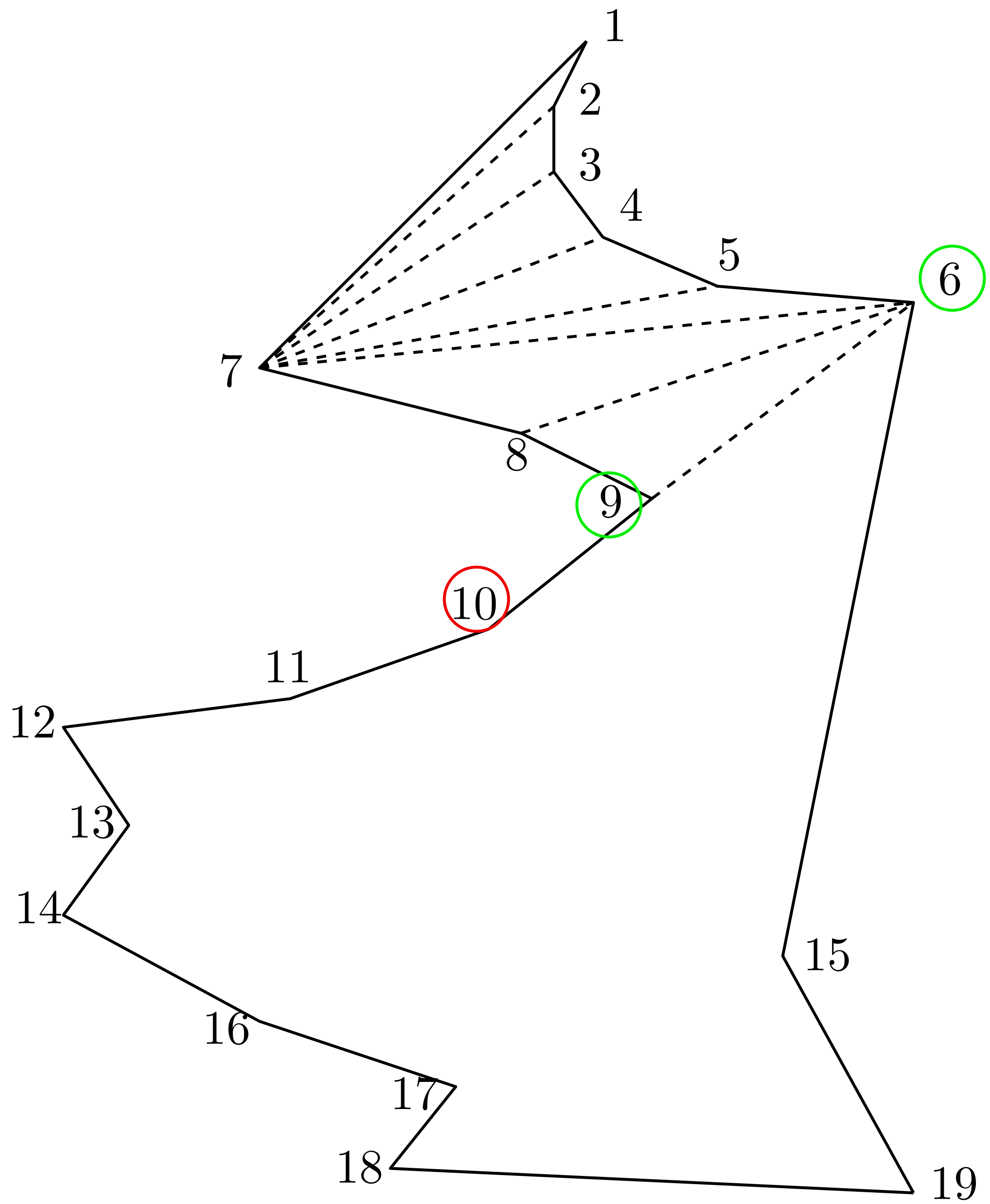
Triangulating a monotone polygon

Current vertex: 10

Add

Queue state:

6, 9, 10



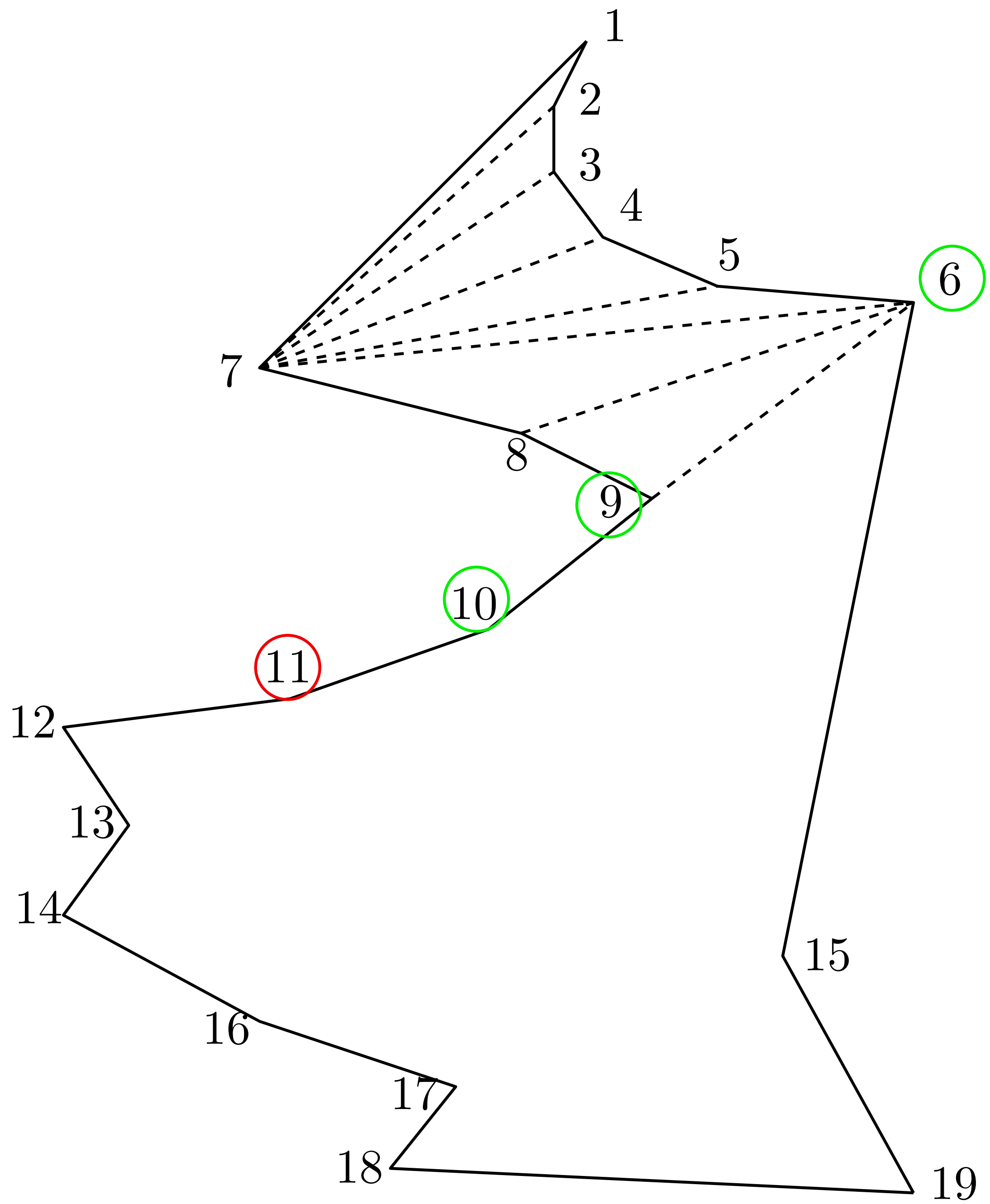
Triangulating a monotone polygon

Current vertex: 11

Add

Queue state:

6, 9, 10, 11



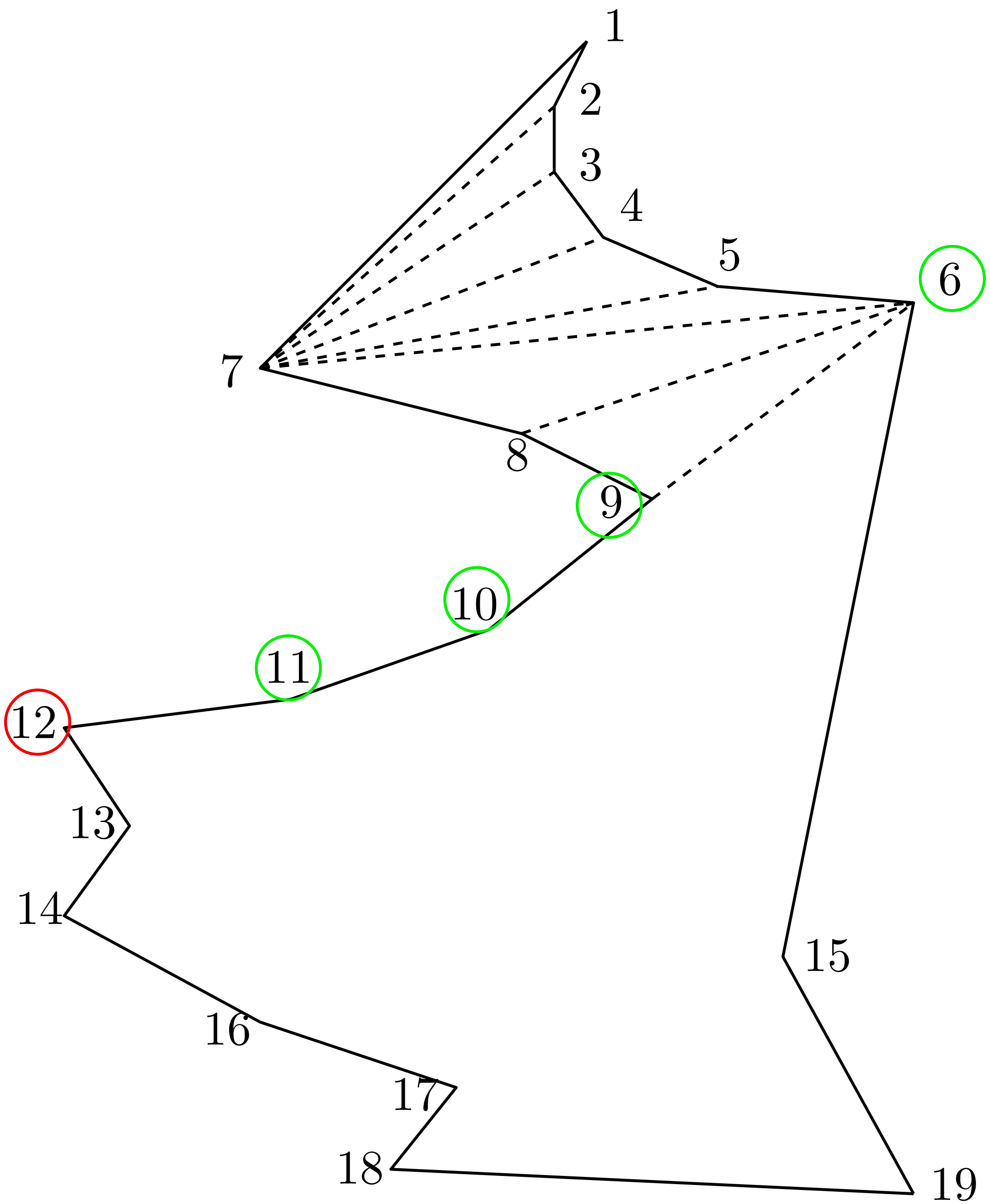
Triangulating a monotone polygon

Current vertex: 12

Add

Queue state:

6, 9, 10, 11, 12



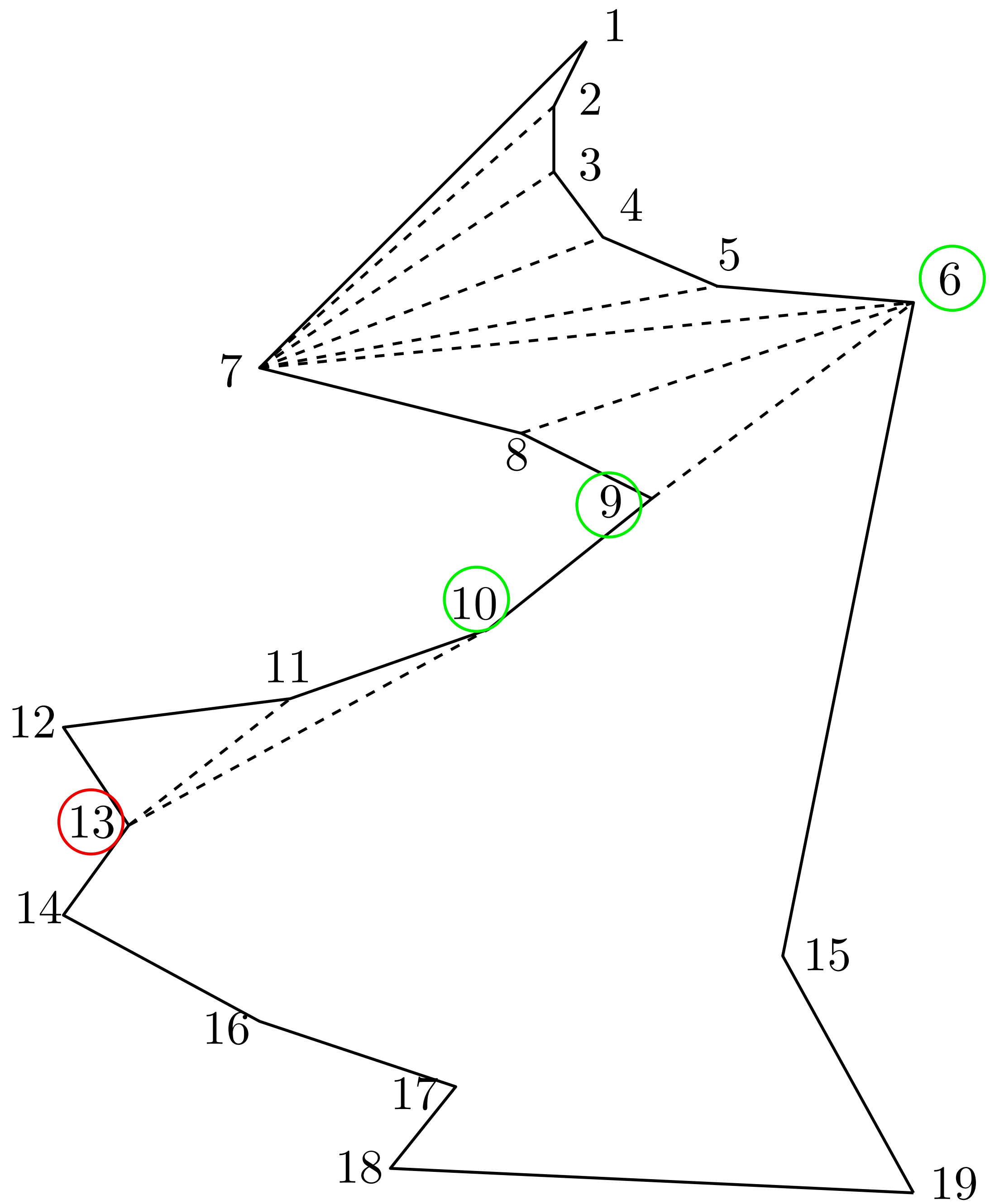
Triangulating a monotone polygon

Current vertex: 13

Ear

Queue state:

6, 9, 10, 13



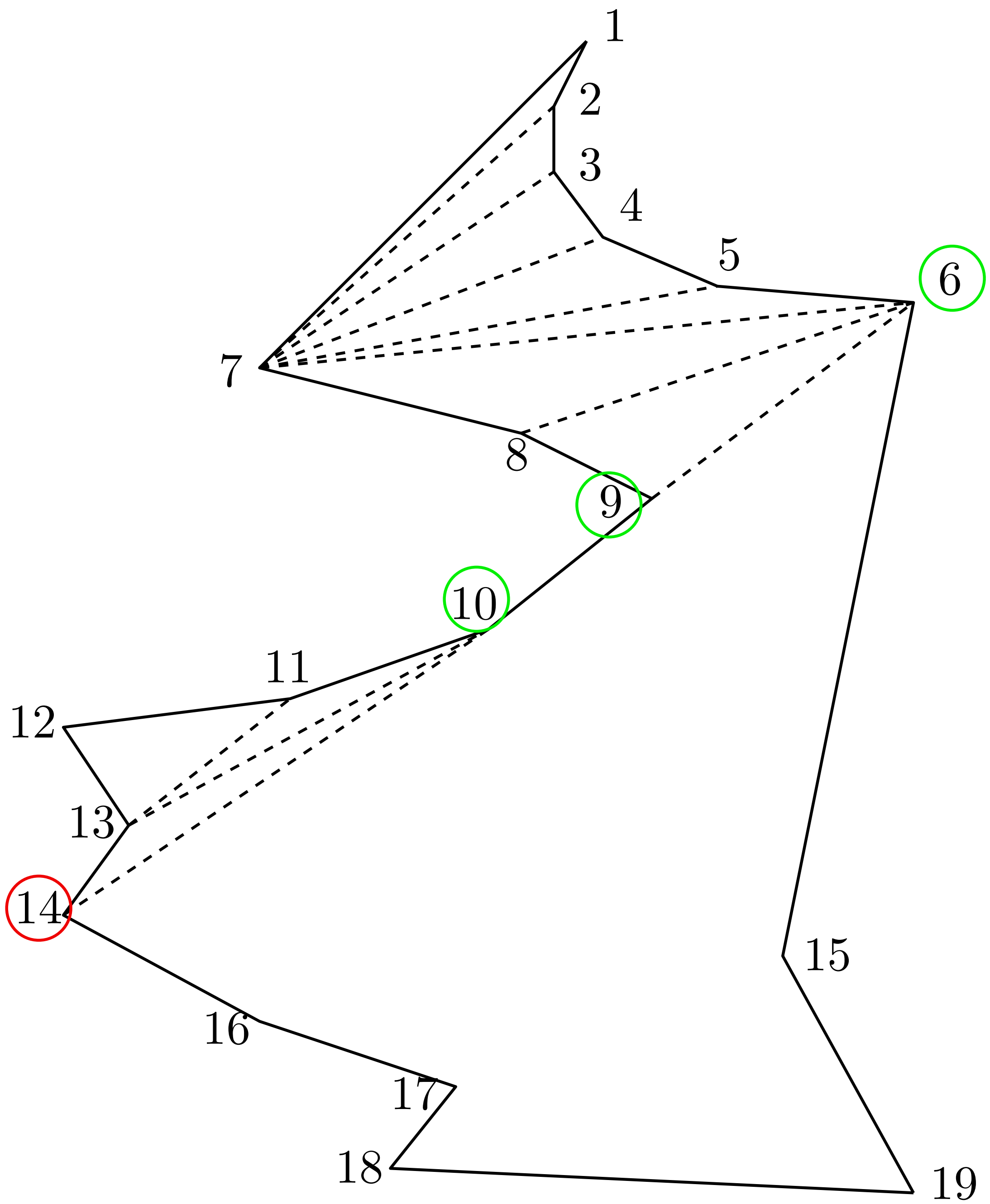
Triangulating a monotone polygon

Current vertex: 14

Ear

Queue state:

6, 9, 10, 14



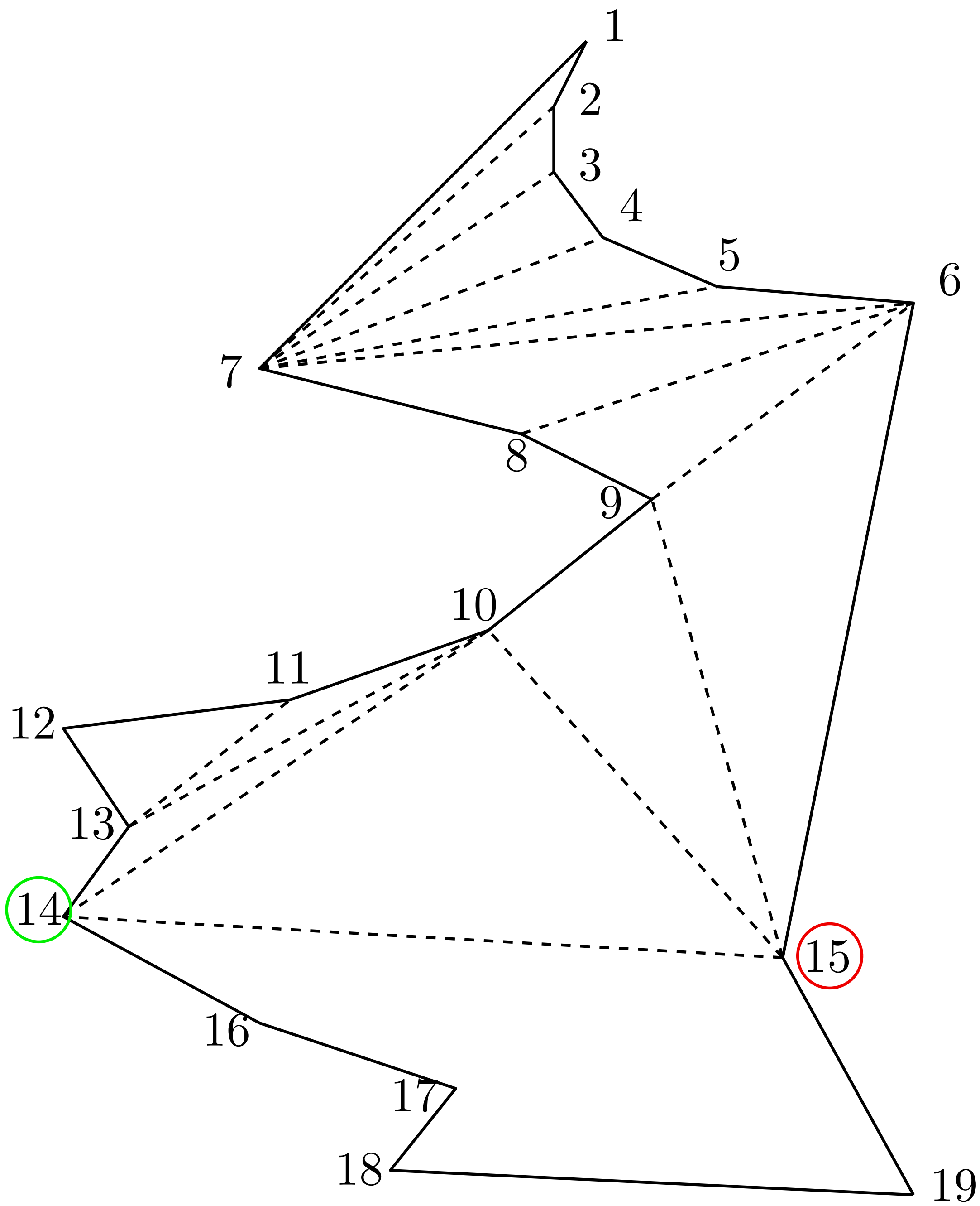
Triangulating a monotone polygon

Current vertex: 15

Opposite chain

Queue state:

14, 15



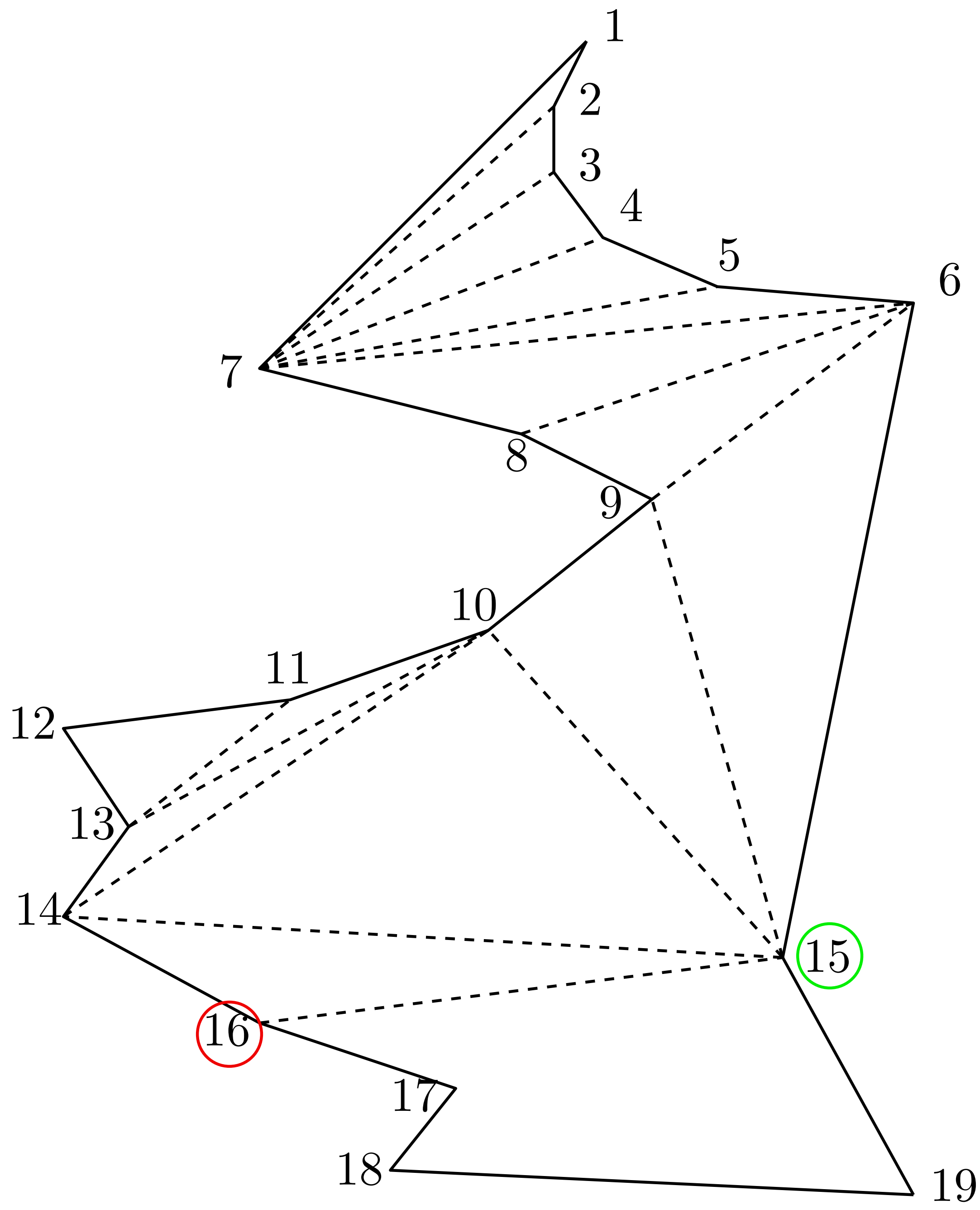
Triangulating a monotone polygon

Current vertex: 16

Opposite chain

Queue state:

15, 16



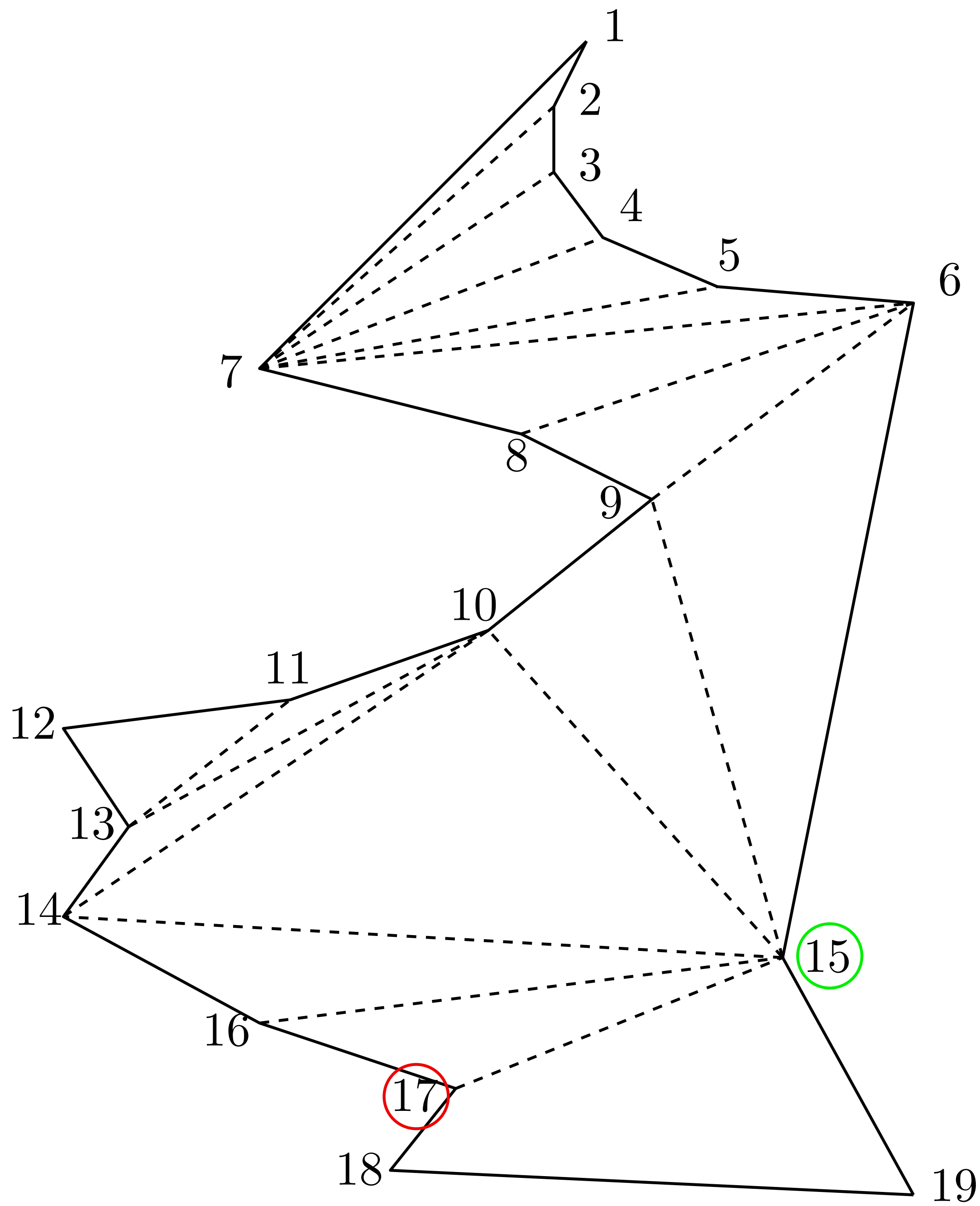
Triangulating a monotone polygon

Current vertex: 17

Ear

Queue state:

15, 17



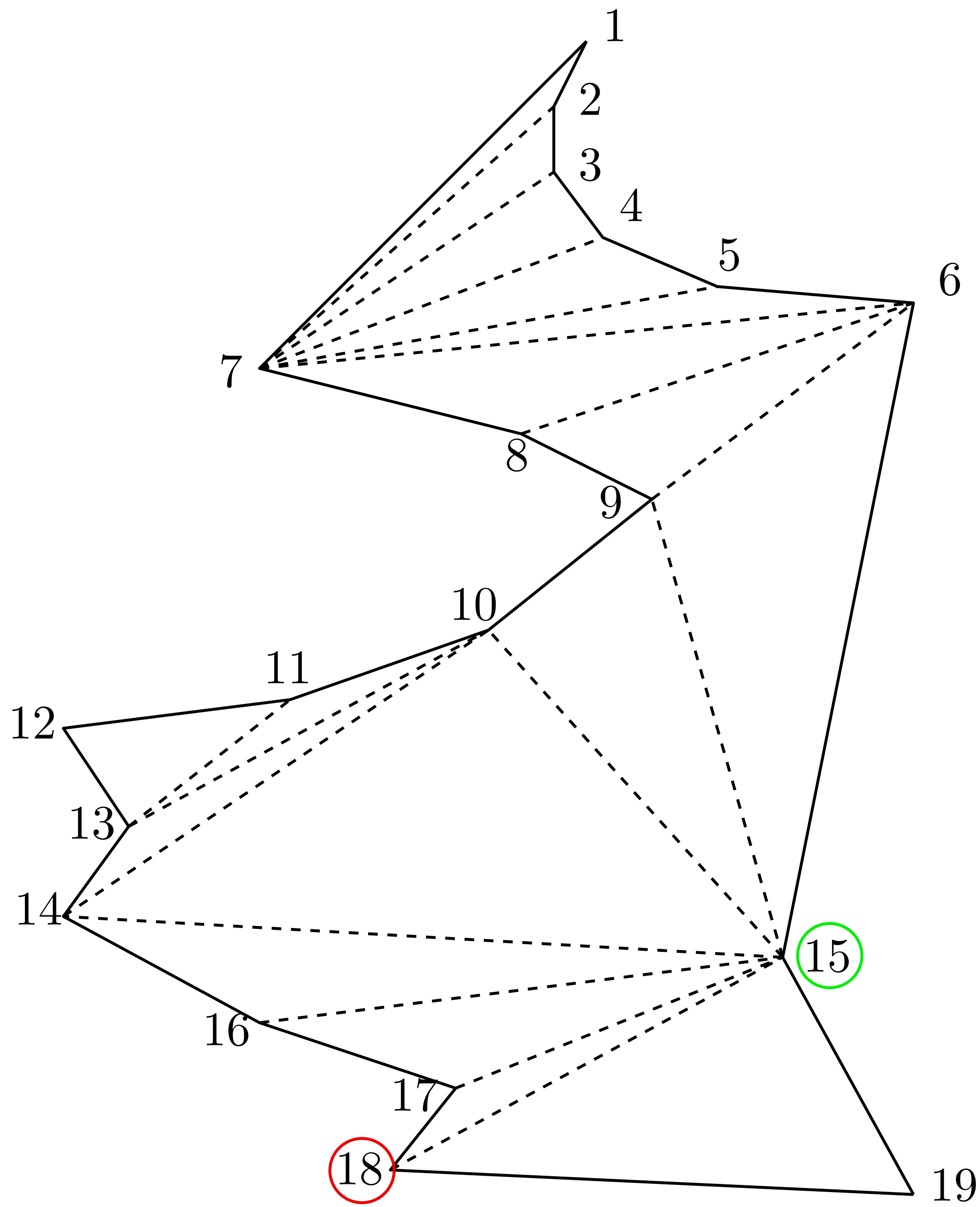
Triangulating a monotone polygon

Current vertex: 18

Ear

Queue state:

15, 18



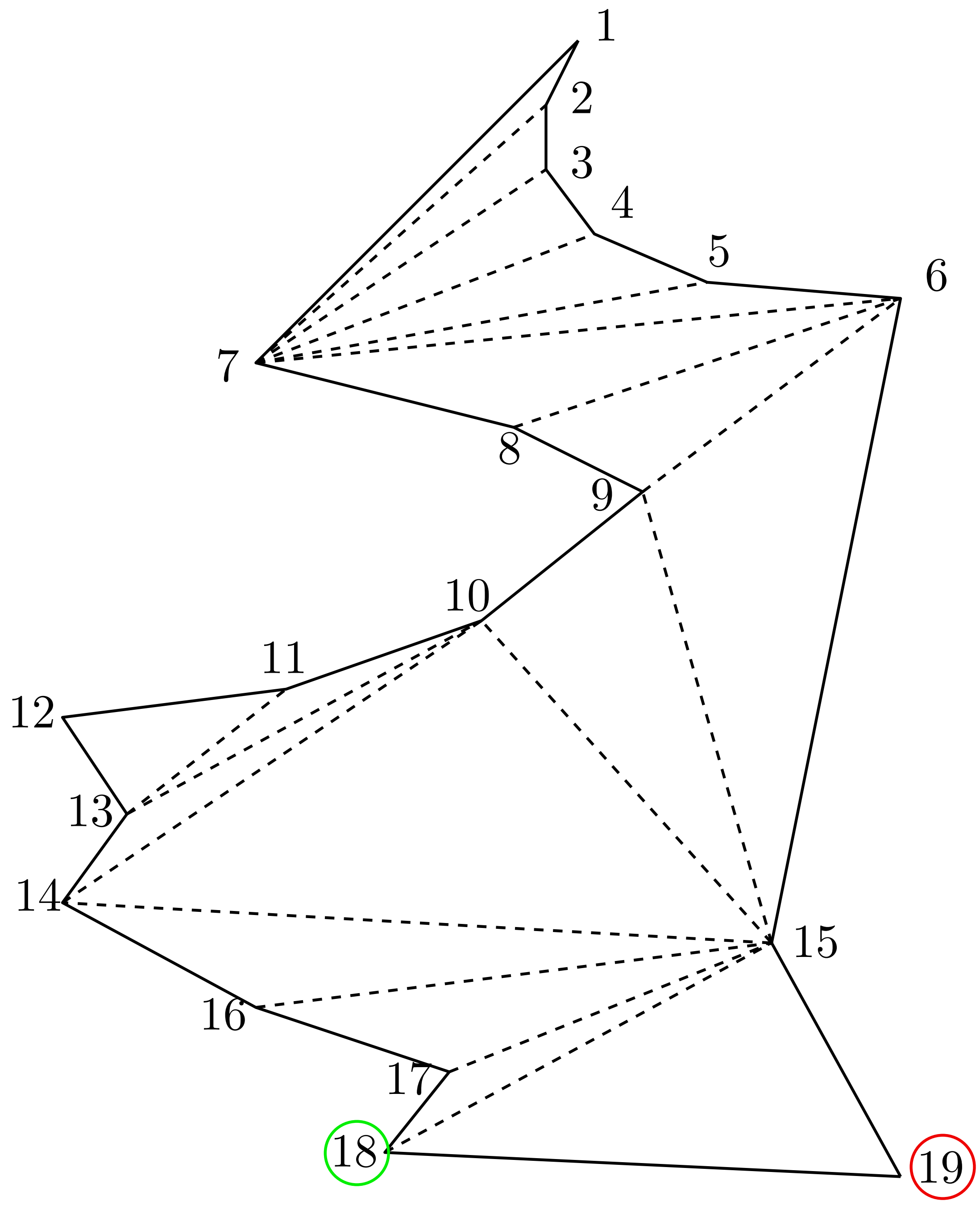
Triangulating a monotone polygon

Current vertex: 19

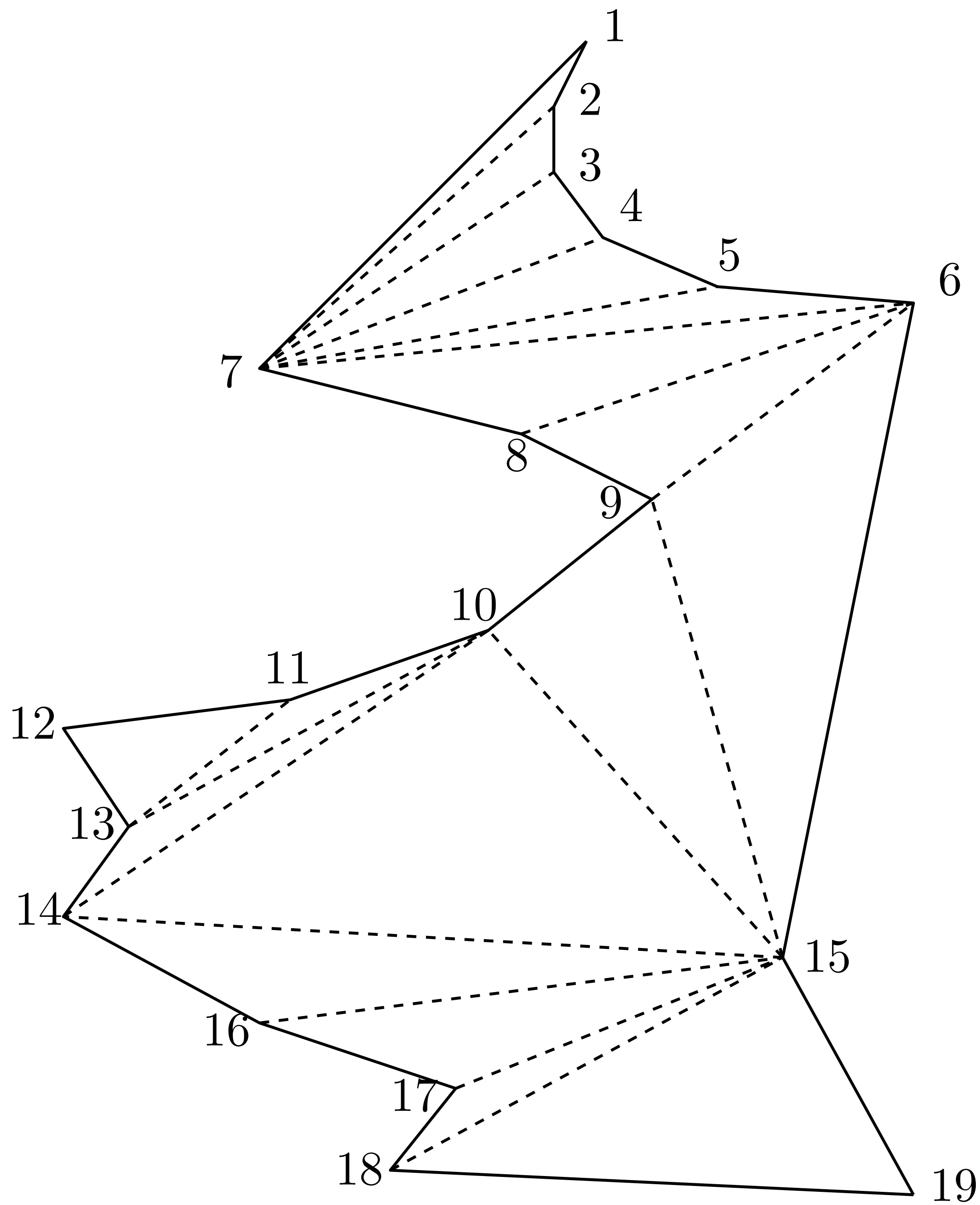
Opposite chain

Queue state:

18, 19



Triangulating a monotone polygon



End

Triangulating a monotone polygon

$O(n \lg n)$

~~**Running time:** $O(n)$~~ ¿¿∃

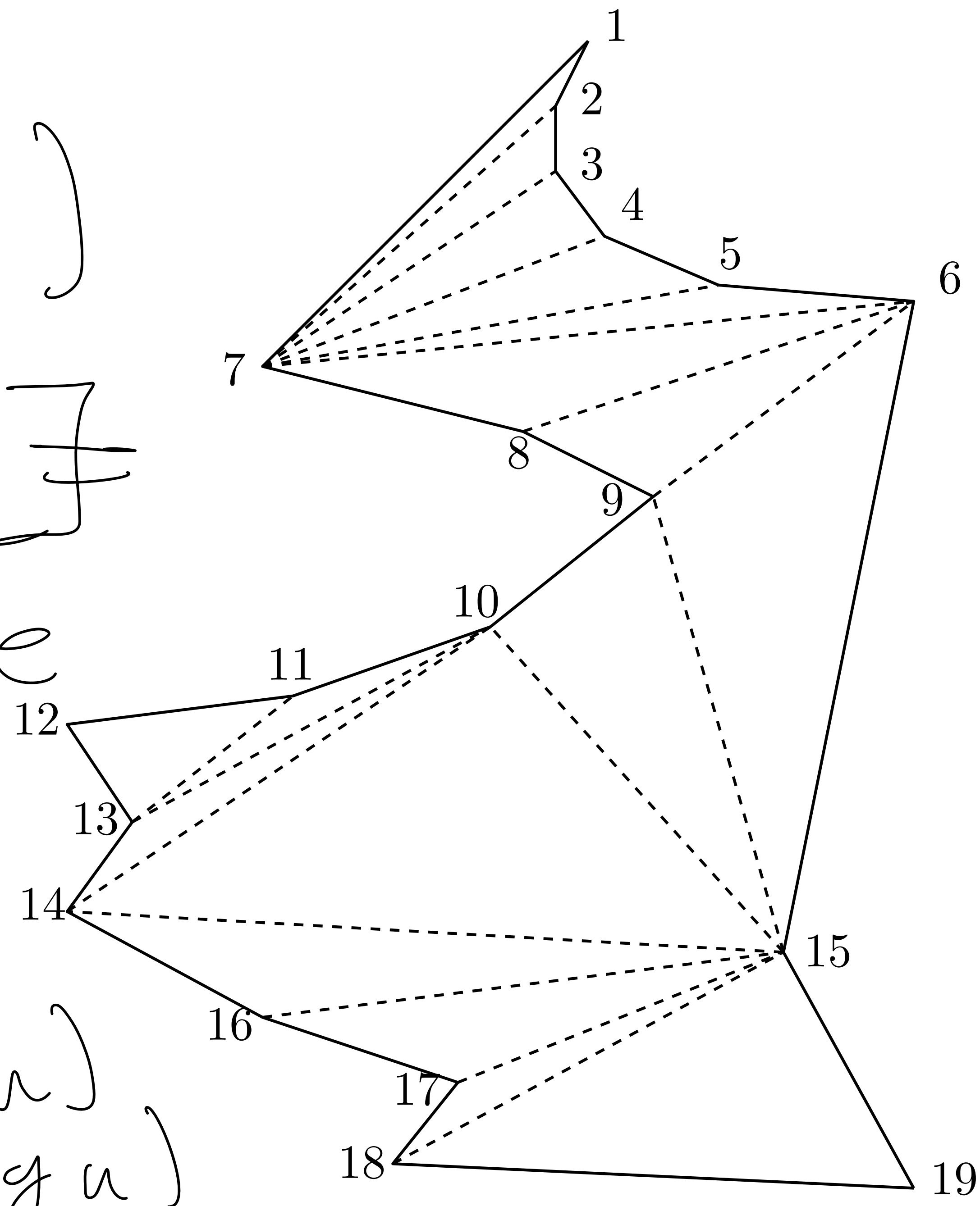
Each vertex is removed from the queue Q in $O(1)$ time.

Q priority queue

prioridad: coord y.

Insertar $O(\lg n)$

Extraer $O(\lg n)$



Summarizing

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

Si la estructura es una pila => O(n)

O(n lg n)

```
Algorithm TRIANGULATEMONOTONEPOLYGON(𝒫)
Input. A strictly y-monotone polygon 𝒫 stored in a doubly-connected edge
  list 𝒟.
Output. A triangulation of 𝒫 stored in the doubly-connected edge list 𝒟.
Merge the vertices on the left chain and the vertices on the right chain of 𝒫
into one sequence, sorted on decreasing y-coordinate. If two vertices have
the same y-coordinate, then the leftmost one comes first. Let u_1, ..., u_n
denote the sorted sequence.
Initialize an empty stack 𝒮, and push u_1 and u_2 onto it.
for j ← 3 to n − 1
  do if u_j and the vertex on top of 𝒮 are on different chains
       then Pop all vertices from 𝒮.
            Insert into 𝒟 a diagonal from u_j to each popped vertex,
            except the last one.
            Push u_{j−1} and u_j onto 𝒮.
       else Pop one vertex from 𝒮.
            Pop the other vertices from 𝒮 as long as the diagonals from
            u_j to them are inside 𝒫. Insert these diagonals into 𝒟. Push
            the last vertex that has been popped back onto 𝒮.
            Push u_j onto 𝒮.
Add diagonals from u_n to all stack vertices except the first and the last one.
```

Summarizing

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 more general polygons?

TRIANGULATING POLYGONS

Summarizing

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:

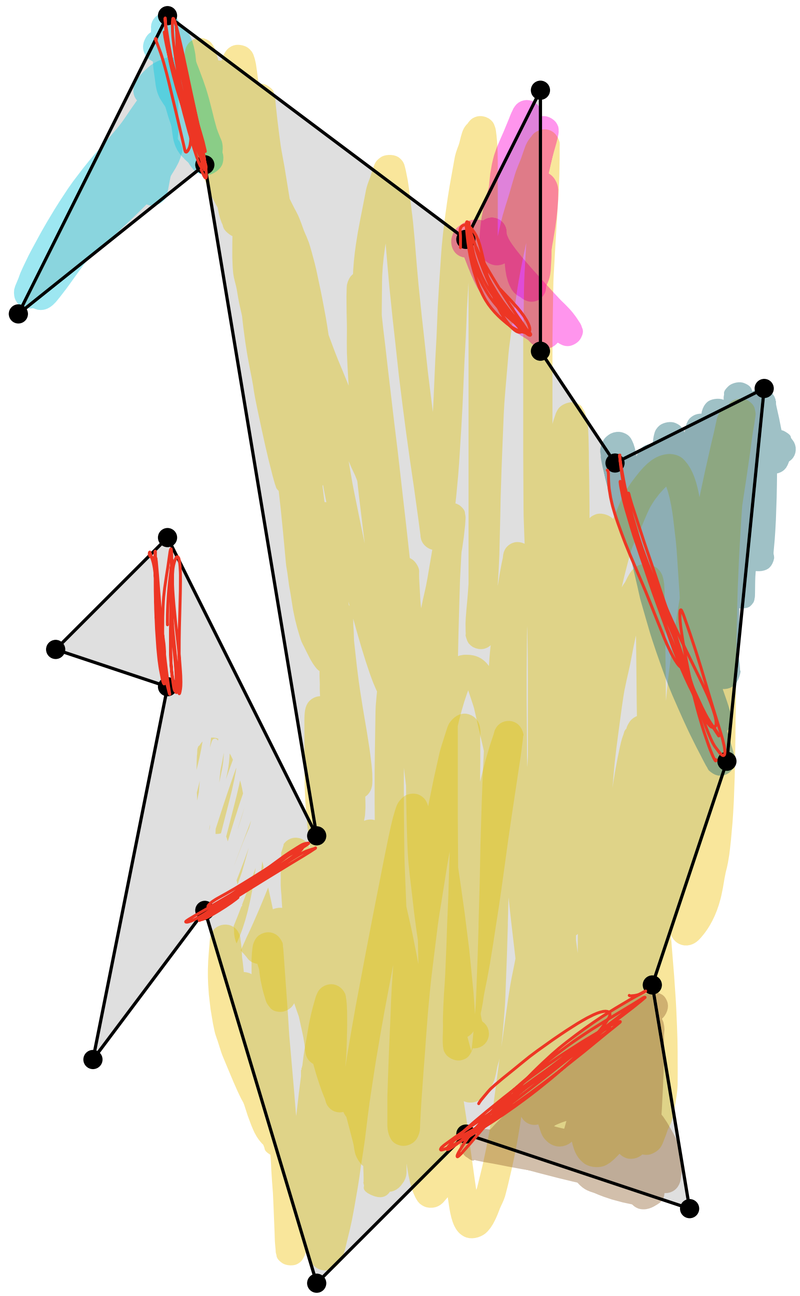
- $O(n)$ scanning the monotone chains in order

Is it possible to be more efficient more general polygons? Yes!

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

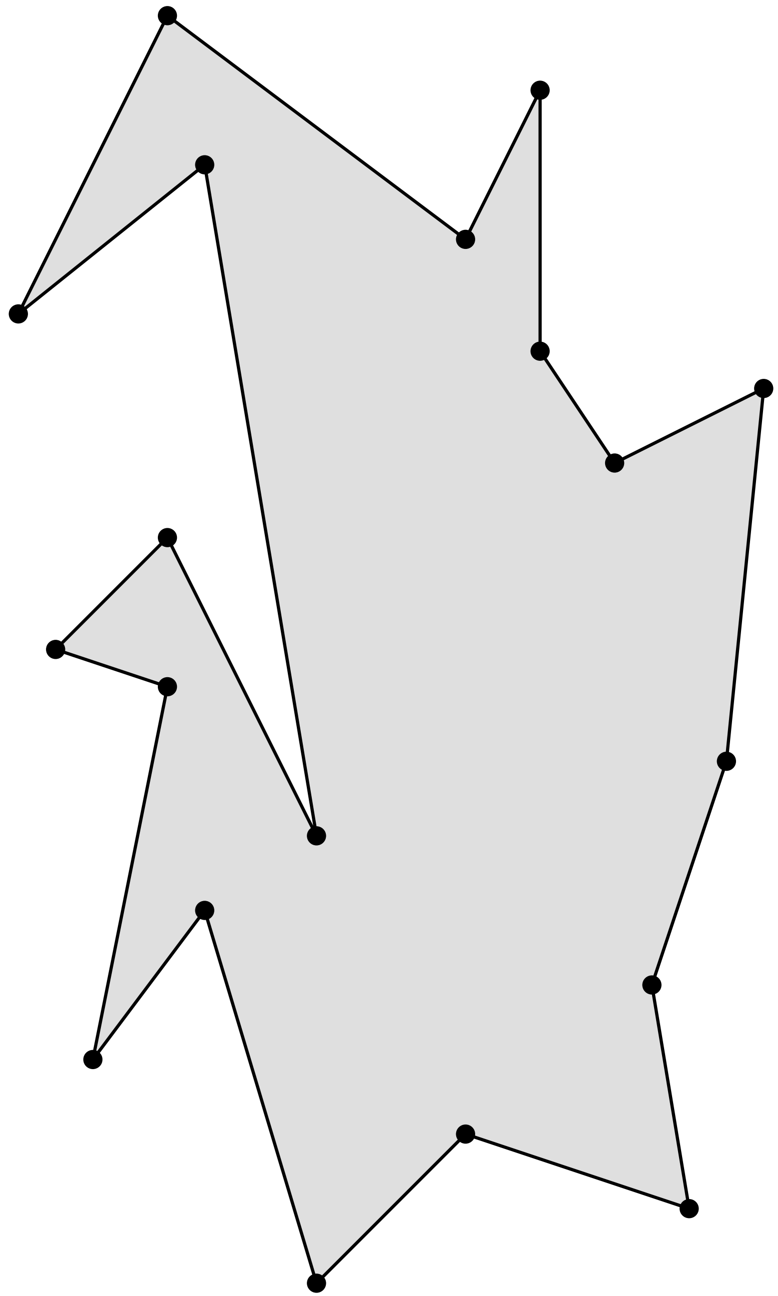
la complejidad dependerá del # de piezas

Monotone partition



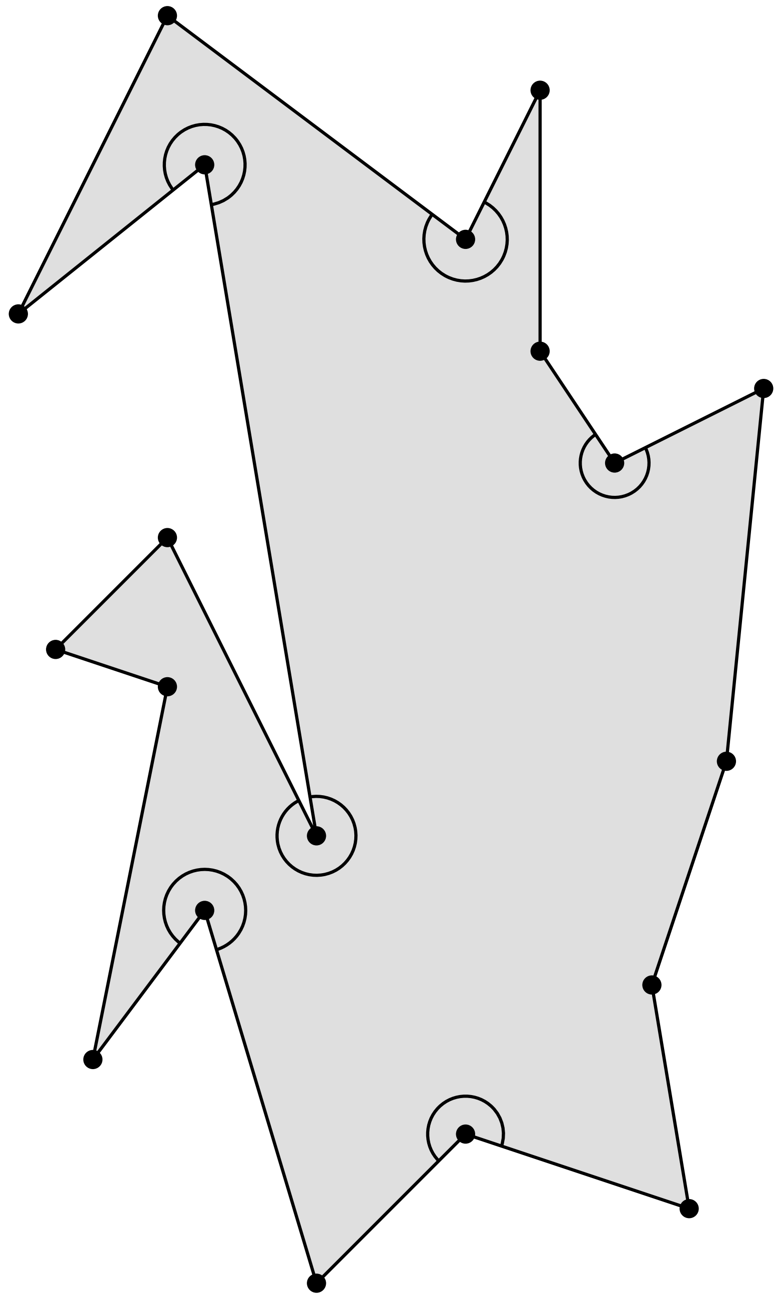
Monotone partition

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



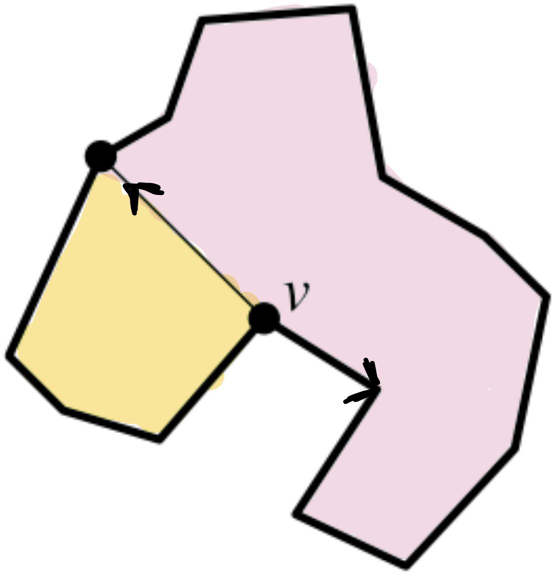
Monotone partition

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

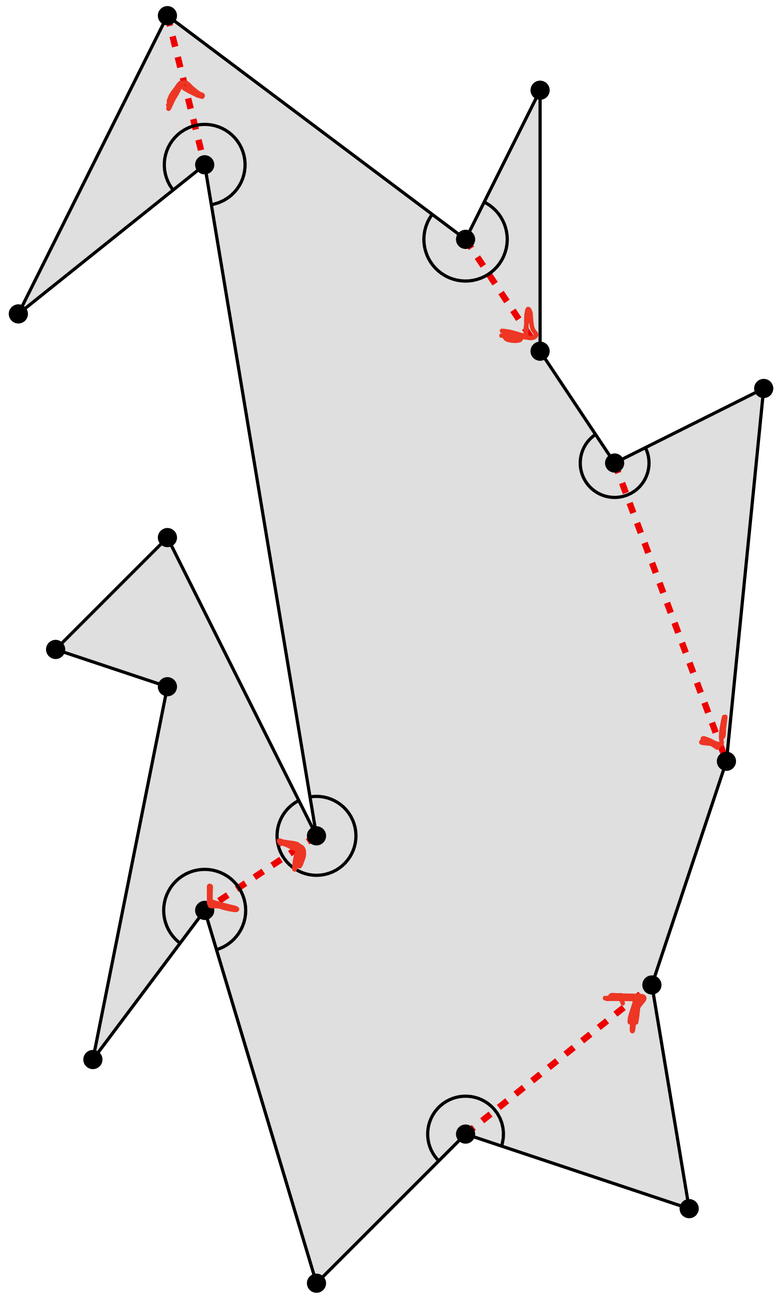


Monotone partition

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

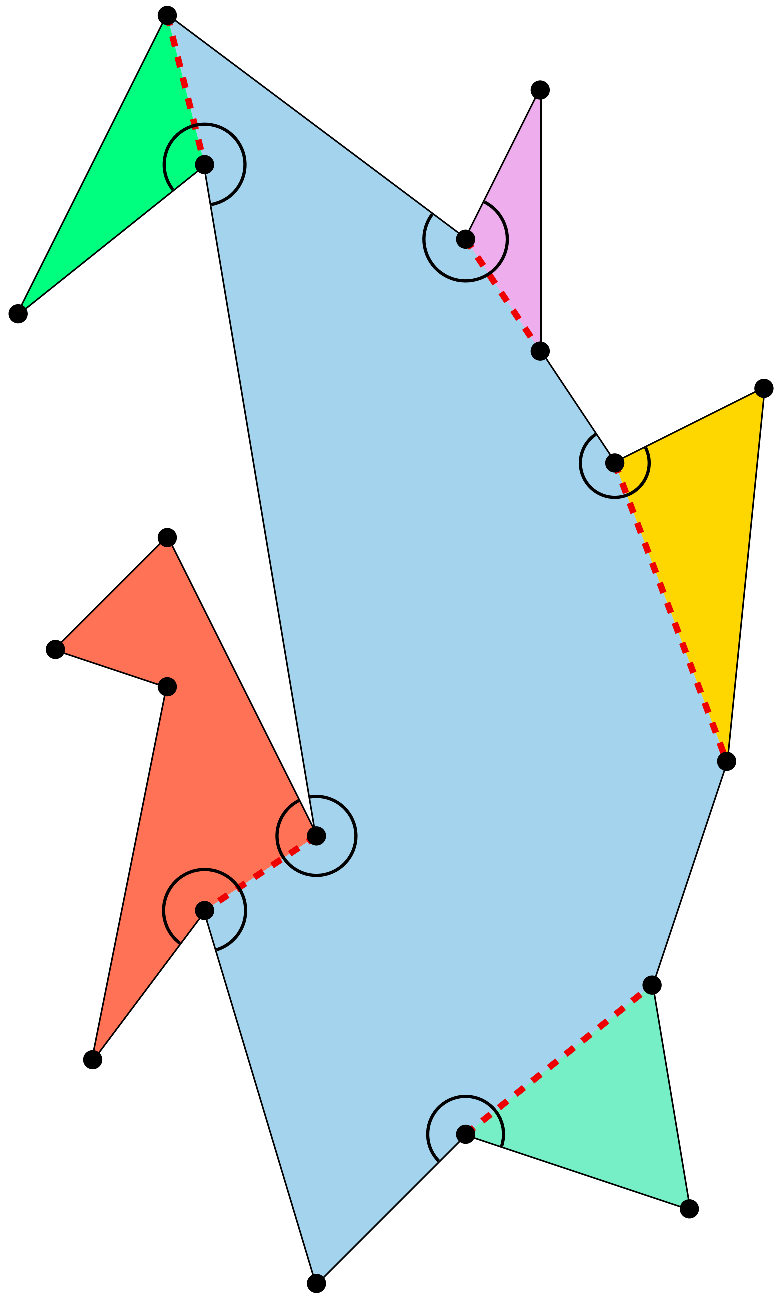


• v deja de ser cúspide



Monotone partition

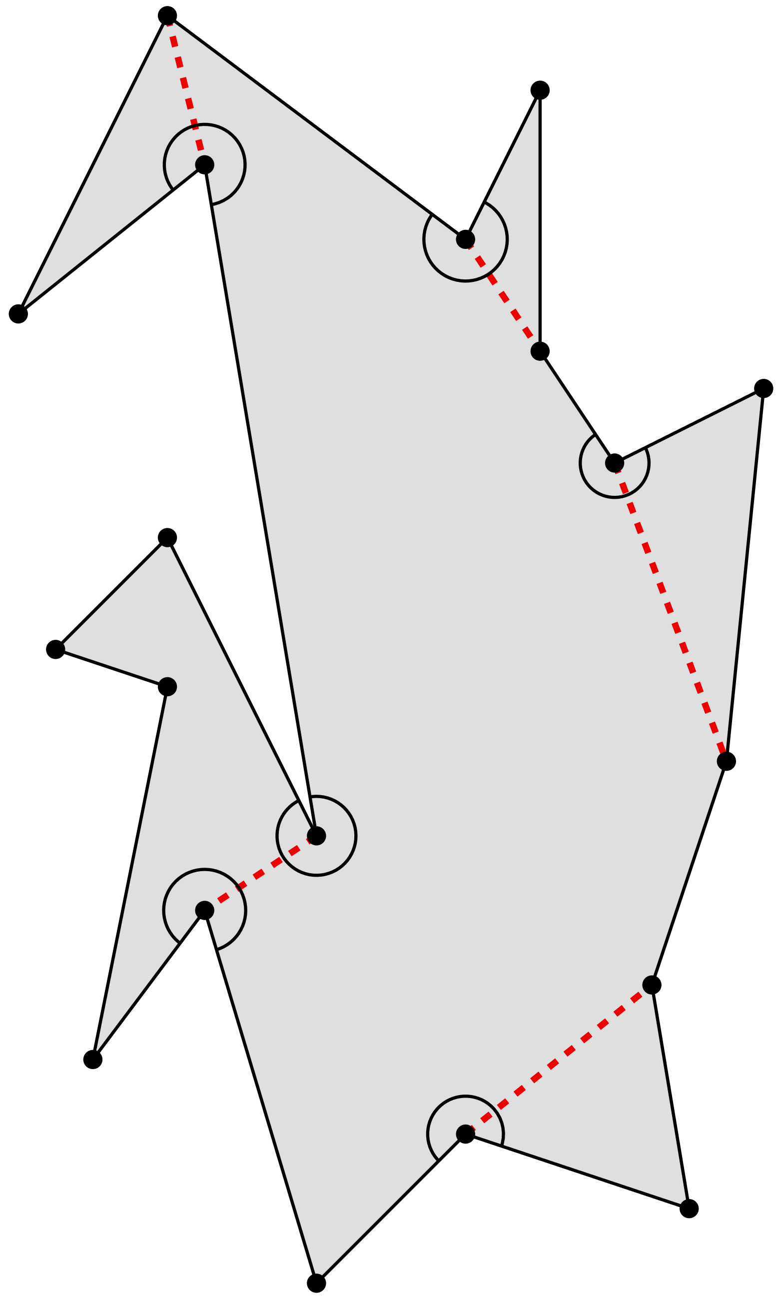
In order to create a monotone partition of a polygon, all cusps need to be “broken” by internal diagonals.



Monotone partition

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

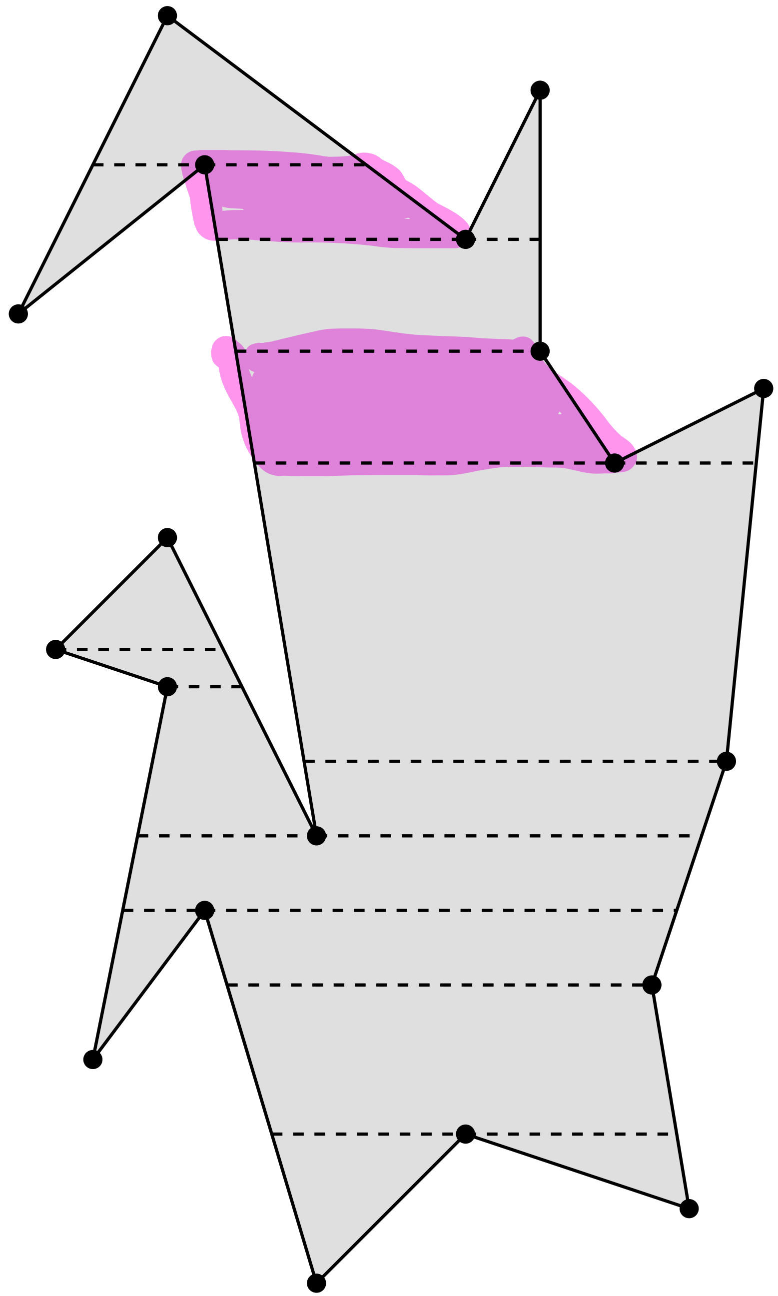
This can be done starting from a **trapezoidal decomposition** of the polygon.



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.

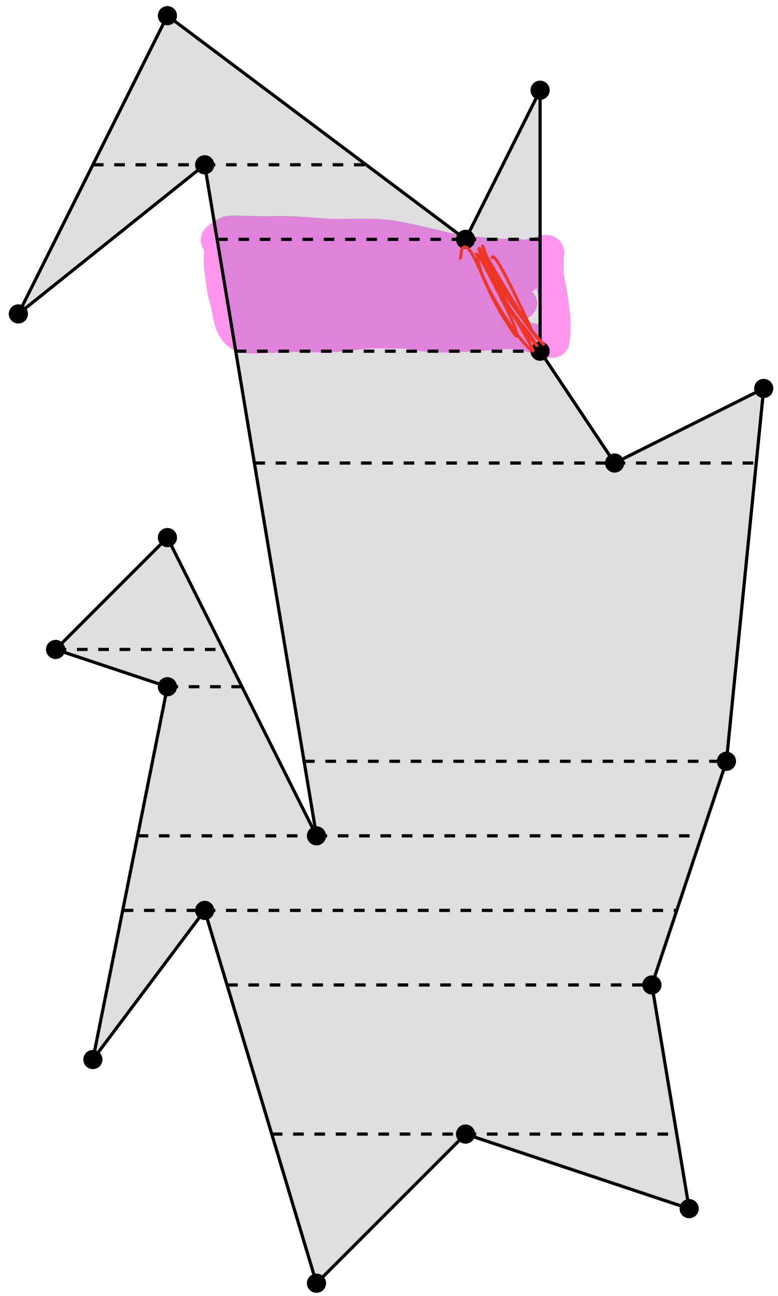


Monotone partition

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).

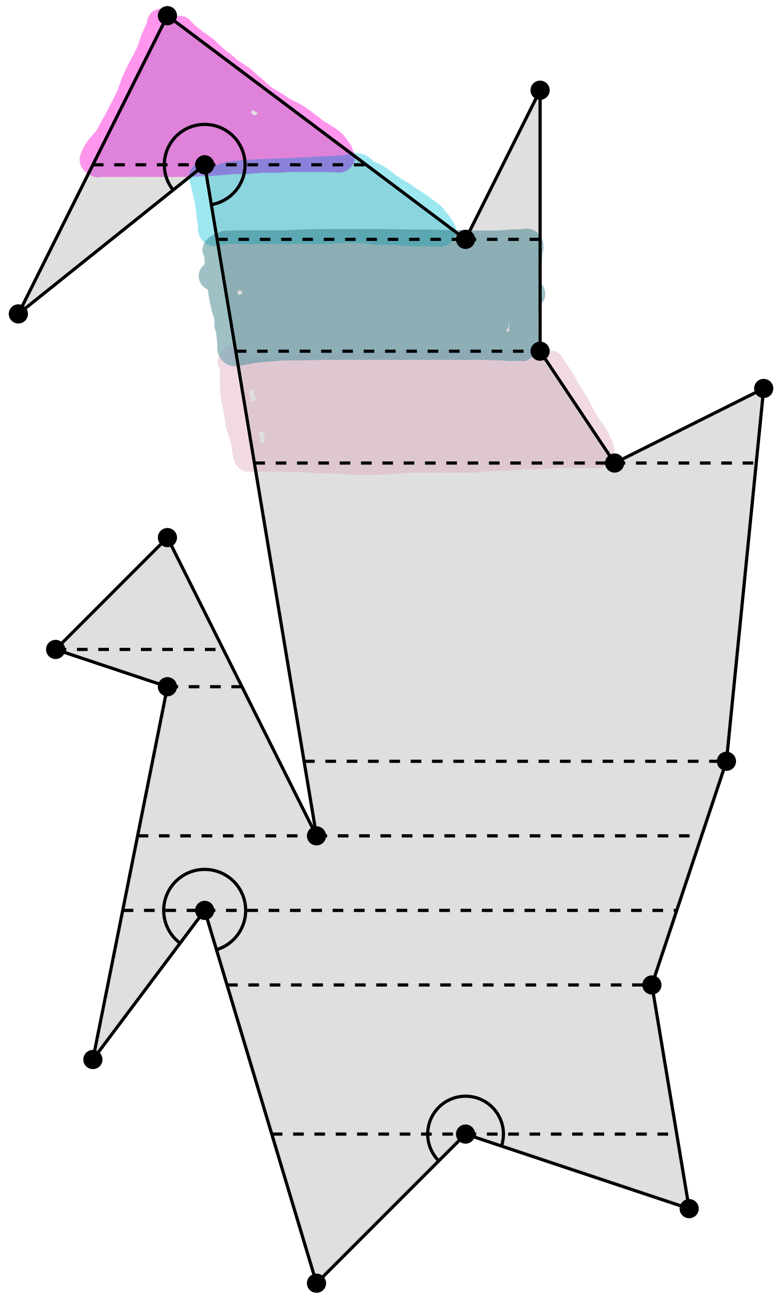


Monotone partition

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).

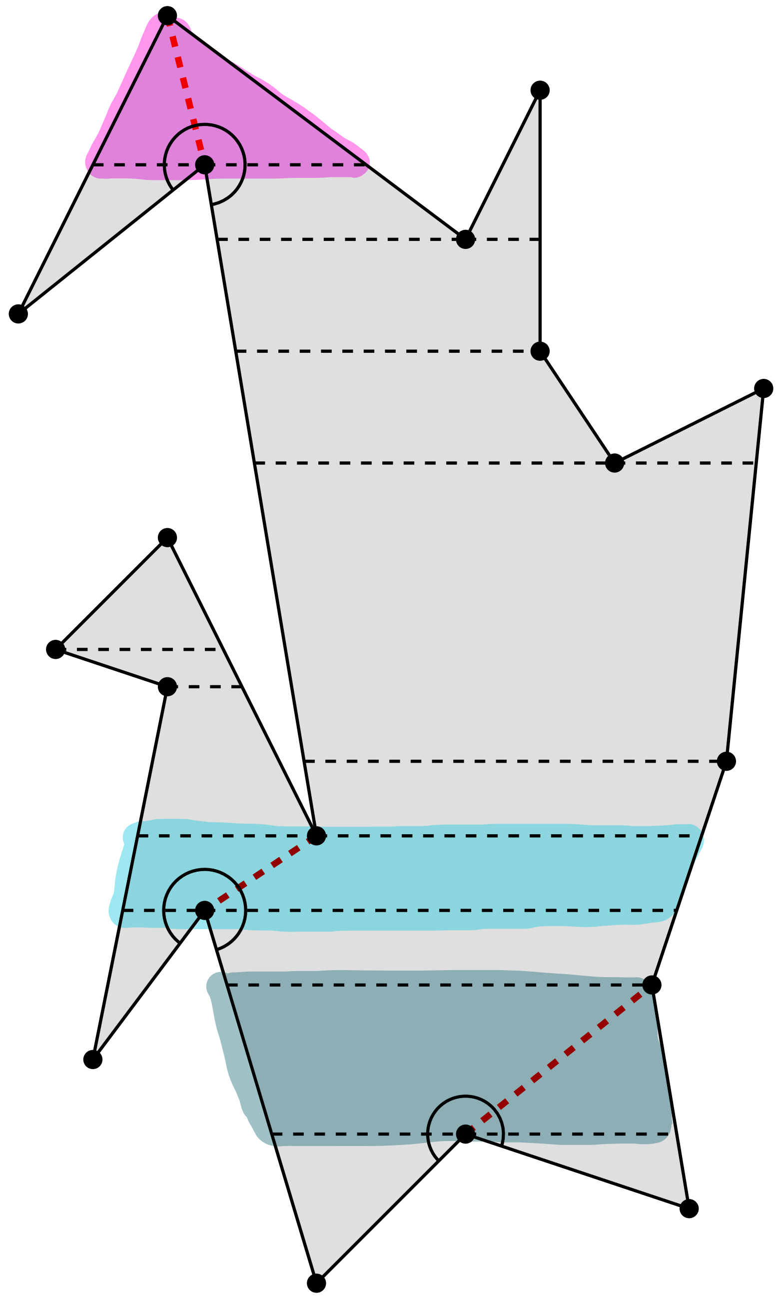


Monotone partition

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).



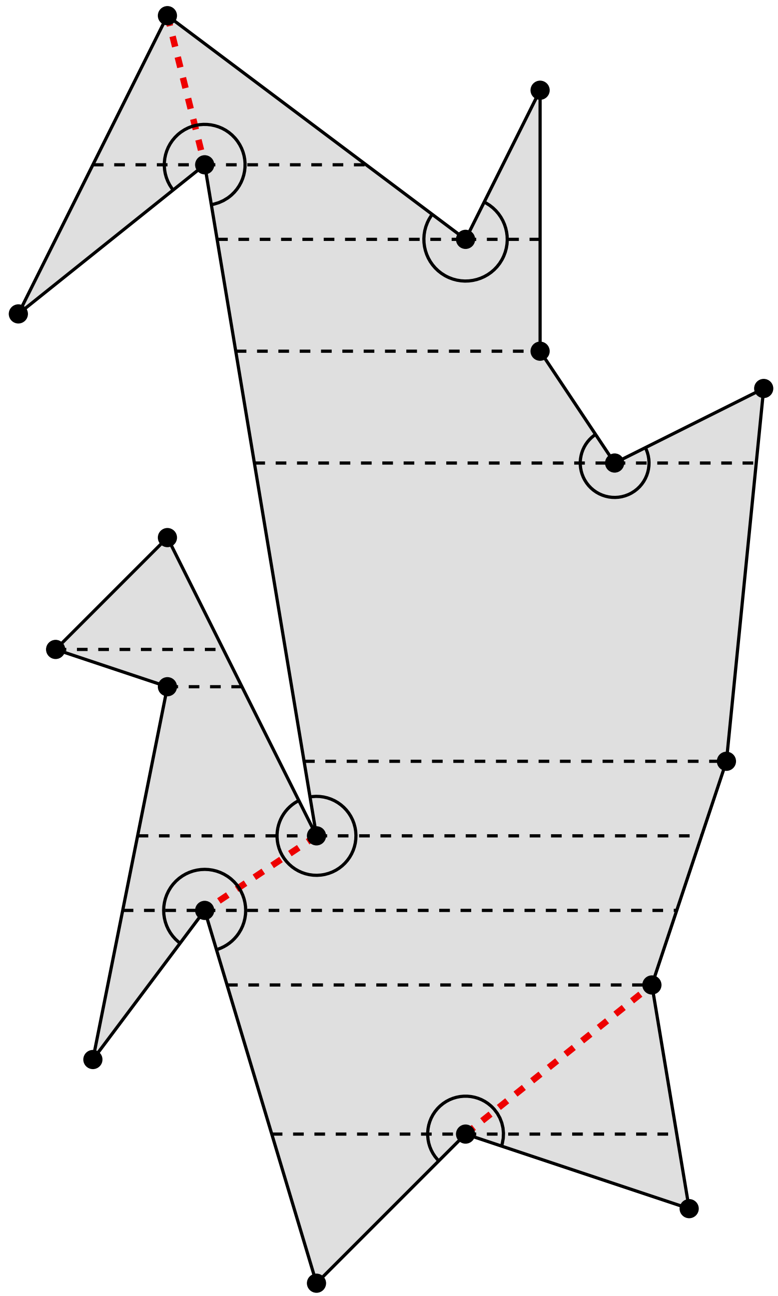
TRIANGULATING POLYGONS

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).

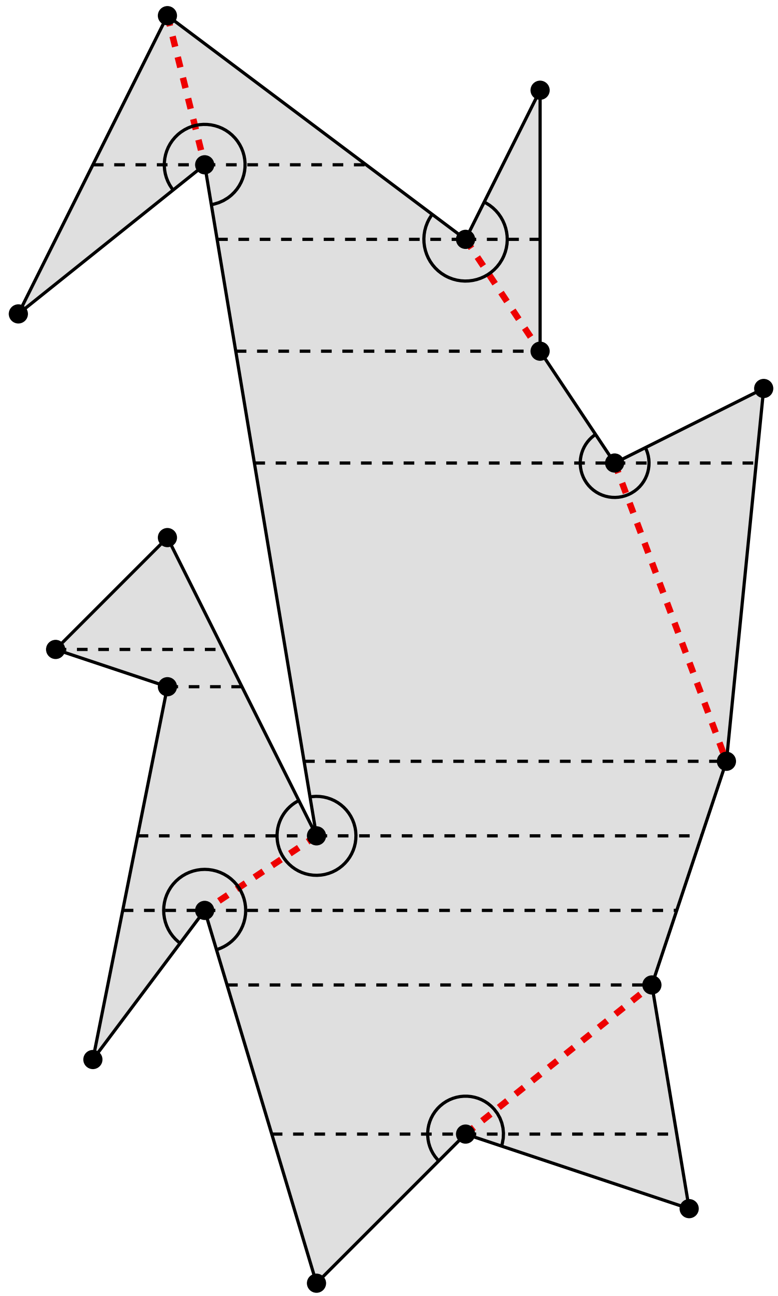


Monotone partition

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.

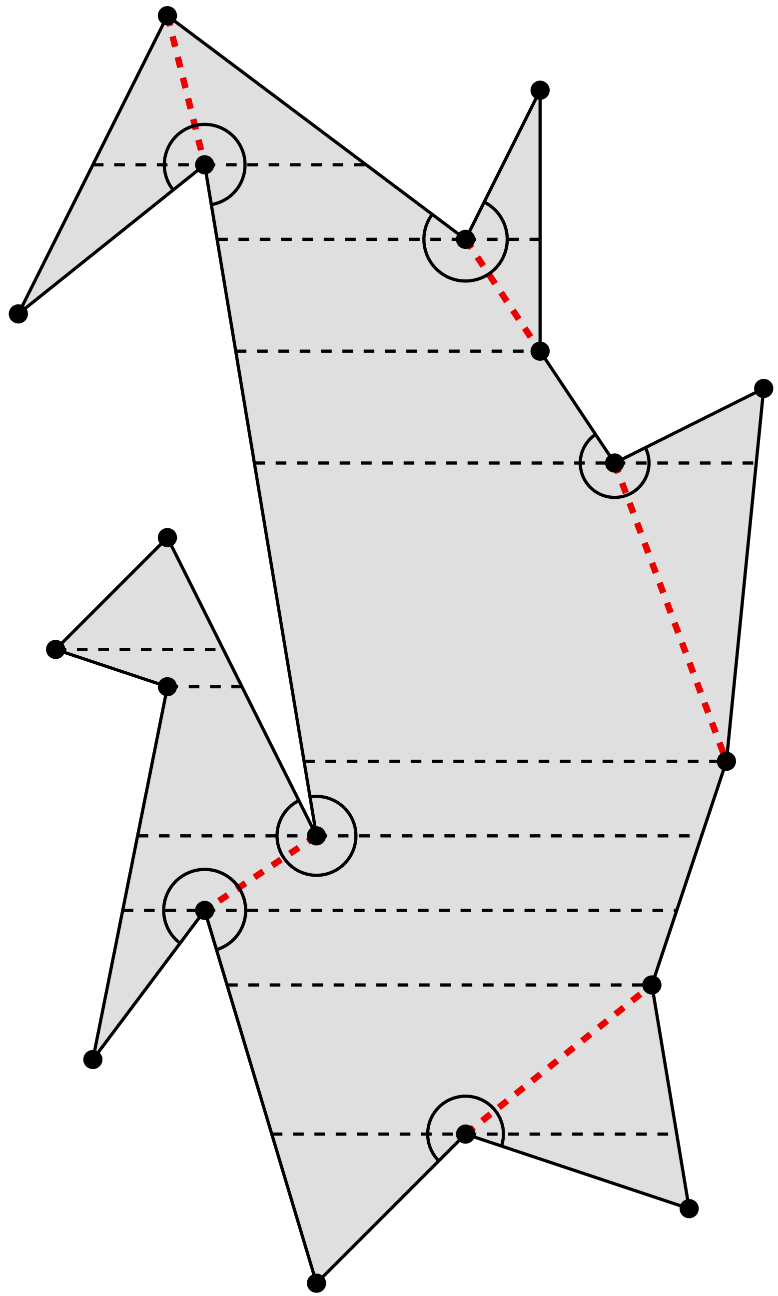
This can be done starting from a **trapezoidal decomposition** of the polygon.

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



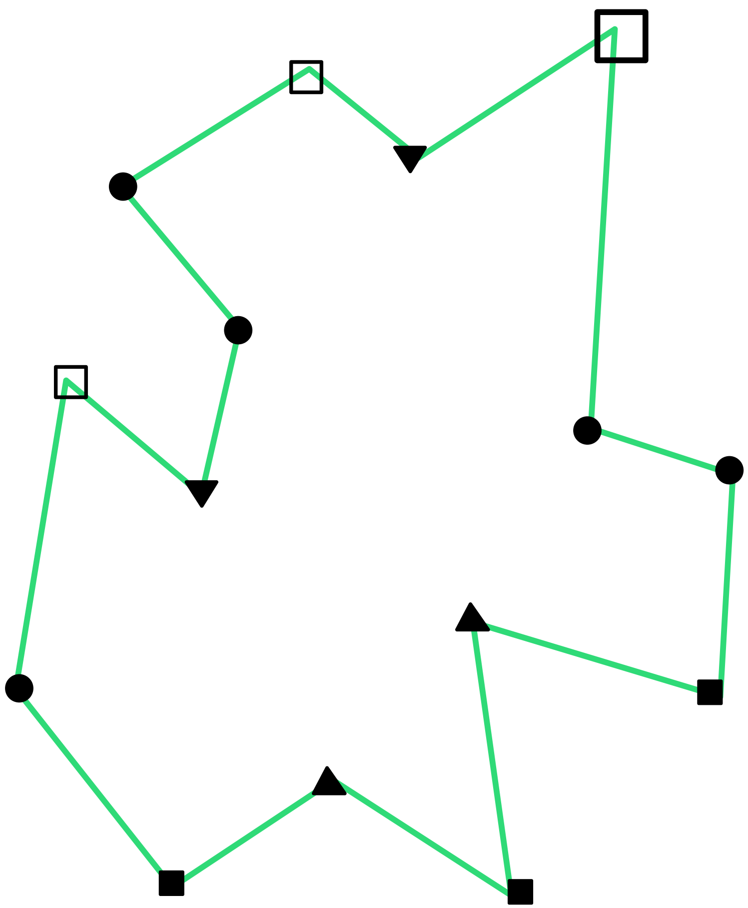
Monotone partition

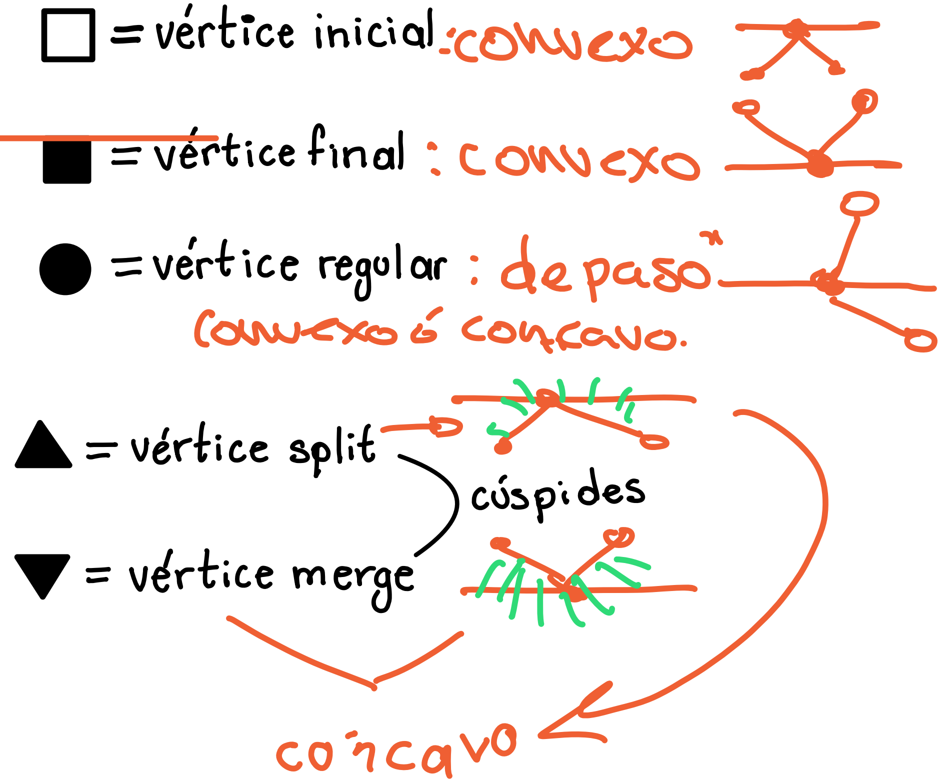
Sweep line algorithm

TRIANGULATING POLYGONS

Monotone partition

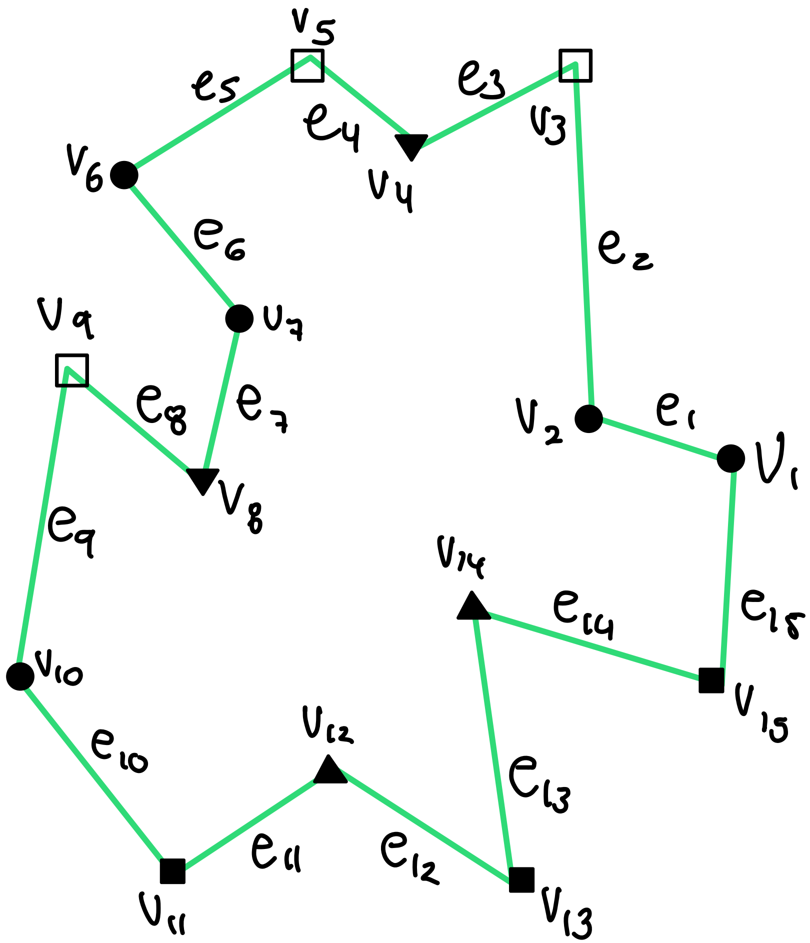
Sweep line algorithm





Monotone partition

Sweep line algorithm

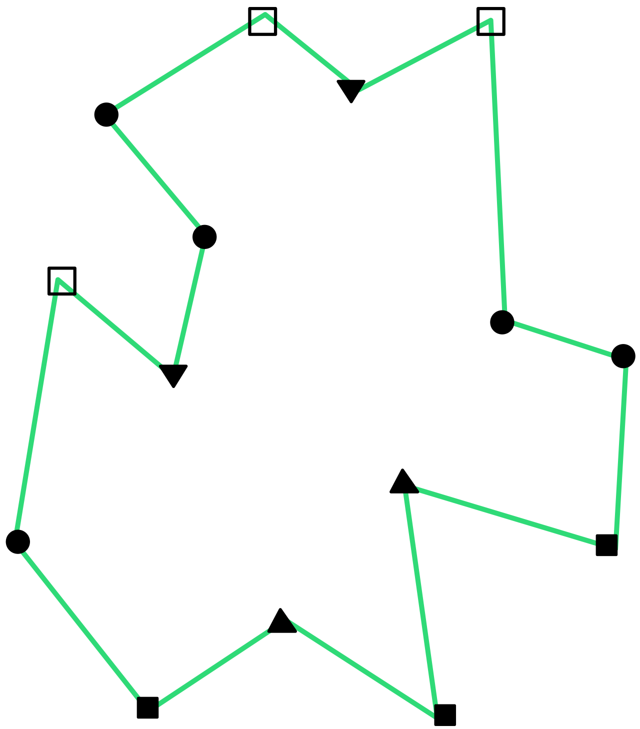


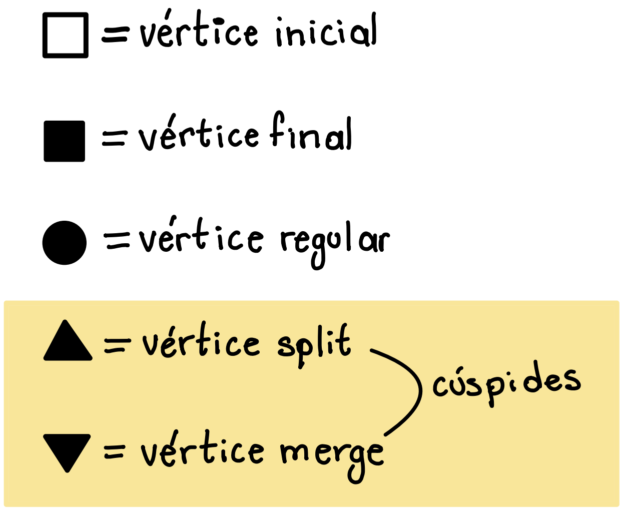
- Eventos: vértices de P. (no se crean eventos durante el barrido.)
- Pila de eventos Q.
- Q es una cola de prioridad, la prioridad de un evento es su coord. y.
- Siguiente evento $O(\lg n)$ si no se ordenan y $O(1)$ si se ordenan.

Monotone partition

Sweep line algorithm

¿Cómo agregamos las diagonales?

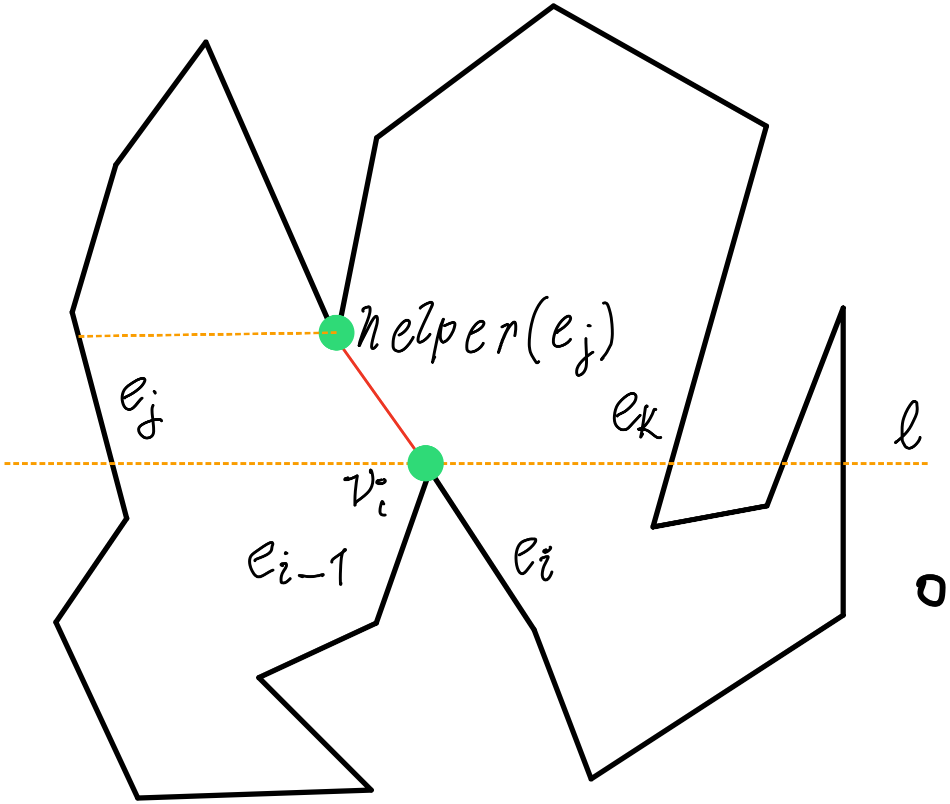




Monotone partition

vértice split

- Agregar diagonales desde cada vértice split hacia un vértice arriba de este.

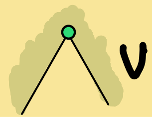


- helper(e_j) = vértice más bajo, arriba de la recta de barrido, tq la recta horizontal que lo conecta con ej queda contenida en el interior de P.

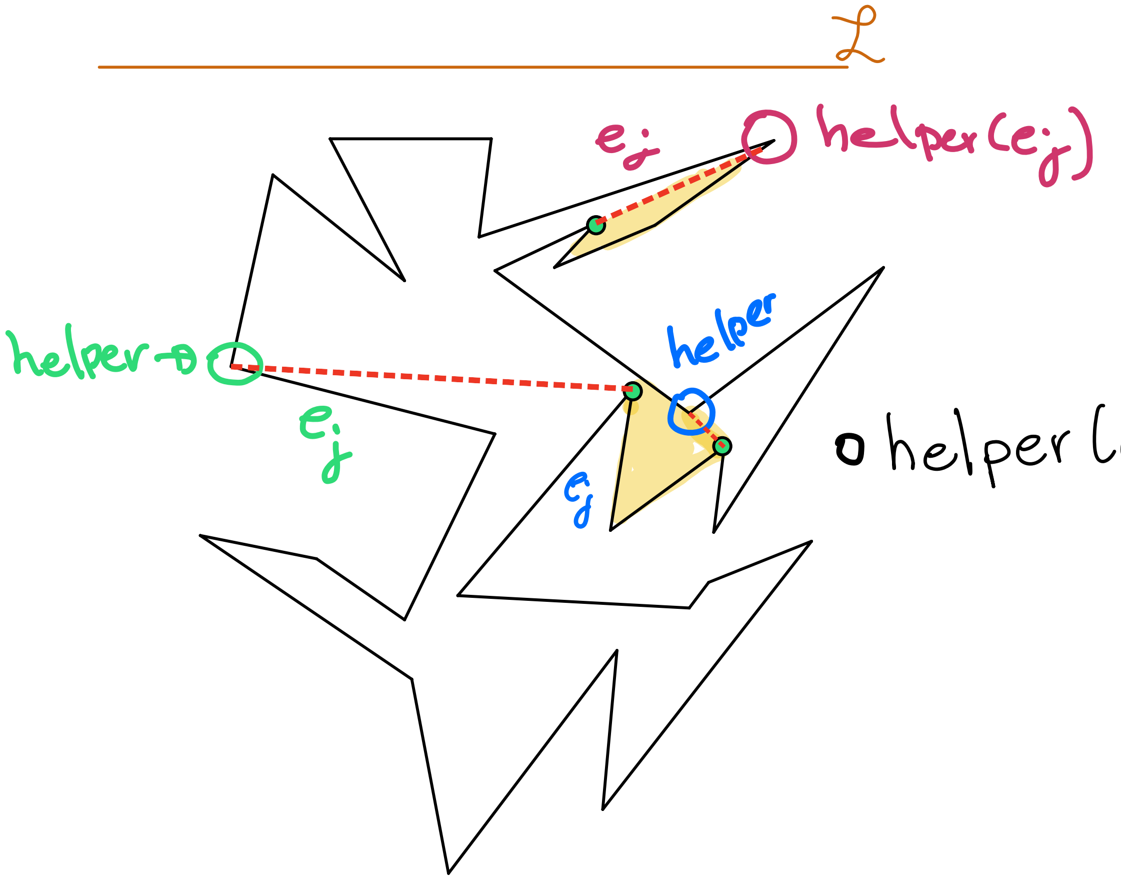
(helper(e_j) podría ser el extremo superior de e_j)

Monotone partition

Sweep line algorithm

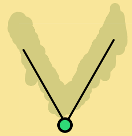


vértice split

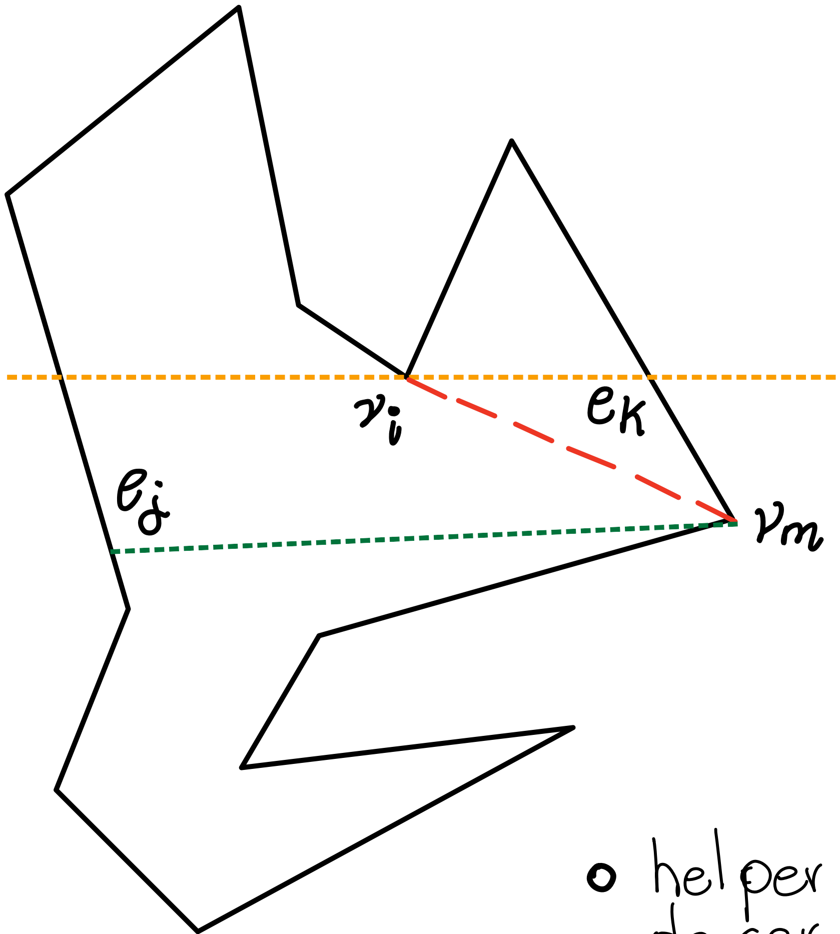


- helper(e_j) = vértice más bajo, arriba de la recta de barrido, tq la recta horizontal que lo conecta con ej queda contenida en el interior de P.

Monotone partition



vértice merge



- Agregar diagonales desde cada vértice merge hacia un vértice abajo de este.

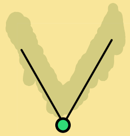
- Deseamos conectar v_i con el vértice más alto, entre e_j y e_k, que está por debajo de l, tq la recta horizontal que lo conecta con e_j queda contenida en el interior de P.

  – No lo hemos explorado.

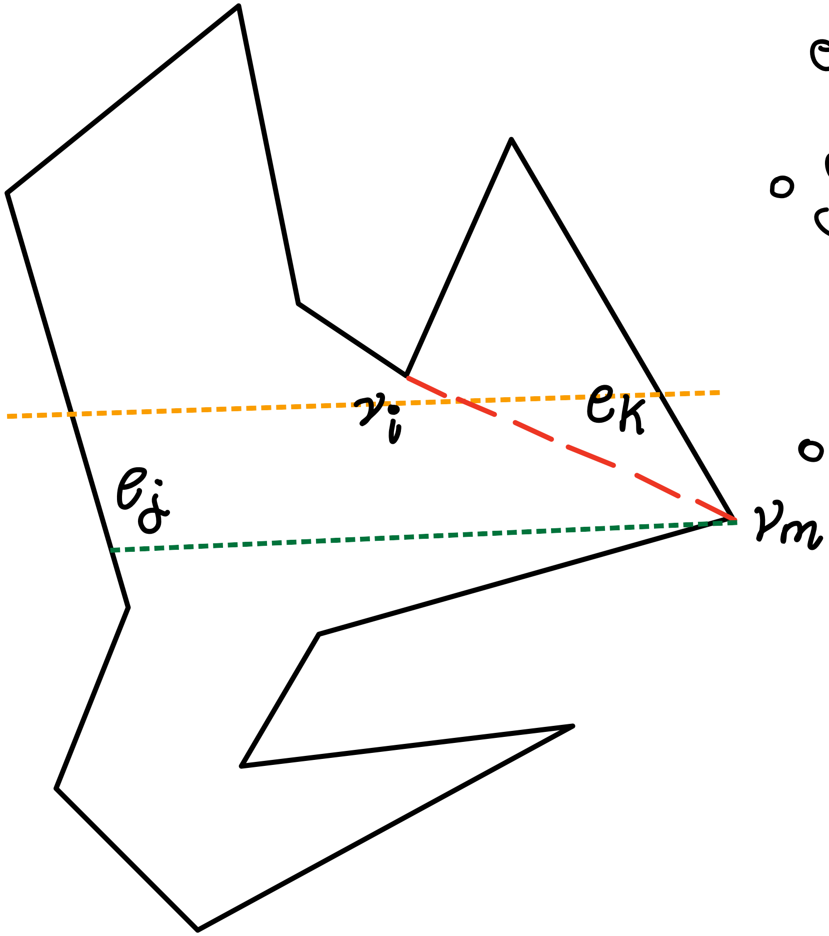
- helper $(e_j) = v_i$, entonces, cuando v_i deje de ser el helper de e_j y sea reemplazado por v_m, conectamos a v_i con v_m.

TRIANGULATING POLYGONS

Monotone partition



vértice merge

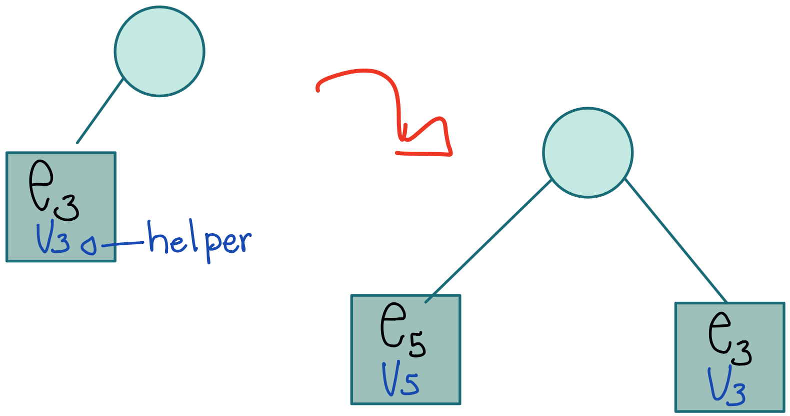


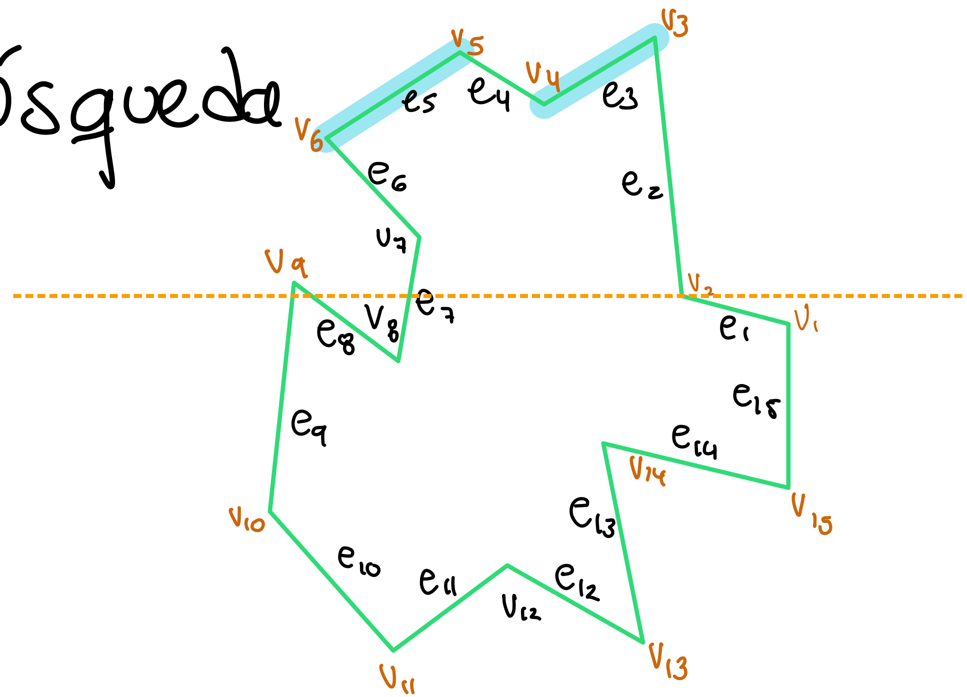
- Supongamos que hasta antes de llegar a v_m el ayudante de e_j es v_i.
- Y que cuando lleguemos a v_m este reemplaza a v_i como ayudante de e_j
- Conectamos a v_i con v_m.

Monotone partition

- Además de Q necesitamos una estructura para almacenar el status del algoritmo: La intersección de la recta con P.

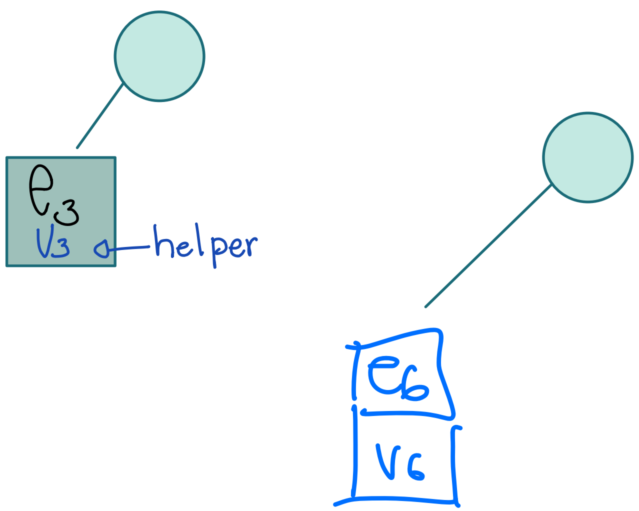
- Árbol binario de búsqueda (dinámico).

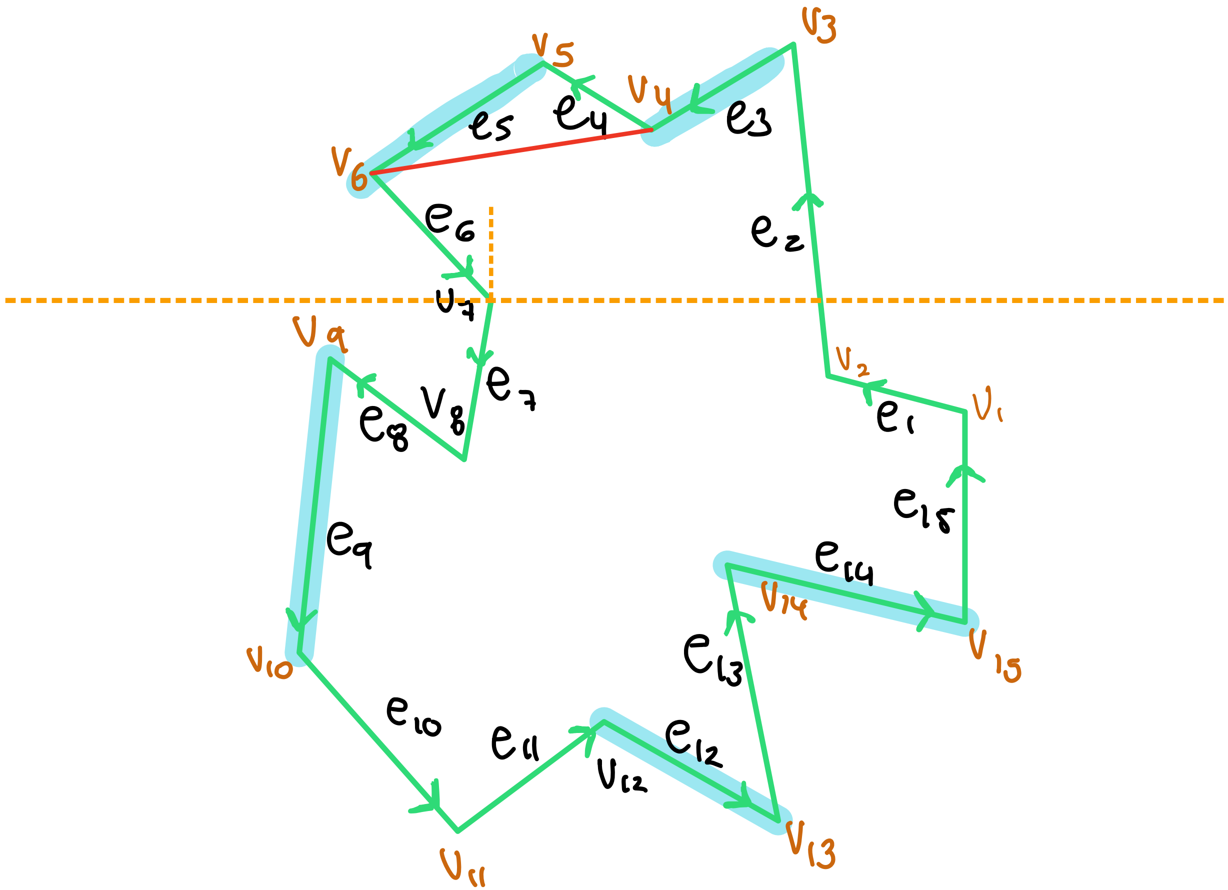




(No almacenaremos todas las aristas, únicamente aquellas que dejen el polígono a su izquierda)

Monotone partition



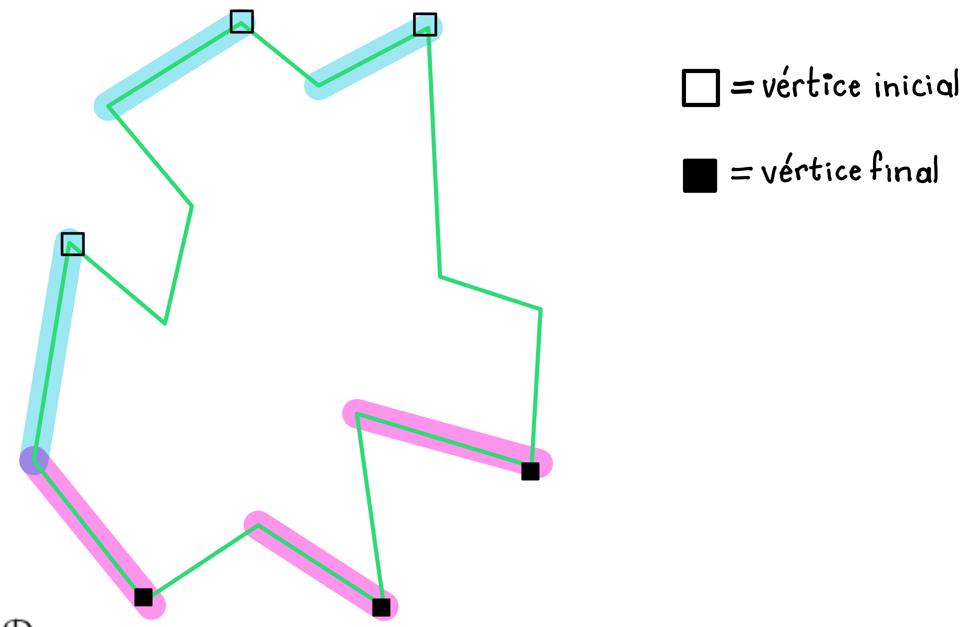


Monotone partition

Algorithm MAKEMONOTONE($\mathcal{P}$)
Input. A simple polygon $\mathcal{P}$ stored in a doubly-connected edge list $\mathcal{D}$.
Output. A partitioning of $\mathcal{P}$ into monotone subpolygons, stored in $\mathcal{D}$.

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

Monotone partition



HANDLESTARTVERTEX(v_i)

1. Insert e_i in $\mathcal{T}$ and set $helper(e_i)$ to v_i.

HANDLEENDVERTEX(v_i)

1. **if** $helper(e_{i-1})$ is a merge vertex
2. **then** Insert the diagonal connecting v_i to $helper(e_{i-1})$ in $\mathcal{D}$.
3. Delete e_{i-1} from $\mathcal{T}$.

TRIANGULATING POLYGONS

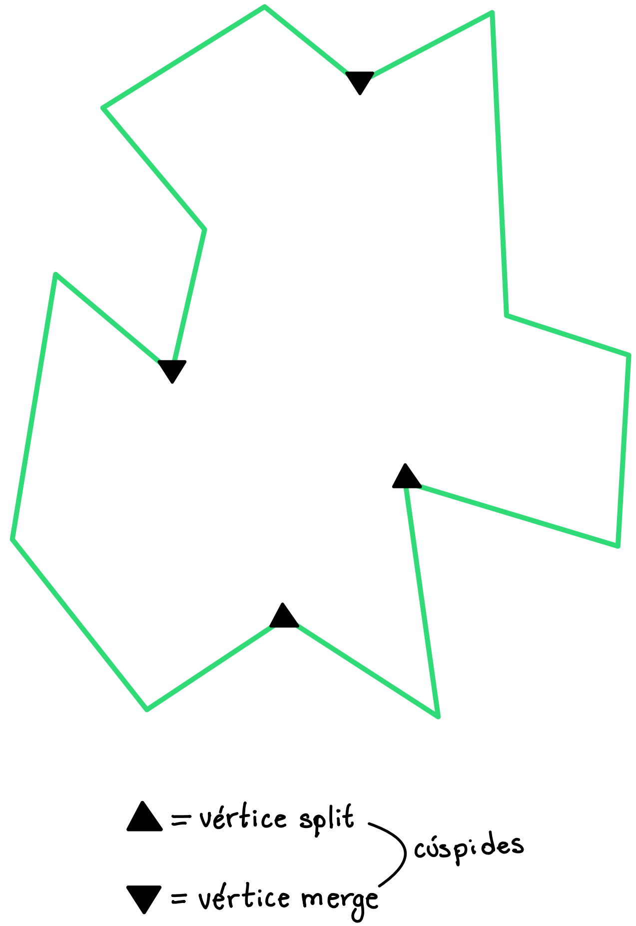
Monotone partition

HANDLESPLITVERTEX(v_i)

1. Search in $\mathcal{T}$ to find the edge e_j directly left of v_i.
2. Insert the diagonal connecting v_i to $helper(e_j)$ in $\mathcal{D}$.
3. $helper(e_j) \leftarrow v_i$
4. Insert e_i in $\mathcal{T}$ and set $helper(e_i)$ to v_i.

HANDLEMERGEVERTEX(v_i)

1. **if** $helper(e_{i-1})$ is a merge vertex
2. **then** Insert the diagonal connecting v_i to $helper(e_{i-1})$ in $\mathcal{D}$.
3. Delete e_{i-1} from $\mathcal{T}$.
4. Search in $\mathcal{T}$ to find the edge e_j directly left of v_i.
5. **if** $helper(e_j)$ is a merge vertex
6. **then** Insert the diagonal connecting v_i to $helper(e_j)$ in $\mathcal{D}$.
7. $helper(e_j) \leftarrow v_i$

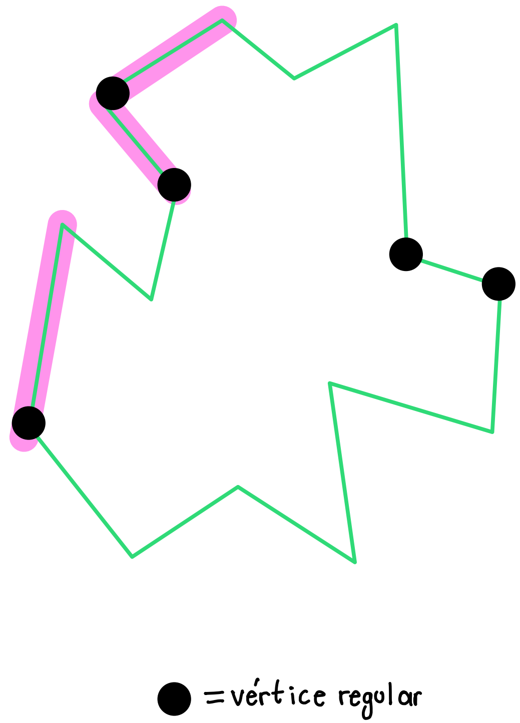


TRIANGULATING POLYGONS

Monotone partition

HANDLEREGULARVERTEX(v_i)

if the interior of $\mathcal{P}$ lies to the right of v_i
  then if $helper(e_{i-1})$ is a merge vertex
      then Insert the diagonal connecting v_i to $helper(e_{i-1})$ in $\mathcal{D}$.
    Delete e_{i-1} from $\mathcal{T}$.
    Insert e_i in $\mathcal{T}$ and set $helper(e_i)$ to v_i.
  else Search in $\mathcal{T}$ to find the edge e_j directly left of v_i.
    if $helper(e_j)$ is a merge vertex
      then Insert the diagonal connecting v_i to $helper(e_j)$ in $\mathcal{D}$.
    $helper(e_j) \leftarrow v_i$



HANDLEMERGEVERTEX(v_i)

if $helper(e_{i-1})$ is a merge vertex
  then Insert the diagonal connecting v_i to $helper(e_{i-1})$ in $\mathcal{D}$.
Delete e_{i-1} from $\mathcal{T}$.
Search in $\mathcal{T}$ to find the edge e_j directly left of v_i.
if $helper(e_j)$ is a merge vertex
  then Insert the diagonal connecting v_i to $helper(e_j)$ in $\mathcal{D}$.
$helper(e_j) \leftarrow v_i$

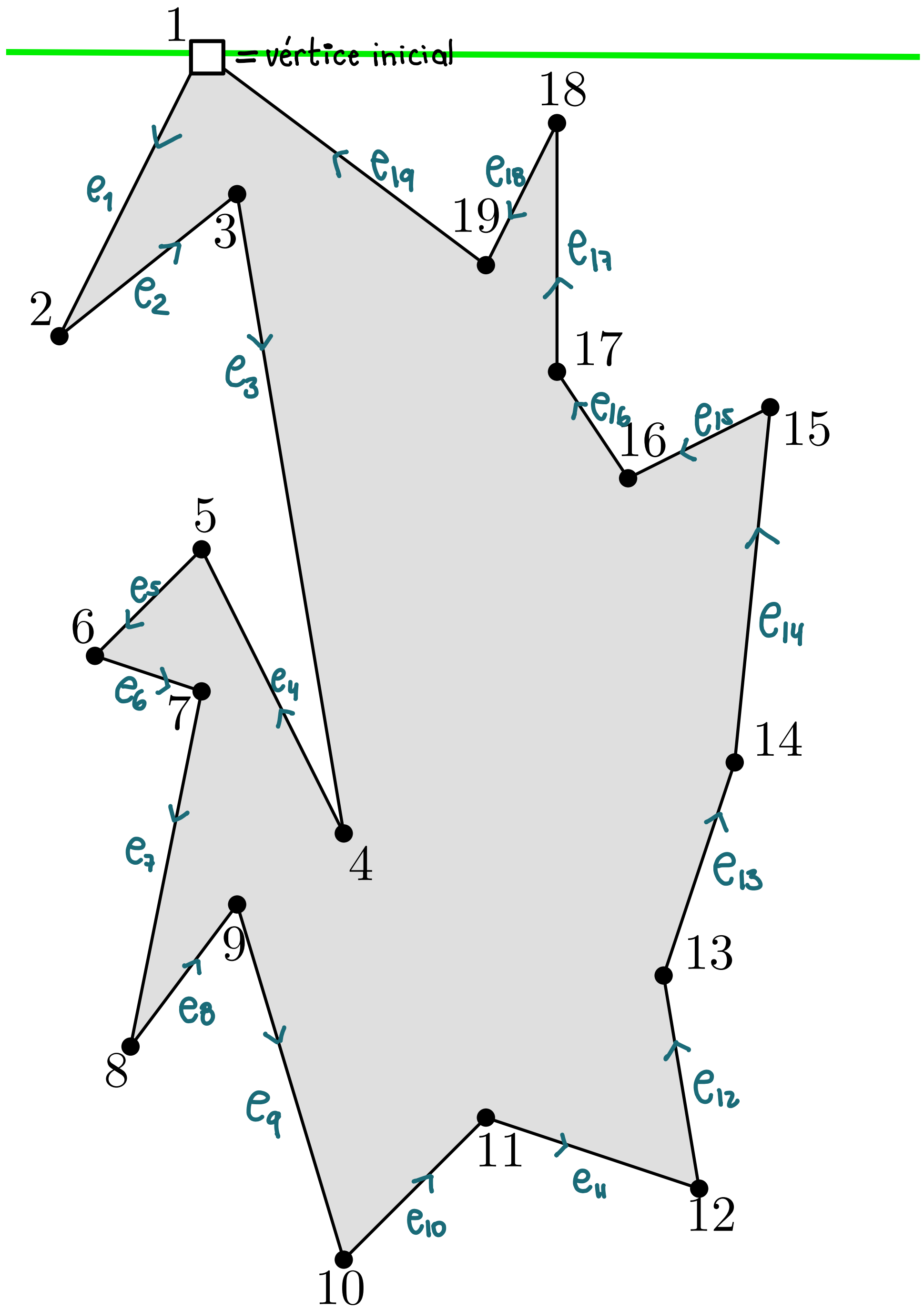
Monotone partition

D :

T :

HANDLESTARTVERTEX(v_i)

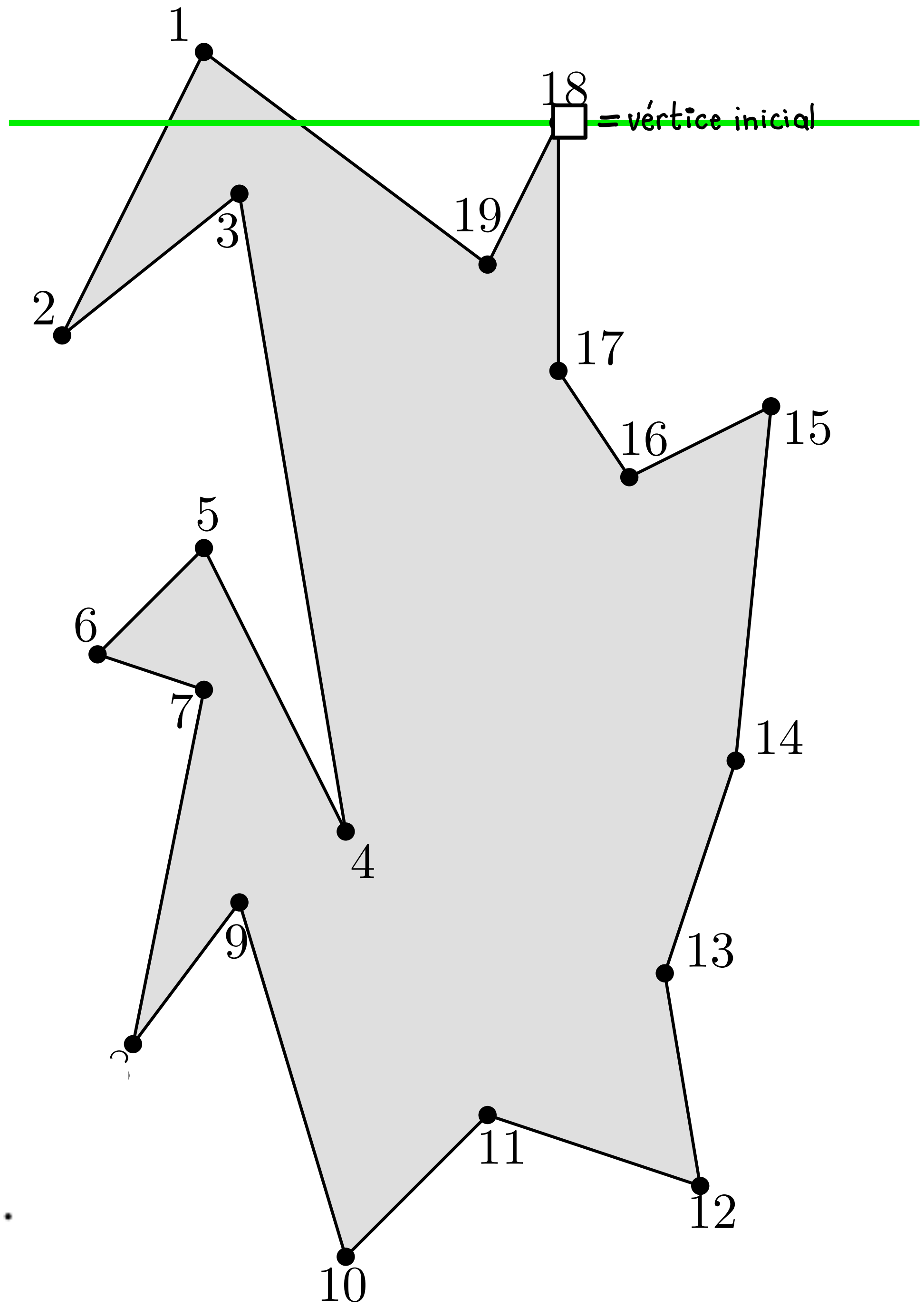
1. Insert e_i in $\mathcal{T}$ and set $helper(e_i)$ to v_i.



Monotone partition

D :

$\mathcal{T}$:



HANDLESTARTVERTEX(v_i)

1. Insert e_i in $\mathcal{T}$ and set *helper*(e_i) to v_i.

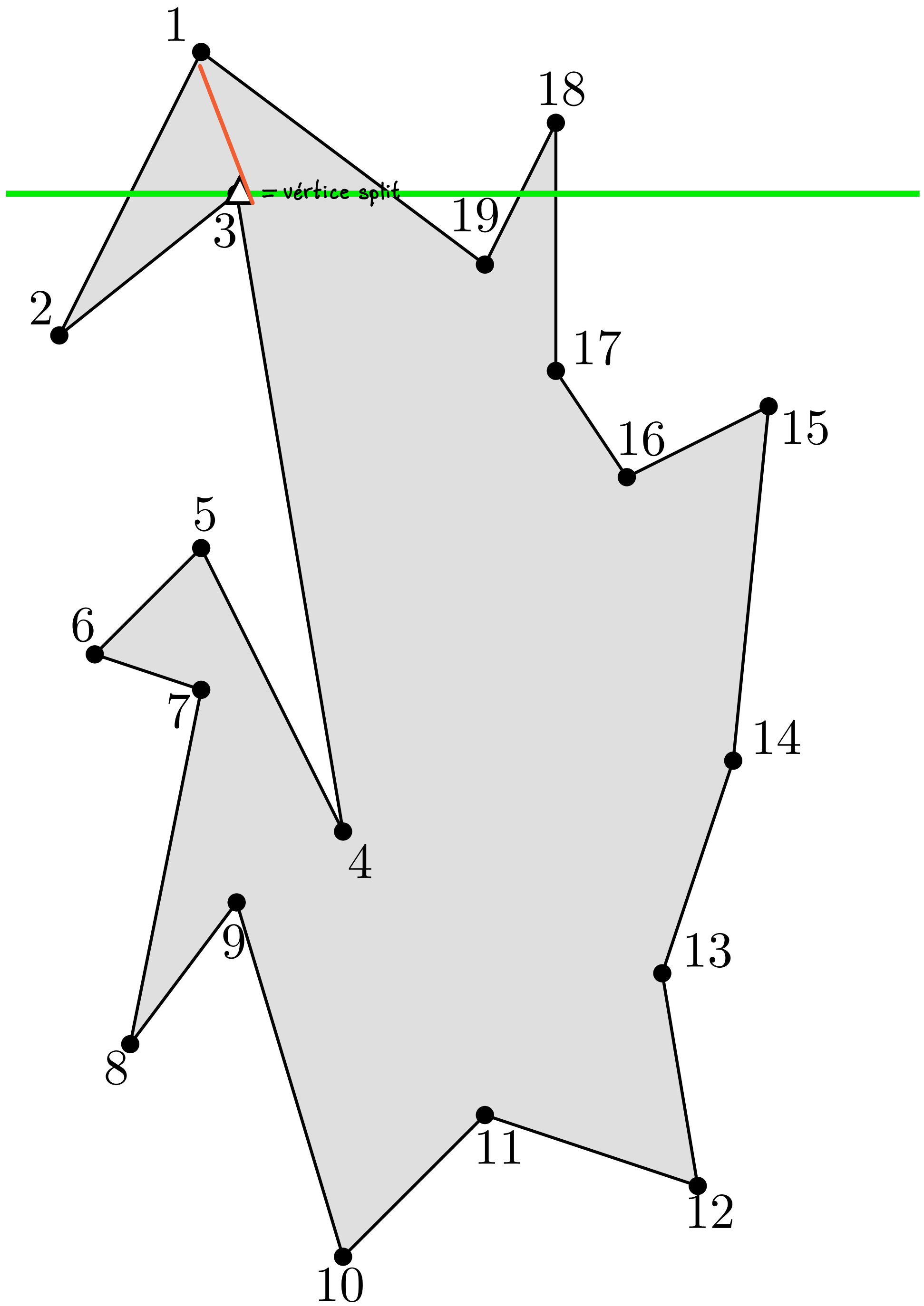
Monotone partition

𝒟 :

𝒯 :

HANDLESPLITVERTEX(v_i)

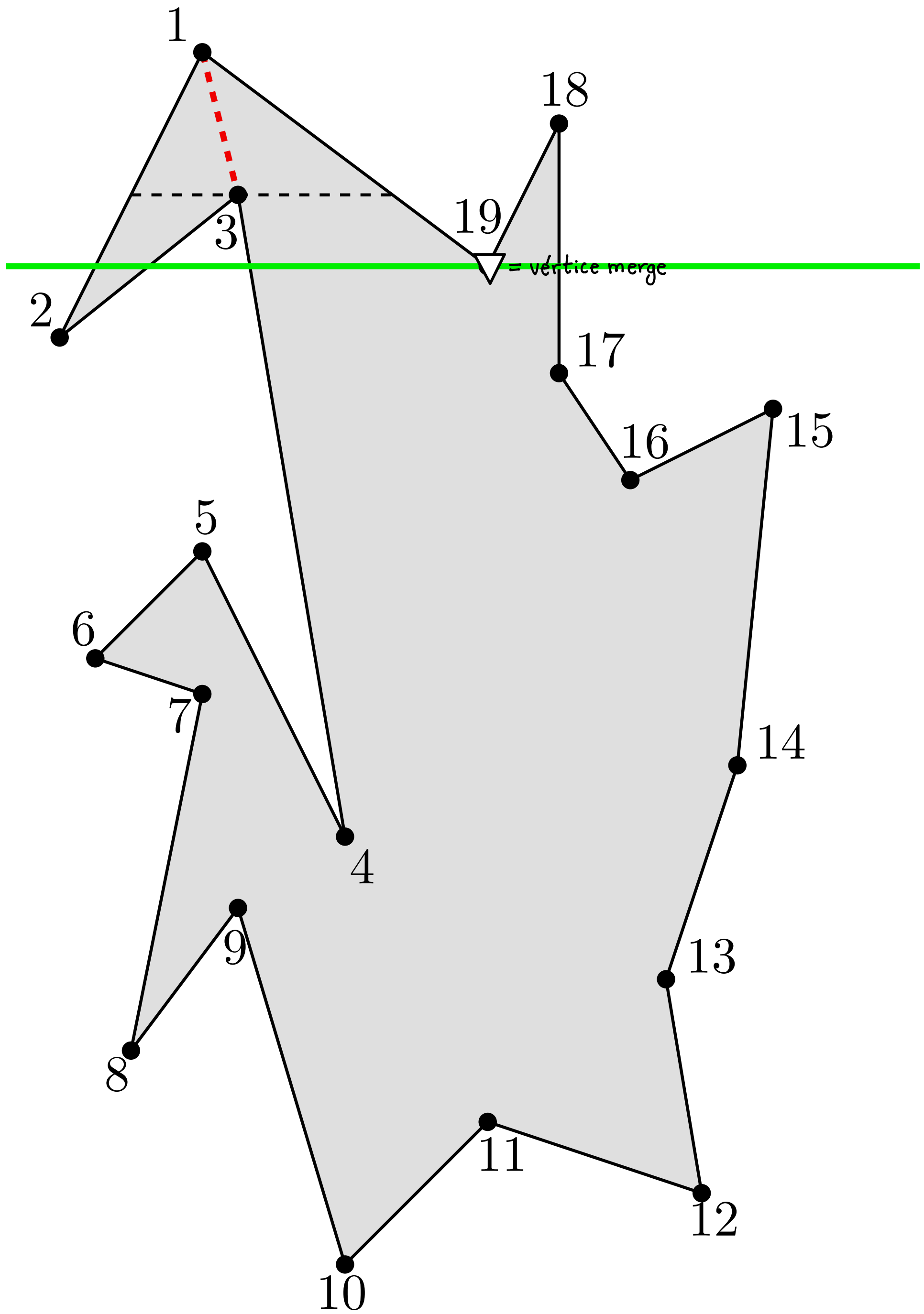
1. Search in $\mathcal{T}$ to find the edge e_j directly left of v_i.
2. Insert the diagonal connecting v_i to $helper(e_j)$ in $\mathcal{D}$.
3. $helper(e_j) \leftarrow v_i$
4. Insert e_i in $\mathcal{T}$ and set $helper(e_i)$ to v_i.



Monotone partition

$\mathcal{D}$:

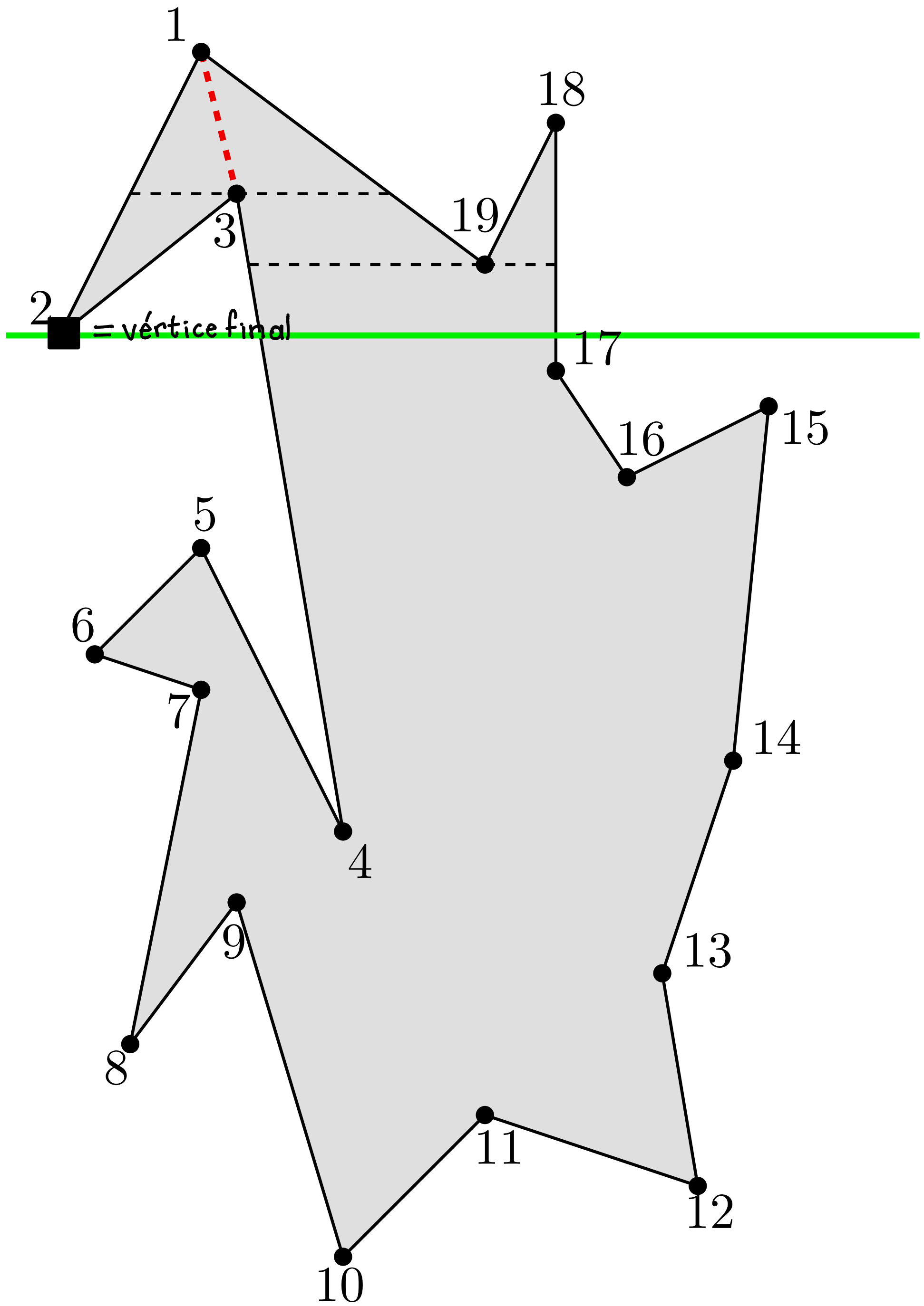
$\mathcal{T}$:



HANDLEMERGEVERTEX(v_i)

if *helper*(e_{i-1}) is a merge vertex
  then Insert the diagonal connecting v_i to *helper*(e_{i-1}) in $\mathcal{D}$.
Delete e_{i-1} from $\mathcal{T}$.
Search in $\mathcal{T}$ to find the edge e_j directly left of v_i.
if *helper*(e_j) is a merge vertex
  then Insert the diagonal connecting v_i to *helper*(e_j) in $\mathcal{D}$.
helper(e_j) $\leftarrow v_i$

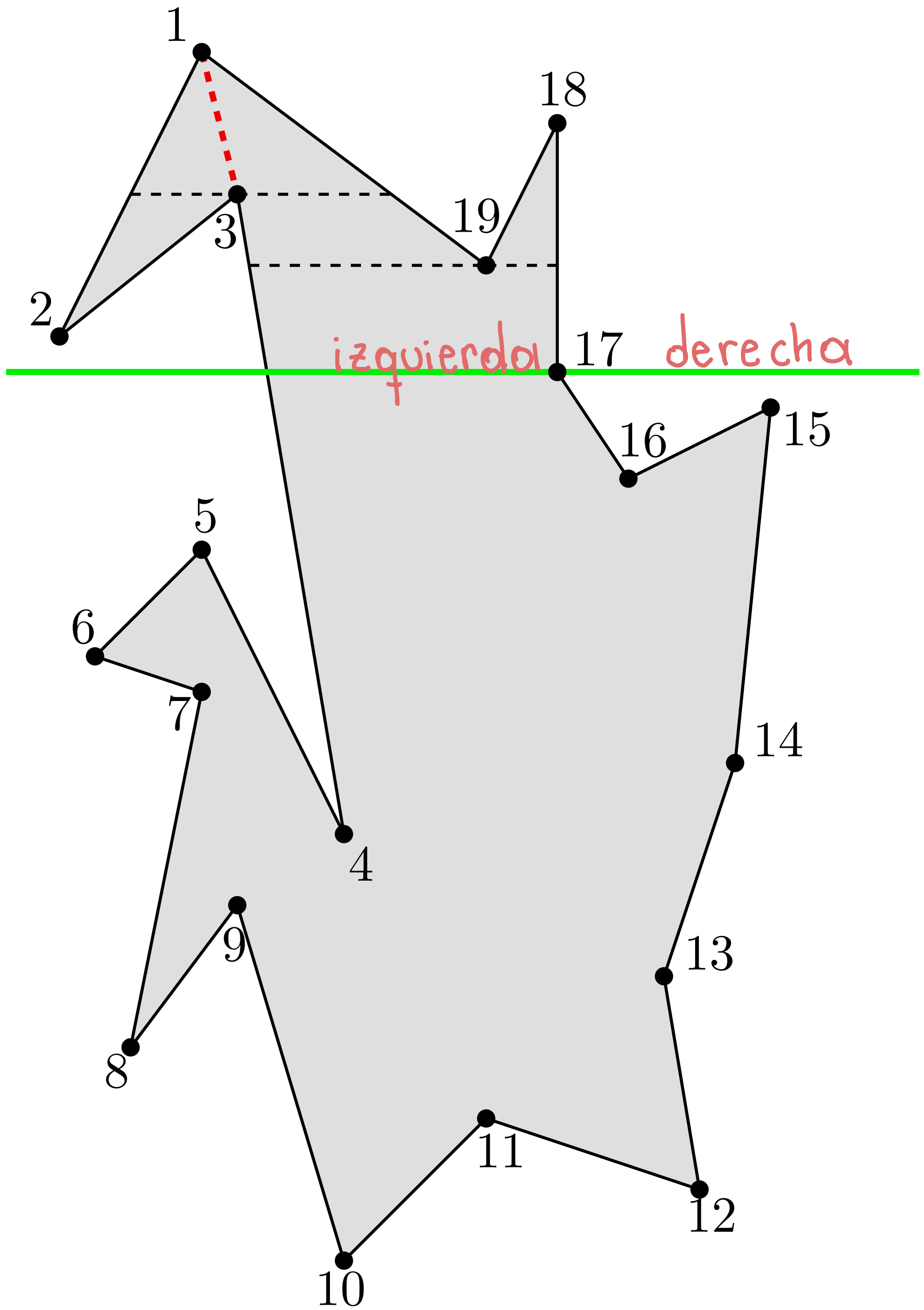
Monotone partition



HANDLEENDVERTEX(v_i)

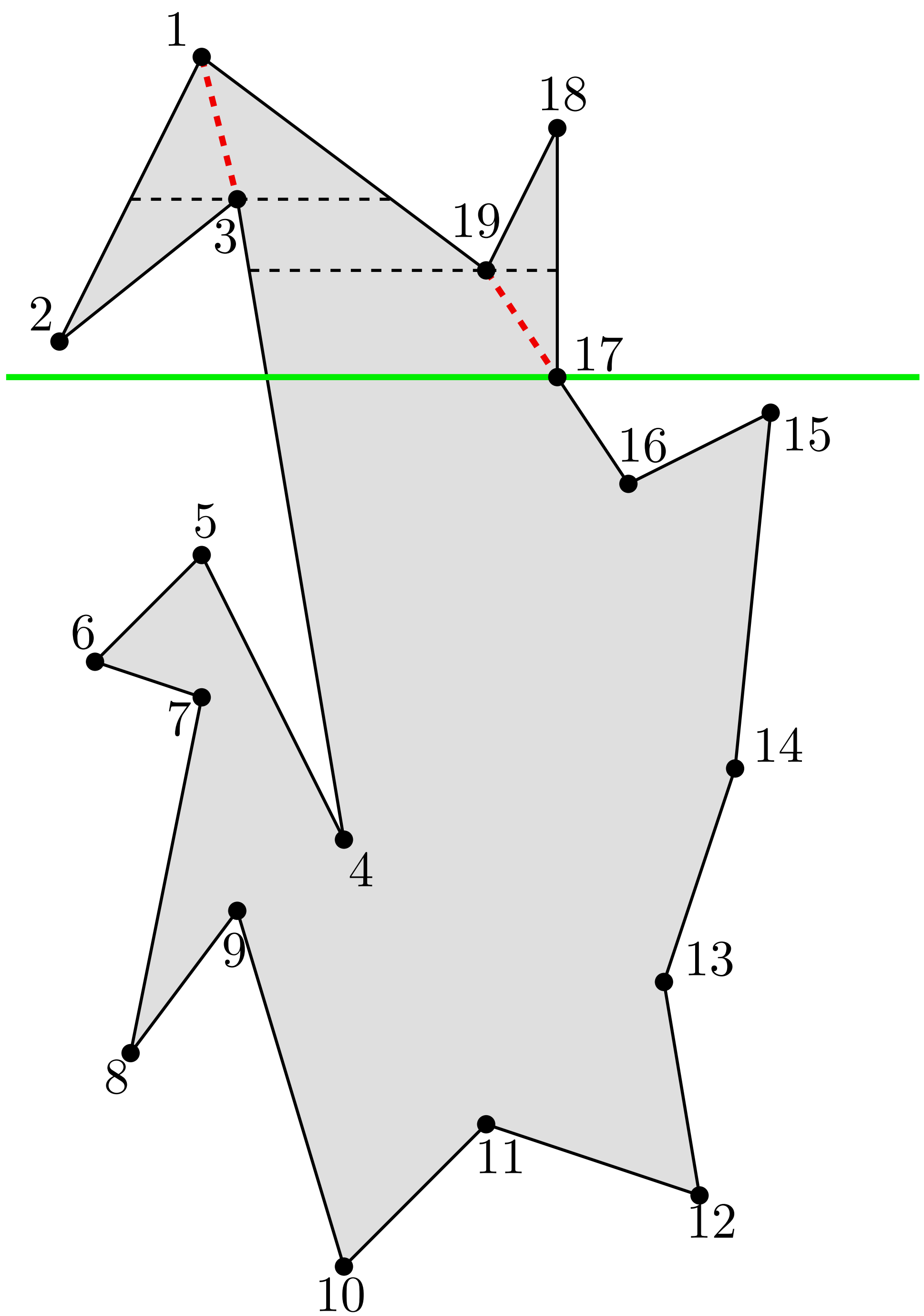
1. **if** $helper(e_{i-1})$ is a merge vertex
2. **then** Insert the diagonal connecting v_i to $helper(e_{i-1})$ in $\mathcal{D}$.
3. Delete e_{i-1} from $\mathcal{T}$.

Monotone partition



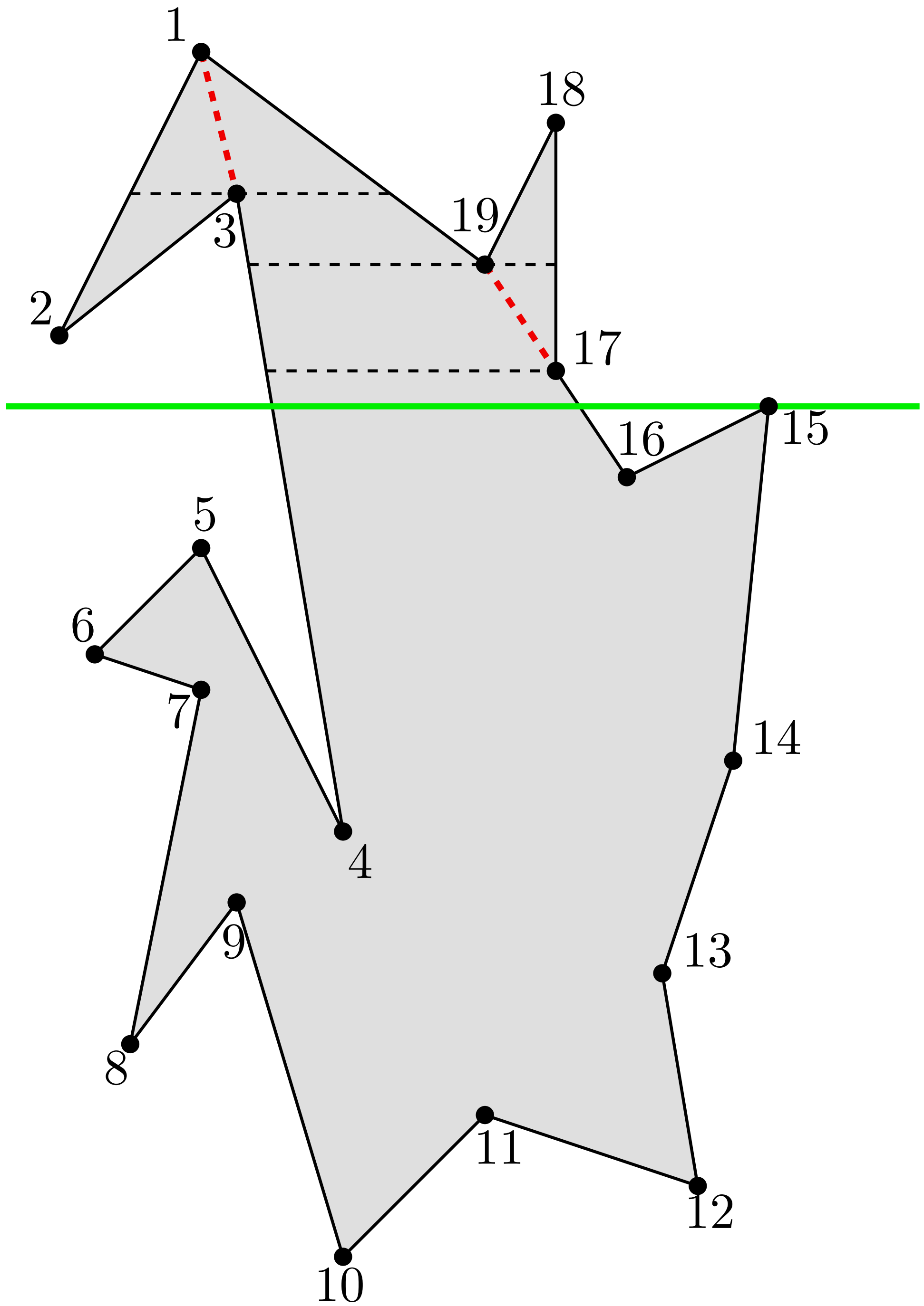

```
HANDLEREGULARVERTEX(v_i)
if the interior of P lies to the right of v_i
  then if helper(e_{i−1}) is a merge vertex
         then Insert the diagonal connecting v_i to helper(e_{i−1}) in D.
       Delete e_{i−1} from T.
       Insert e_i in T and set helper(e_i) to v_i.
  else Search in T to find the edge e_j directly left of v_i.
       if helper(e_j) is a merge vertex
         then Insert the diagonal connecting v_i to helper(e_j) in D.
       helper(e_j) ← v_i
```

Monotone partition



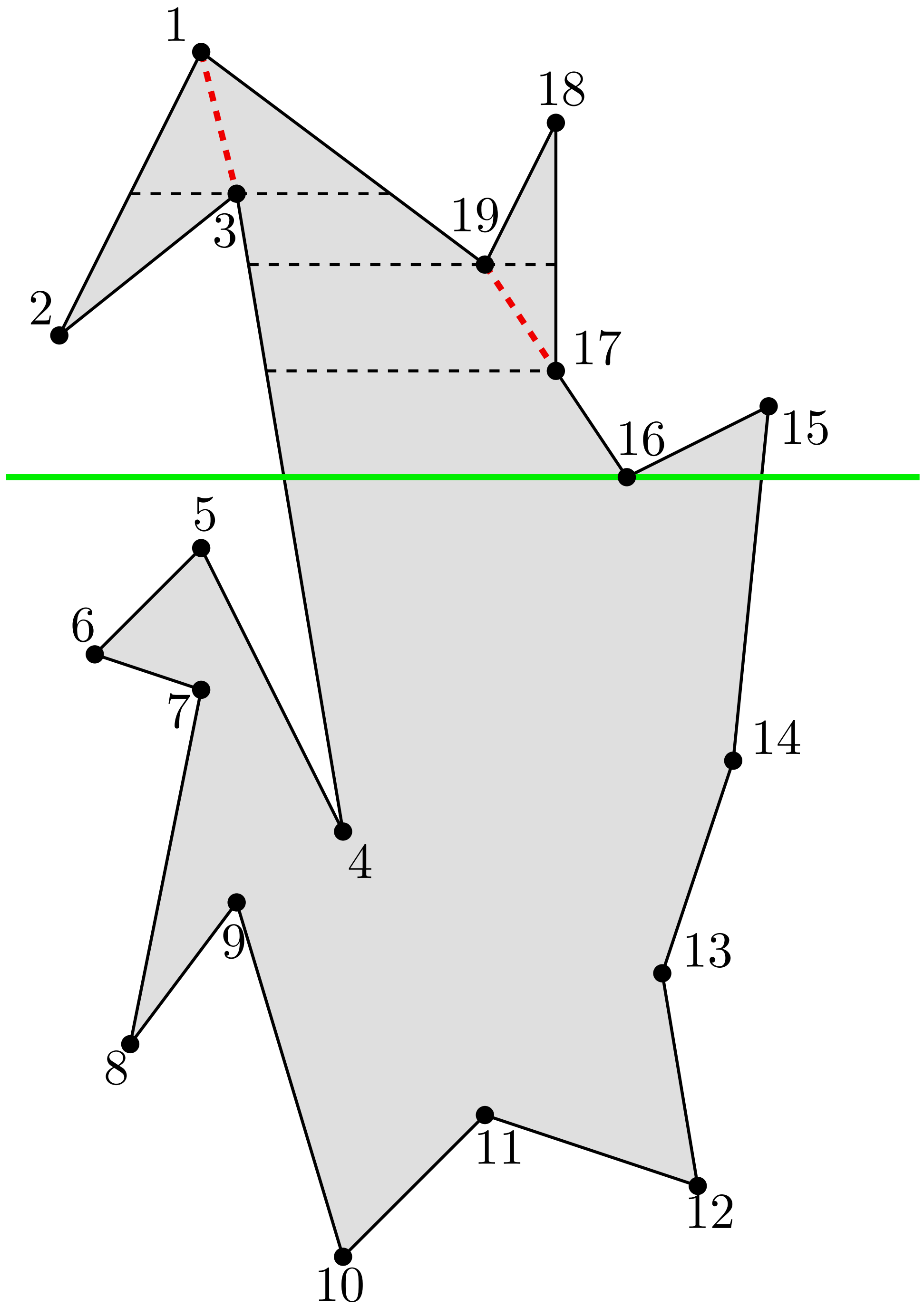

```
HANDLEREGULARVERTEX(v_i)
if the interior of P lies to the right of v_i
   then if helper(e_{i-1}) is a merge vertex
           then Insert the diagonal connecting v_i to helper(e_{i-1}) in D.
        Delete e_{i-1} from T.
        Insert e_i in T and set helper(e_i) to v_i.
   else Search in T to find the edge e_j directly left of v_i.
        if helper(e_j) is a merge vertex
           then Insert the diagonal connecting v_i to helper(e_j) in D.
        helper(e_j) ← v_i
```

Monotone partition



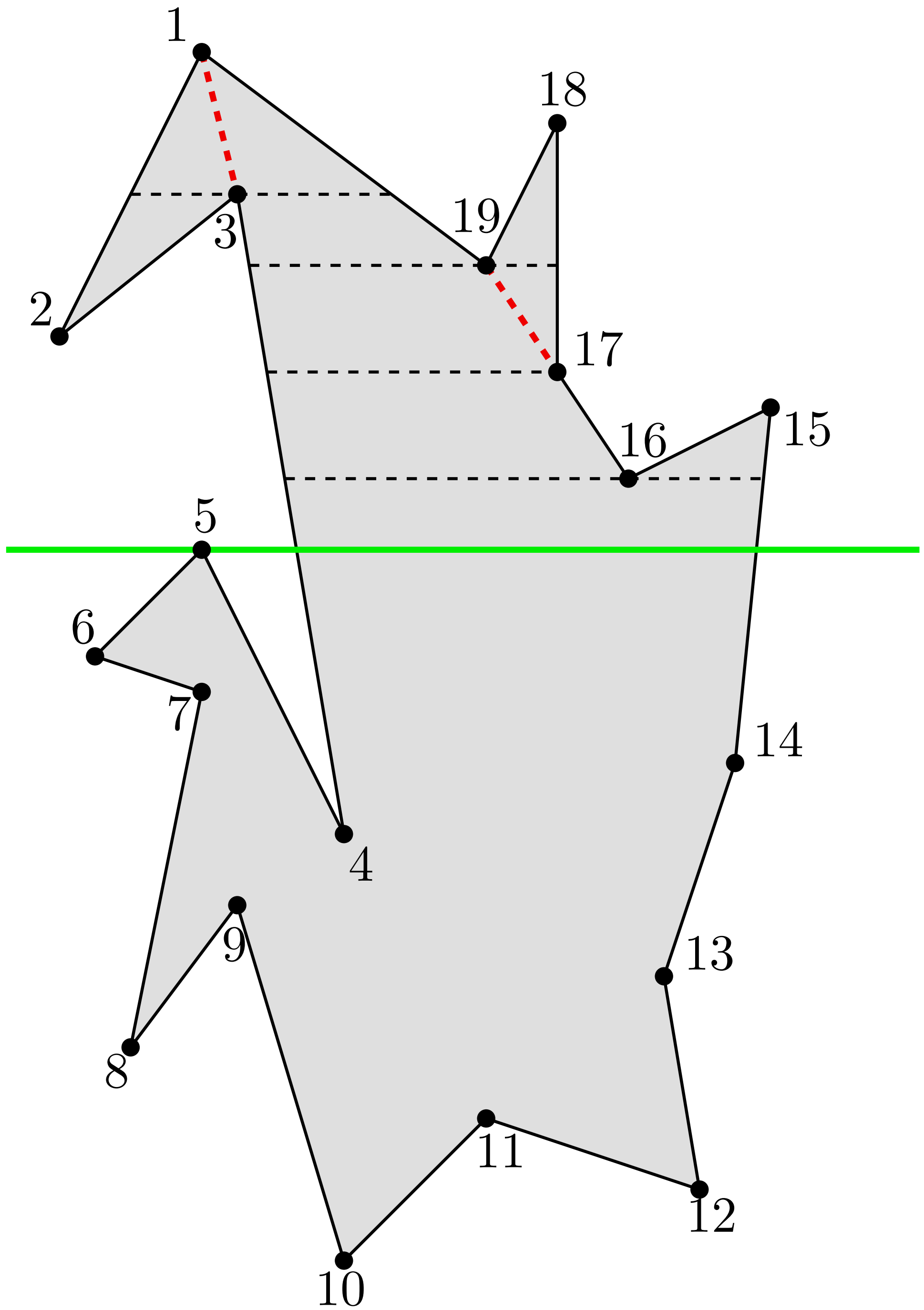
HANDLESTARTVERTEX(v_i)
1. Insert e_i in $\mathcal{T}$ and set $helper(e_i)$ to v_i.

Monotone partition



HANDLEMERGEVERTEX(v_i)
1. **if** *helper*(e_{i-1}) is a merge vertex
2. **then** Insert the diagonal connecting v_i to *helper*(e_{i-1}) in $\mathcal{D}$.
3. Delete e_{i-1} from $\mathcal{T}$.
4. Search in $\mathcal{T}$ to find the edge e_j directly left of v_i.
5. **if** *helper*(e_j) is a merge vertex
6. **then** Insert the diagonal connecting v_i to *helper*(e_j) in $\mathcal{D}$.
7. *helper*(e_j) $\leftarrow v_i$

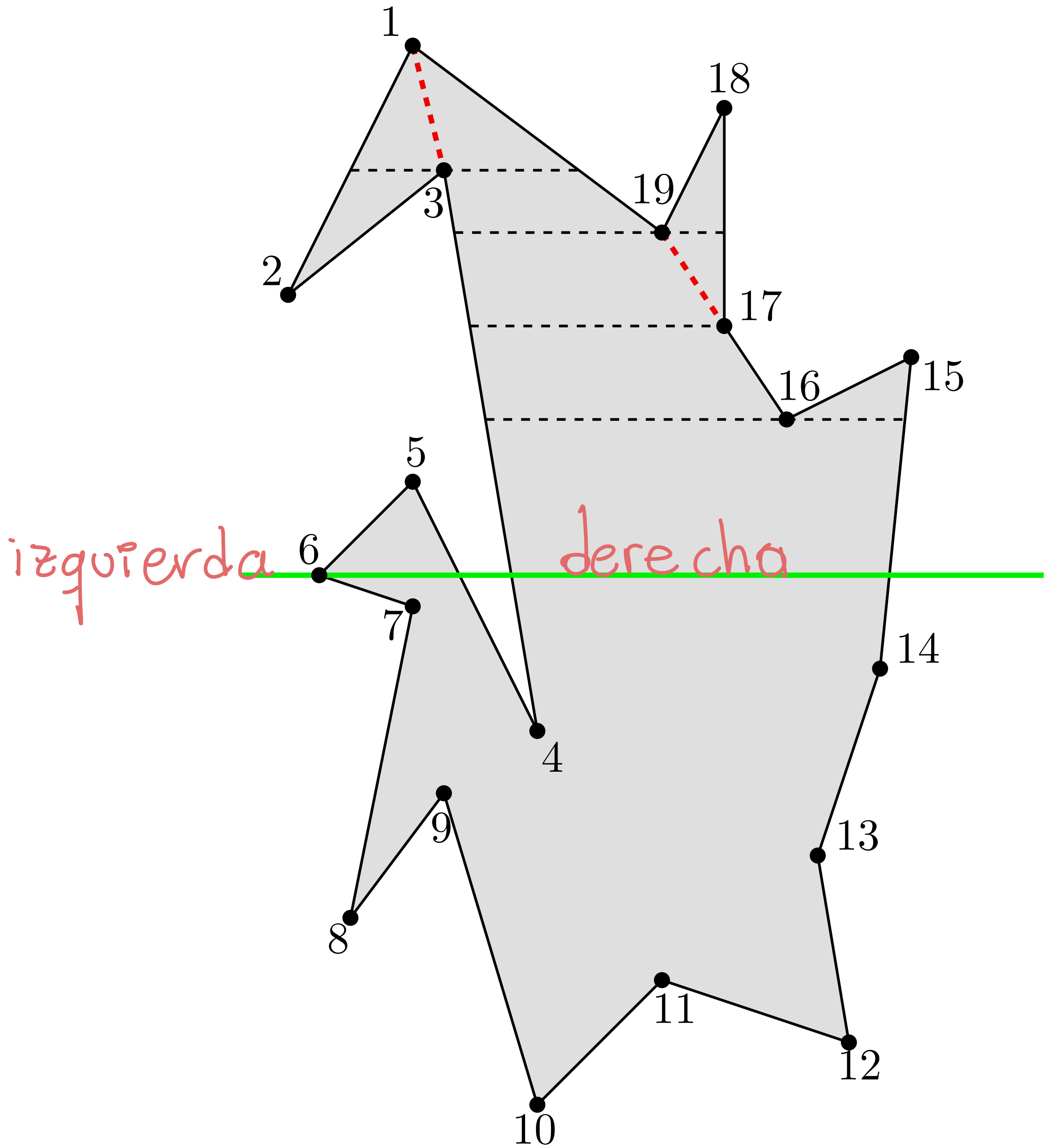
Monotone partition



HANDLESTARTVERTEX(v_i)

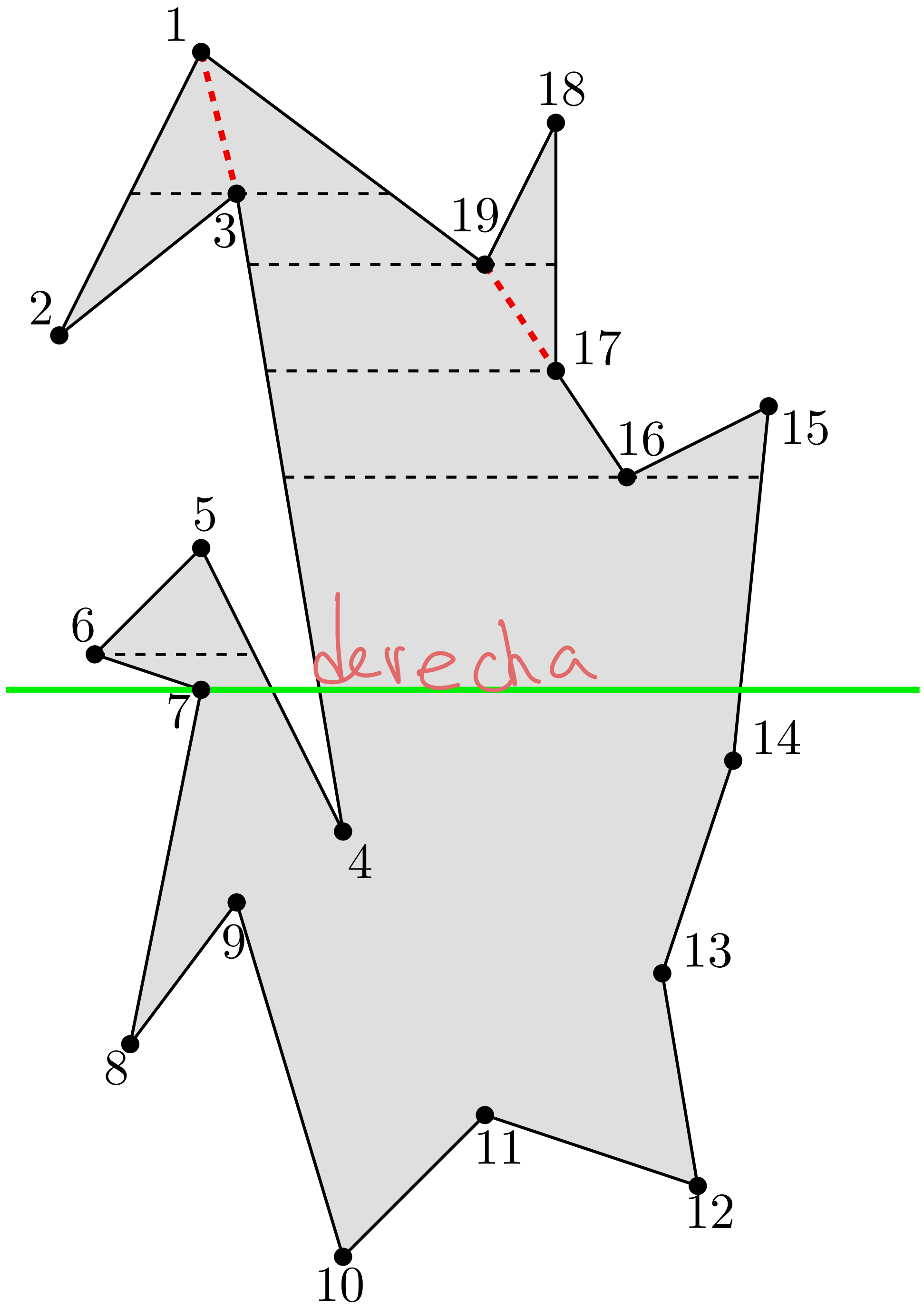
1. Insert e_i in $\mathcal{T}$ and set $helper(e_i)$ to v_i.

Monotone partition




```
HANDLEREGULARVERTEX(v_i)
if the interior of P lies to the right of v_i
  then if helper(e_{i-1}) is a merge vertex
         then Insert the diagonal connecting v_i to helper(e_{i-1}) in D.
       Delete e_{i-1} from T.
       Insert e_i in T and set helper(e_i) to v_i.
  else Search in T to find the edge e_j directly left of v_i.
       if helper(e_j) is a merge vertex
         then Insert the diagonal connecting v_i to helper(e_j) in D.
       helper(e_j) ← v_i
```

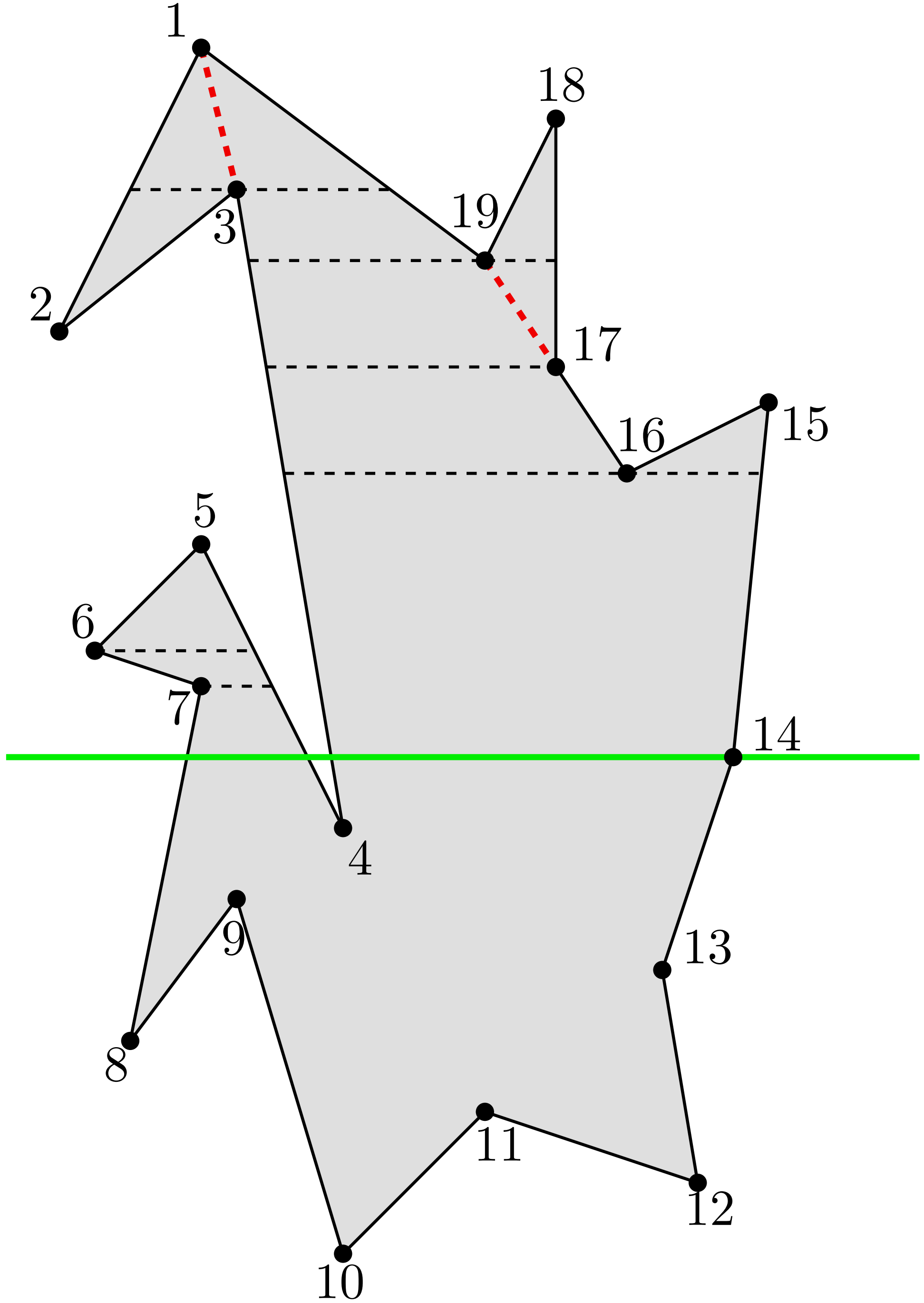
Monotone partition



HANDLEREGULARVERTEX(v_i)

```
if the interior of 𝒫 lies to the right of v_i
    then if helper(e_{i-1}) is a merge vertex
            then Insert the diagonal connecting v_i to helper(e_{i-1}) in 𝒟.
         Delete e_{i-1} from 𝒯.
         Insert e_i in 𝒯 and set helper(e_i) to v_i.
    else Search in 𝒯 to find the edge e_j directly left of v_i.
         if helper(e_j) is a merge vertex
            then Insert the diagonal connecting v_i to helper(e_j) in 𝒟.
         helper(e_j) ← v_i
```

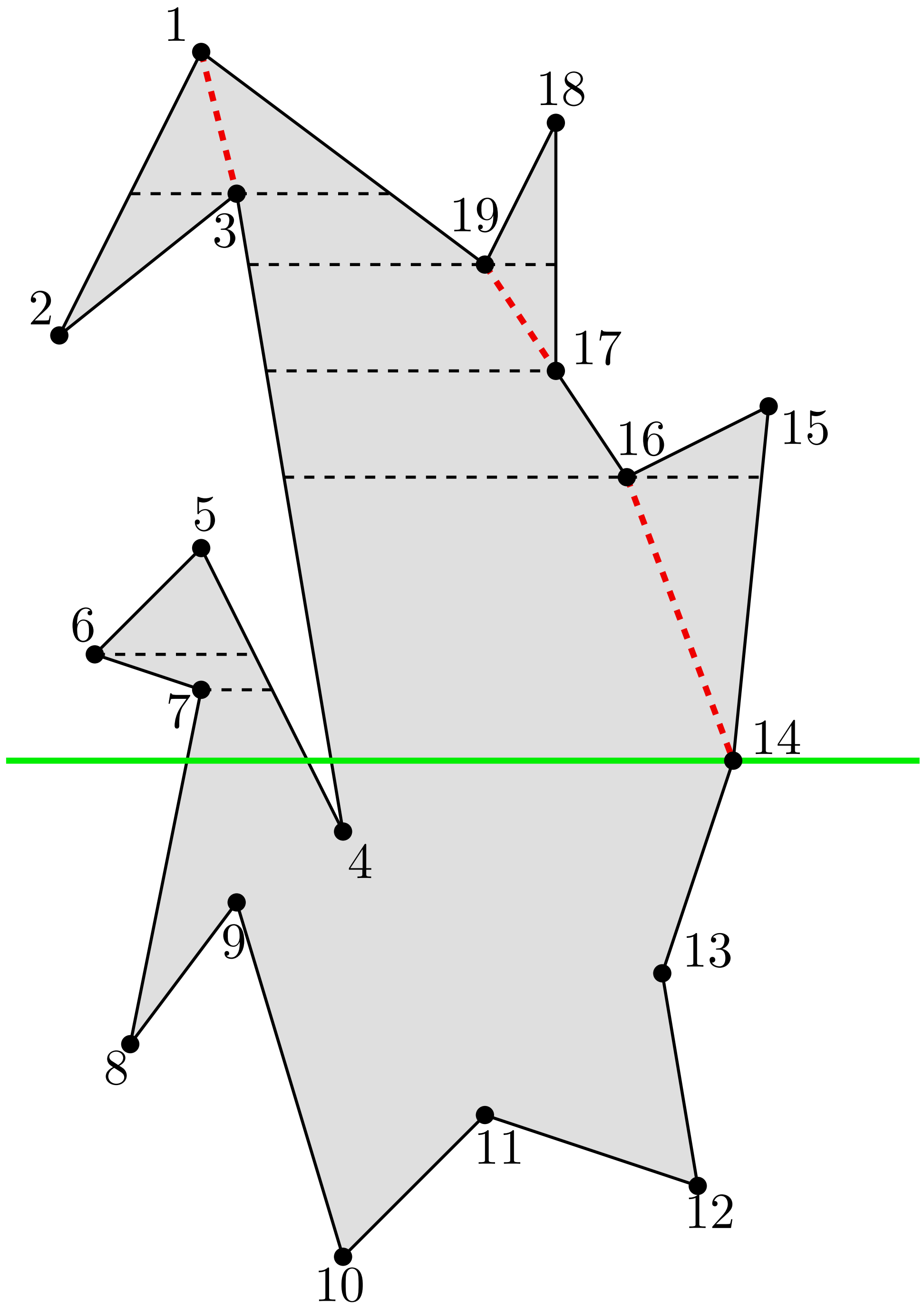
Monotone partition



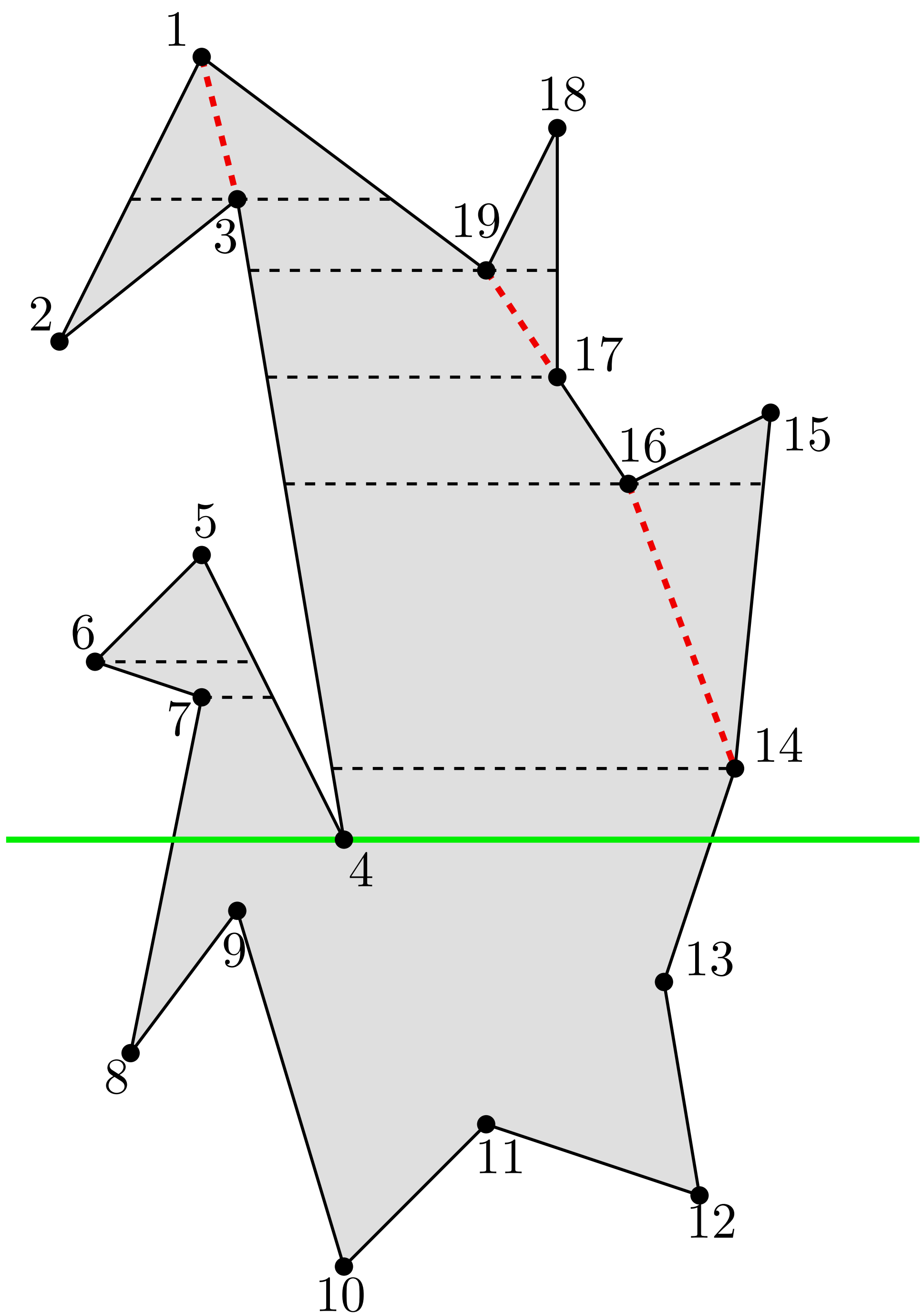
HandleRegularVertex(v_i)

if the interior of $\mathcal{P}$ lies to the right of v_i
  then if $helper(e_{i-1})$ is a merge vertex
    then Insert the diagonal connecting v_i to $helper(e_{i-1})$ in $\mathcal{D}$.
   Delete e_{i-1} from $\mathcal{T}$.
   Insert e_i in $\mathcal{T}$ and set $helper(e_i)$ to v_i.
  else Search in $\mathcal{T}$ to find the edge e_j directly left of v_i.
   if $helper(e_j)$ is a merge vertex
    then Insert the diagonal connecting v_i to $helper(e_j)$ in $\mathcal{D}$.
   $helper(e_j) \leftarrow v_i$

Monotone partition



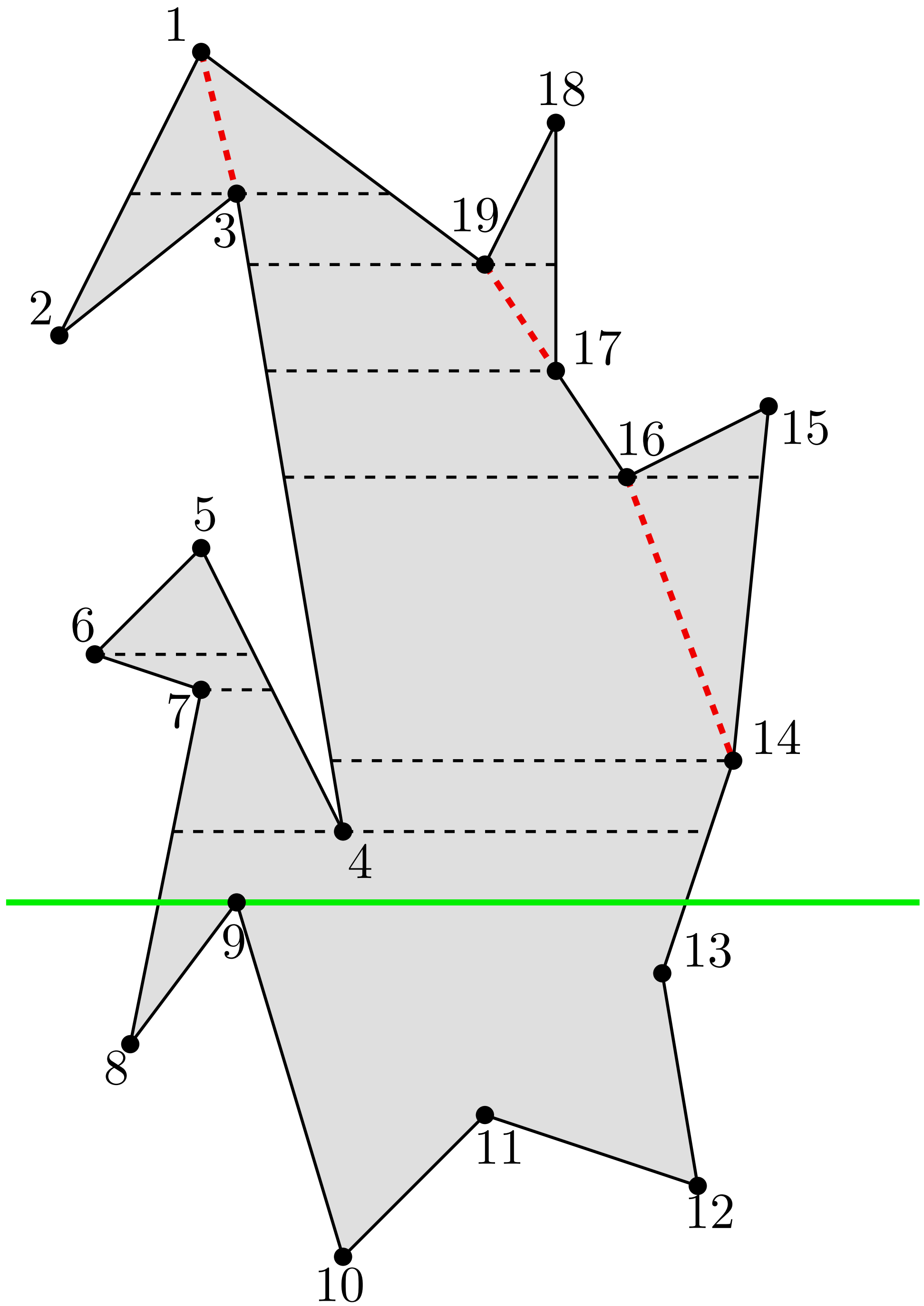
Monotone partition



HANDLEMERGEVERTEX(v_i)
1. **if** *helper*(e_{i-1}) is a merge vertex
2. **then** Insert the diagonal connecting v_i to *helper*(e_{i-1}) in $\mathcal{D}$.
3. Delete e_{i-1} from $\mathcal{T}$.
4. Search in $\mathcal{T}$ to find the edge e_j directly left of v_i.
5. **if** *helper*(e_j) is a merge vertex
6. **then** Insert the diagonal connecting v_i to *helper*(e_j) in $\mathcal{D}$.
7. *helper*(e_j) $\leftarrow v_i$

TRIANGULATING POLYGONS

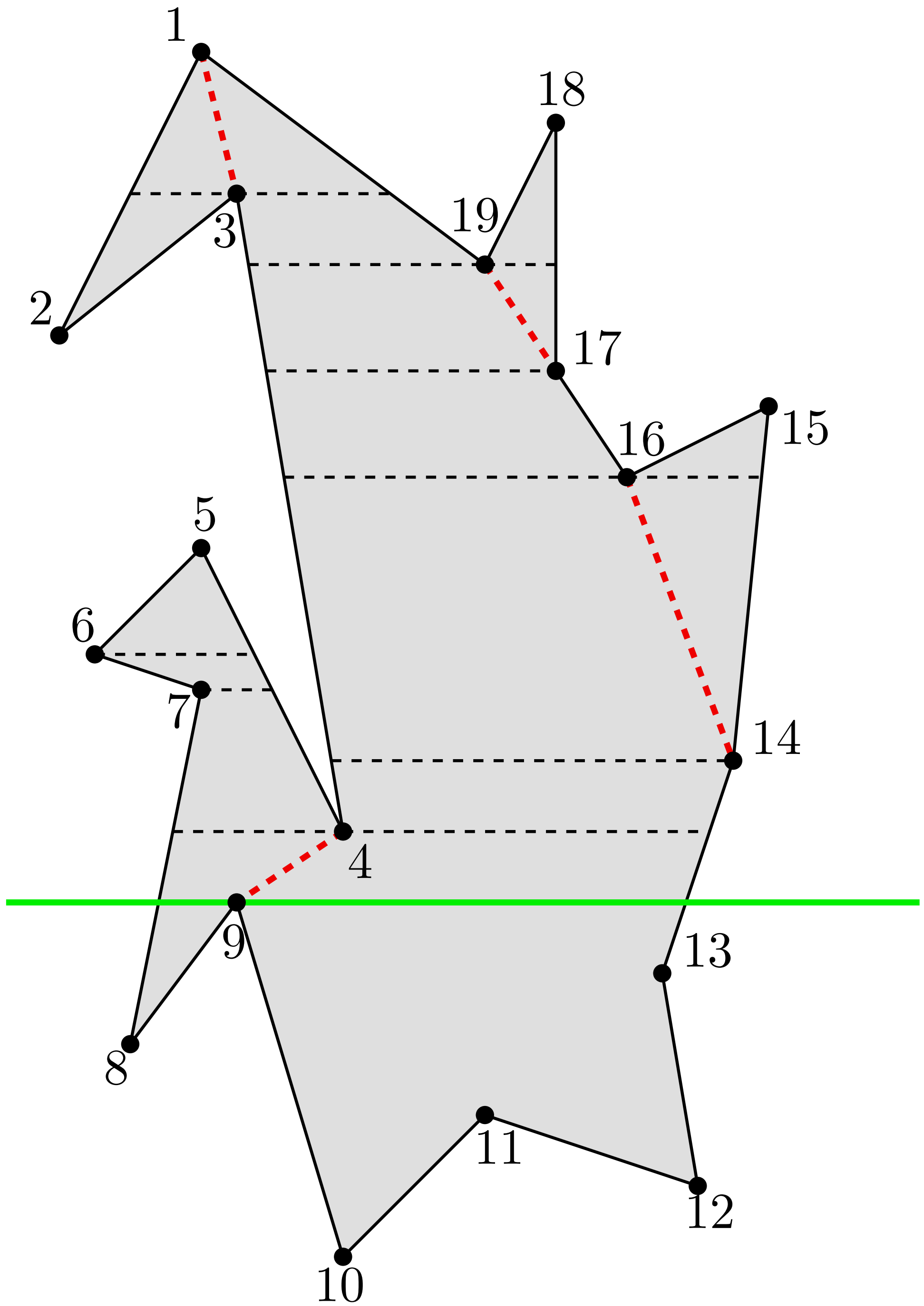
Monotone partition



HANDLESPLITVERTEX(v_i)

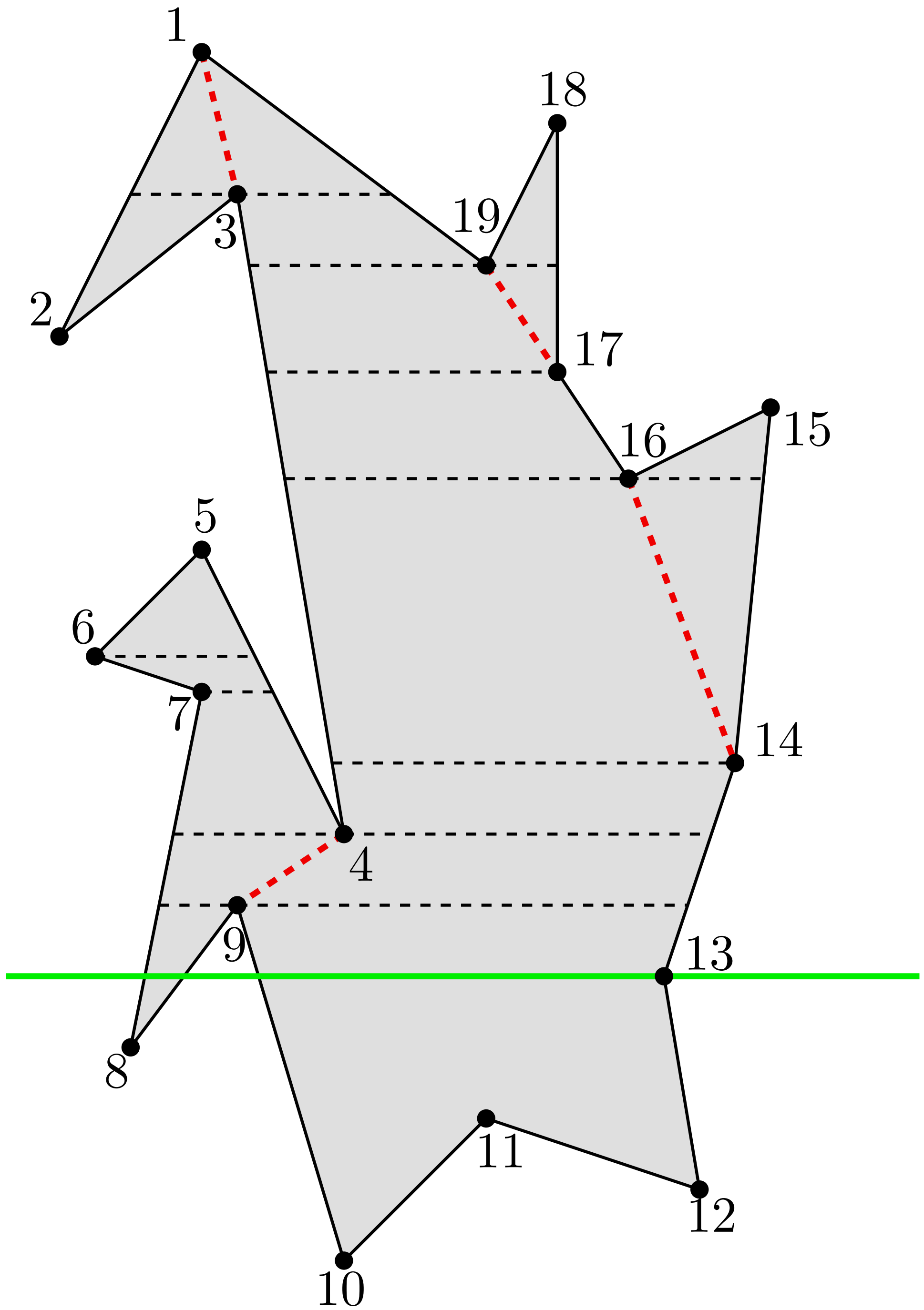
1. Search in $\mathcal{T}$ to find the edge e_j directly left of v_i.
2. Insert the diagonal connecting v_i to $helper(e_j)$ in $\mathcal{D}$.
3. $helper(e_j) \leftarrow v_i$
4. Insert e_i in $\mathcal{T}$ and set $helper(e_i)$ to v_i.

Monotone partition

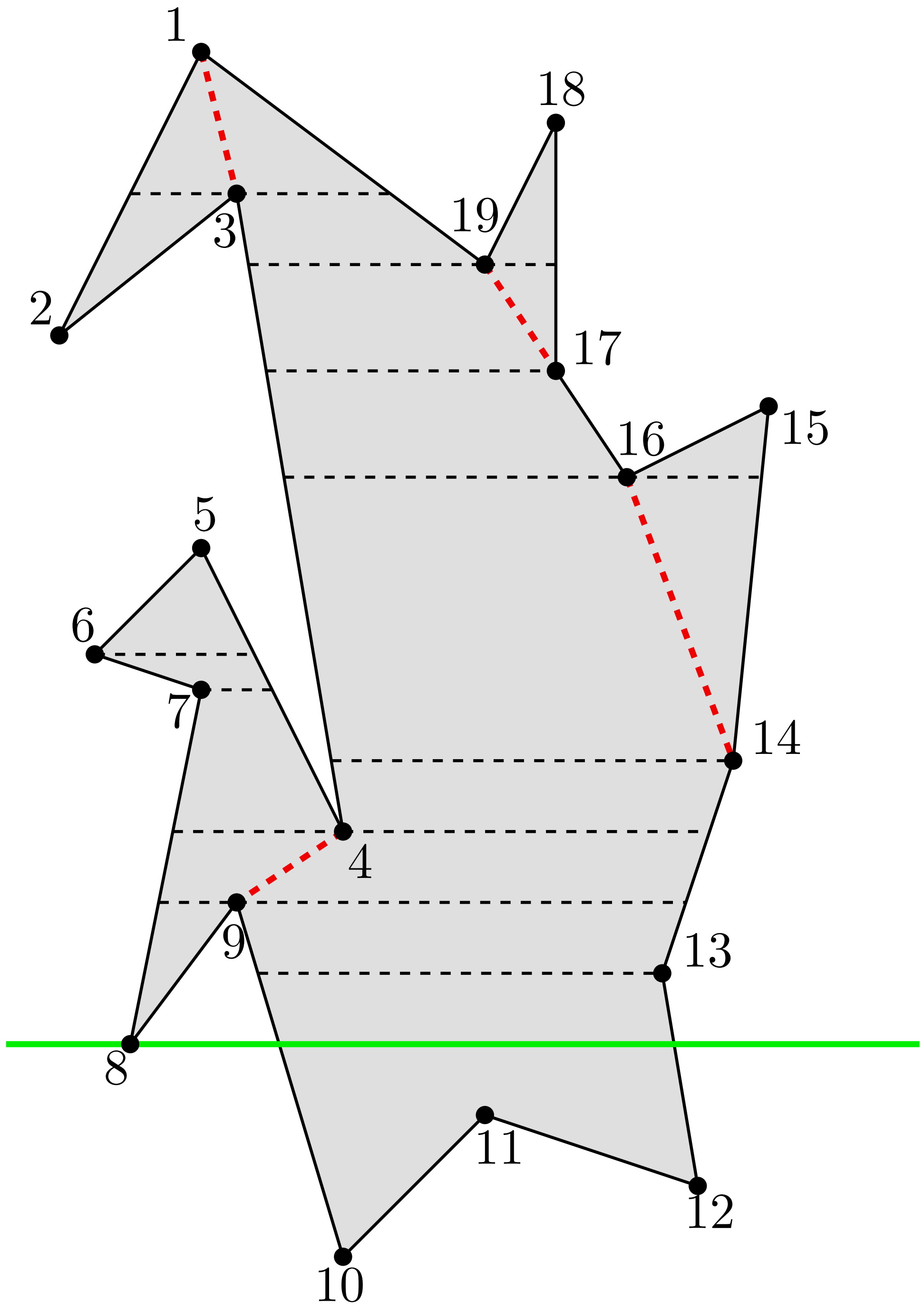


HANDLESPLITVERTEX(v_i)
1. Search in $\mathcal{T}$ to find the edge e_j directly left of v_i.
2. Insert the diagonal connecting v_i to $helper(e_j)$ in $\mathcal{D}$.
3. $helper(e_j) \leftarrow v_i$
4. Insert e_i in $\mathcal{T}$ and set $helper(e_i)$ to v_i.

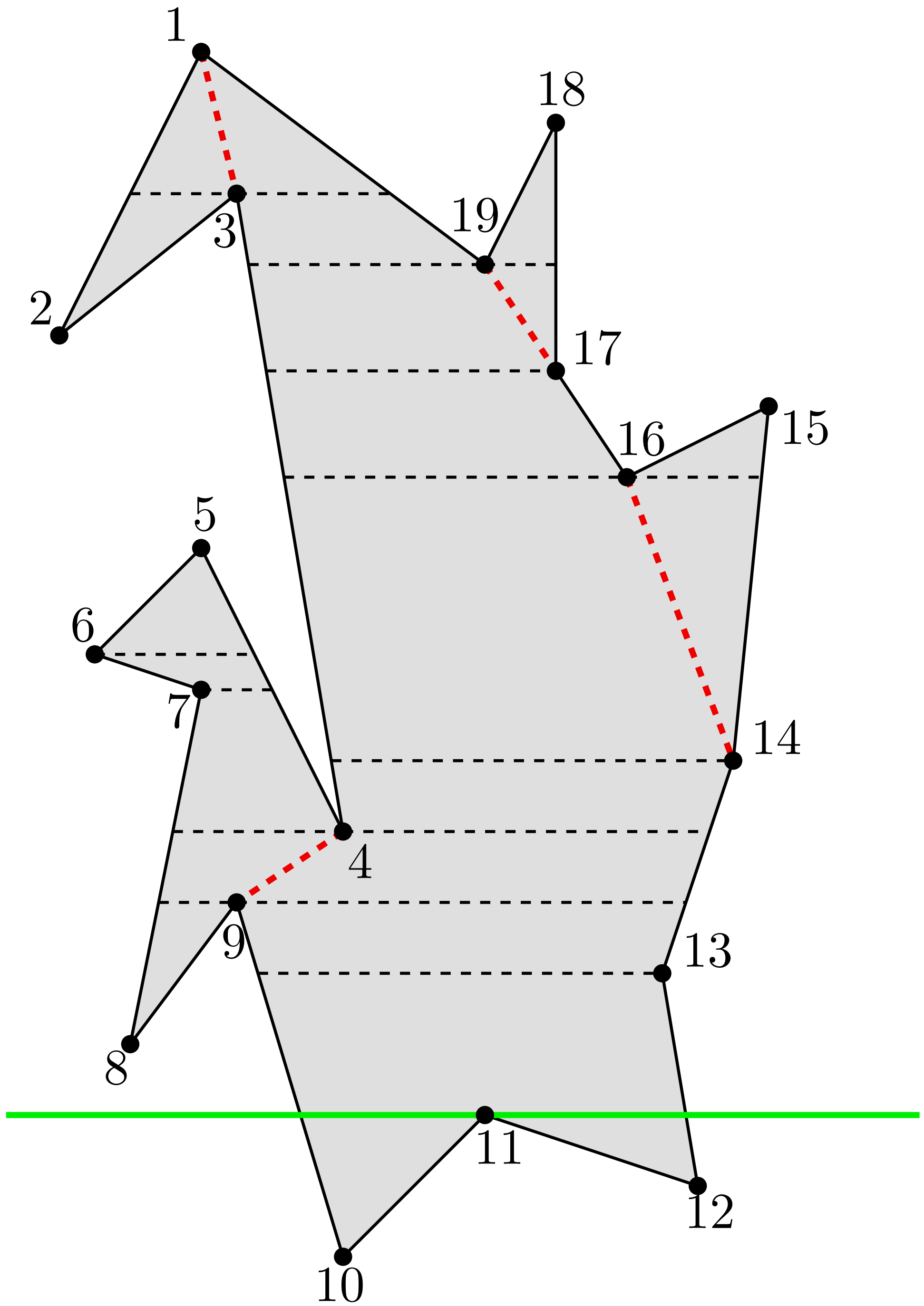
Monotone partition



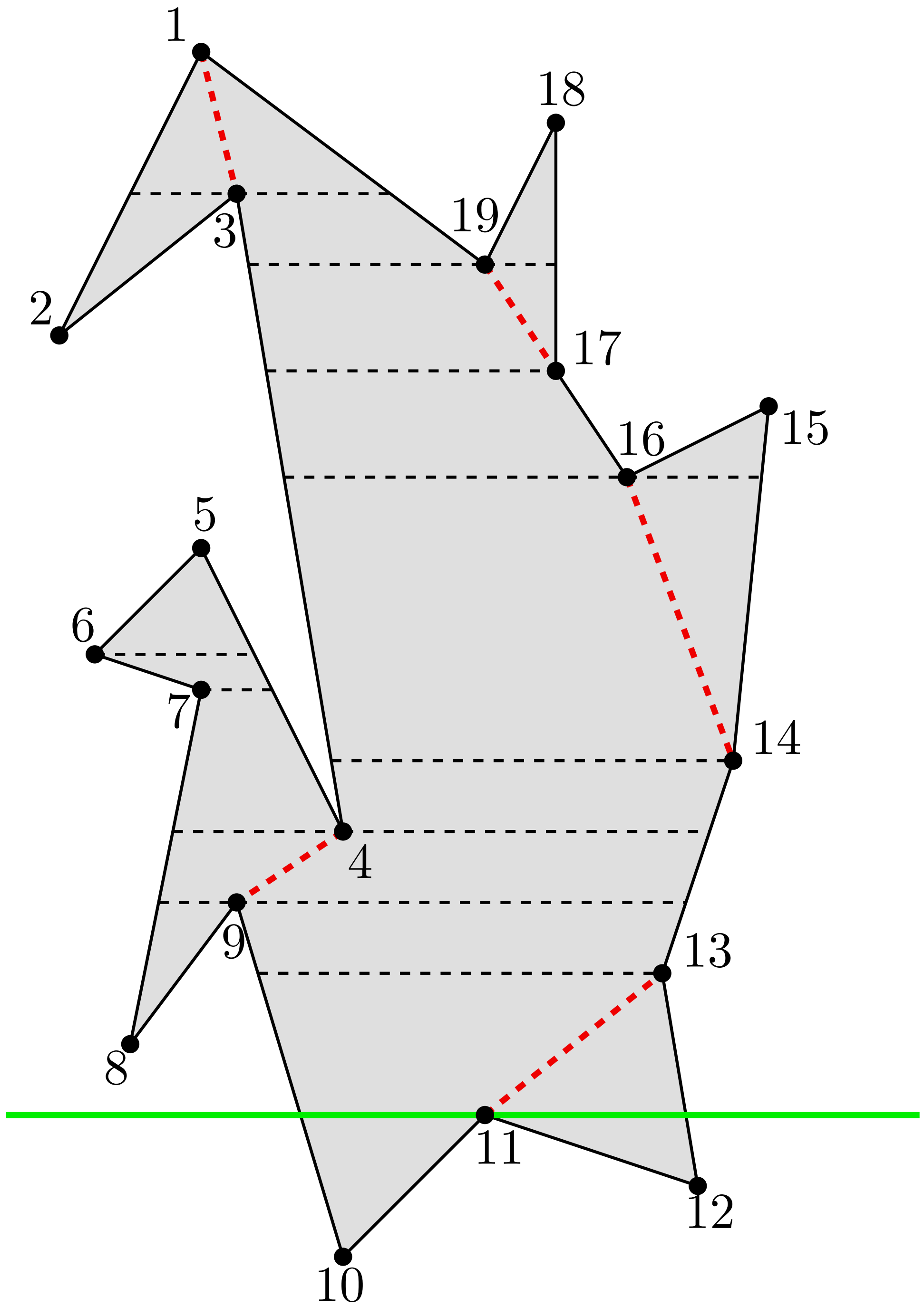
Monotone partition



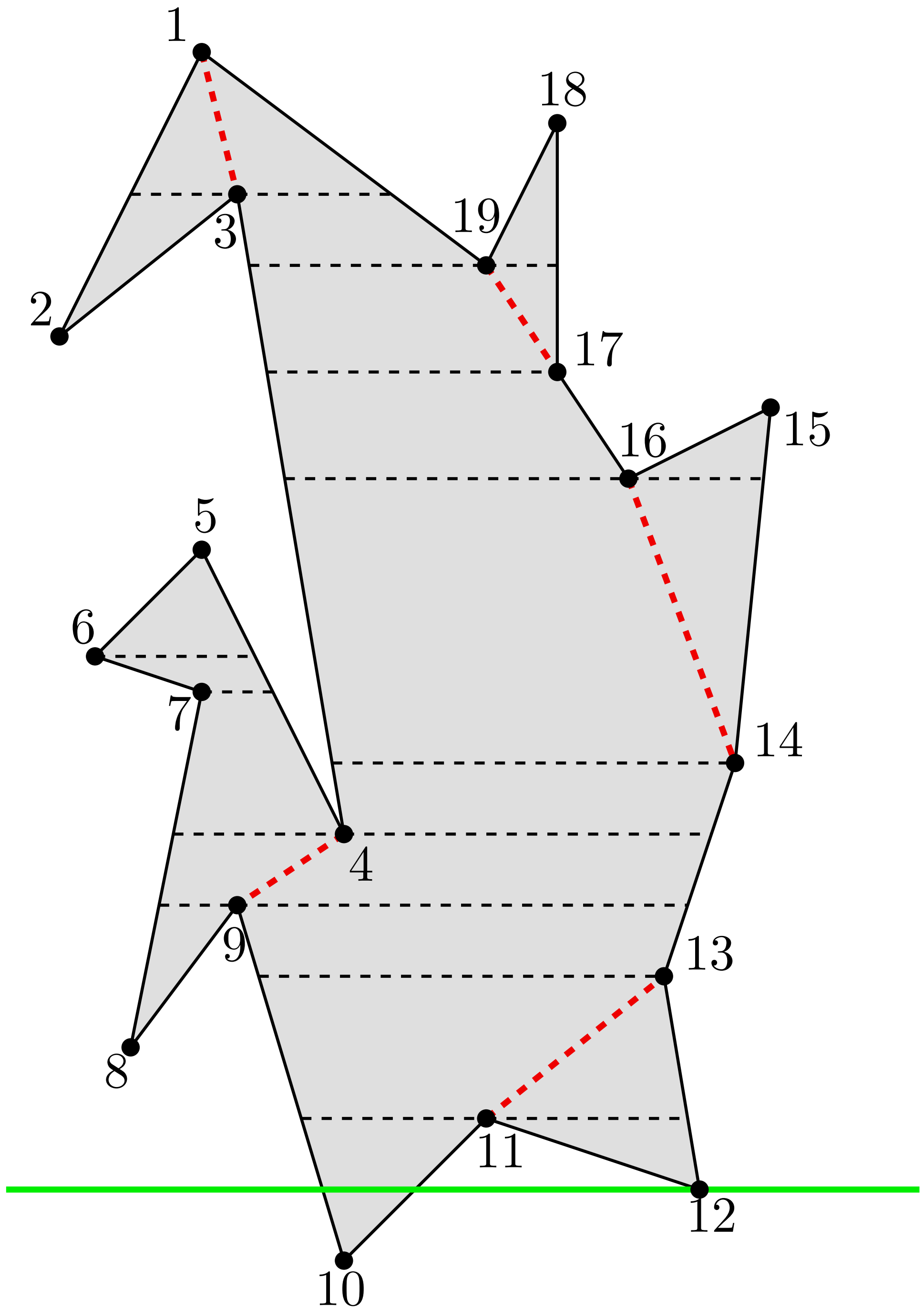
Monotone partition



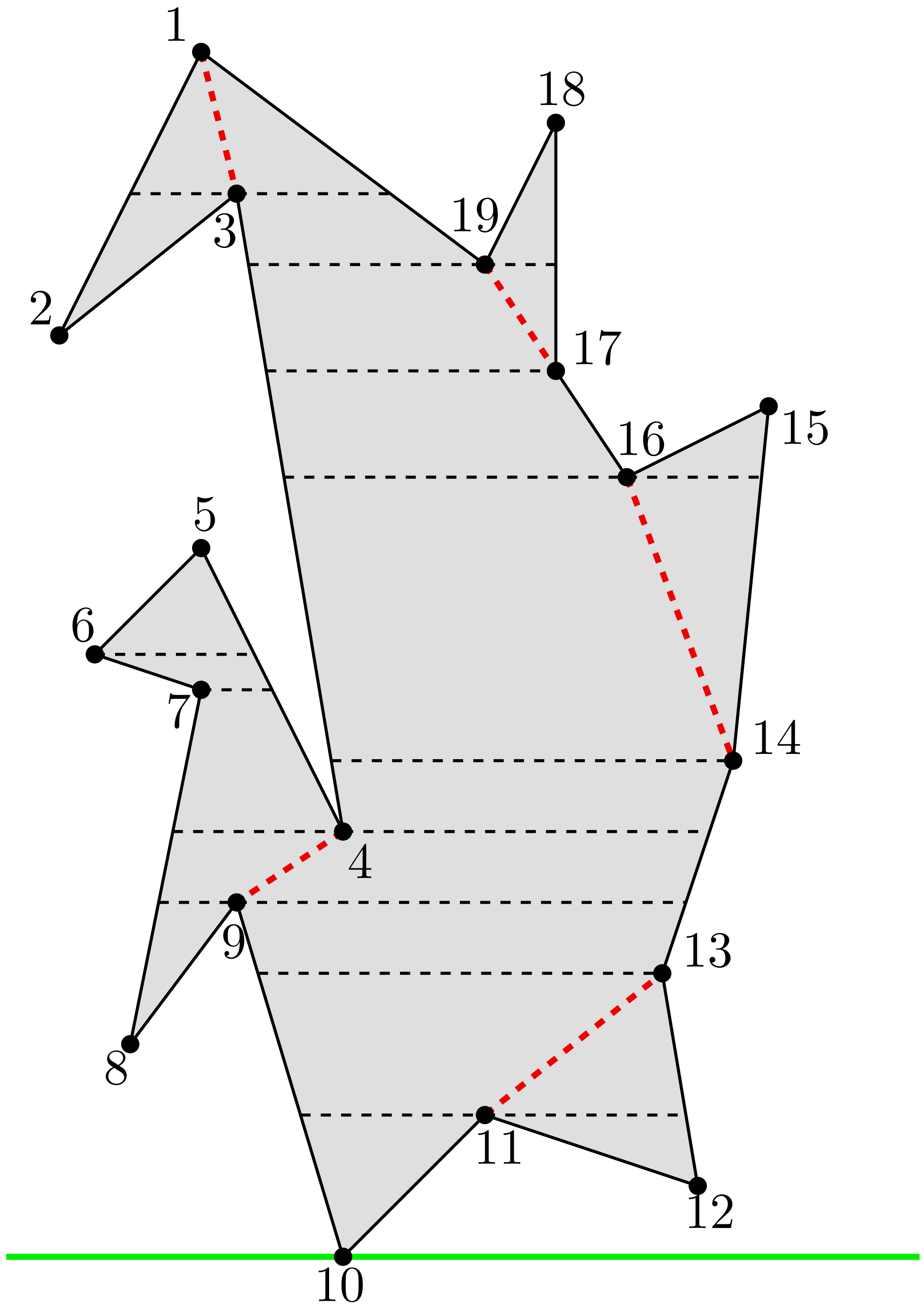
Monotone partition



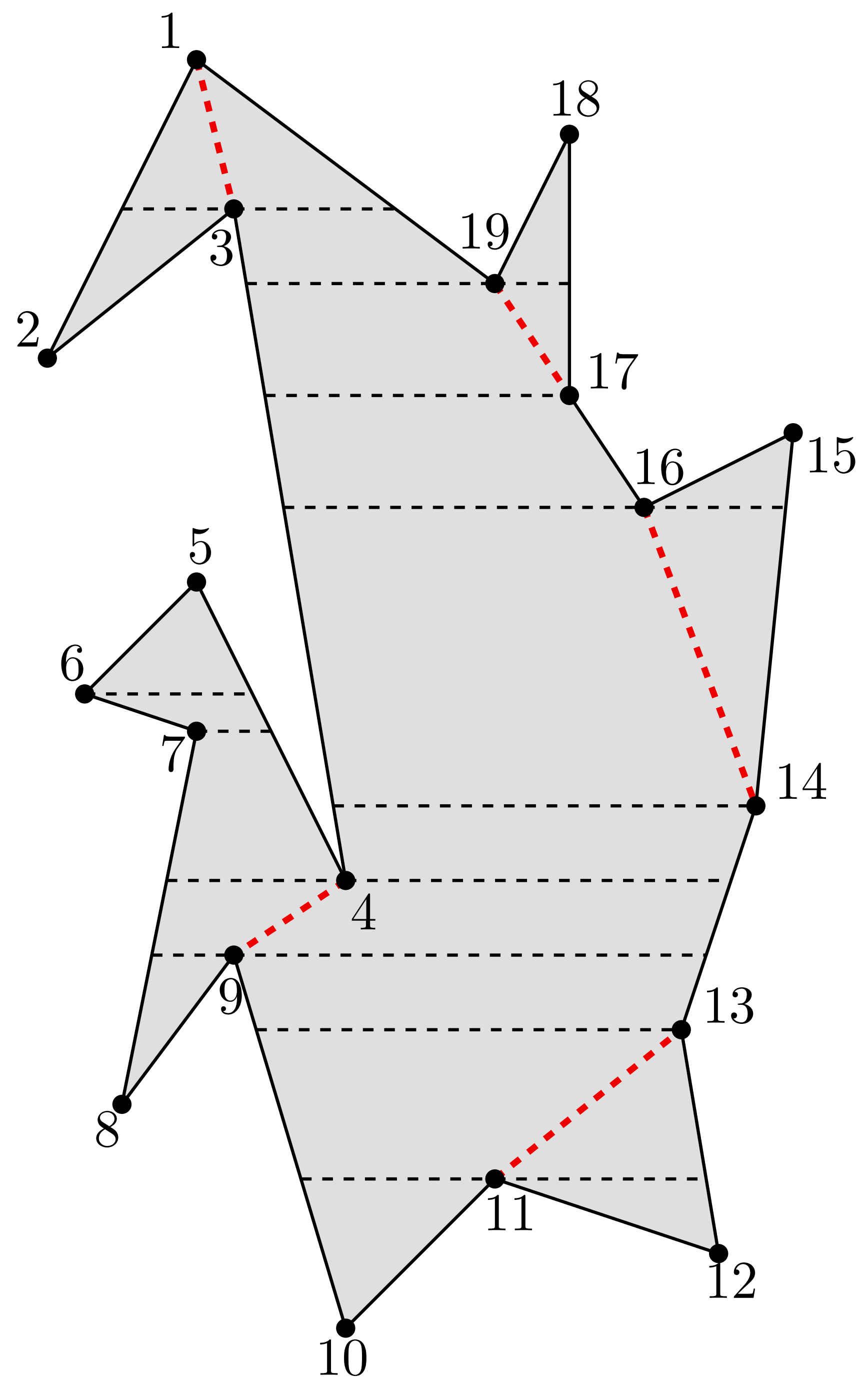
Monotone partition



Monotone partition

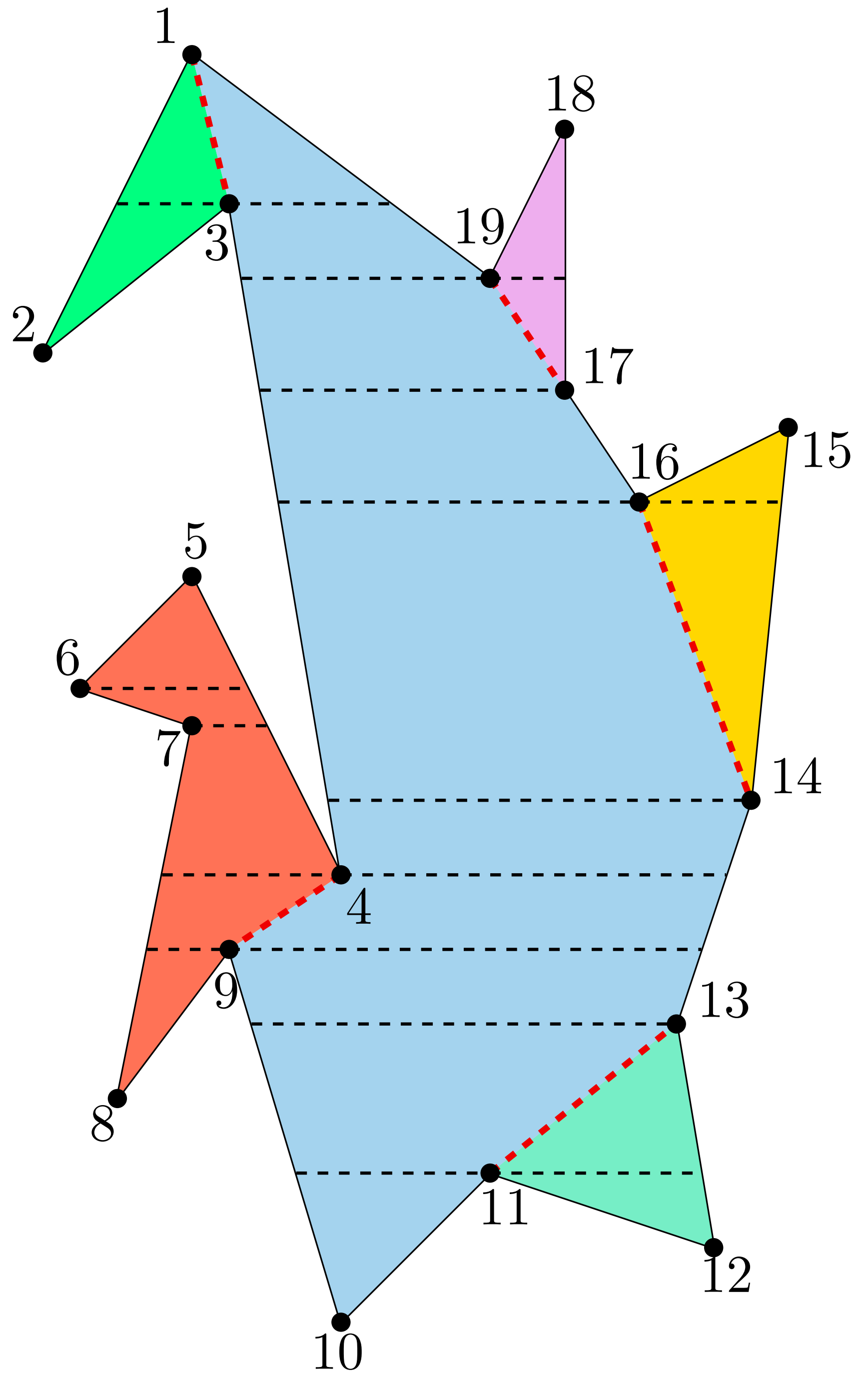


Monotone partition



TRIANGULATING POLYGONS

Monotone partition

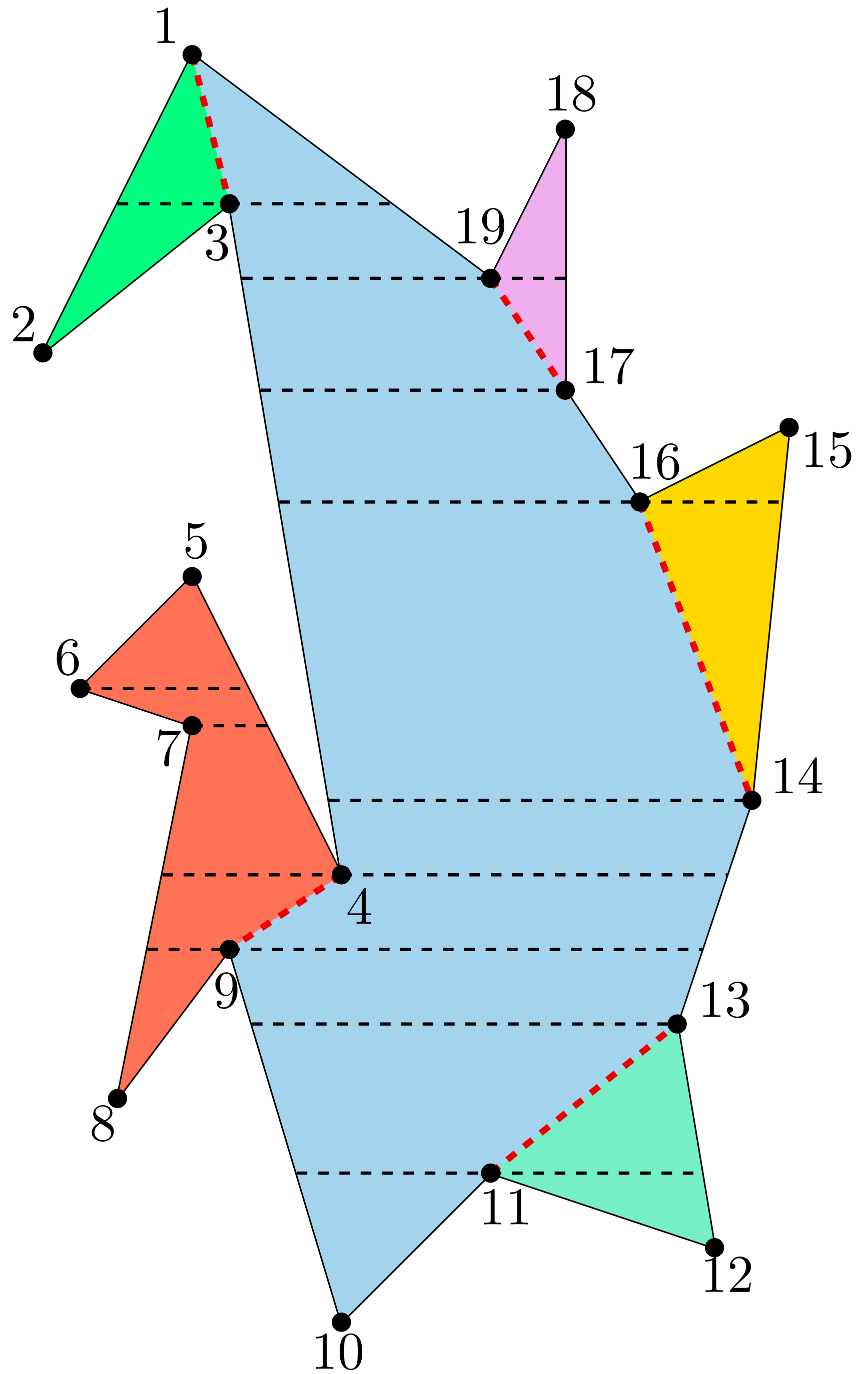


Monotone partition

Running time:

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

The algorithm runs 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

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?

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

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

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.

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.

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.

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.

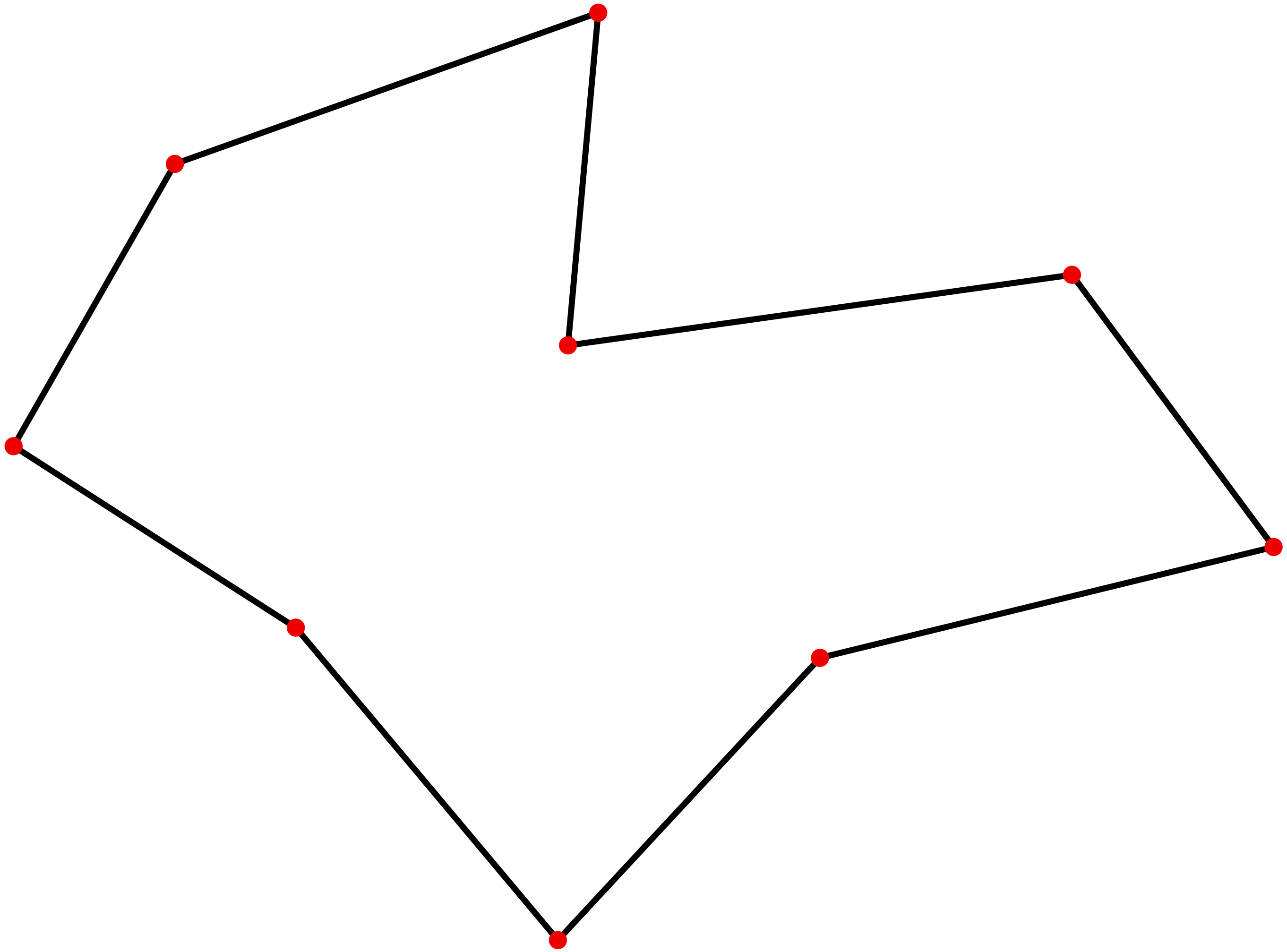
Disadvantage: the stored data is redundant and it uses more space than required.

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.

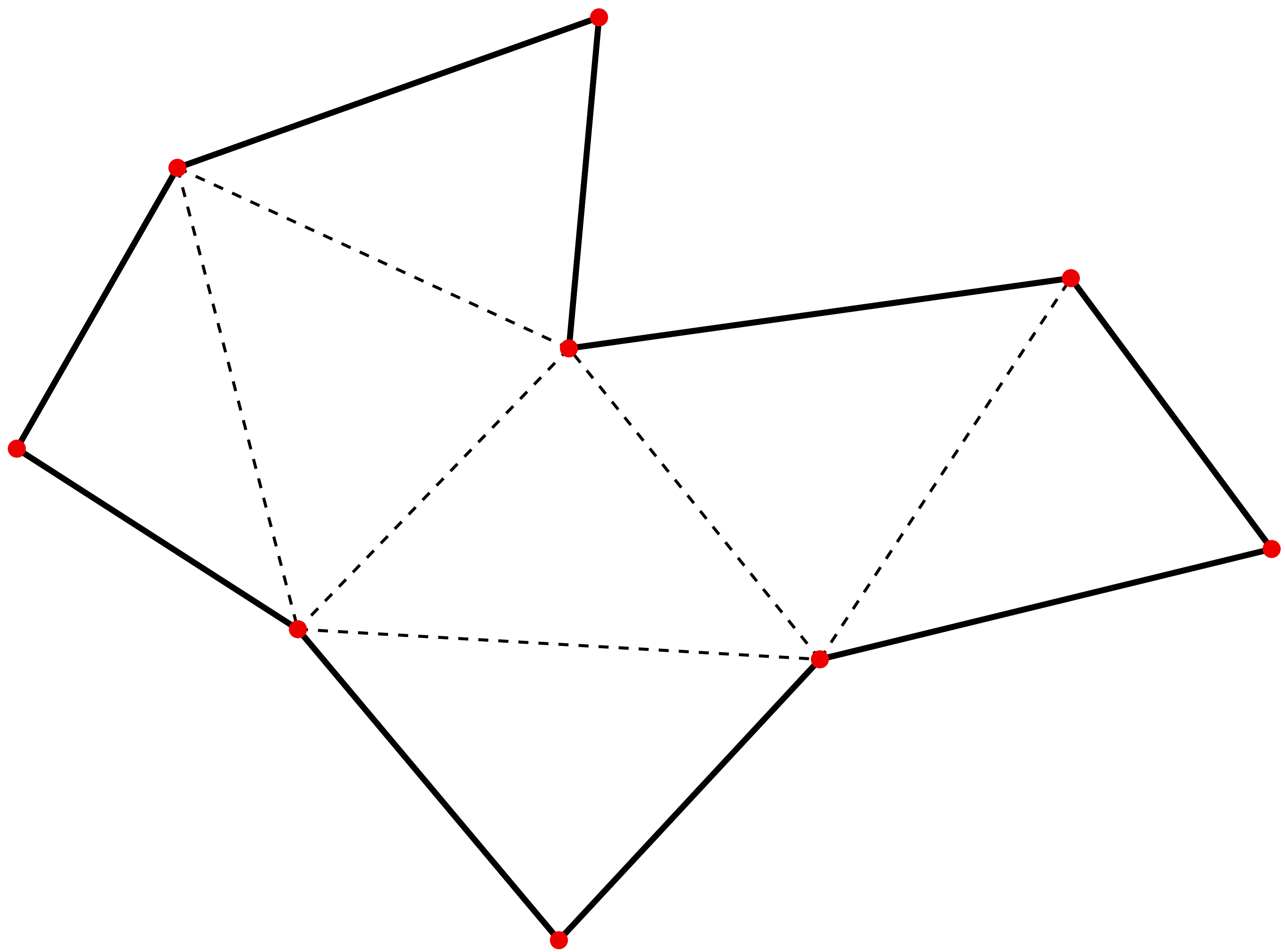
Storing the polygon triangulation

DCEL



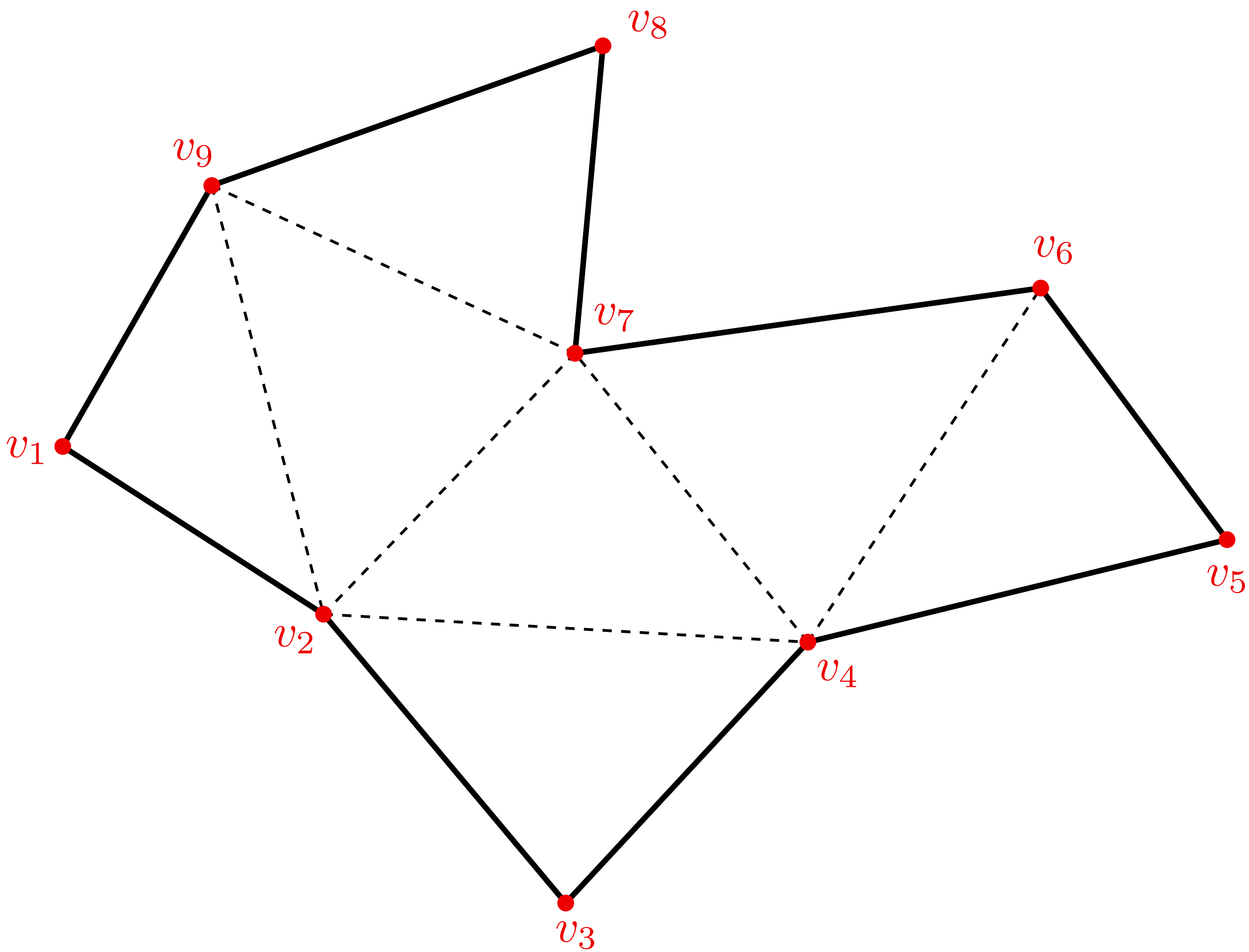
Storing the polygon triangulation

DCEL



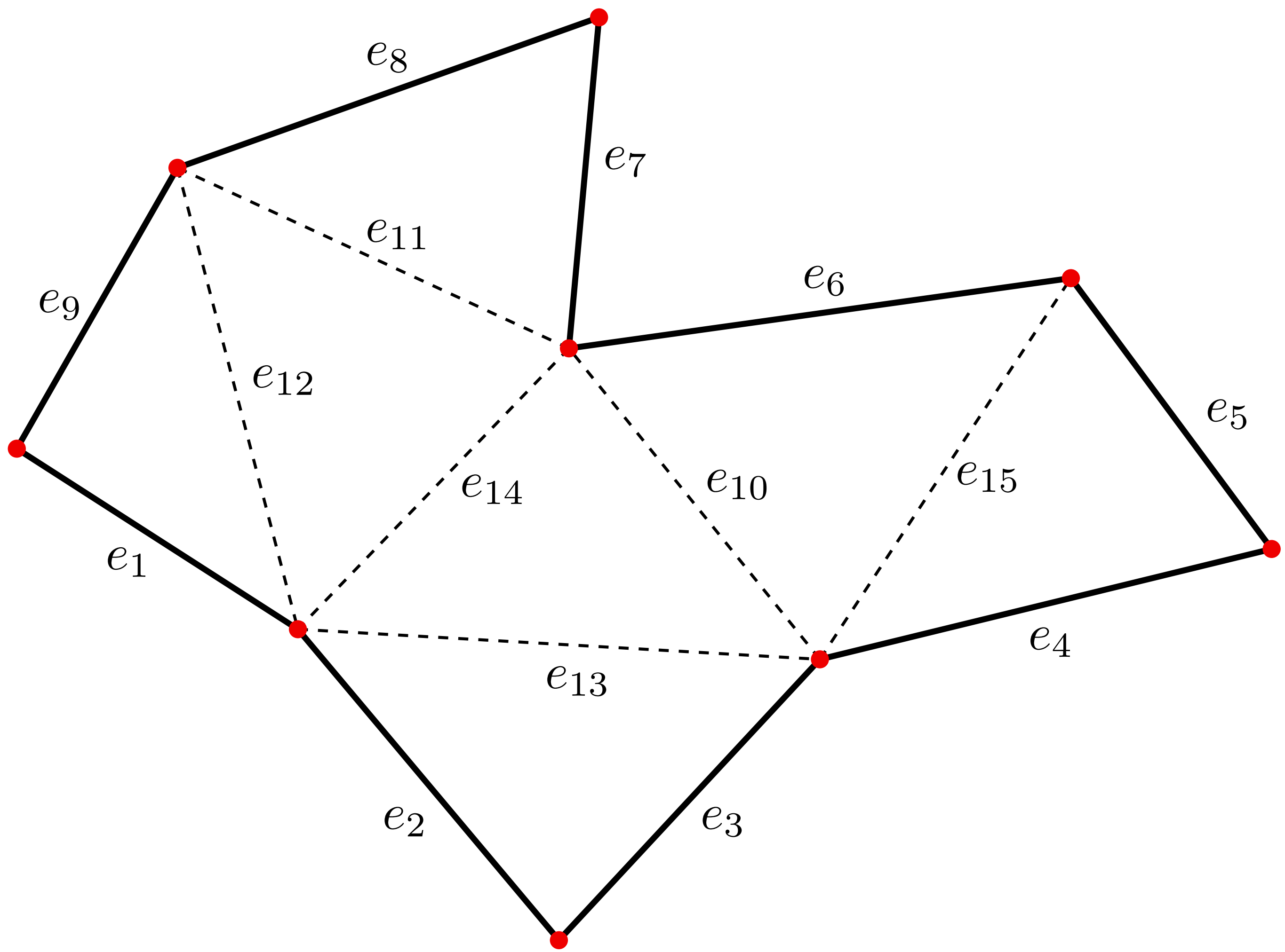
Storing the polygon triangulation

DCEL



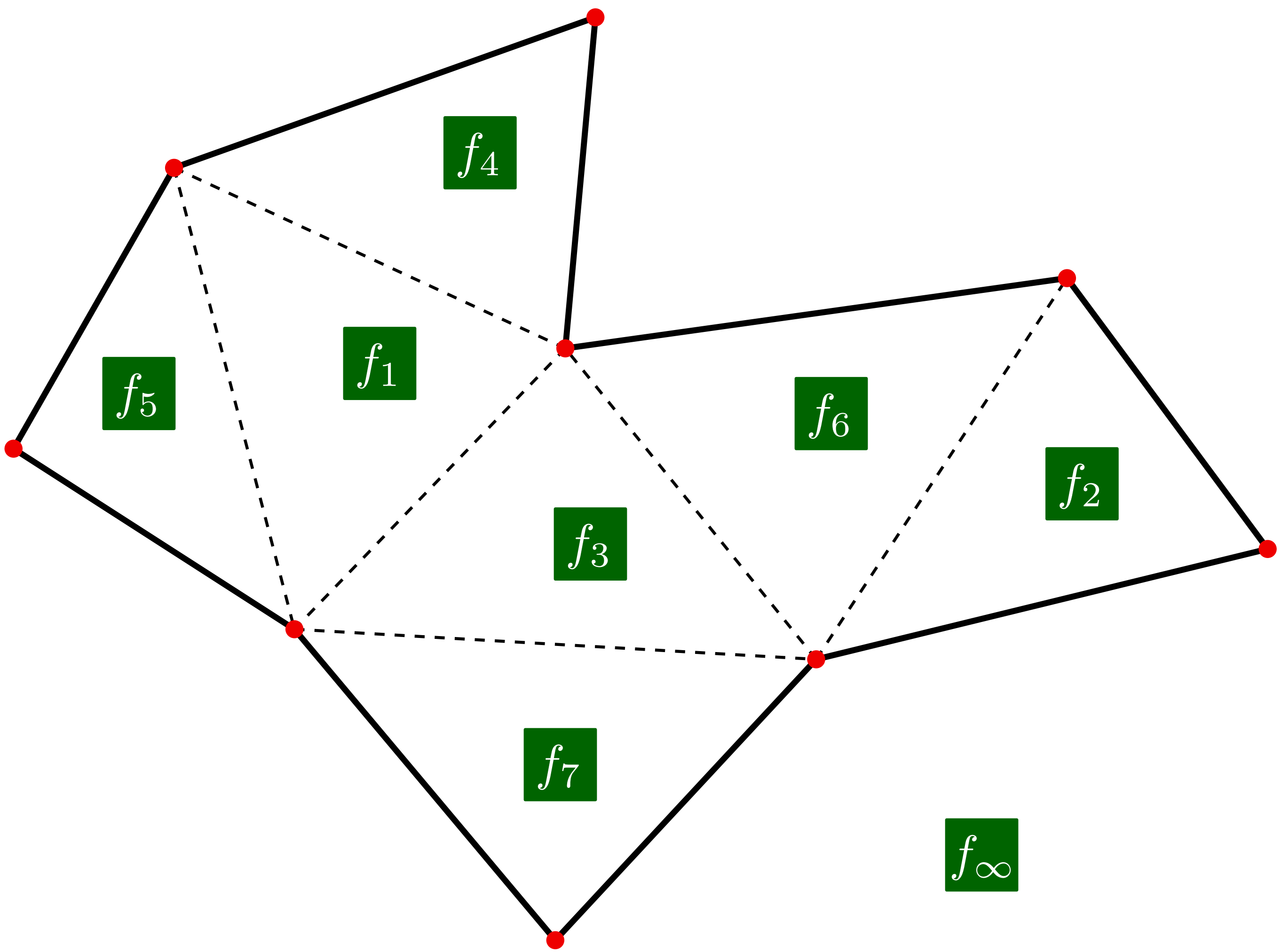
Storing the polygon triangulation

DCEL



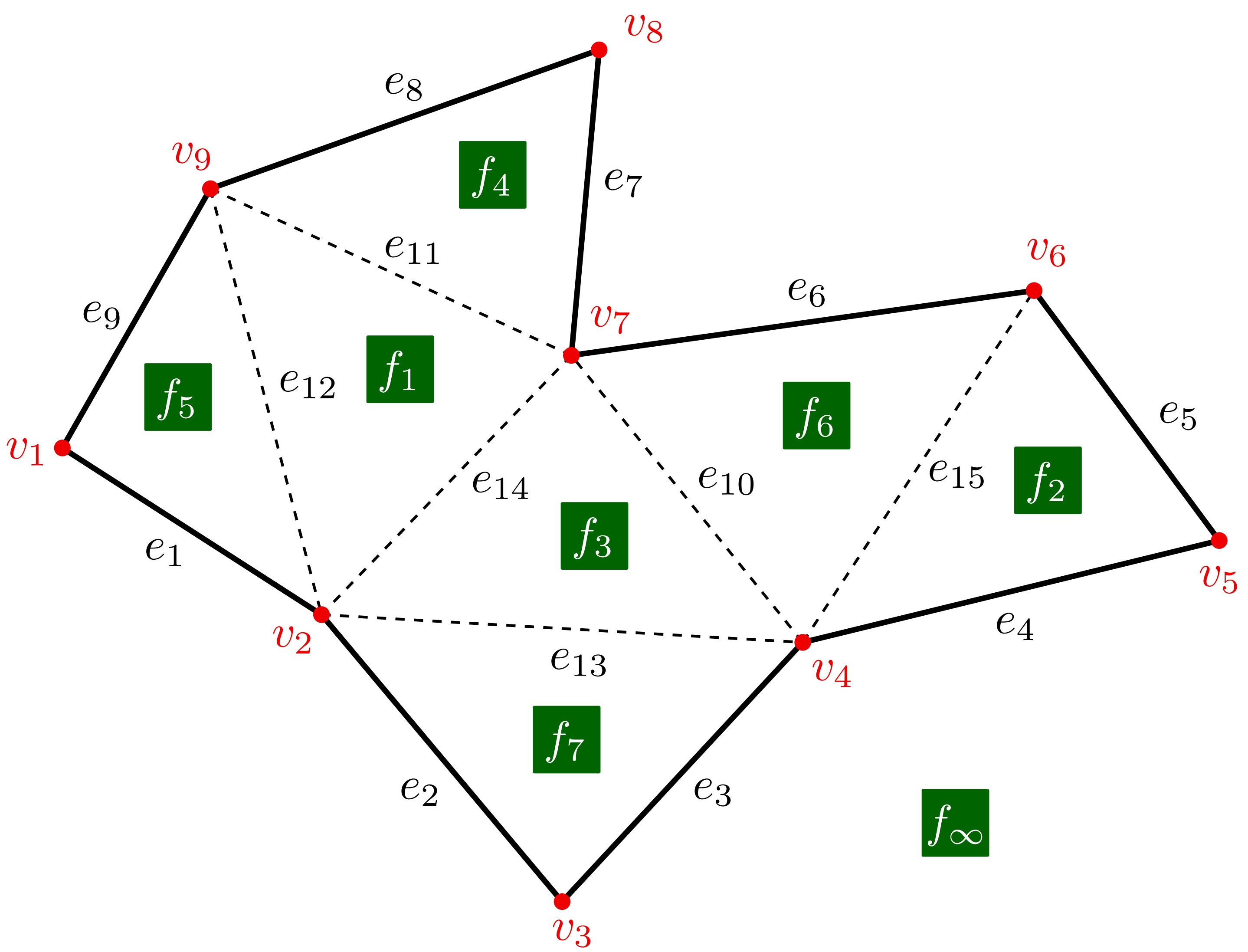
Storing the polygon triangulation

DCEL



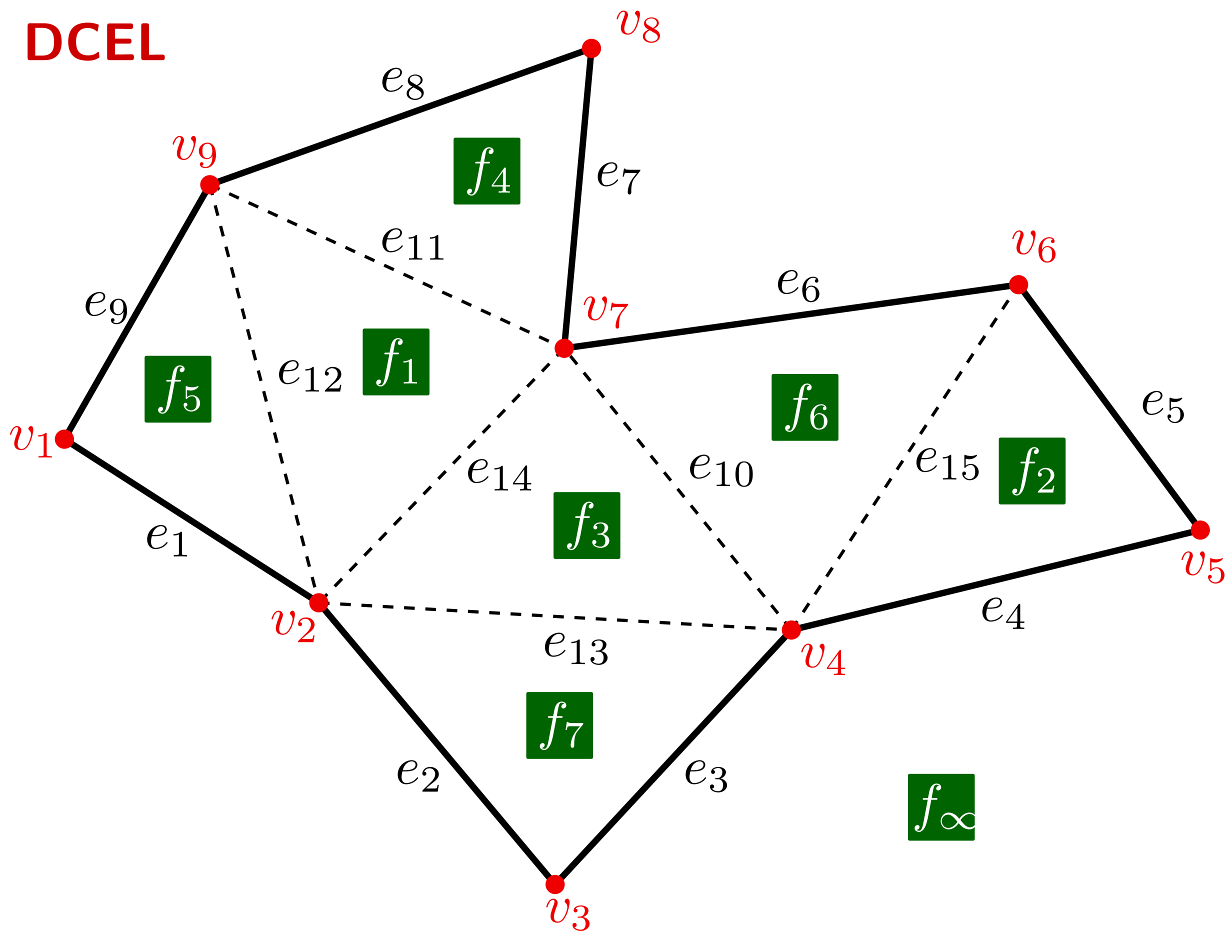
Storing the polygon triangulation

DCEL



Storing the polygon triangulation

DCEL

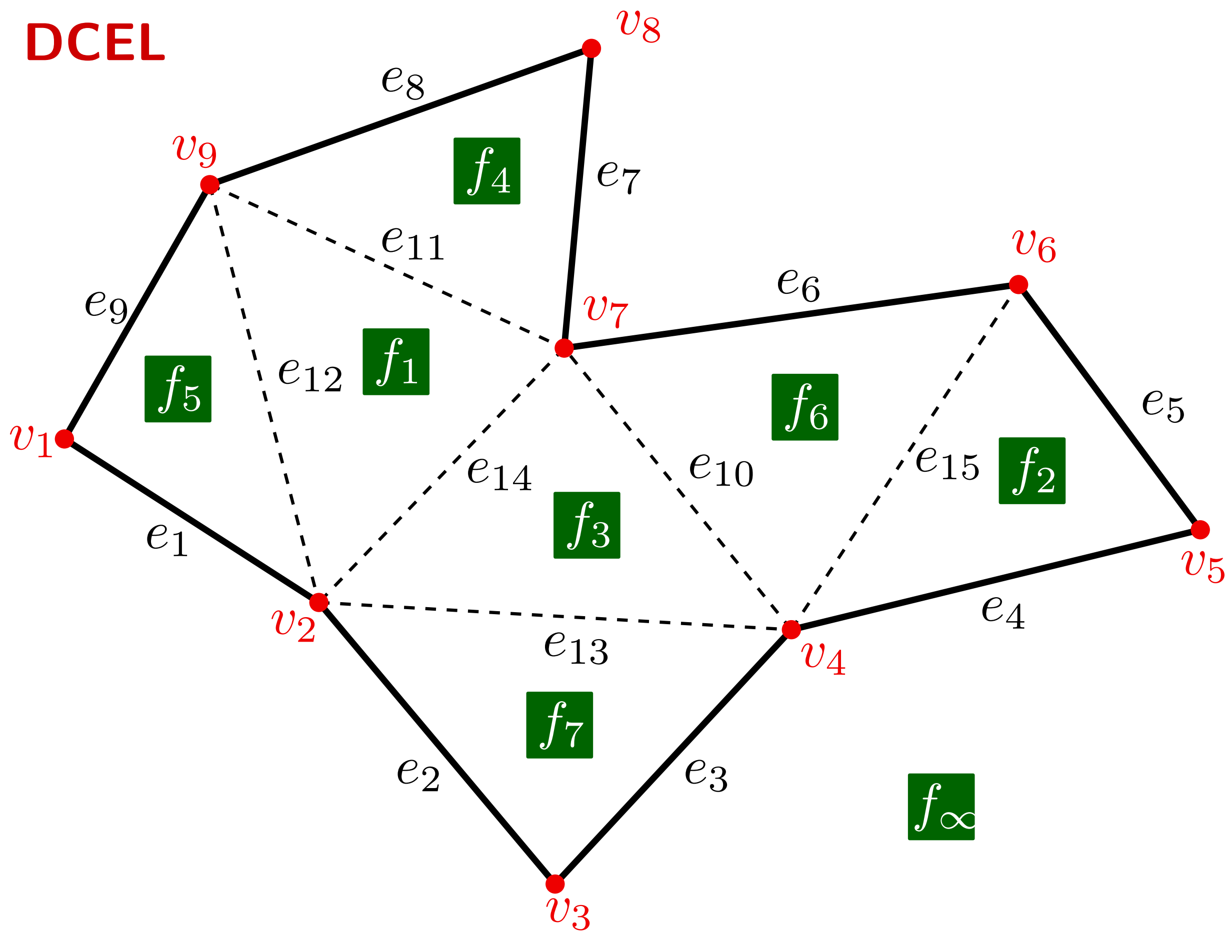


Storing the polygon triangulation

Table of vertices

v	x	y	e
1	x_1	y_1	1
2	x_2	y_2	1
3	x_3	y_3	2
4	x_4	y_4	10
5	x_5	y_5	4
6	x_6	y_6	6
7	x_7	y_7	10
8	x_8	y_8	8
9	x_9	y_9	9

DCEL

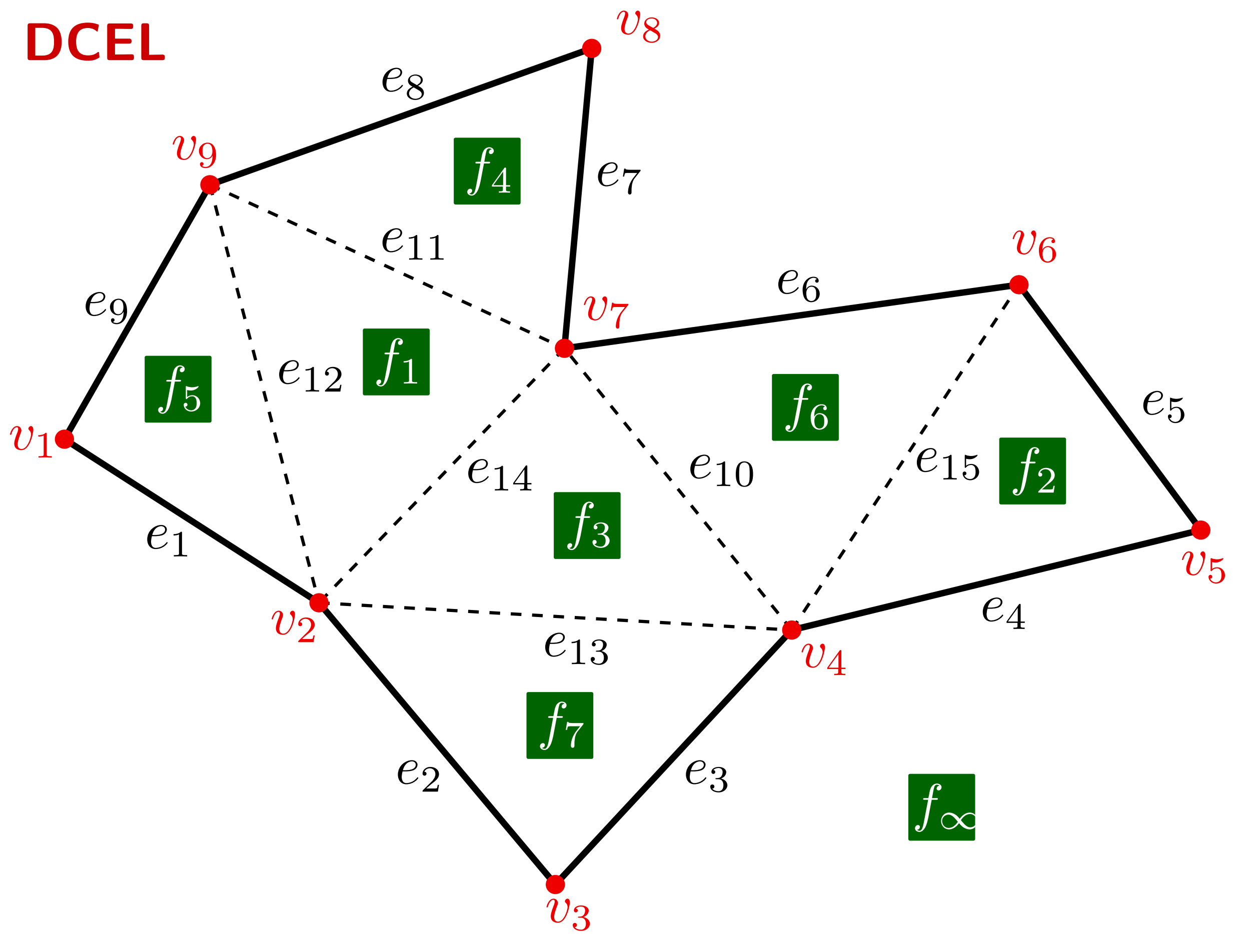


Storing the polygon triangulation

Table of faces

f	e
1	11
2	4
3	10
4	11
5	1
6	6
7	2
∞	9

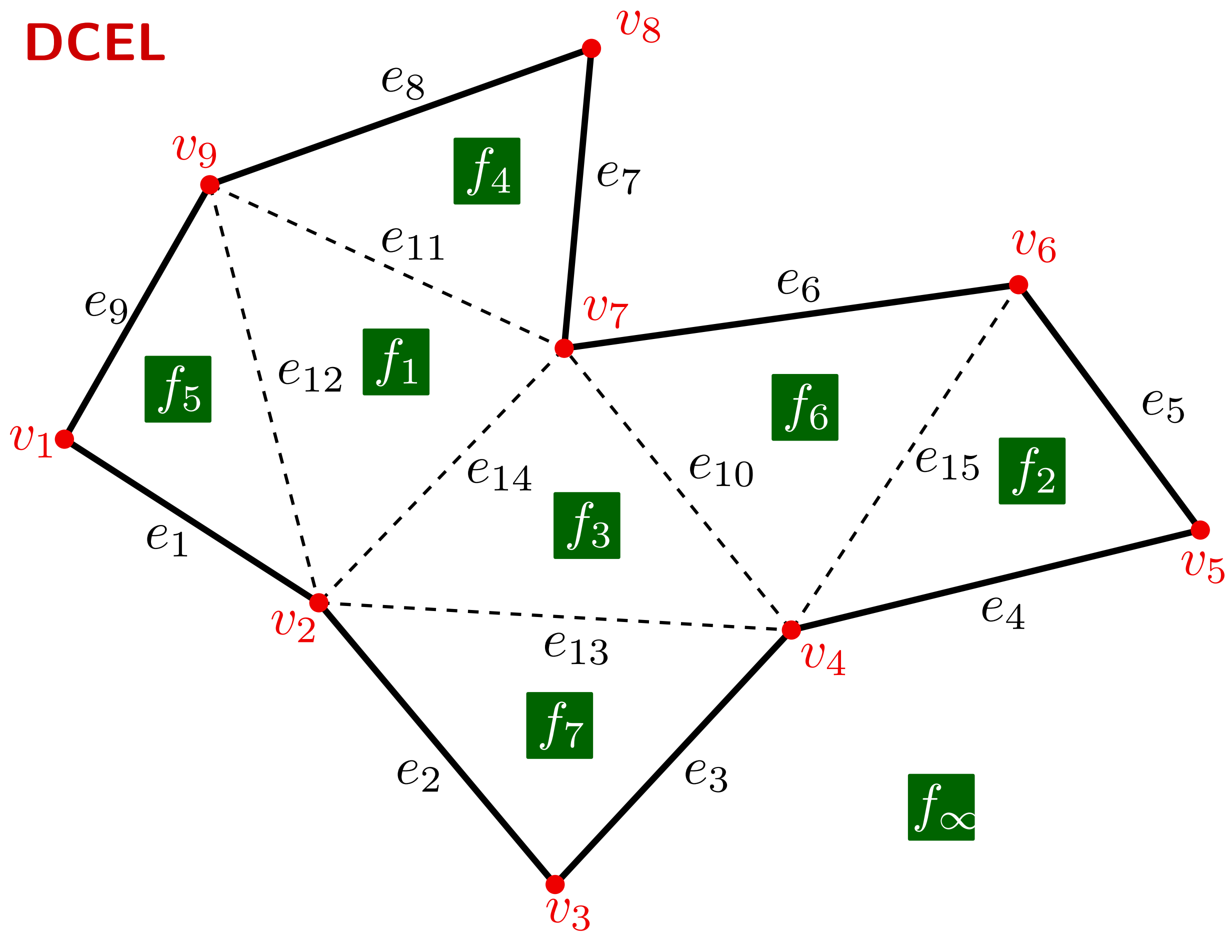
DCEL



DCEL

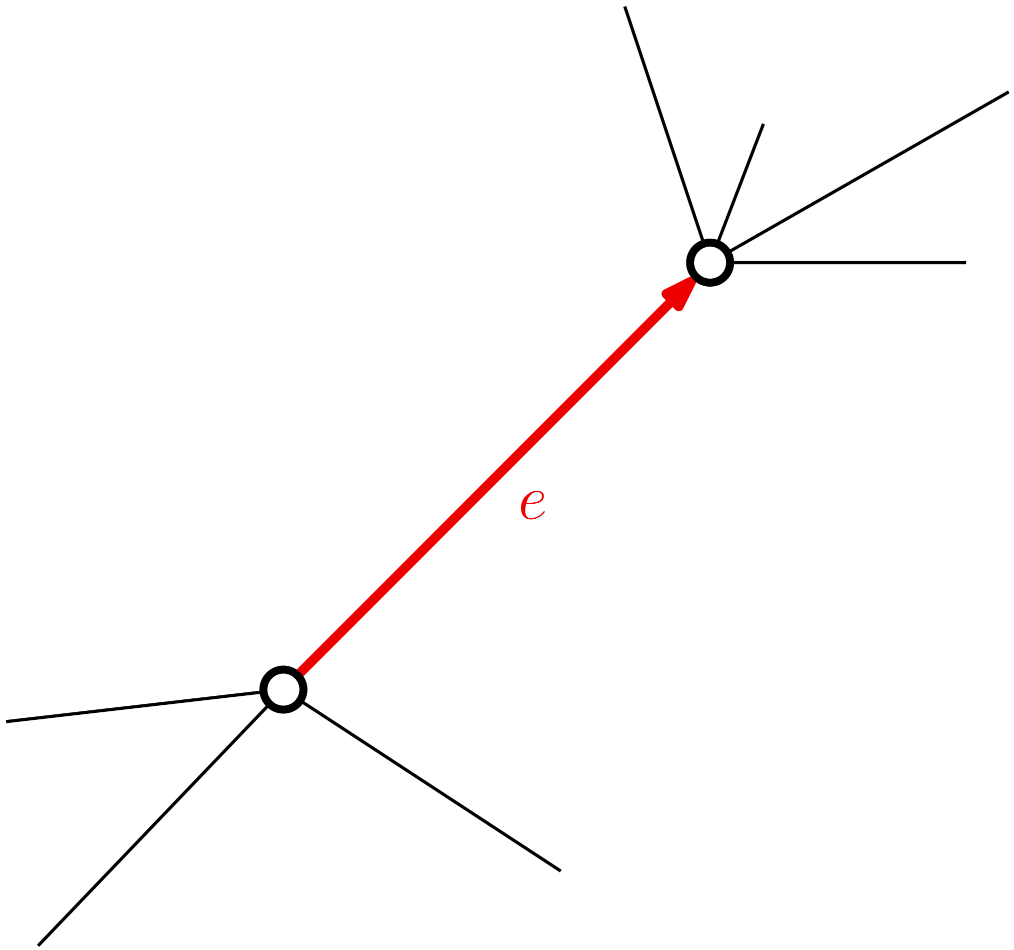
e	v_B	v_E	f_L	f_R	e_P	e_N
1	1	2	5	∞	9	2
2	2	3	7	∞	13	3
3	4	3	∞	7	4	2
4	4	5	2	∞	15	5
5	5	6	2	∞	4	6
6	6	7	6	∞	15	7
7	7	8	4	∞	11	8
8	8	9	4	∞	7	9
9	9	1	5	∞	12	1
10	4	7	3	6	13	6
11	9	7	4	1	8	14
12	2	9	5	1	1	11
13	2	4	3	7	14	3
14	2	7	1	3	12	10
15	4	6	6	2	10	5

DCEL



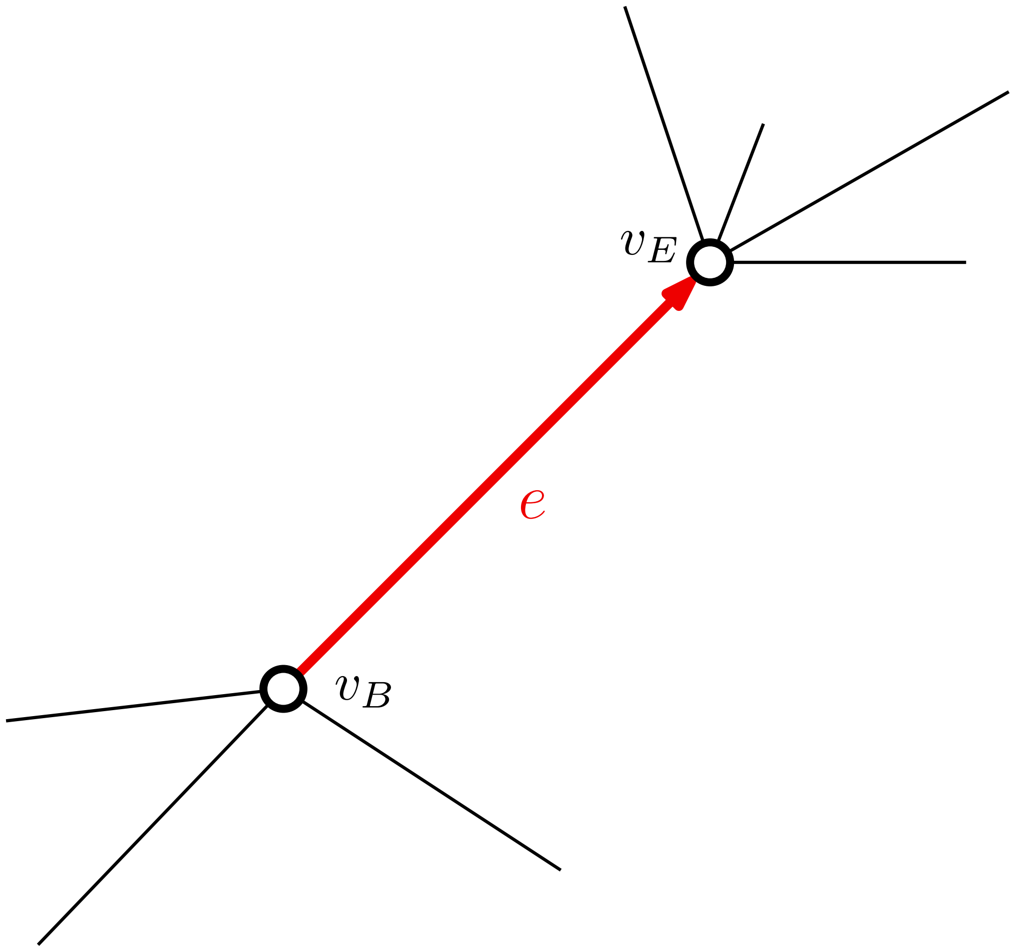
Storing the polygon triangulation

DCEL



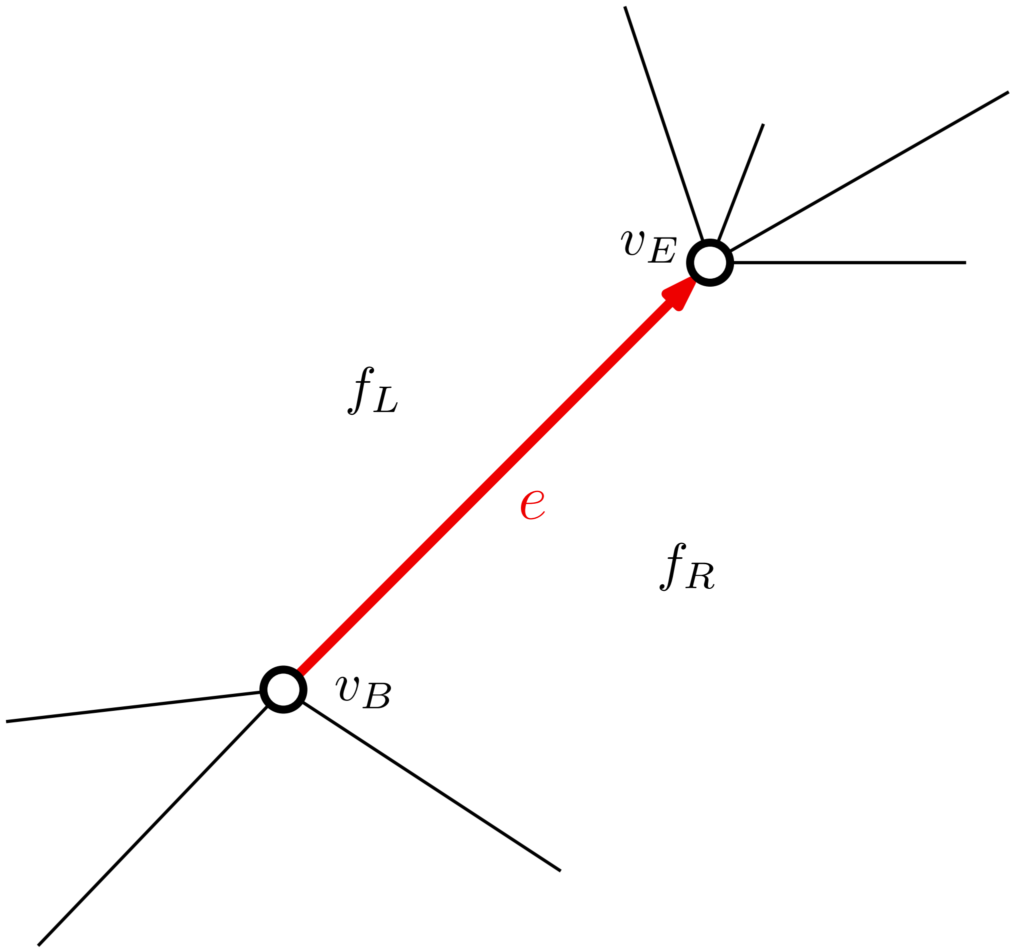
Storing the polygon triangulation

DCEL



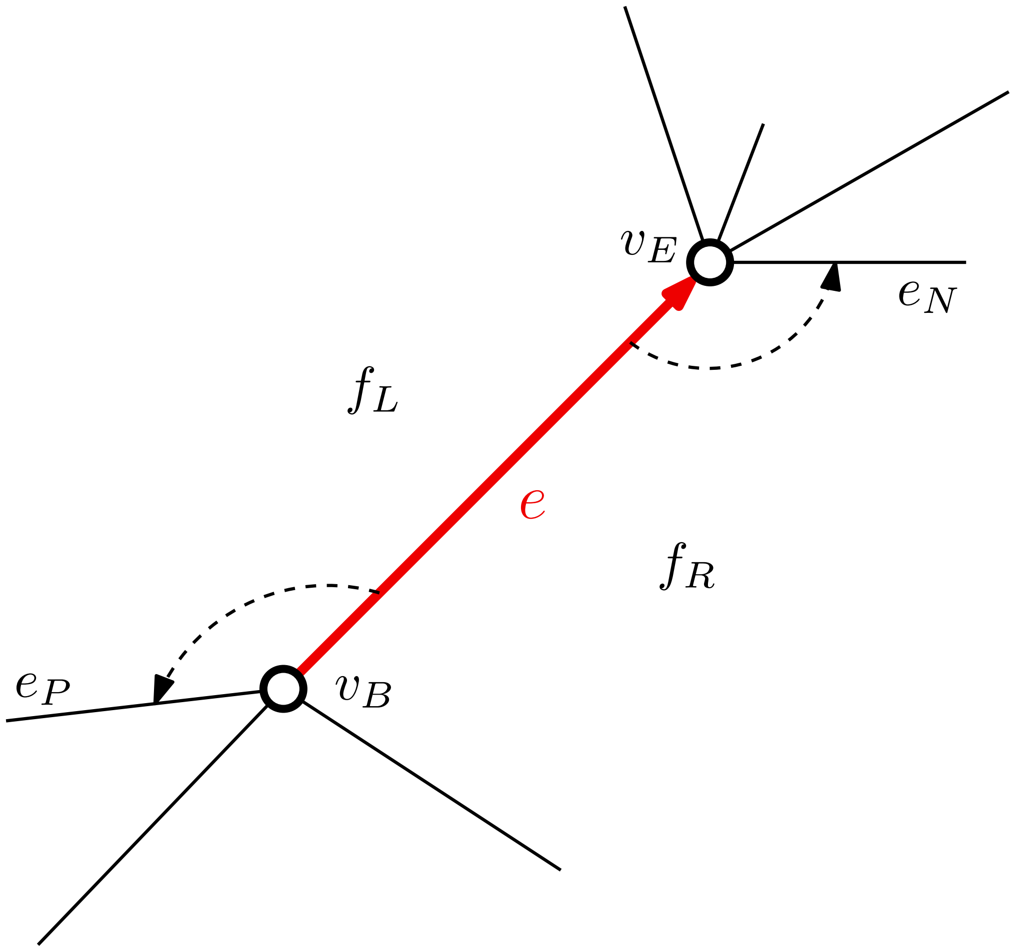
Storing the polygon triangulation

DCEL



Storing the polygon triangulation

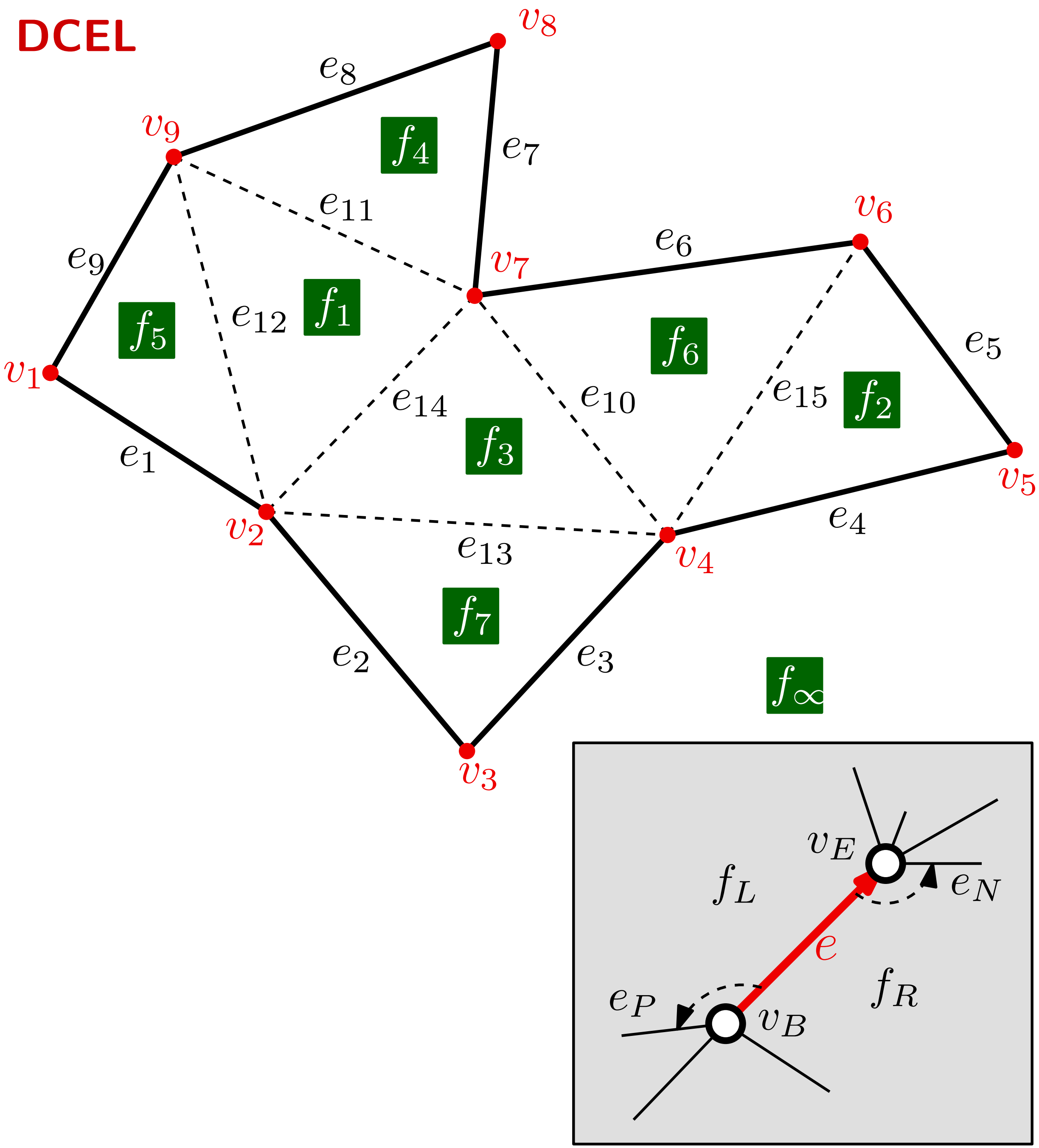
DCEL



Storing the polygon triangulation

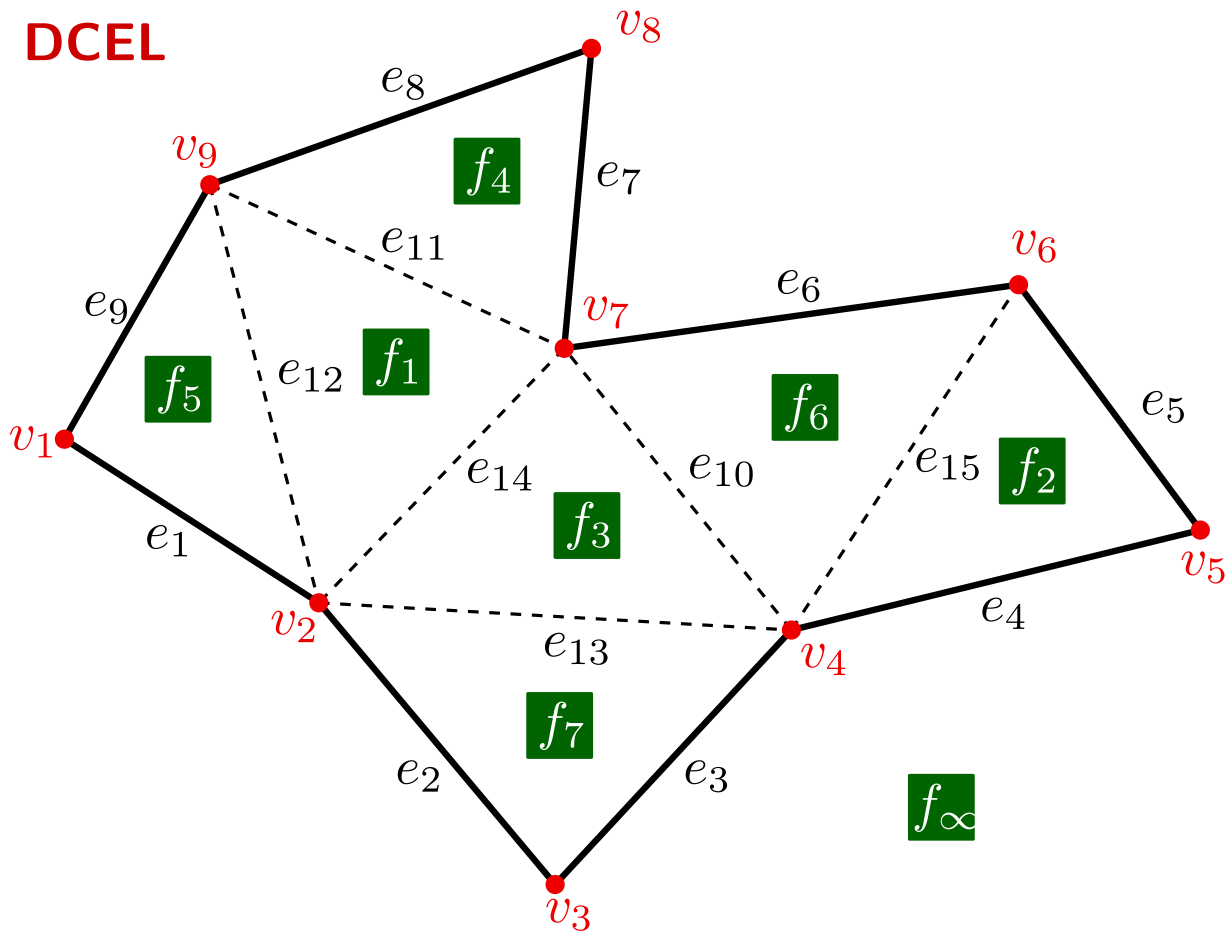
DCEL

e	v_B	v_E	f_L	f_R	e_P	e_N
1	1	2	5	∞	9	2
2	2	3	7	∞	13	3
3	4	3	∞	7	4	2
4	4	5	2	∞	15	5
5	5	6	2	∞	4	6
6	6	7	6	∞	15	7
7	7	8	4	∞	11	8
8	8	9	4	∞	7	9
9	9	1	5	∞	12	1
10	4	7	3	6	13	6
11	9	7	4	1	8	14
12	2	9	5	1	1	11
13	2	4	3	7	14	3
14	2	7	1	3	12	10
15	4	6	6	2	10	5



Storing the polygon triangulation

DCEL



Storage space

- For each face:

  1 pointer

- For each vertex:

  2 coordinates + 1 pointer

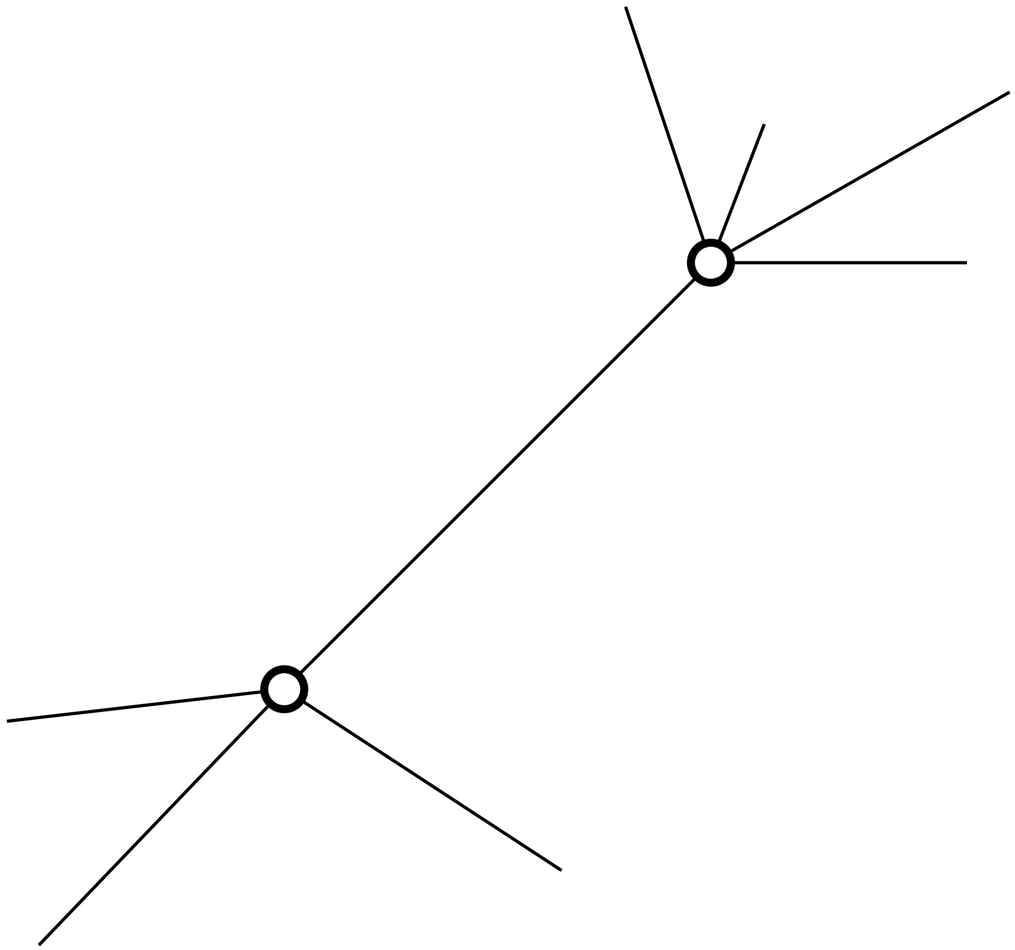
- For each edge:

  6 pointers

In total, the storage space is $O(n)$.

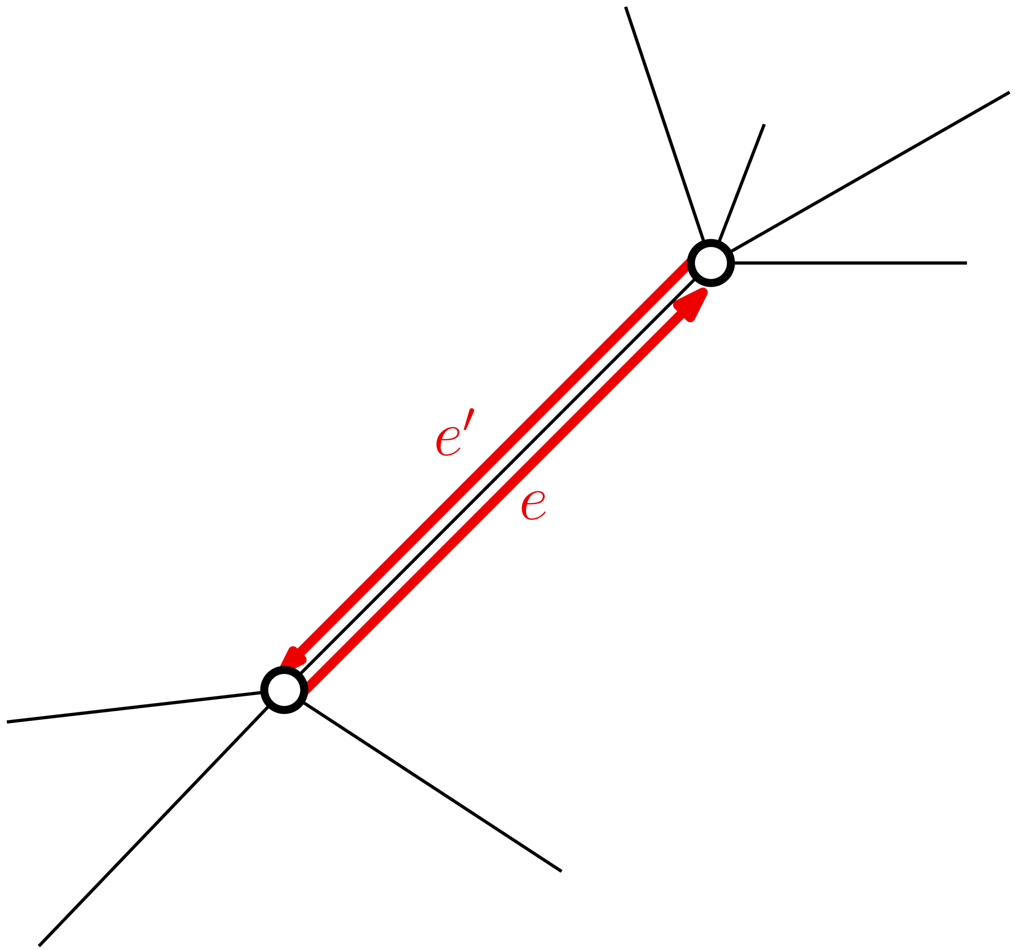
DCEL

There are other DCEL variants, as for example:



DCEL

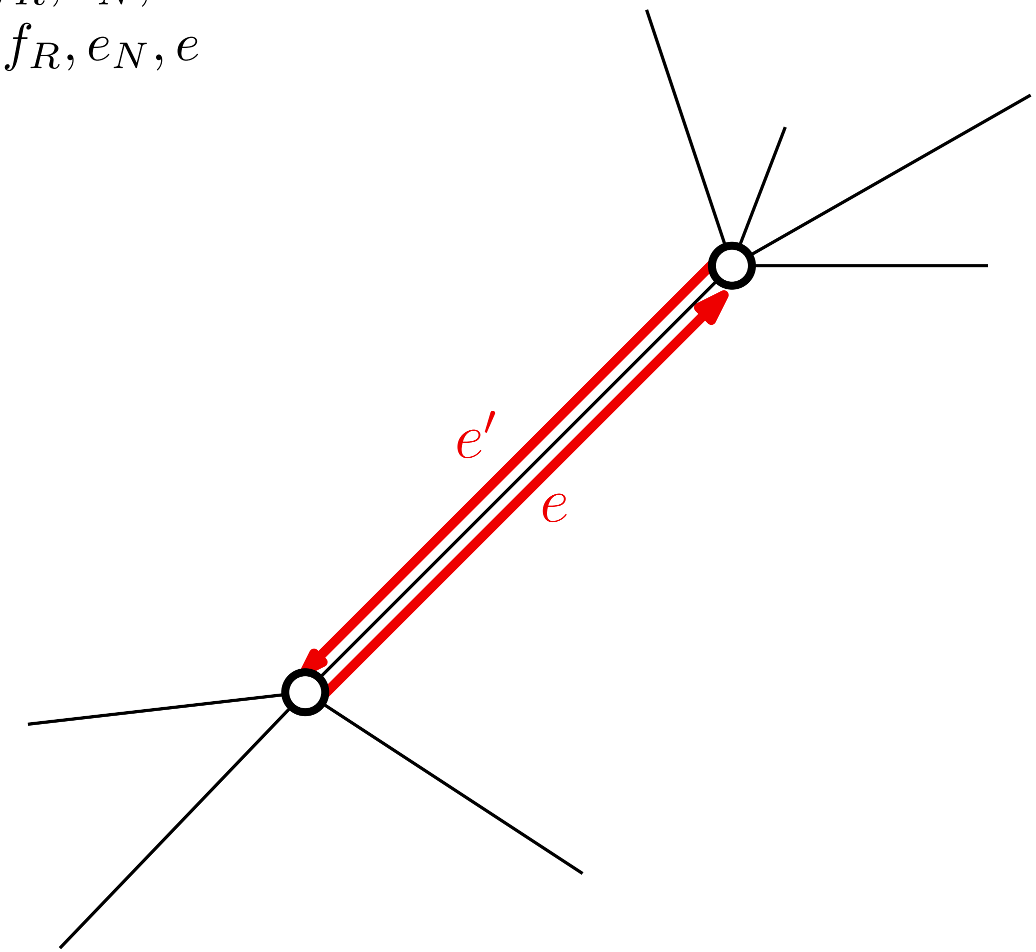
There are other DCEL variants, as for example:



DCEL

There are other DCEL variants, as for example:

$$e \longrightarrow v_B, f_R, e_N, e'$$
$$e' \longrightarrow v_B, f_R, e_N, e$$

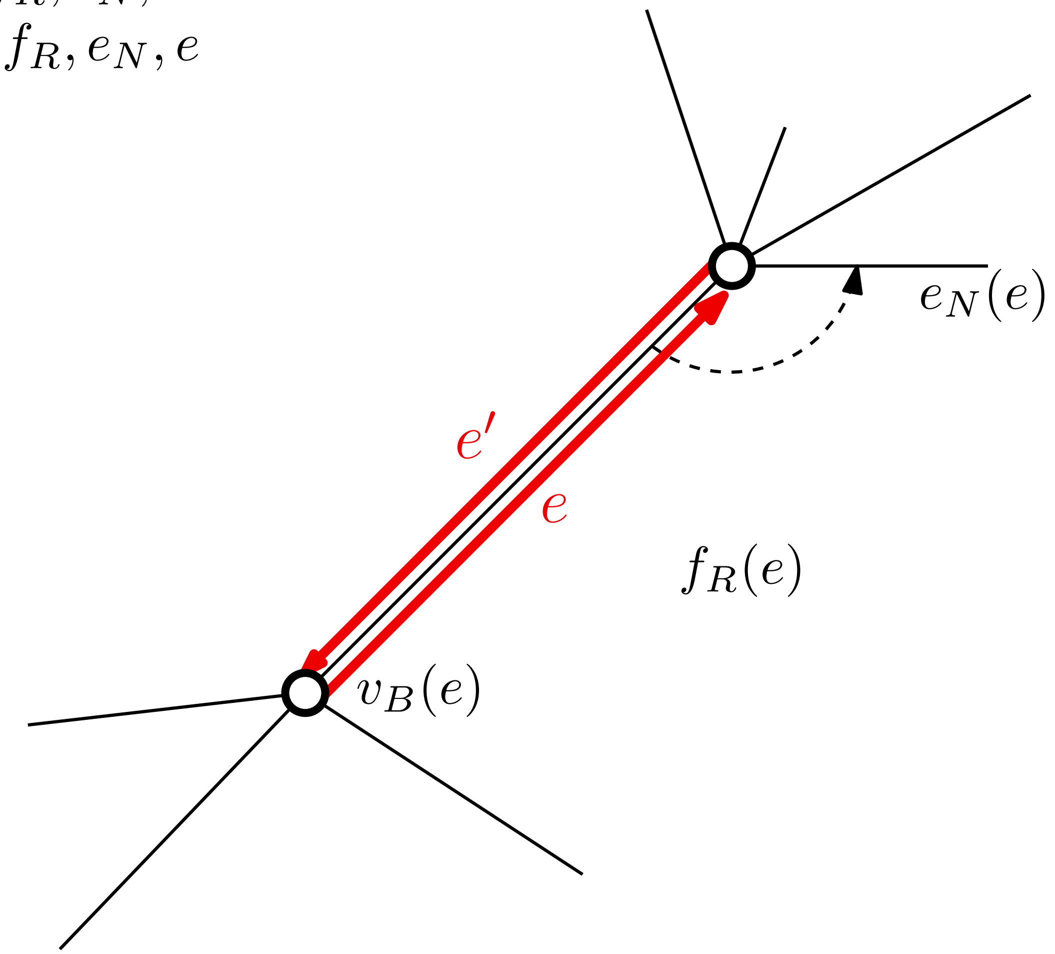


Storing the polygon triangulation

DCEL

There are other DCEL variants, as for example:

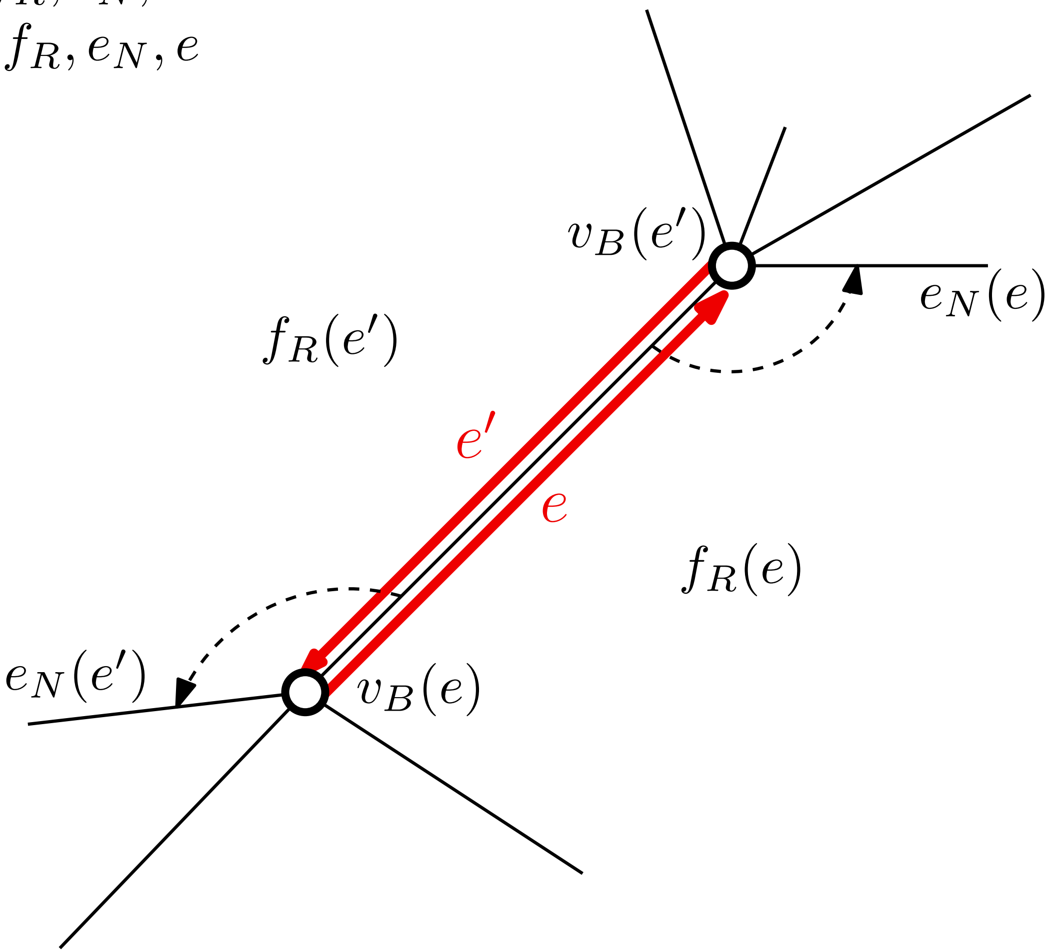
$$e \longrightarrow v_B, f_R, e_N, e'$$
$$e' \longrightarrow v_B, f_R, e_N, e$$



DCEL

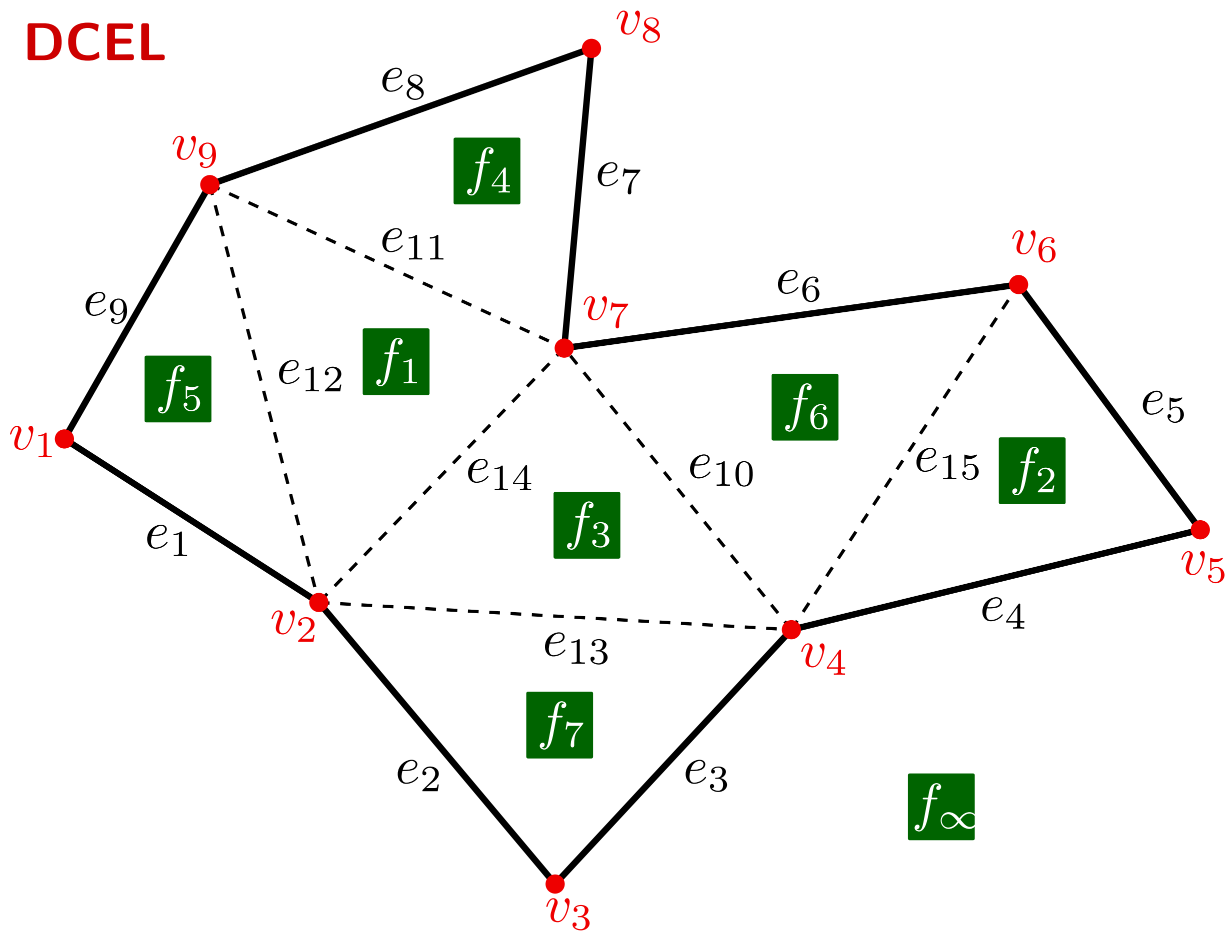
There are other DCEL variants, as for example:

$$e \longrightarrow v_B, f_R, e_N, e'$$
$$e' \longrightarrow v_B, f_R, e_N, e$$



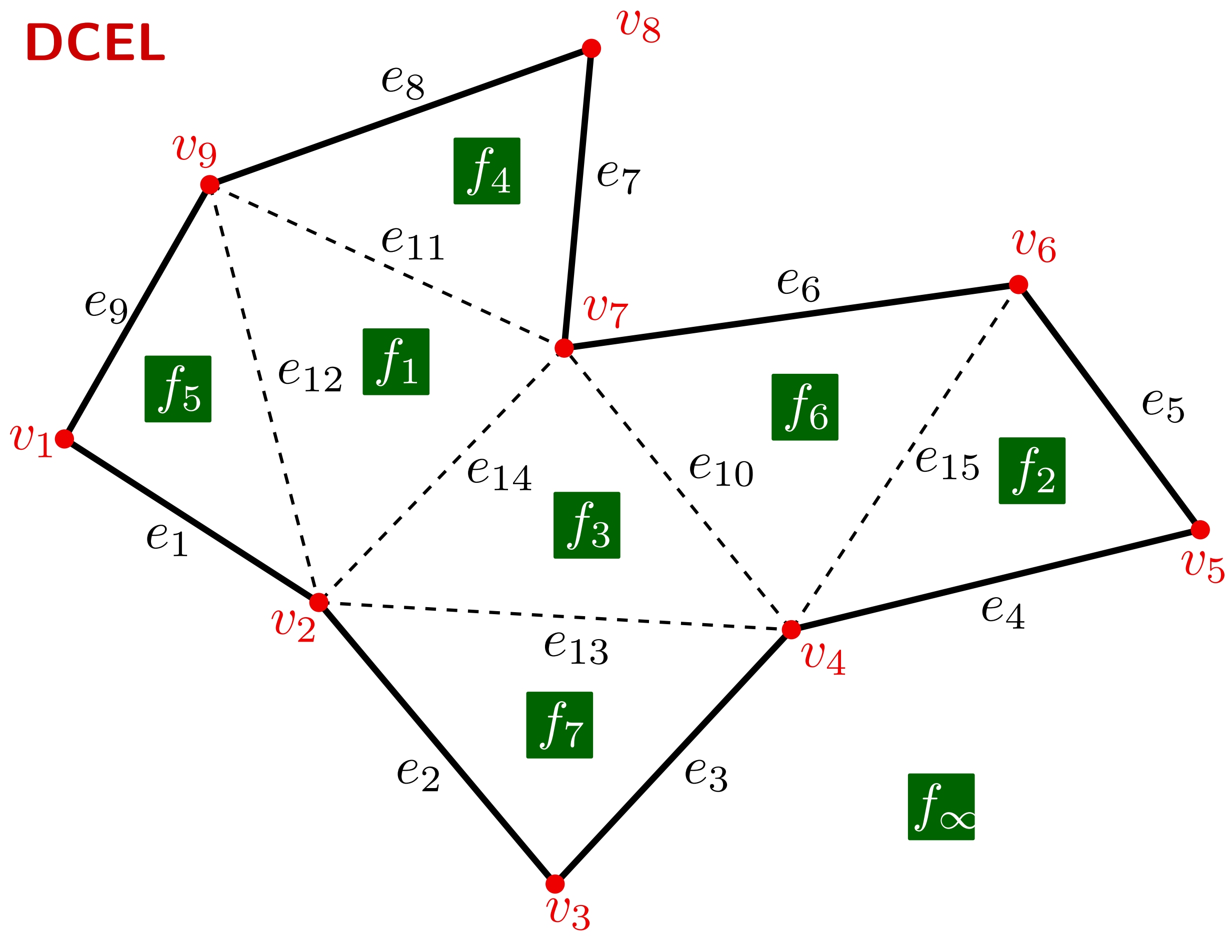
Storing the polygon triangulation

DCEL



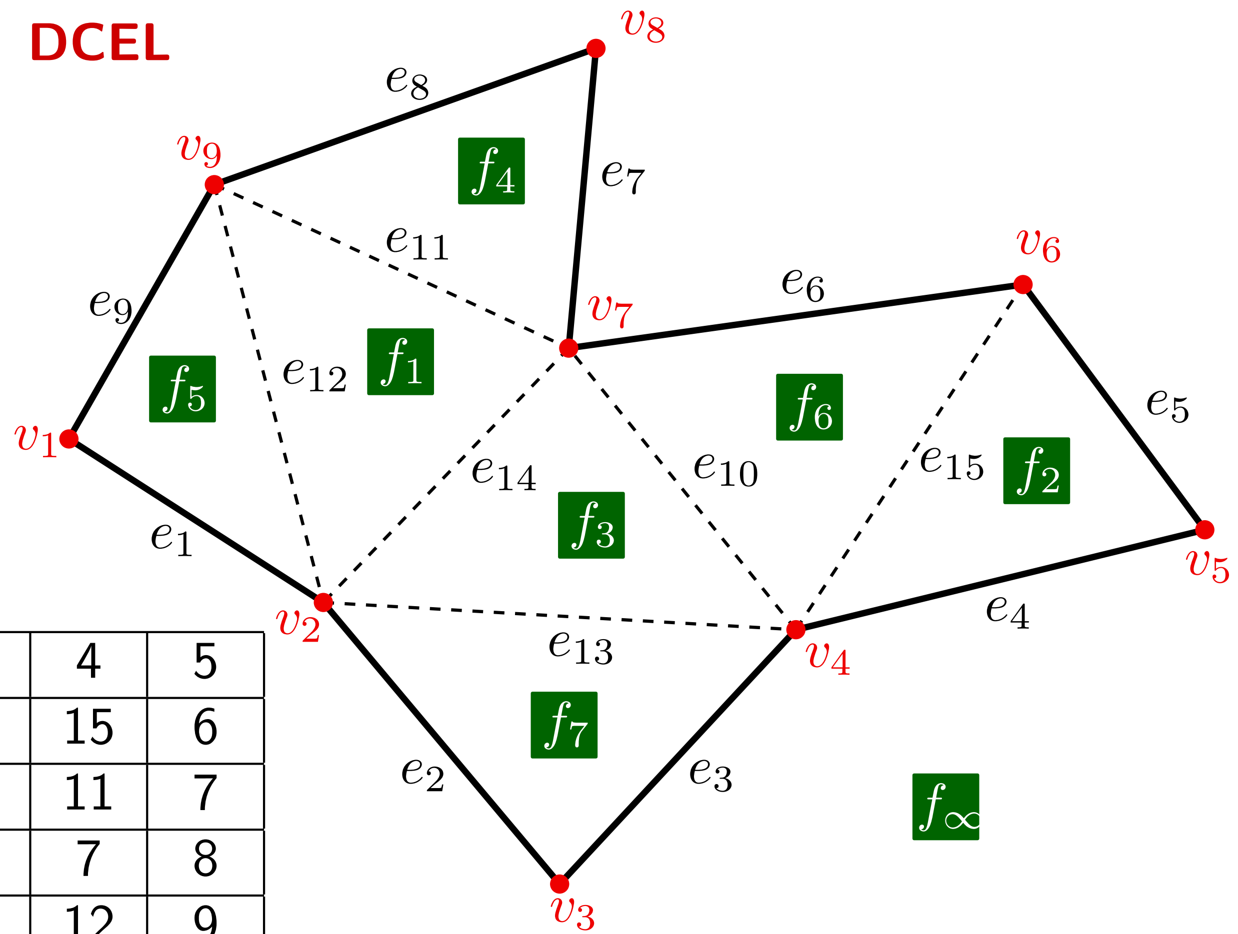
Storing the polygon triangulation

e	v_B	v_E	f_L	f_R	e_P	e_N
1	1	2	5	∞	9	2
2	2	3	7	∞	13	3
3	4	3	∞	7	4	2
4	4	5	2	∞	15	5
5	5	6	2	∞	4	6
6	6	7	6	∞	15	7
7	7	8	4	∞	11	8
8	8	9	4	∞	7	9
9	9	1	5	∞	12	1
10	4	7	3	6	13	6
11	9	7	4	1	8	14
12	2	9	5	1	1	11
13	2	4	3	7	14	3
14	2	7	1	3	12	10
15	4	6	6	2	10	5



Storing the polygon triangulation

e	v_B	f_R	e_N	e'
1	1	∞	2	1'
2	2	∞	3	2'
3	4	7	2	3'
4	4	∞	5	4'
5	5	∞	6	5'
6	6	∞	7	6'
7	7	∞	8	7'
8	8	∞	9	8'
9	9	∞	1	9'
10	4	6	6	10'
11	9	1	14	11'
12	2	1	11	12'
13	2	7	3	13'
14	2	3	10	14'
15	4	2	5	15'
1'	2	5	9	1
2'	3	7	13	2
3'	3	∞	4	3
4'	5	2	15	4
5'	6	2	4	5
6'	7	6	15	6
7'	8	4	11	7
8'	9	4	7	8
9'	1	5	12	9
10'	7	3	13	10
11'	7	4	8	11
12'	9	5	1	12
13'	4	3	14	13
14'	7	1	12	14
15'	6	6	10	15



Storing the polygon triangulation

How to build the DCEL

Storing the polygon triangulation

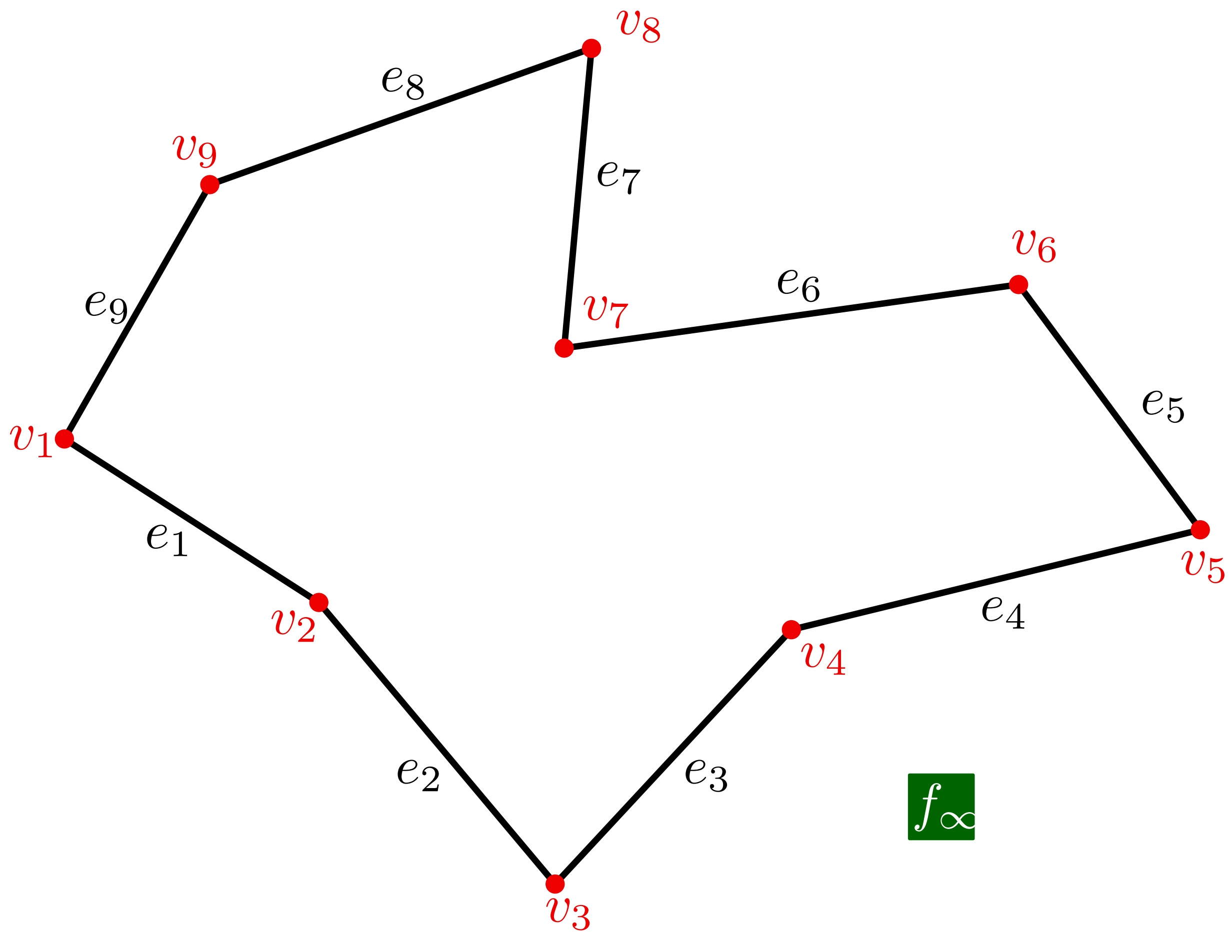
How to build the DCEL

Algorithm 1: substracting ears

Storing the polygon triangulation

How to build the DCEL

Algorithm 1: substracting ears

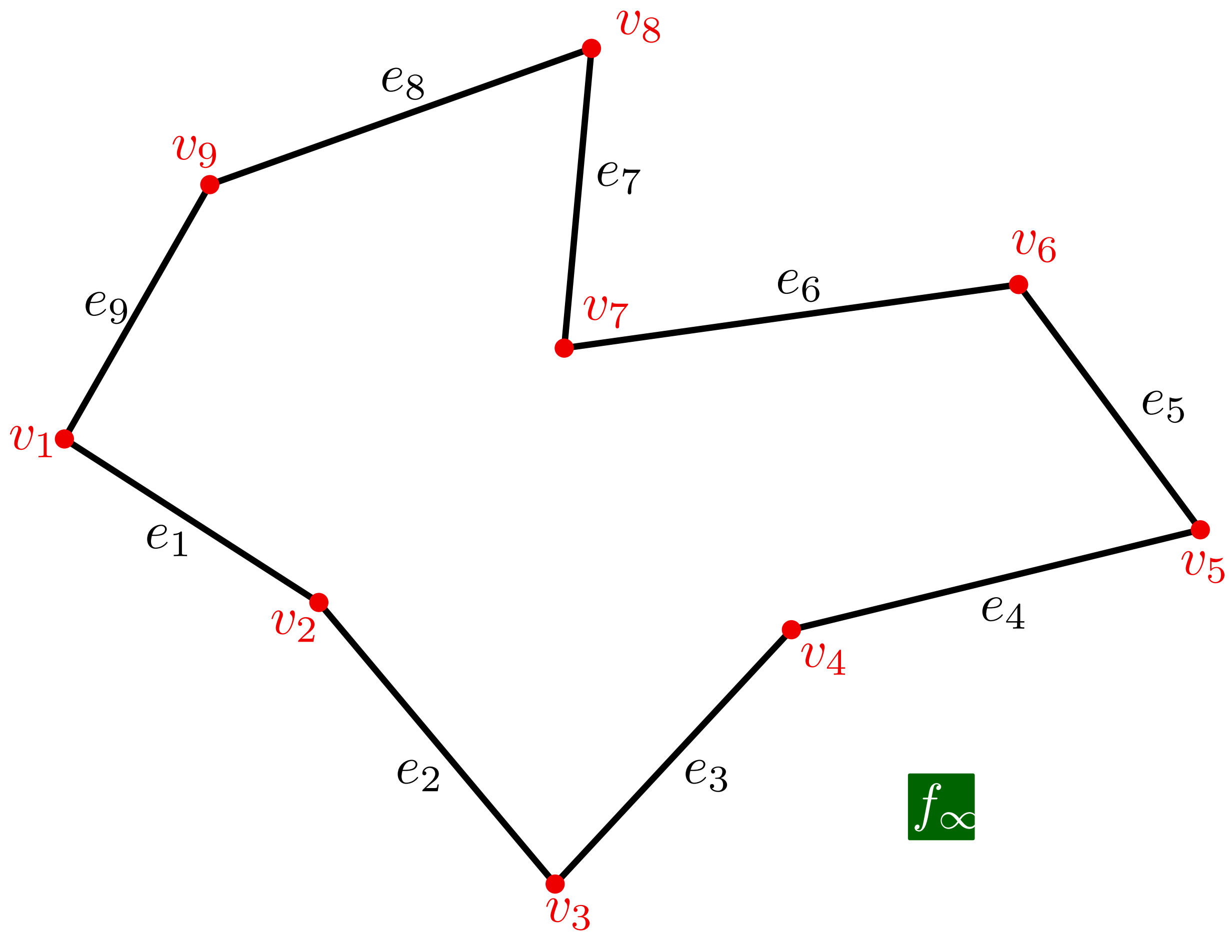


Storing the polygon triangulation

How to build the DCEL

Algorithm 1: substracting ears

Initialize



Storing the polygon triangulation

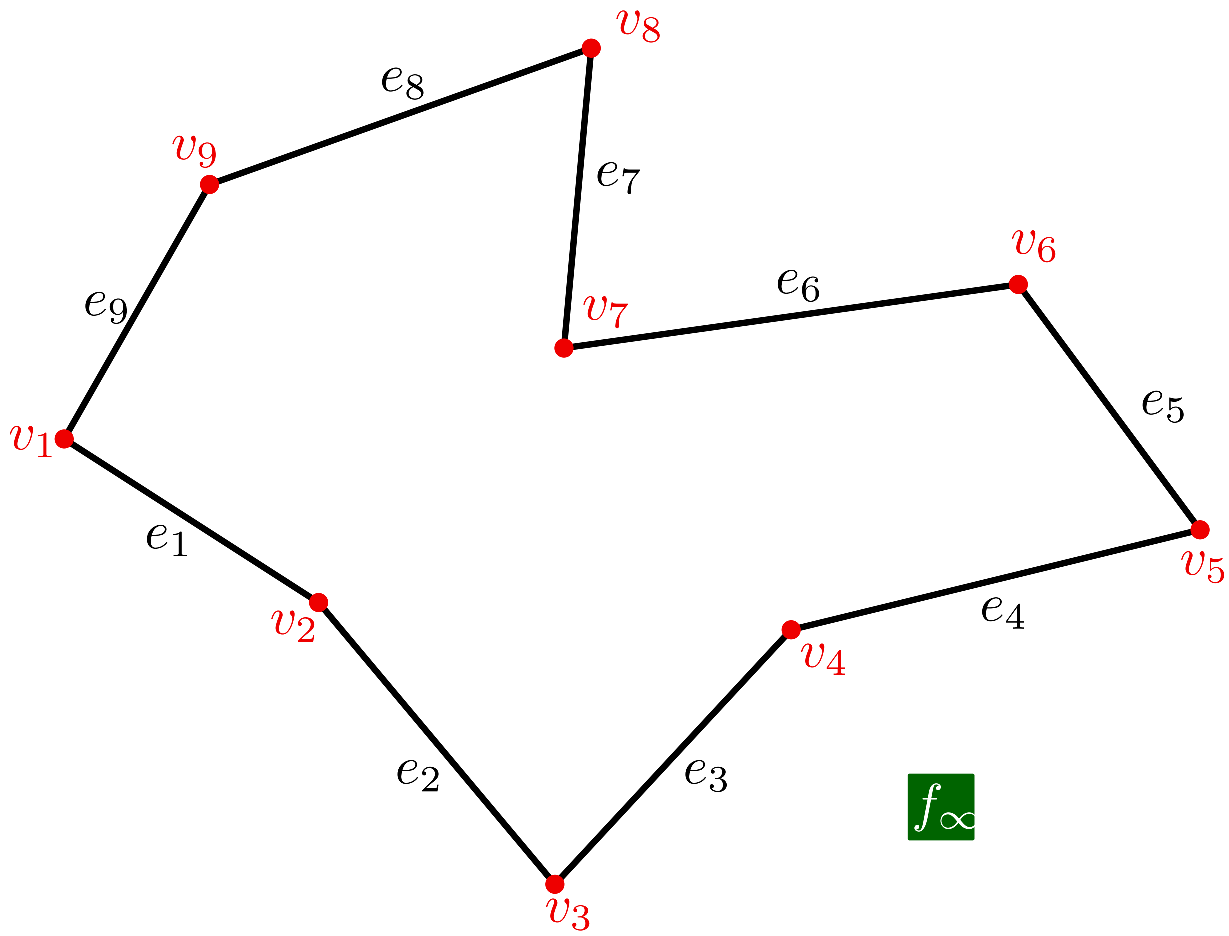
How to build the DCEL

Algorithm 1: substracting ears

Initialize

Table of vertices

v	x	y	e
1	x_1	y_1	1
2	x_2	y_2	2
3	x_3	y_3	3
4	x_4	y_4	4
5	x_5	y_5	5
6	x_6	y_6	6
7	x_7	y_7	7
8	x_8	y_8	8
9	x_9	y_9	9



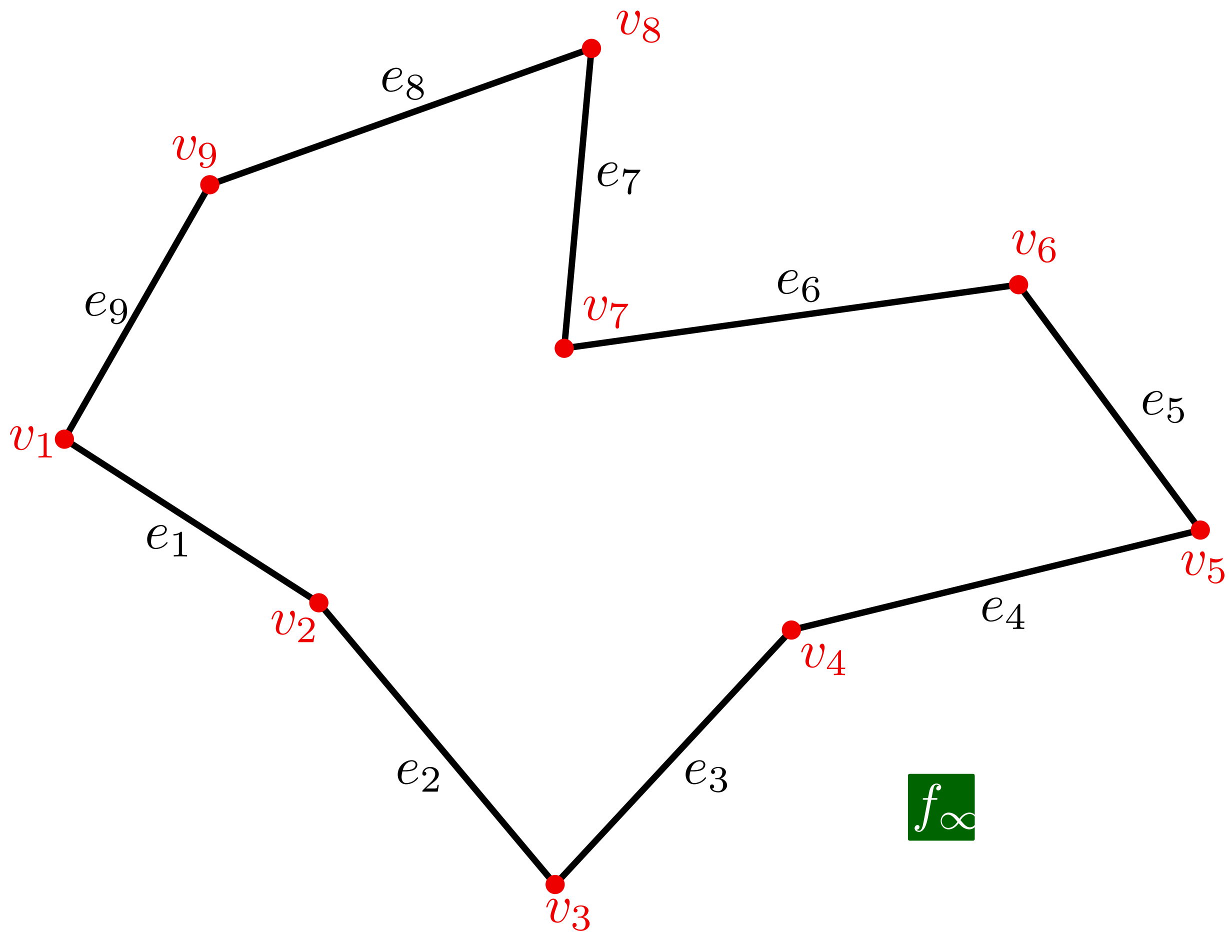
How to build the DCEL

Algorithm 1: substracting ears

Initialize

Table of faces

f	e
∞	9



Storing the polygon triangulation

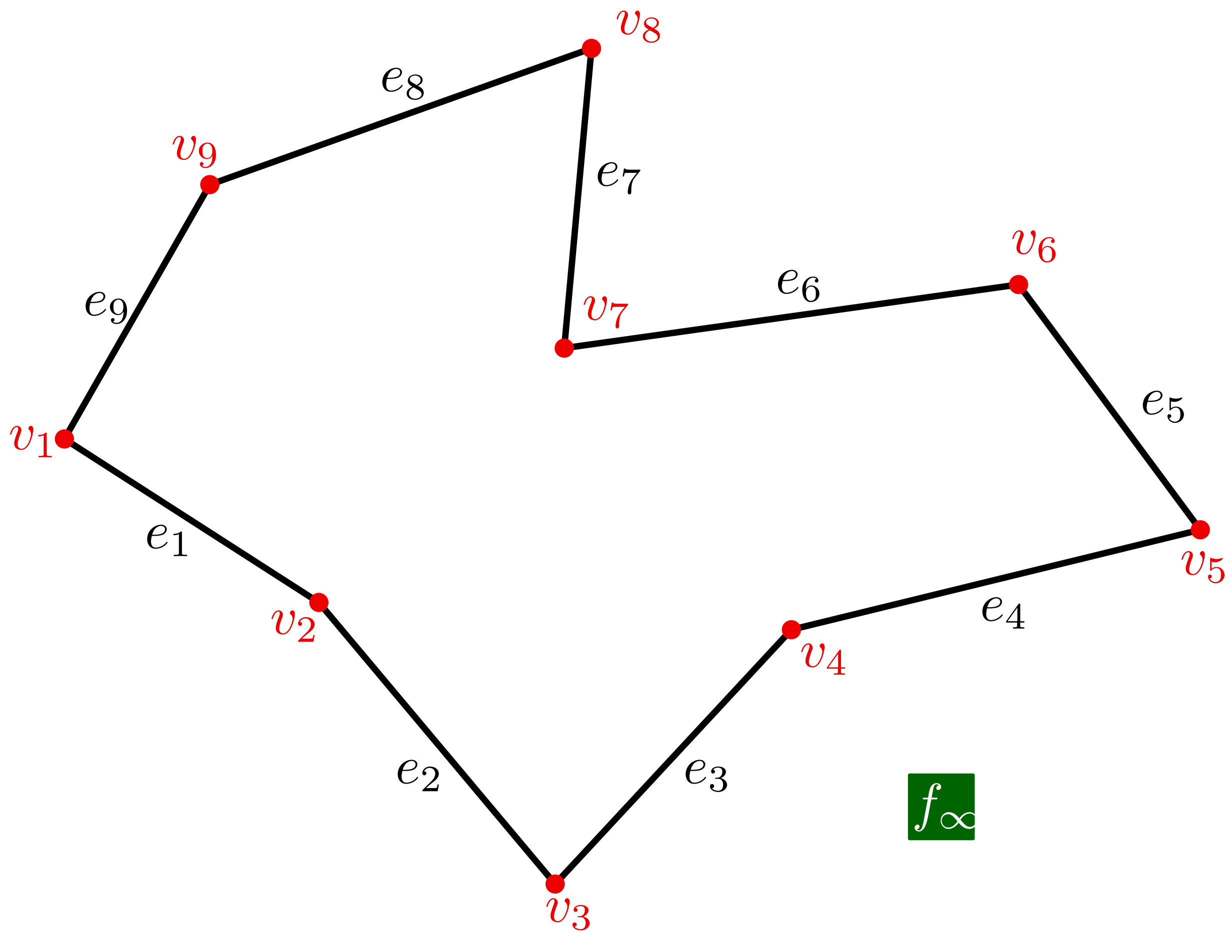
How to build the DCEL

Algorithm 1: substracting ears

Initialize

DCEL

e	v_B	v_E	f_L	f_R	e_P	e_N
1	1	2		∞		2
2	2	3		∞		3
3	3	4		∞		4
4	4	5		∞		5
5	5	6		∞		6
6	6	7		∞		7
7	7	8		∞		8
8	8	9		∞		9
9	9	1		∞		1



Storing the polygon triangulation

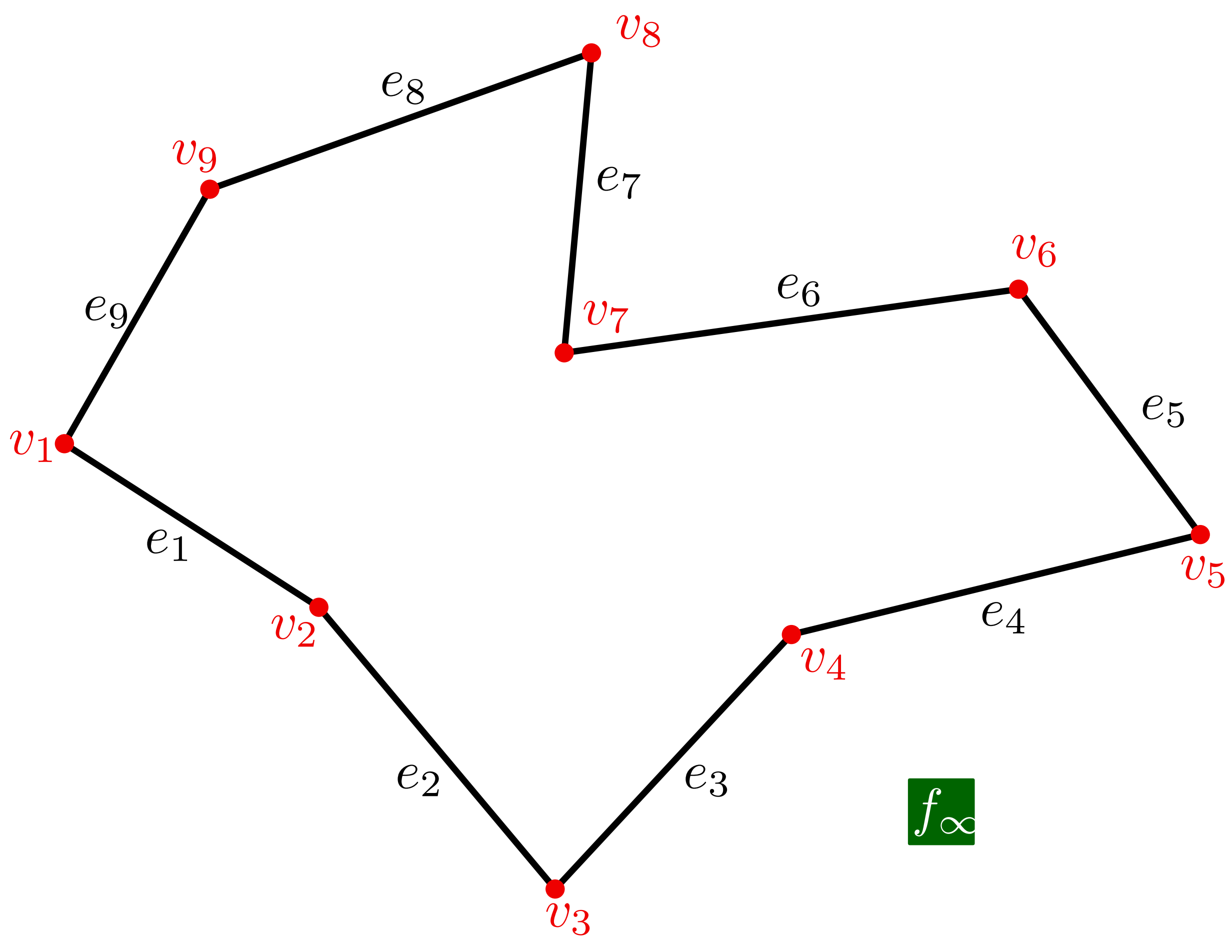
How to build the DCEL

Algorithm 1: substracting ears

Initialize

DCEL

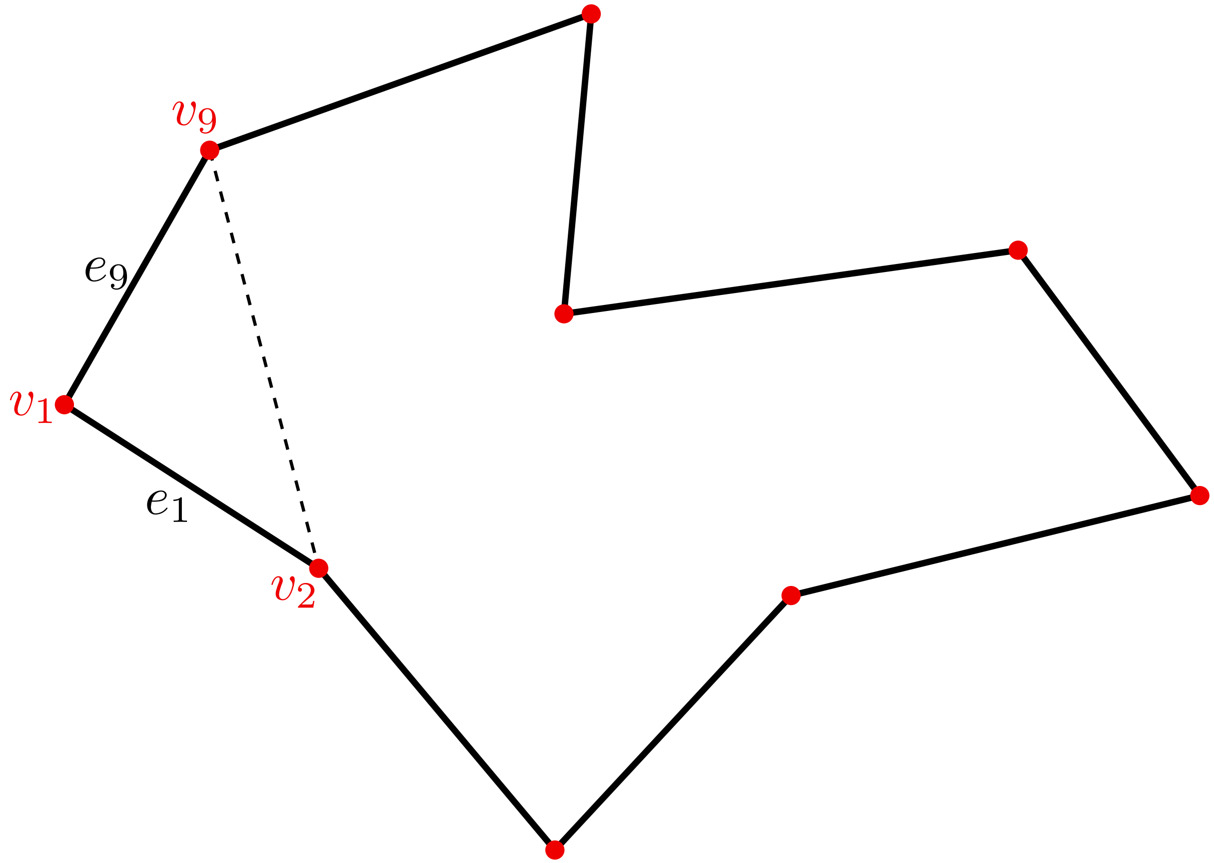
e	v_B	v_E	f_L	f_R	e_P	e_N
1	1	2		∞		2
2	2	3		∞		3
3	4	3	∞		4	
4	4	5		∞		5
5	5	6		∞		6
6	6	7		∞		7
7	7	8		∞		8
8	8	9		∞		9
9	9	1		∞		1



How to build the DCEL

Algorithm 1: substracting ears

Advance

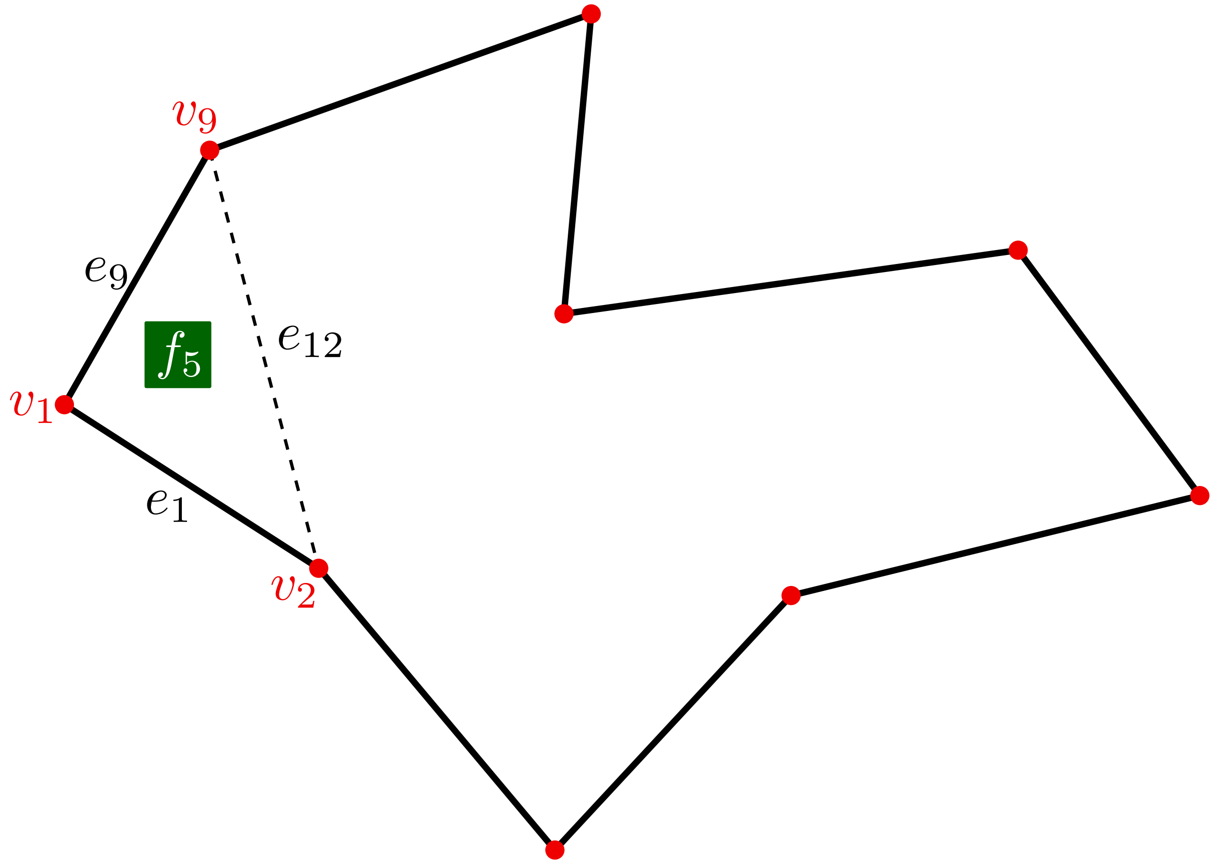


Storing the polygon triangulation

How to build the DCEL

Algorithm 1: substracting ears

Advance



Storing the polygon triangulation

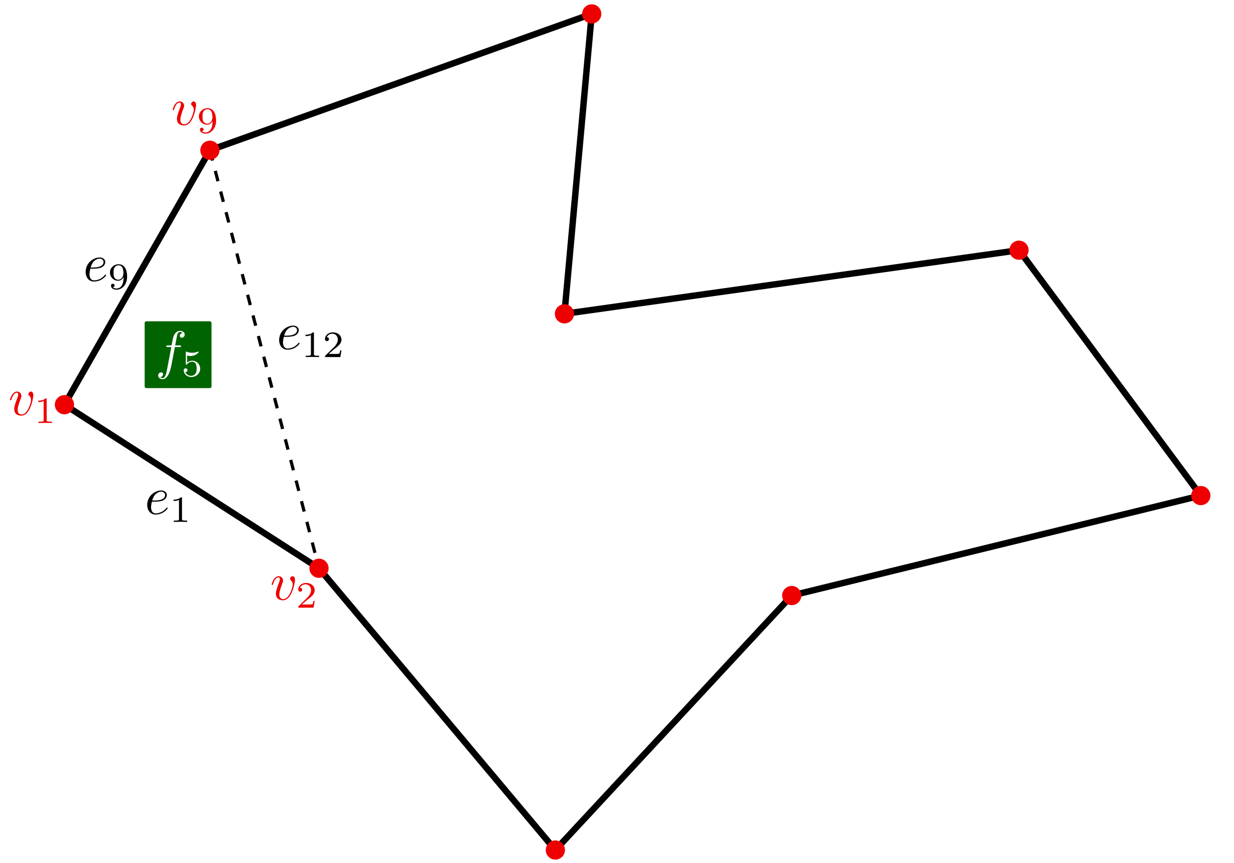
How to build the DCEL

Algorithm 1: substracting ears

Advance

Table of faces

f	e
5	9



How to build the DCEL

Algorithm 1: substracting ears

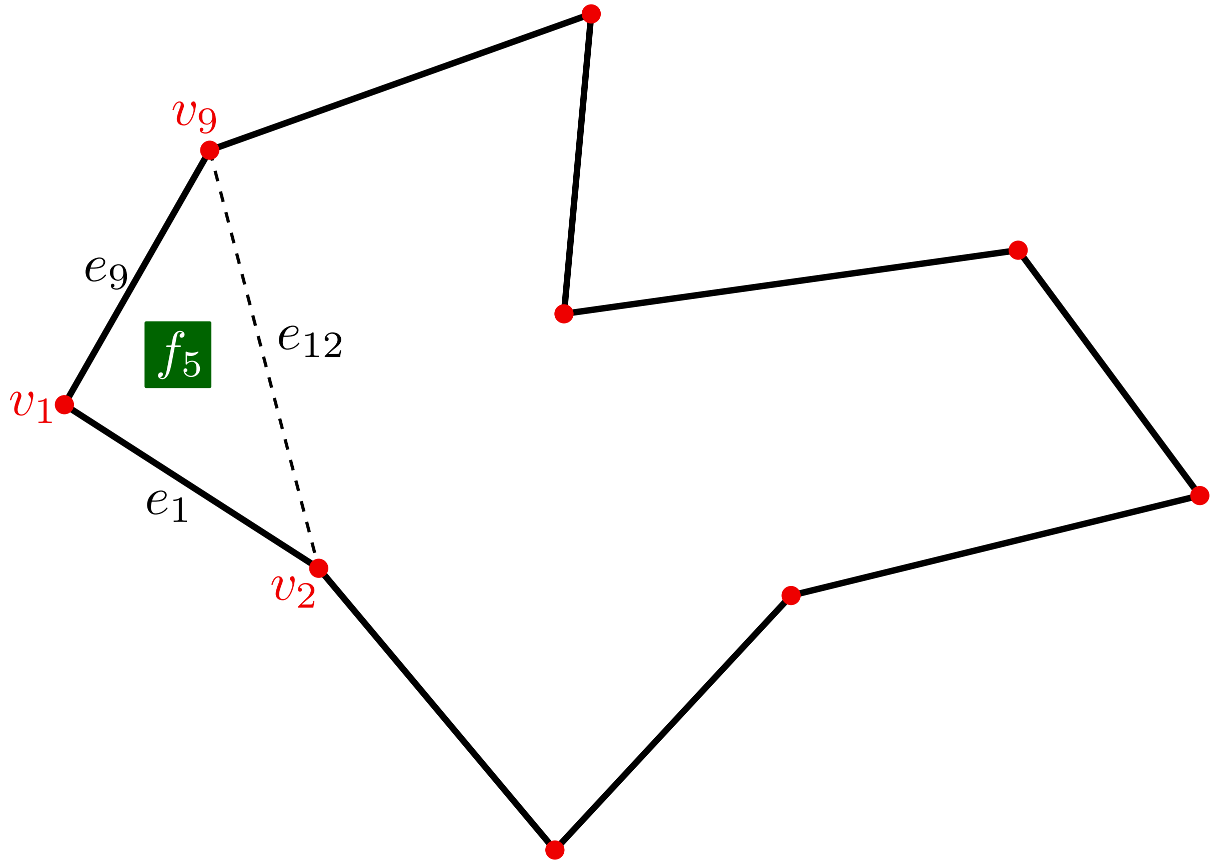
Advance

Table of faces

f	e
5	9

DCEL

e	v_B	v_E	f_L	f_R	e_P	e_N
1	1	2	5	∞	9	2
9	9	1	5	∞	12	1
12	2	9	5		1	



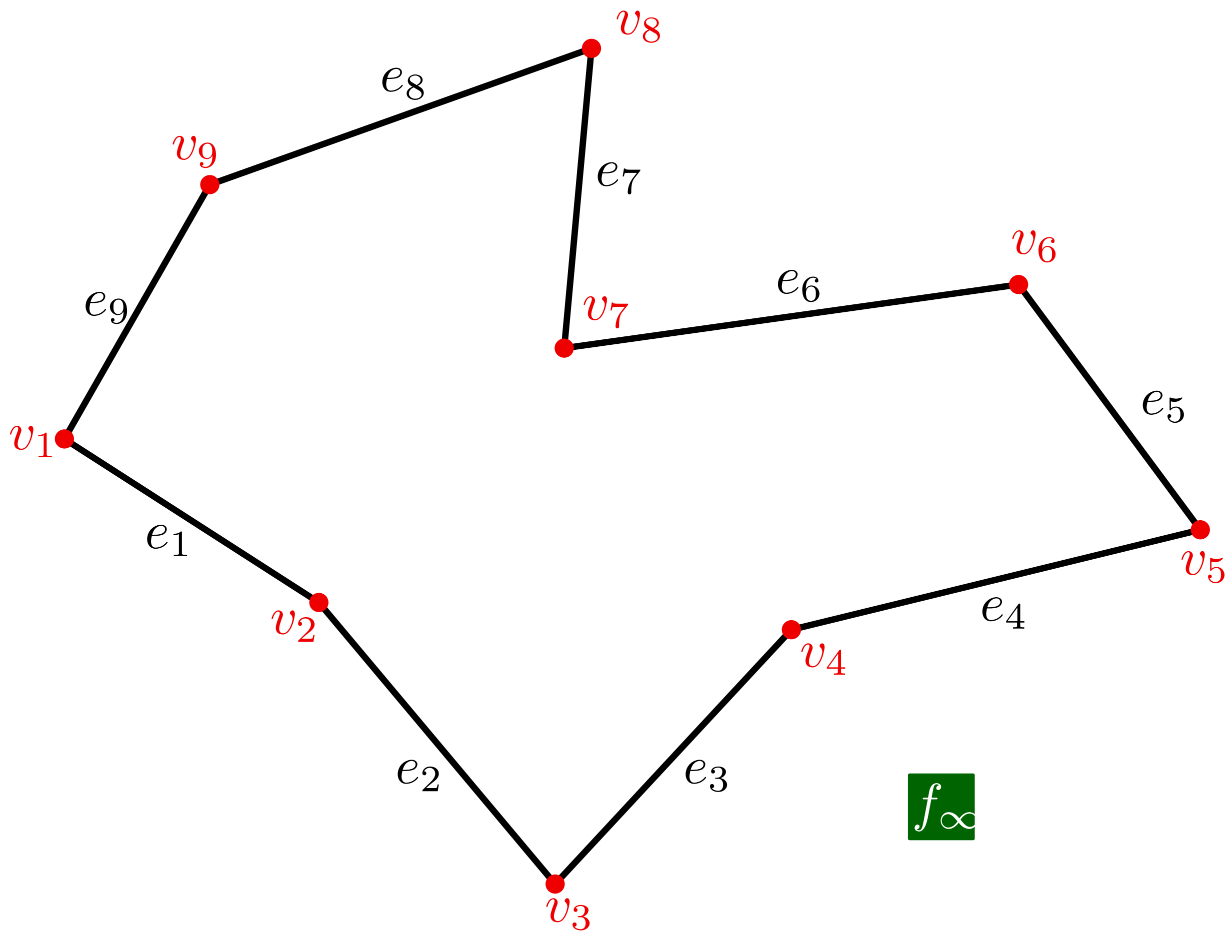
Storing the polygon triangulation

How to build the DCEL

Algorithm 2: inserting diagonals

How to build the DCEL

Algorithm 2: inserting diagonals

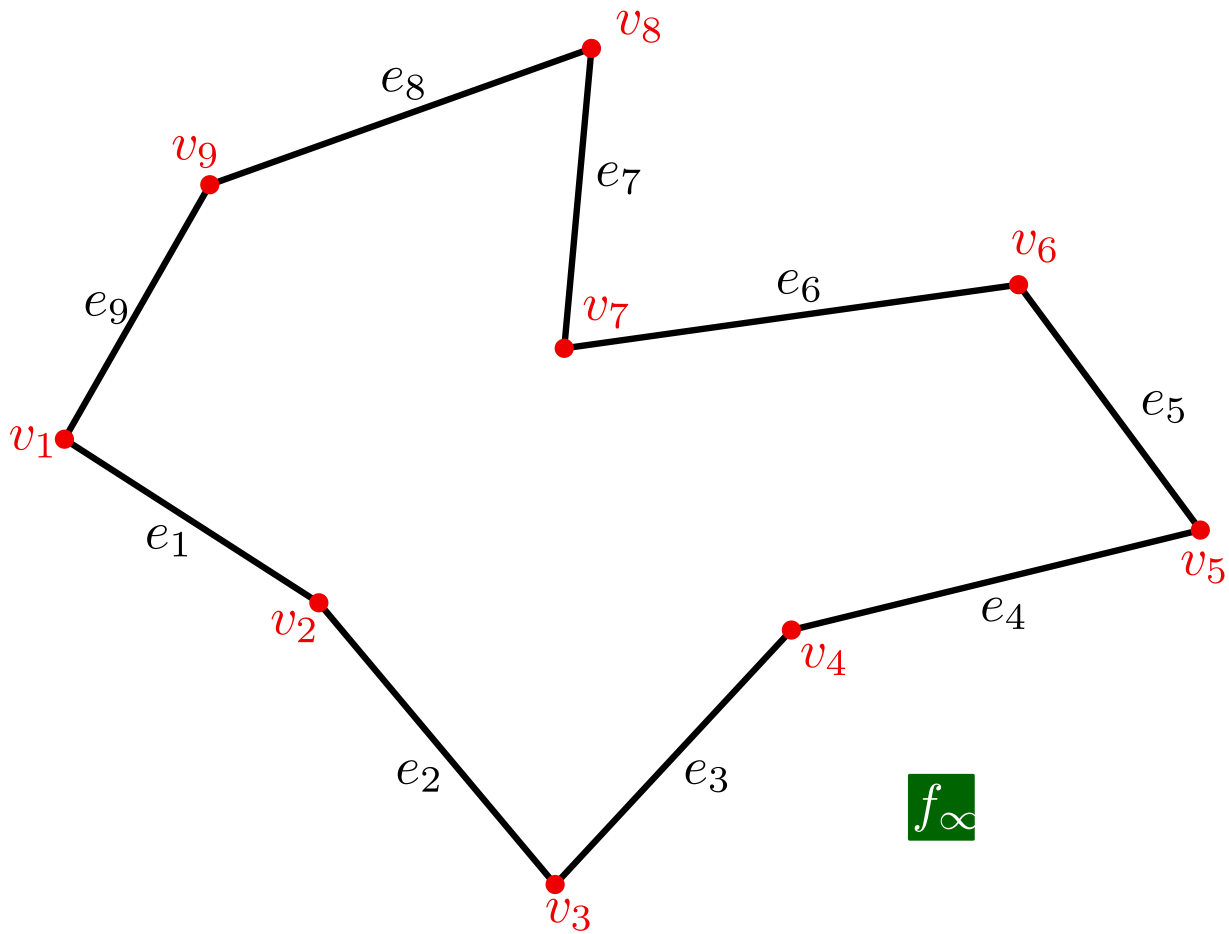


Storing the polygon triangulation

How to build the DCEL

Algorithm 2: inserting diagonals

Initialize



Storing the polygon triangulation

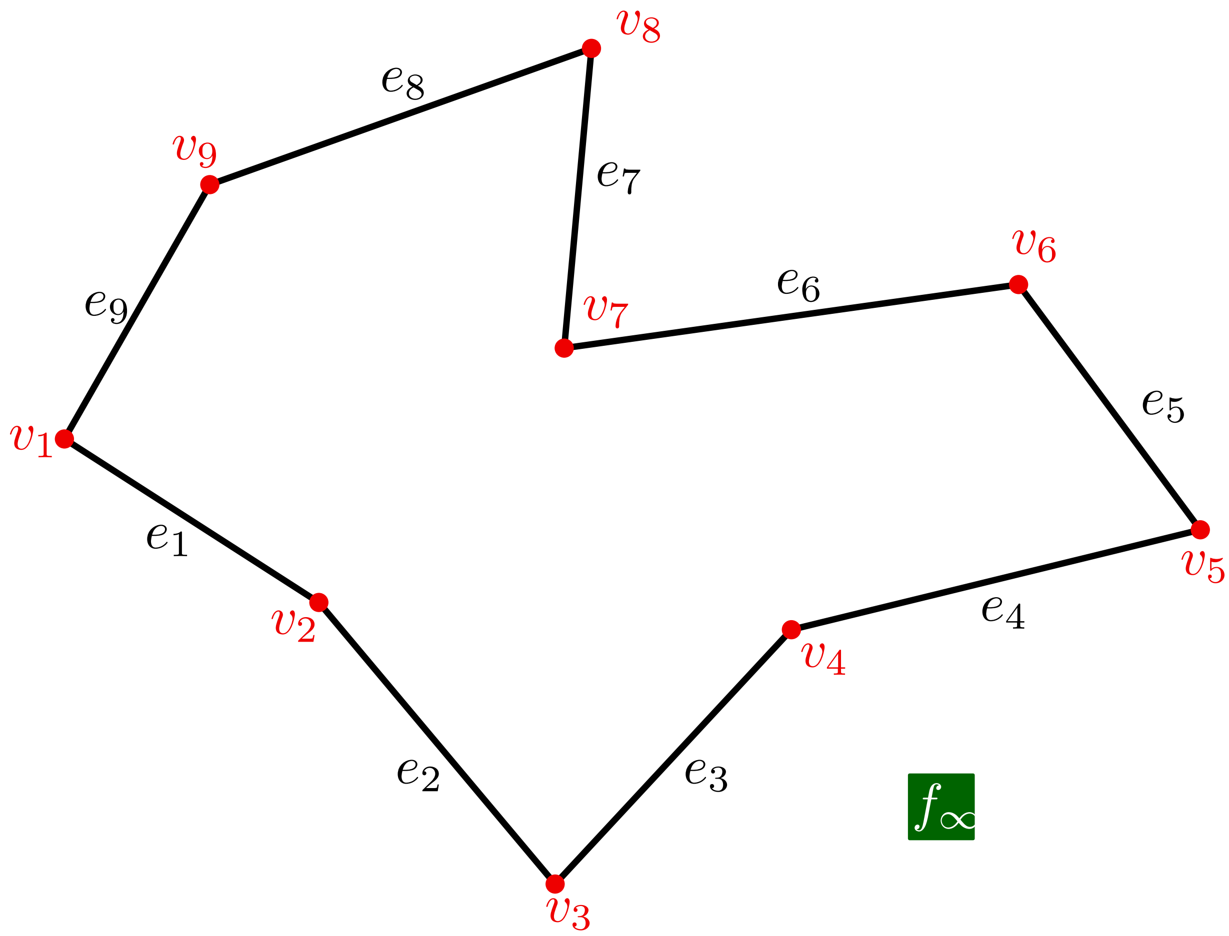
How to build the DCEL

Algorithm 2: inserting diagonals

Initialize

Table of vertices

v	x	y	e
1	x_1	y_1	1
2	x_2	y_2	2
3	x_3	y_3	3
4	x_4	y_4	4
5	x_5	y_5	5
6	x_6	y_6	6
7	x_7	y_7	7
8	x_8	y_8	8
9	x_9	y_9	9



Storing the polygon triangulation

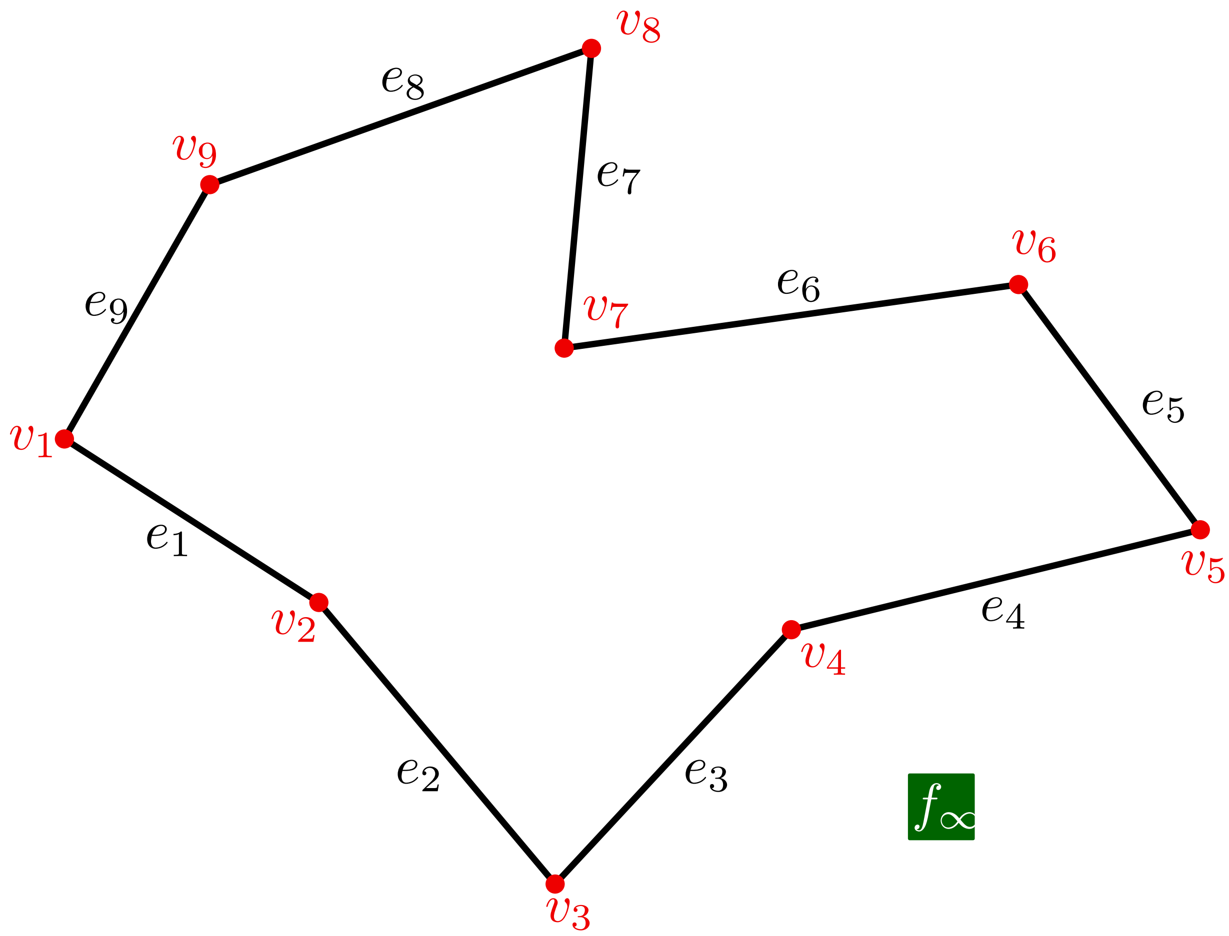
How to build the DCEL

Algorithm 2: inserting diagonals

Initialize

Table of faces

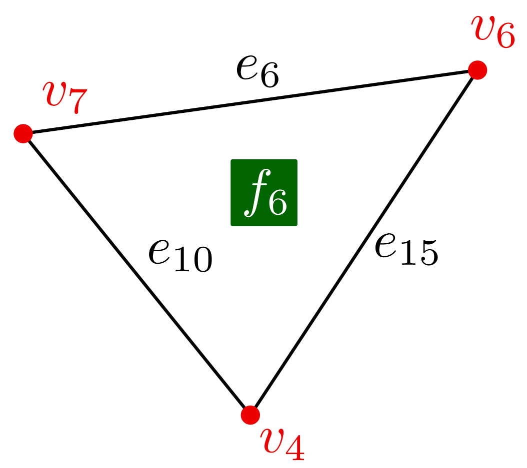
f	e
∞	9



How to build the DCEL

Algorithm 2: inserting diagonals

Base step



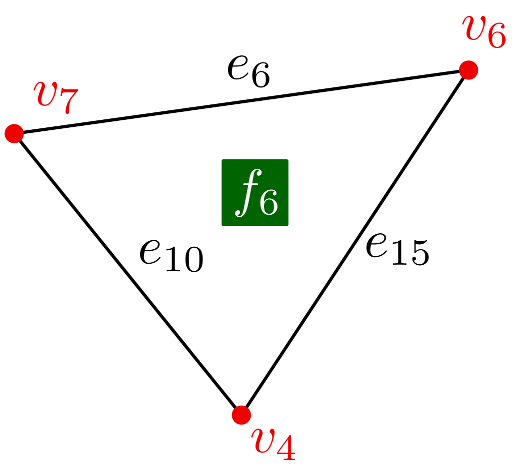
How to build the DCEL

Algorithm 2: inserting diagonals

Base step

Table of faces

f	e
6	10



How to build the DCEL

Algorithm 2: inserting diagonals

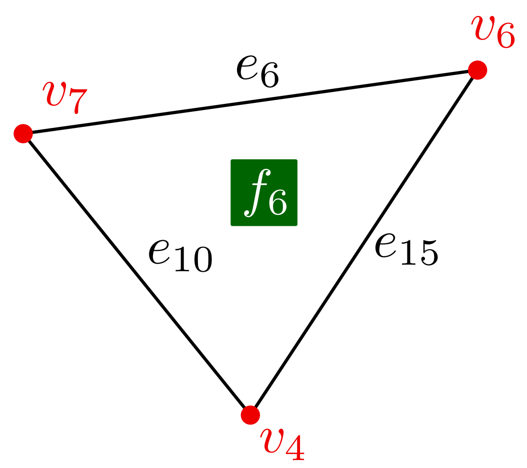
Base step

Table of faces

f	e
6	10

DCEL

e	v_B	v_E	f_L	f_R	e_P	e_N
6	6	7	6	∞	15	10
10	7	4	6	∞	6	15
15	4	6	6	∞	10	6

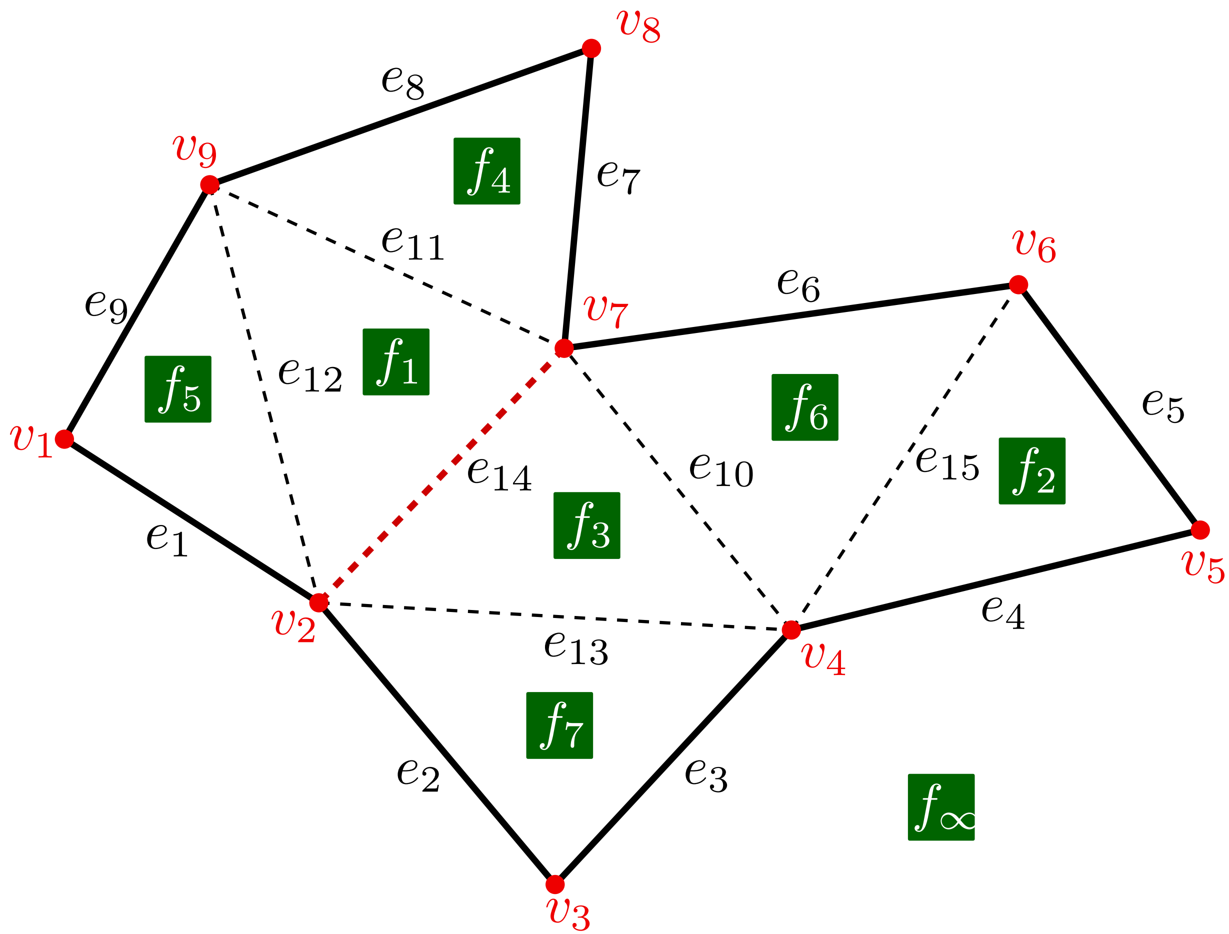


Storing the polygon triangulation

How to build the DCEL

Algorithm 2: inserting diagonals

Merge step



How to build the DCEL

Algorithm 2: inserting diagonals

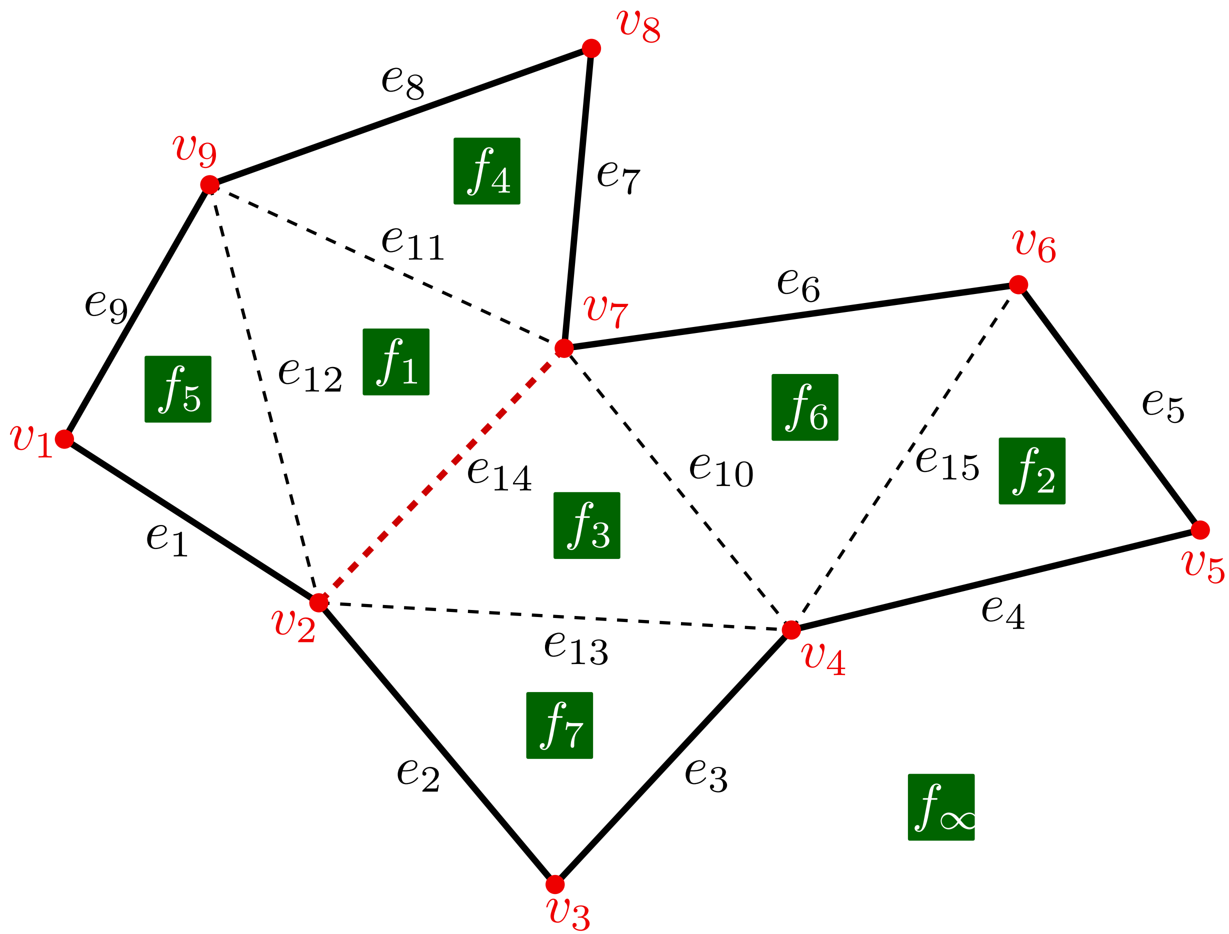
Merge step

DCEL 1

e	v_B	v_E	f_L	f_R	e_P	e_N
1	1	2	5	∞	9	14
14	2	7	1	∞	12	7

DCEL 2

e	v_B	v_E	f_L	f_R	e_P	e_N
6	6	7	6	∞	15	14
14	2	7	∞	3	2	10



Storing the polygon triangulation

How to build the DCEL

Algorithm 2: inserting diagonals

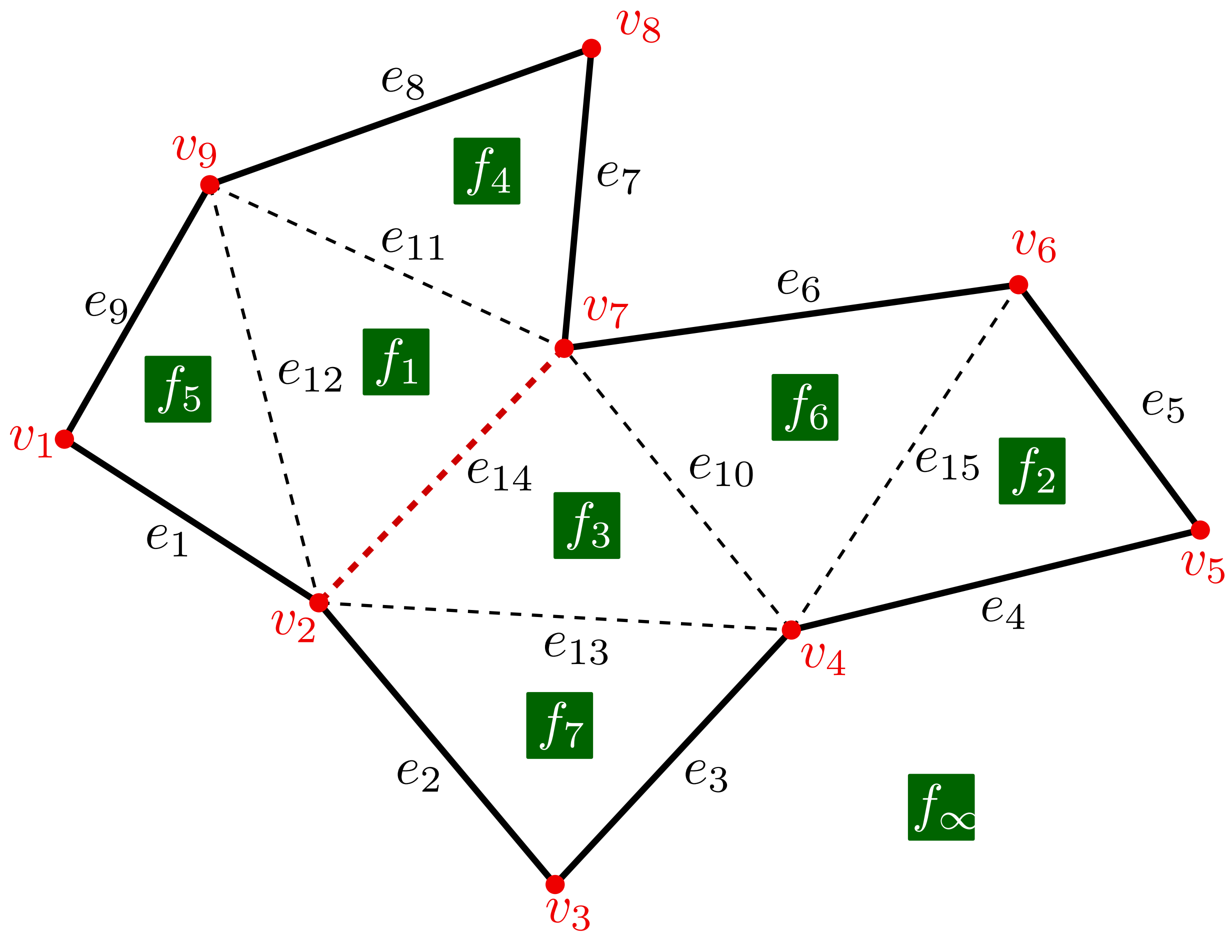
Merge step

DCEL 1

e	v_B	v_E	f_L	f_R	e_P	e_N
1	1	2	5	∞	9	~~14~~
14	2	7	1	~~∞~~	12	~~7~~

DCEL 2

e	v_B	v_E	f_L	f_R	e_P	e_N
6	6	7	6	∞	15	~~14~~
14	2	7	~~∞~~	3	~~2~~	10



How to build the DCEL

Algorithm 2: inserting diagonals

Merge step

DCEL 1

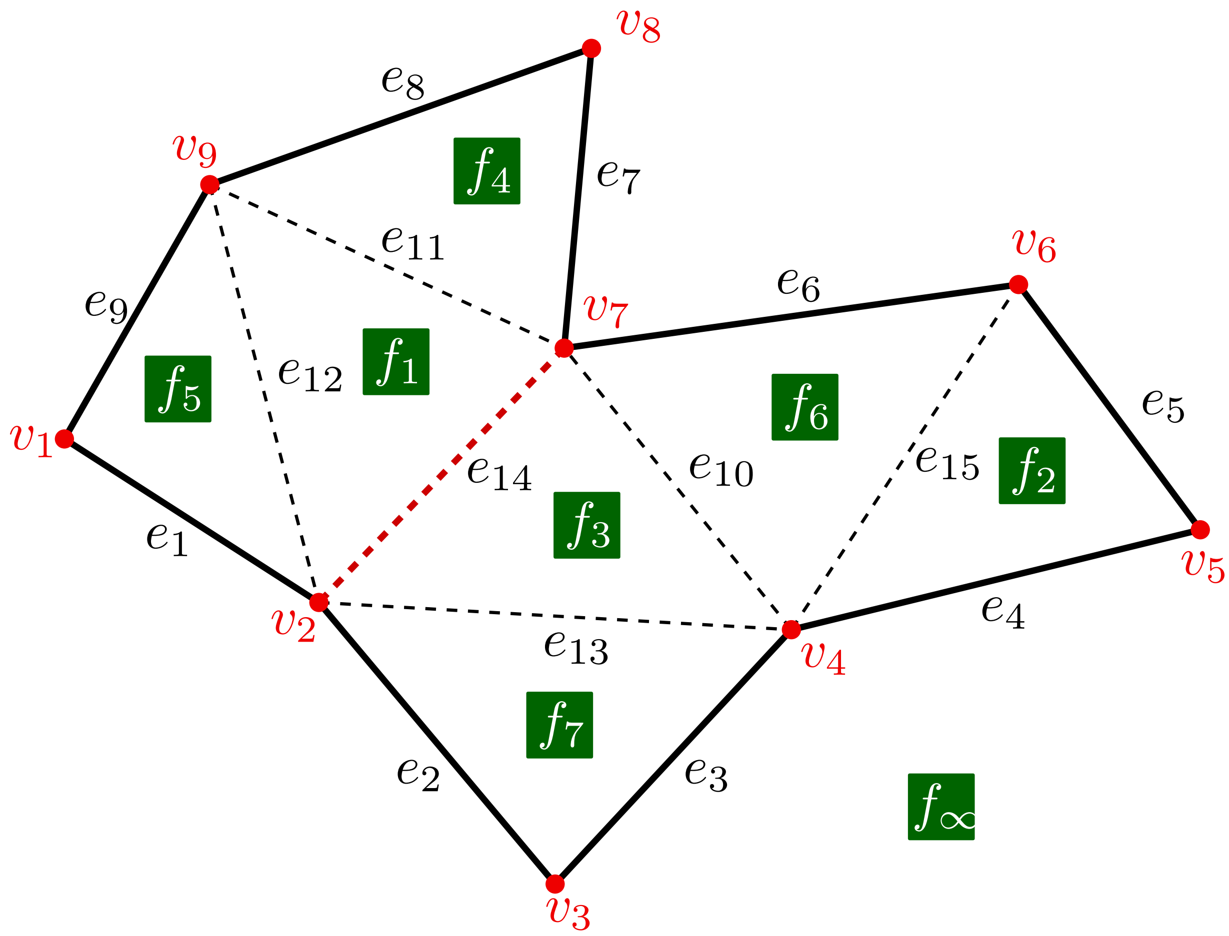
e	v_B	v_E	f_L	f_R	e_P	e_N
1	1	2	5	∞	9	~~14~~
14	2	7	1	~~∞~~	12	~~7~~

DCEL 2

e	v_B	v_E	f_L	f_R	e_P	e_N
6	6	7	6	∞	15	~~14~~
14	2	7	~~∞~~	3	~~2~~	10

Merged DCEL

e	v_B	v_E	f_L	f_R	e_P	e_N
1	1	2	5	∞	9	2
6	6	7	6	∞	15	7
14	2	7	1	3	12	10



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

Storing the polygon triangulation

How to build the DCEL

Algorithm 3:

1. Decompose into monotone polygons
2. Triangulate monotone pieces

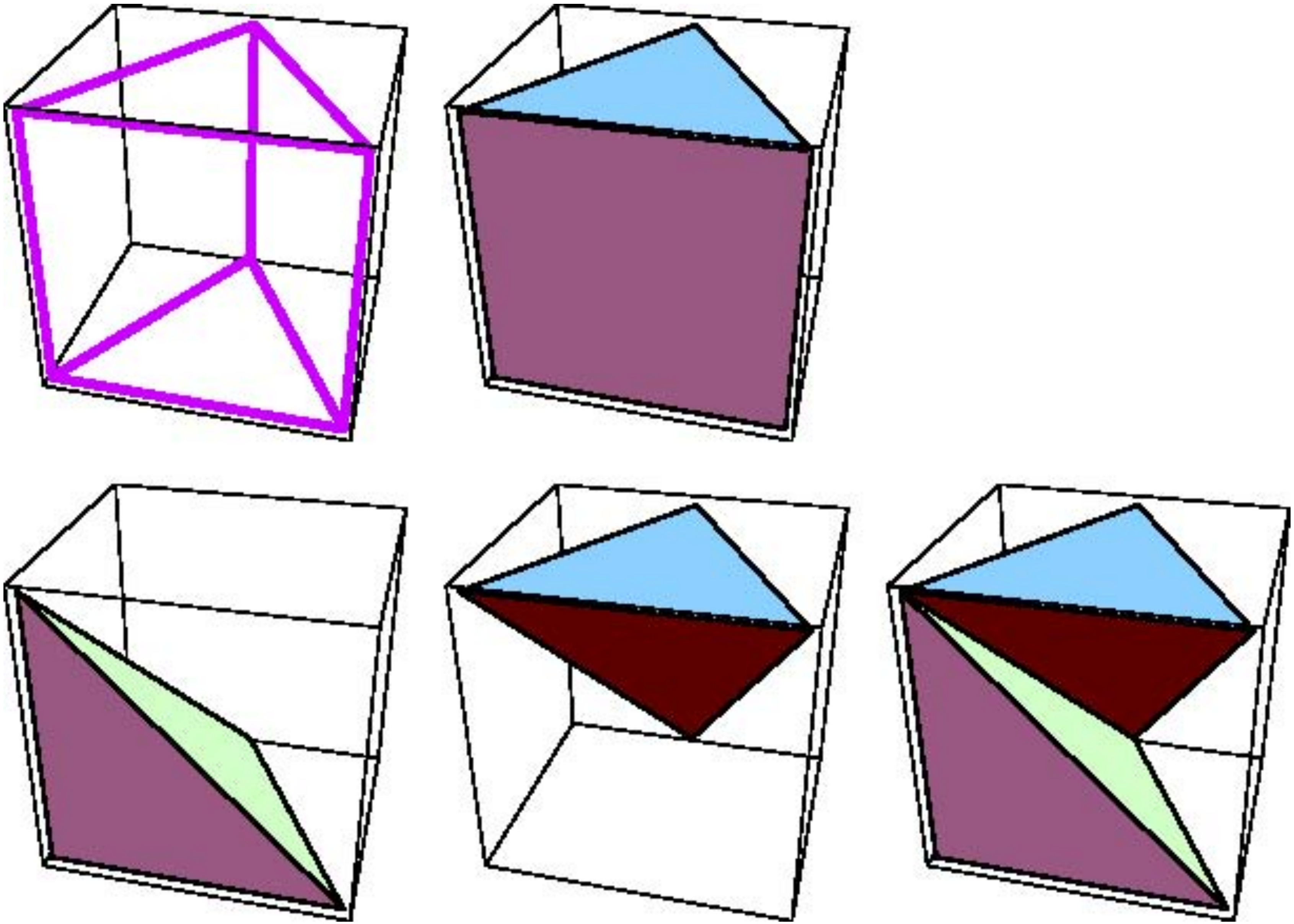
Computing the DCEL is done by combining previous strategies:

- Separating ears for triangulating each monotone subpolygon.
- Merging DCELs for putting together the monotone pieces.

WHAT HAPPENS IN 3D?

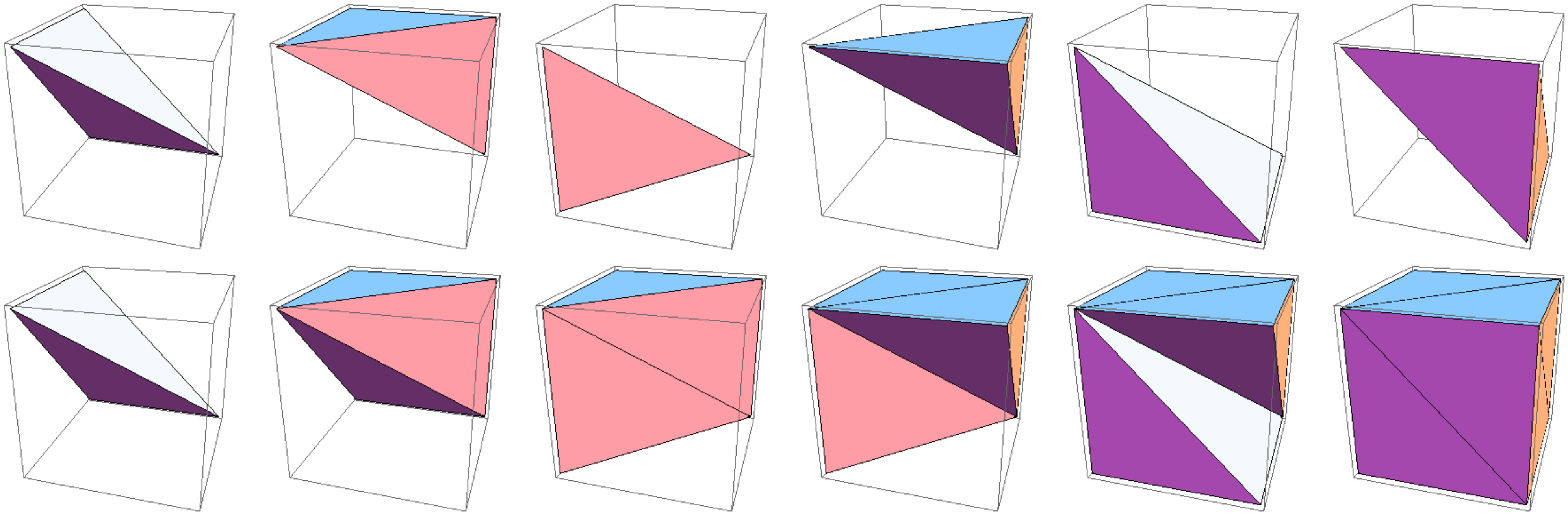
WHAT HAPPENS IN 3D?

A polyhedron that can be *tetrahedralized*:



WHAT HAPPENS IN 3D?

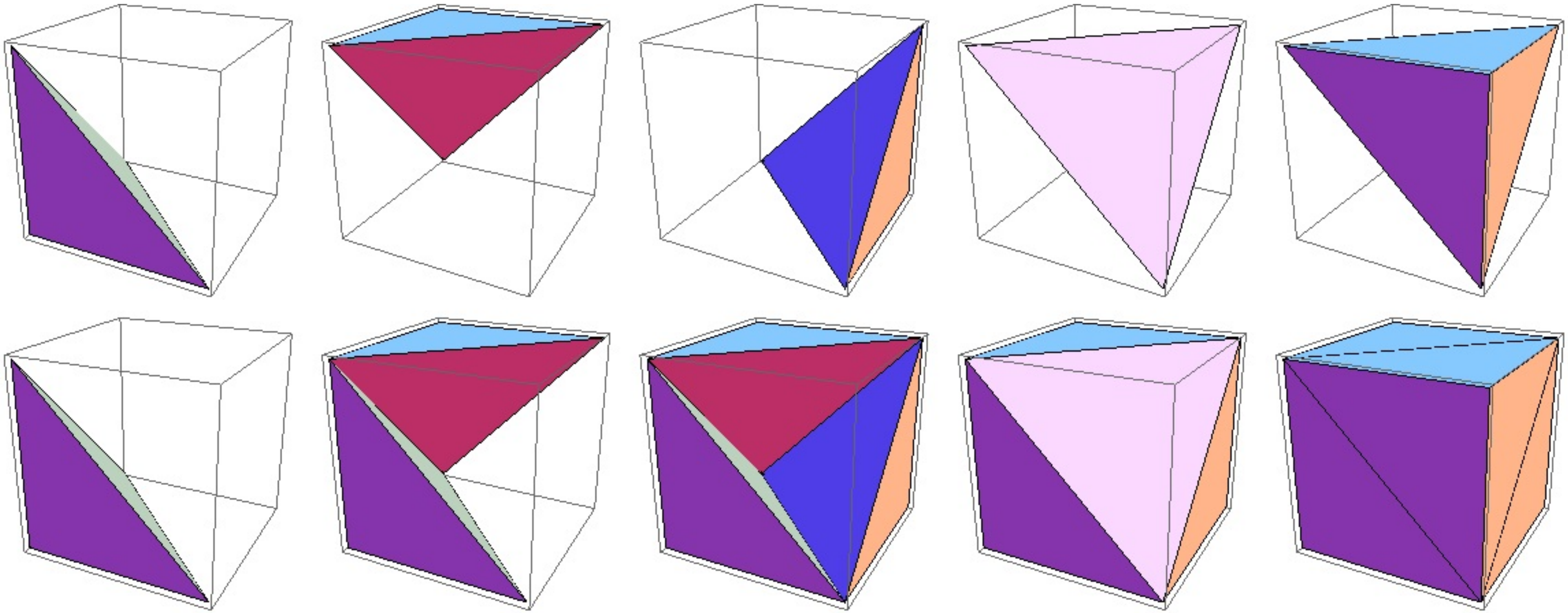
A cube can be decomposed into 6 tetrahedra...



WHAT HAPPENS IN 3D?

A cube can be decomposed into 6 tetrahedra...

but also into 5!



A polyhedron that cannot be tetrahedralized:

Schönhardt polyhedron

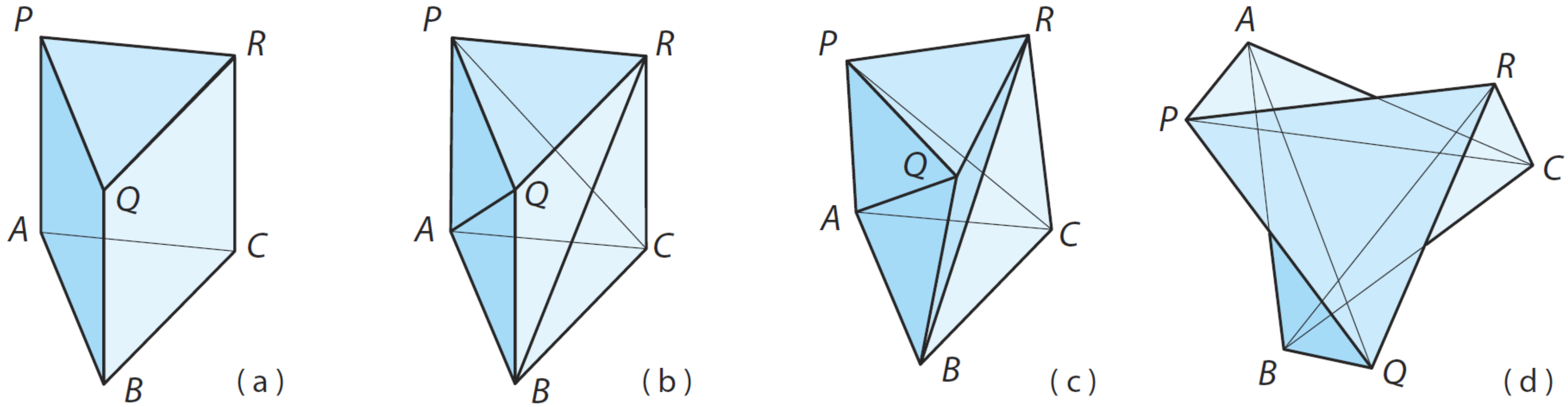


Figure from the book by Devadoss and O'Rourke

Smallest polyhedron that cannot be tetrahedralized

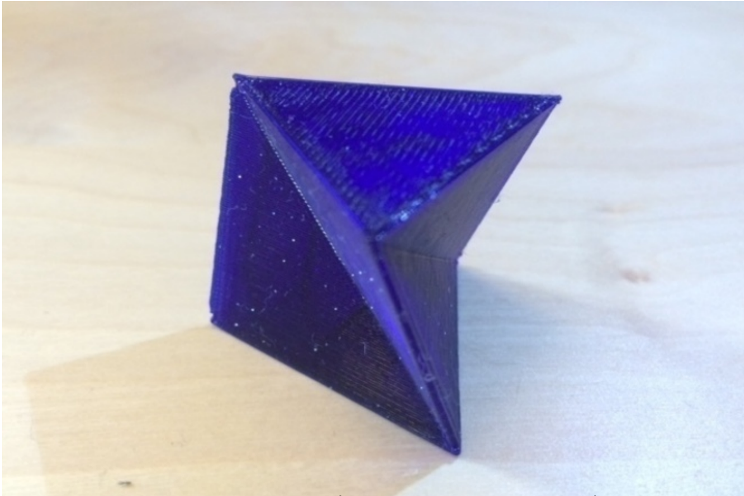


Photo from pinshape.com (designed by *mathgrrl*)

TO LEARN MORE

- J. O'Rourke, **Computational Geometry in C (2nd ed.)**, Cambridge University Press, 1998.
- M. de Berg, O. Cheong, M. van Kreveld, M. Overmars, **Computational Geometry: Algorithms and Applications (3rd rev. ed.)**, Springer, 2008.

TO LEARN MORE

- J. O'Rourke, **Computational Geometry in C (2nd ed.)**, Cambridge University Press, 1998.
- M. de Berg, O. Cheong, M. van Kreveld, M. Overmars, **Computational Geometry: Algorithms and Applications (3rd rev. ed.)**, Springer, 2008.

A NICE APPLICATION

The art gallery theorem

- J. O'Rourke, **Art Gallery Theorems and Algorithms**, Oxford University Press, 1987. http://maven.smith.edu/~orourke/books/ArtGalleryTheorems/art.html