

# Triangolazione di Delaunay

Programmazione e Calcolo Scientifico

ALDO SAMBO

MADDALENA GHIOTTI

EMILIO ZORZI

- 
- Selezione del triangolo iniziale e creazione della mesh
    - Costruzione della Griglia
    - Metodo Snake
  - Aggiunta di punti alla mesh
  - Verifica dell'ipotesi di Delaunay
  - Conclusioni e considerazioni



Preferenza su Punti INTERNI!

Fonte: Leonidas J. Guibas, Donald E. Knuth and Micha Sharir (1992)

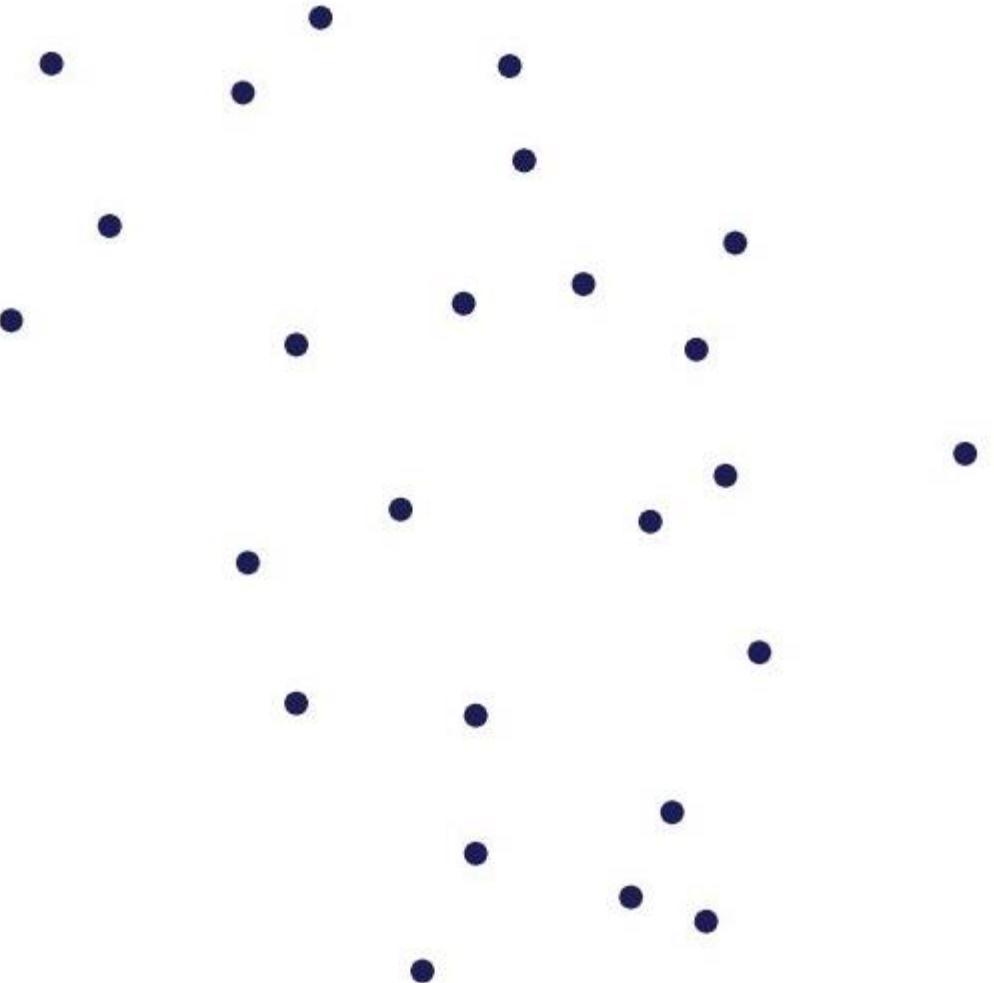
*Randomized Incremental Construction of Delaunay and Voronoi Diagrams*,  
pubblicato su *Algorithmica*, Springer

## DelaunayLibrary

C

Grid

- o int intNum
  - o Matrix<Rectangle, Eigen::Dynamic, Eigen::Dynamic> rectangles
  - o vector<Point\*> pointsGrid
  - o double x\_min
  - o double y\_min
  - o double intervalX
  - o double intervalY
- 
- Grid()
  - Grid(vector<Point\*> points)
  - void PointsInRectangle(vector<Point\*> points)
  - array<Point, 4> PickFourRandomPoints(vector<Point\*> points)
  - **array<Point, 4> Snake()**
  - void Show()

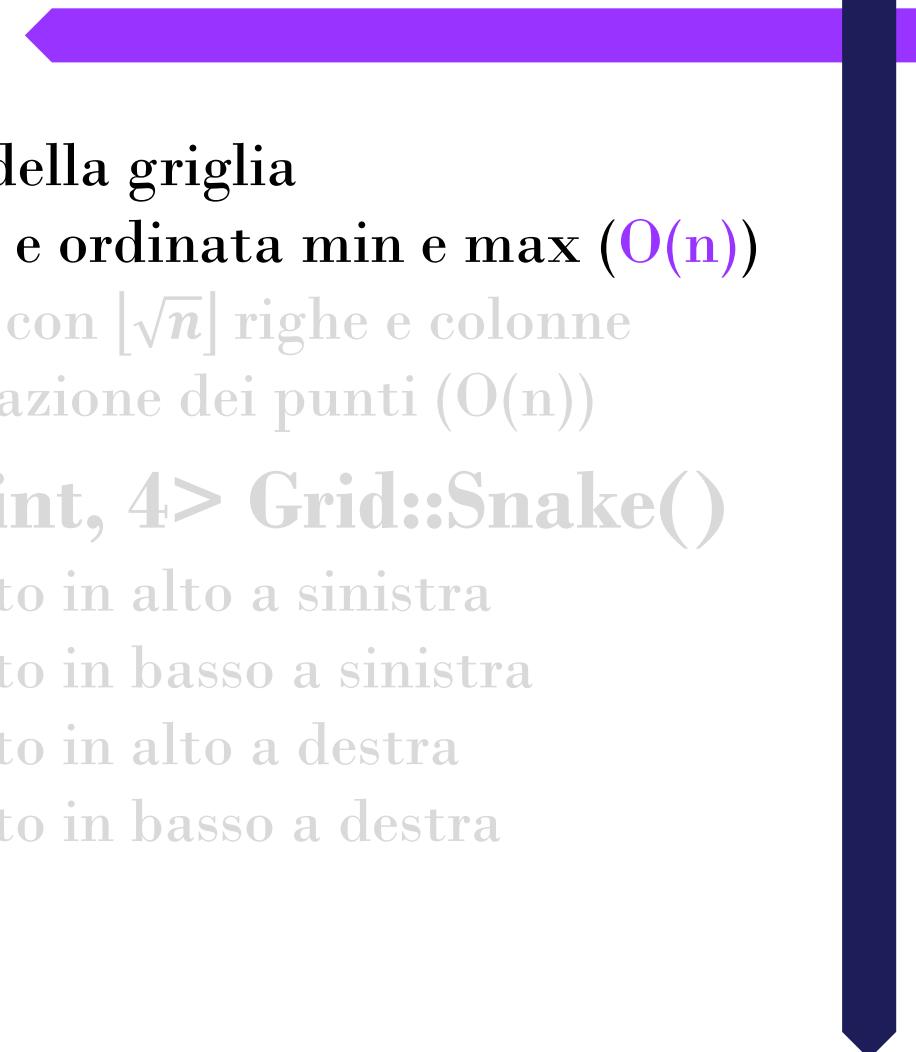


- Creazione della griglia

- Ascissa e ordinata min e max ( $O(n)$ )
- Griglia con  $\lceil \sqrt{n} \rceil$  righe e colonne
- Assegnazione dei punti ( $O(n)$ )

`array<Point, 4> Grid::Snake()`

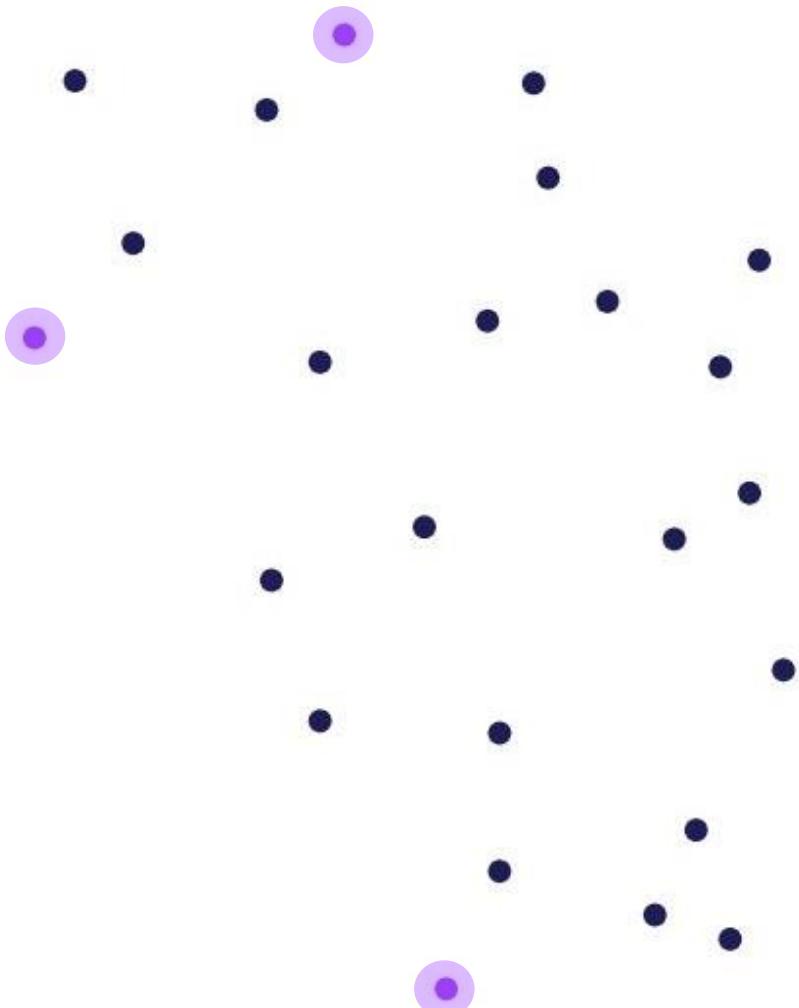
- Scelta punto in alto a sinistra
- Scelta punto in basso a sinistra
- Scelta punto in alto a destra
- Scelta punto in basso a destra

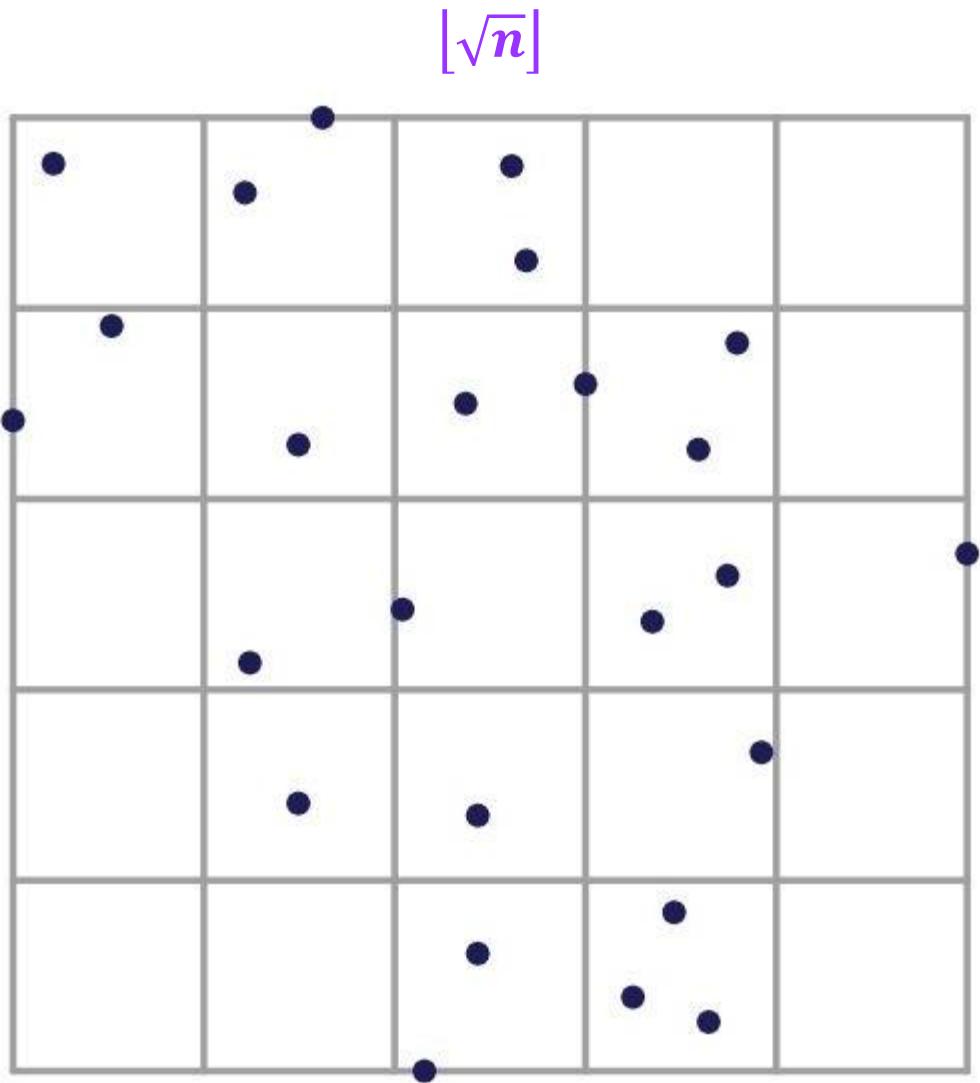


- Creazione della griglia
  - Ascissa e ordinata min e max ( $O(n)$ )
  - Griglia con  $\lceil \sqrt{n} \rceil$  righe e colonne
  - Assegnazione dei punti ( $O(n)$ )

`array<Point, 4> Grid::Snake()`

- Scelta punto in alto a sinistra
- Scelta punto in basso a sinistra
- Scelta punto in alto a destra
- Scelta punto in basso a destra



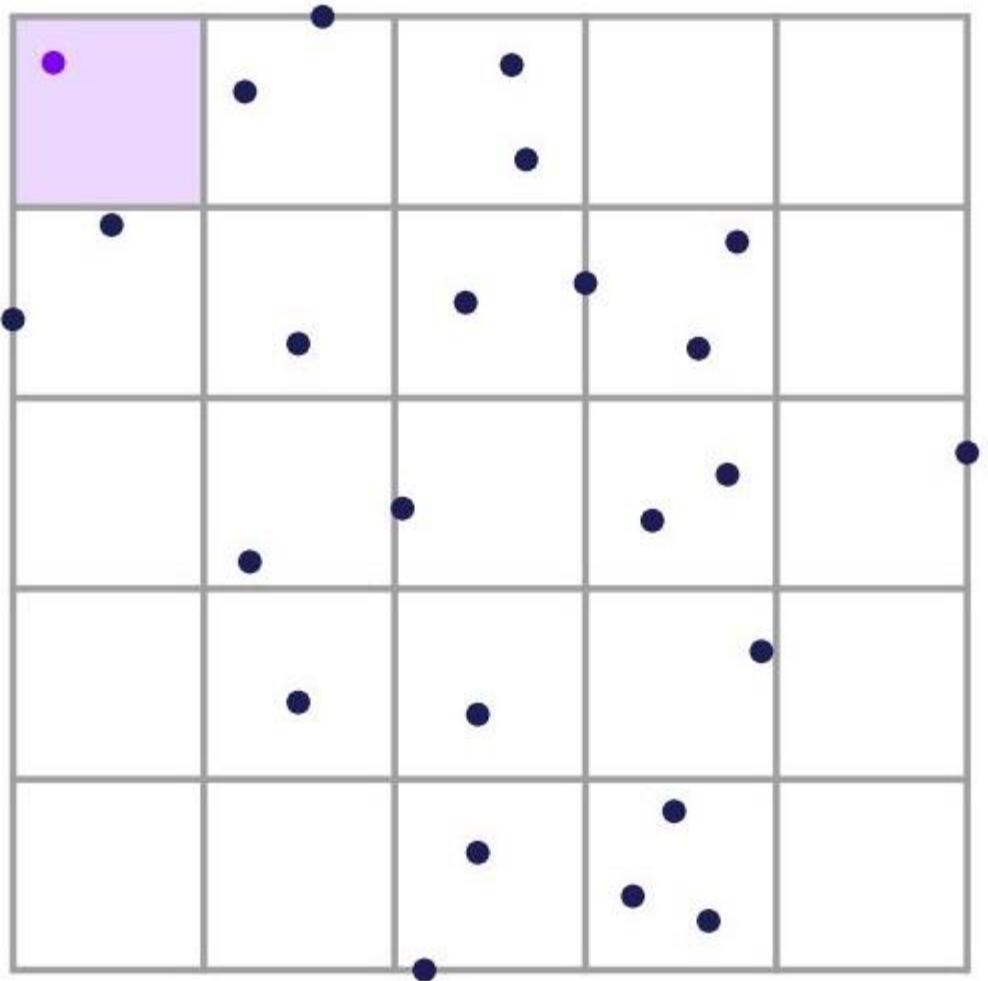


- Creazione della griglia

- Ascissa e ordinata min e max ( $O(n)$ )
- Griglia con  $[\sqrt{n}]$  righe e colonne
- Assegnazione dei punti ( $O(n)$ )

`array<Point, 4> Grid::Snake()`

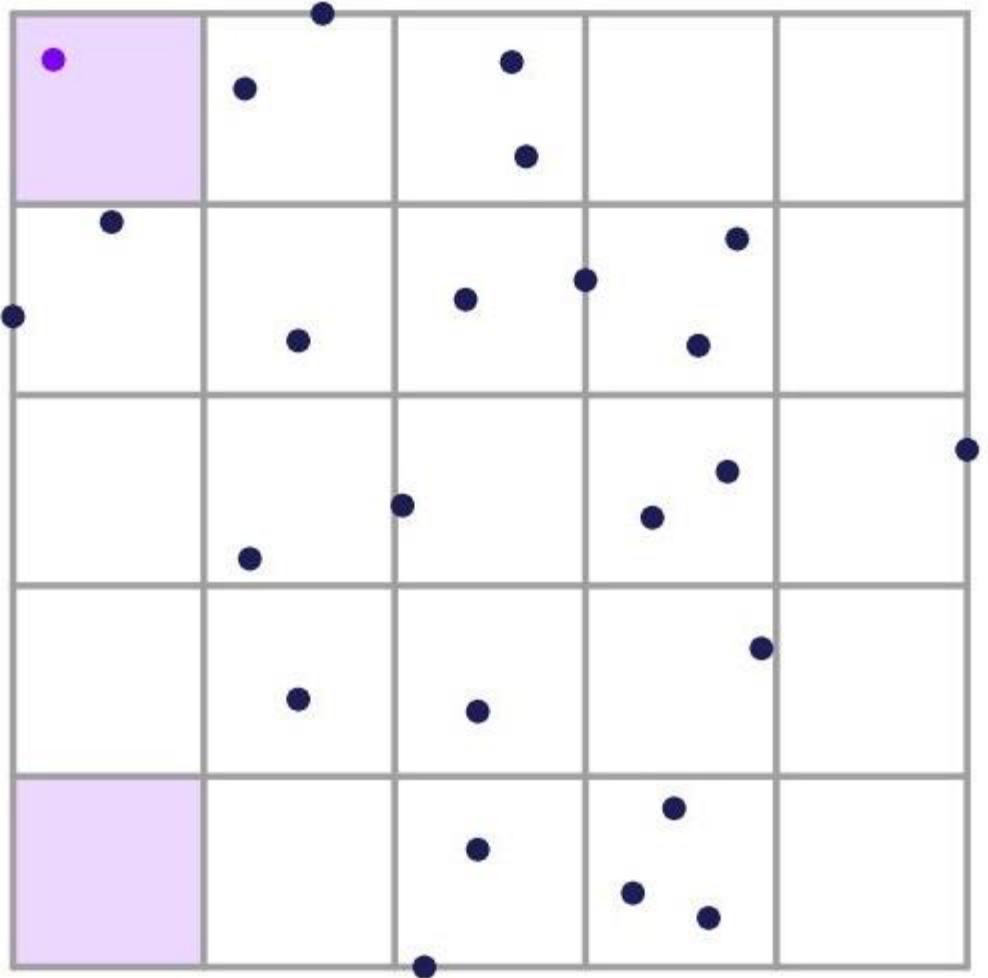
- Scelta punto in alto a sinistra
- Scelta punto in basso a sinistra
- Scelta punto in alto a destra
- Scelta punto in basso a destra



- Creazione della griglia
  - Ascissa e ordinata min e max ( $O(n)$ )
  - Griglia con  $\lfloor \sqrt{n} \rfloor$  righe e colonne
  - Assegnazione dei punti ( $O(n)$ )

**array<Point, 4> Grid::Snake()**

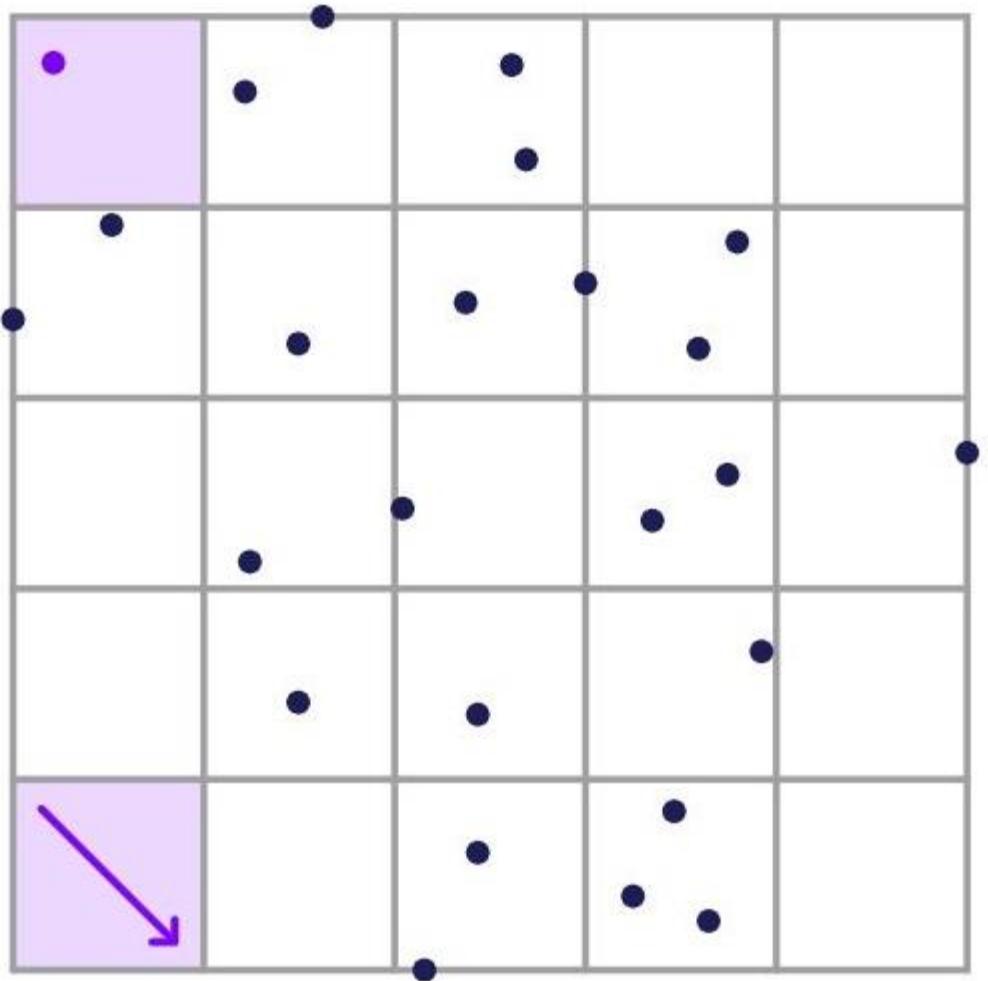
- Scelta punto in alto a sinistra
- Scelta punto in basso a sinistra
- Scelta punto in alto a destra
- Scelta punto in basso a destra



- Creazione della griglia
  - Ascissa e ordinata min e max ( $O(n)$ )
  - Griglia con  $\lfloor \sqrt{n} \rfloor$  righe e colonne
  - Assegnazione dei punti ( $O(n)$ )

**array<Point, 4> Grid::Snake()**

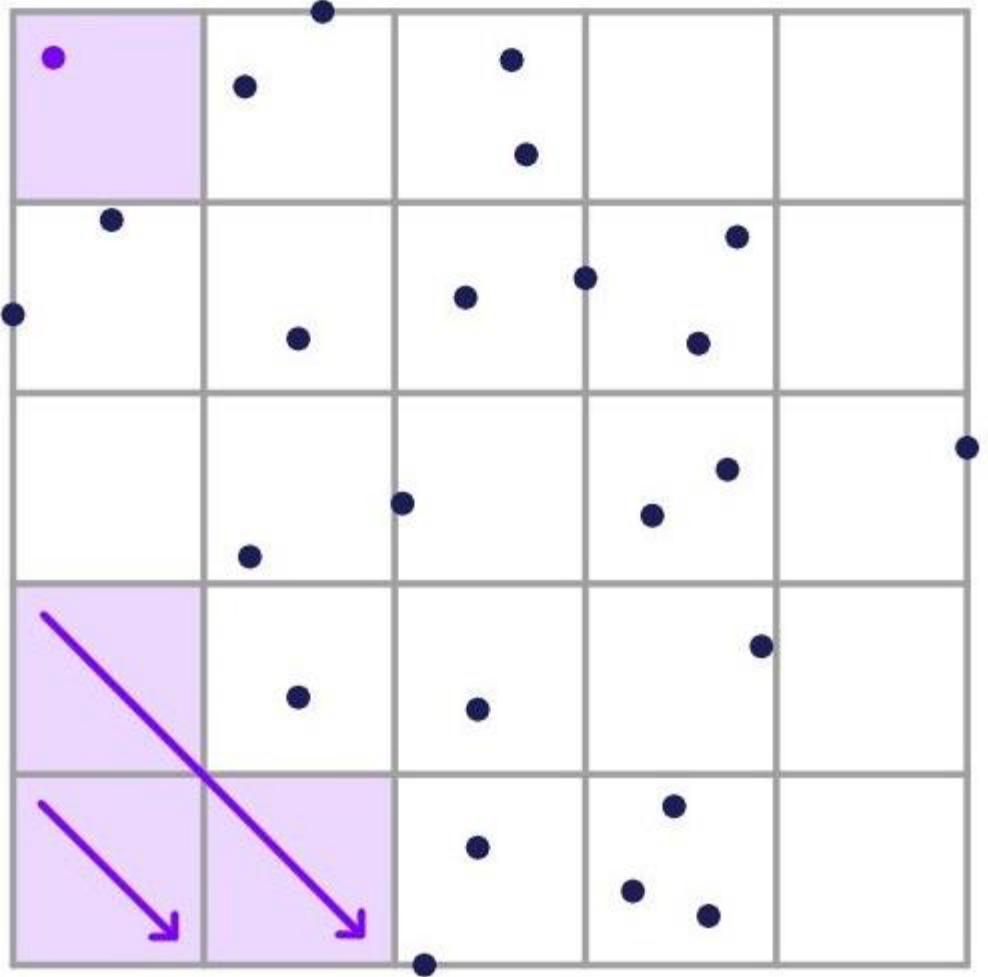
- Scelta punto in alto a sinistra
- Scelta punto in basso a sinistra
- Scelta punto in alto a destra
- Scelta punto in basso a destra



- Creazione della griglia
  - Ascissa e ordinata min e max ( $O(n)$ )
  - Griglia con  $\lfloor \sqrt{n} \rfloor$  righe e colonne
  - Assegnazione dei punti ( $O(n)$ )

**array<Point, 4> Grid::Snake()**

- Scelta punto in alto a sinistra
- Scelta punto in basso a sinistra
- Scelta punto in alto a destra
- Scelta punto in basso a destra

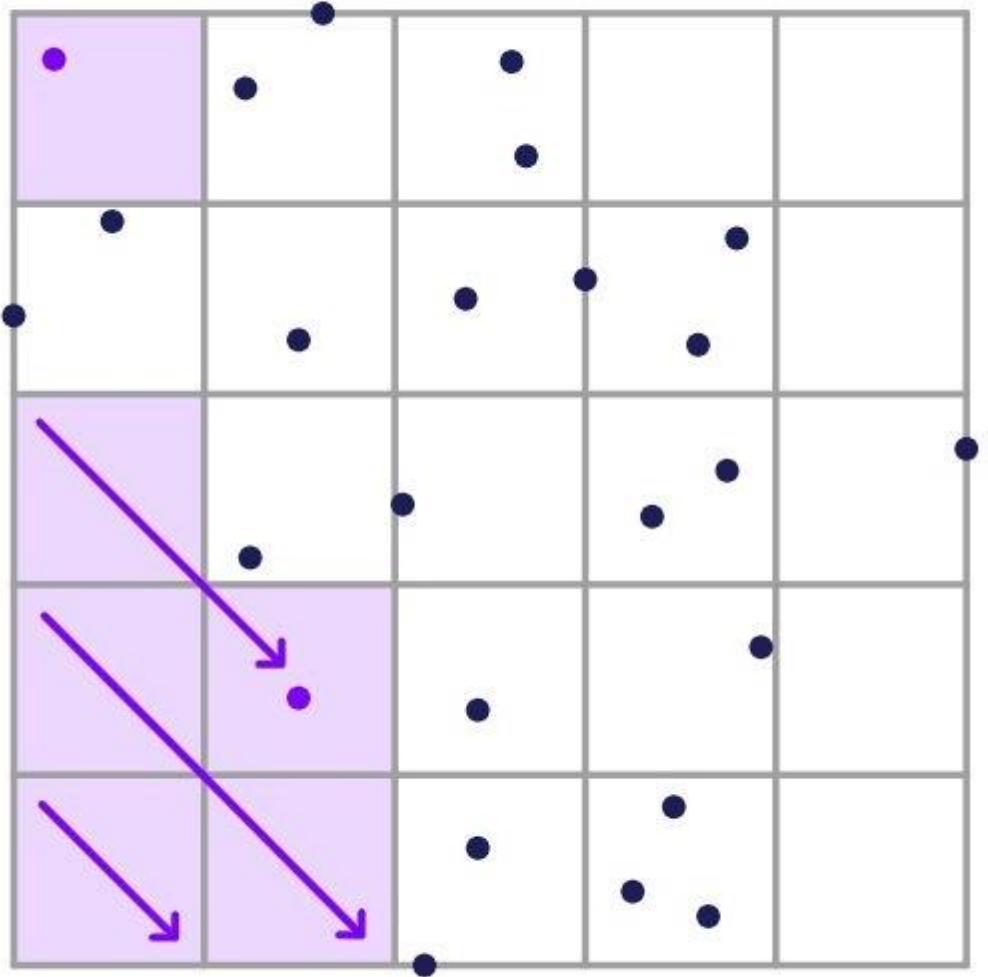


- Creazione della griglia

- Ascissa e ordinata min e max ( $O(n)$ )
- Griglia con  $\lfloor \sqrt{n} \rfloor$  righe e colonne
- Assegnazione dei punti ( $O(n)$ )

### **array<Point, 4> Grid::Snake()**

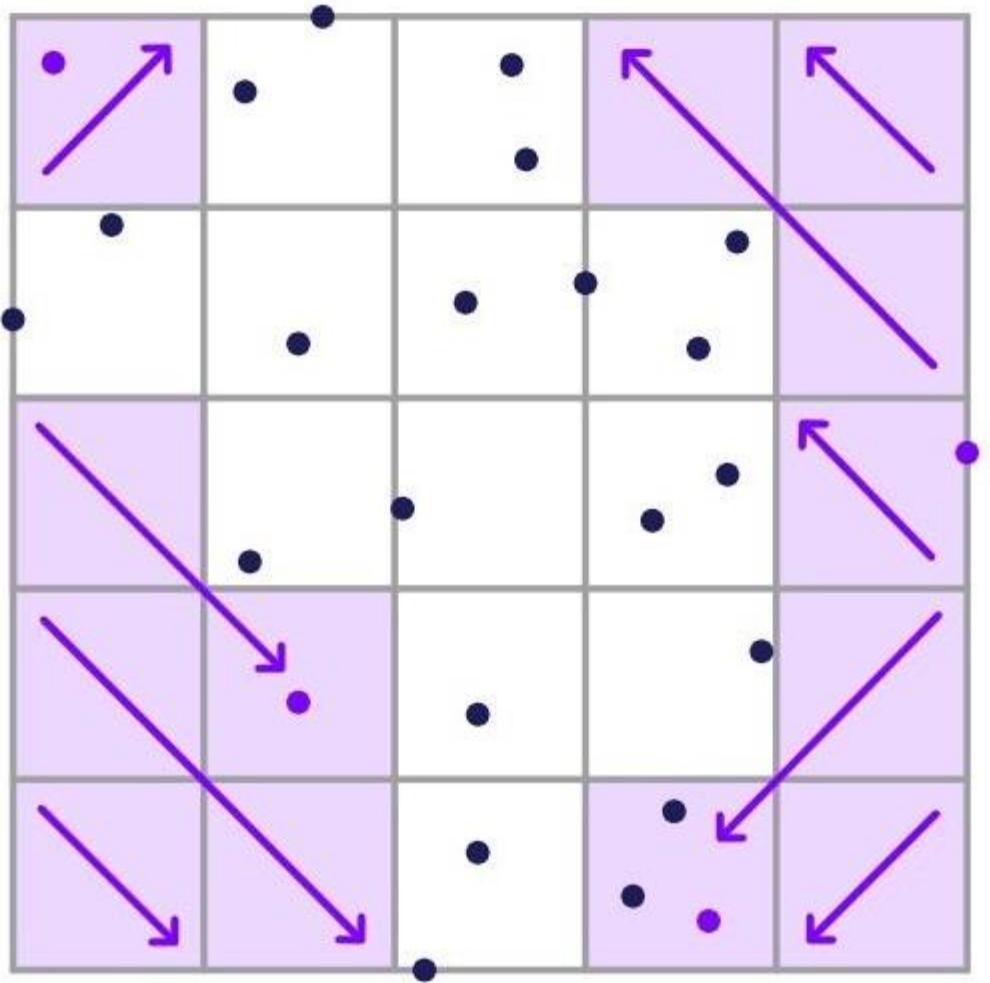
- Scelta punto in alto a sinistra
- Scelta punto in basso a sinistra
- Scelta punto in alto a destra
- Scelta punto in basso a destra



- Creazione della griglia
  - Ascissa e ordinata min e max ( $O(n)$ )
  - Griglia con  $\lfloor \sqrt{n} \rfloor$  righe e colonne
  - Assegnazione dei punti ( $O(n)$ )

**array<Point, 4> Grid::Snake()**

- Scelta punto in alto a sinistra
- Scelta punto in basso a sinistra
- Scelta punto in alto a destra
- Scelta punto in basso a destra

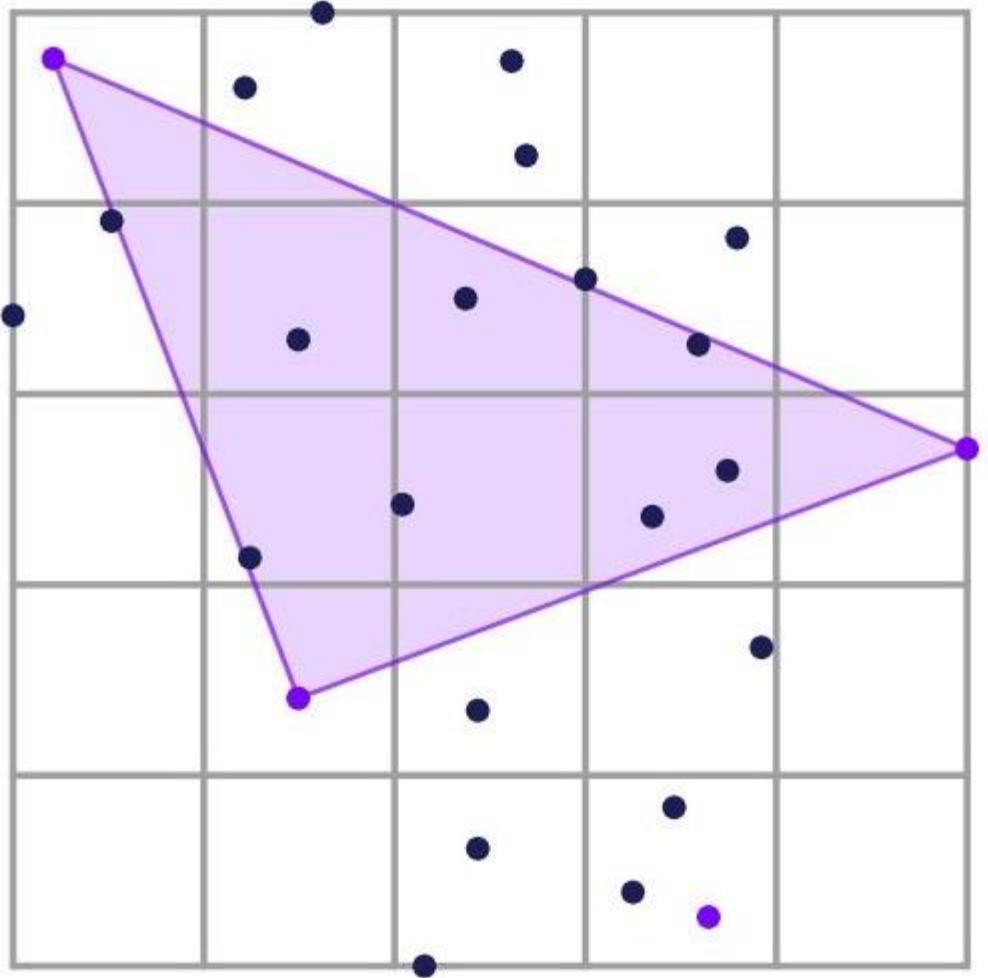


- Creazione della griglia

- Ascissa e ordinata min e max ( $O(n)$ )
- Griglia con  $\lfloor \sqrt{n} \rfloor$  righe e colonne
- Assegnazione dei punti ( $O(n)$ )

### **array<Point, 4> Grid::Snake()**

- Scelta punto in alto a sinistra
- Scelta punto in basso a sinistra
- Scelta punto in alto a destra
- Scelta punto in basso a destra
- Terminazione della serpentina sulla diagonale
- Costo computazionale:  $2O(n)=O(n)$

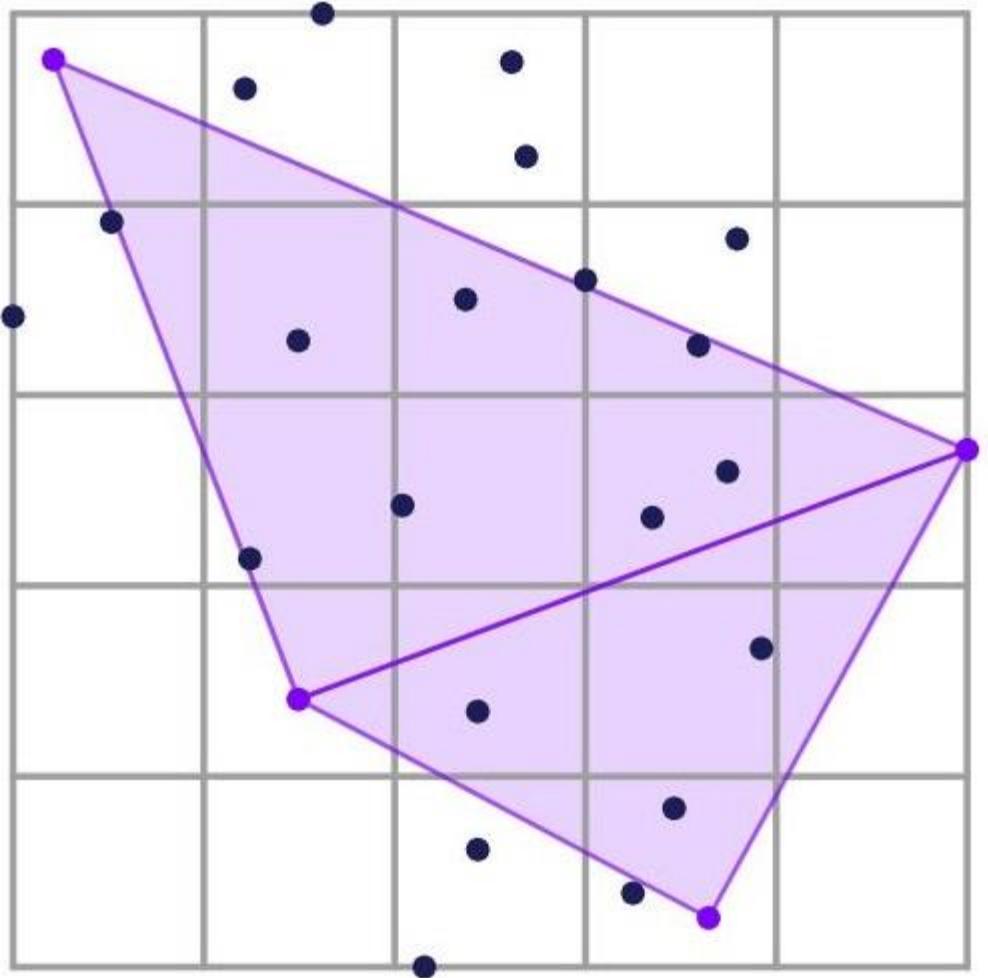


- Creazione della griglia

- Ascissa e ordinata min e max ( $O(n)$ )
- Griglia con  $\lceil \sqrt{n} \rceil$  righe e colonne
- Assegnazione dei punti ( $O(n)$ )

### **array<Point, 4> Grid::Snake()**

- Scelta punto in alto a sinistra
- Scelta punto in basso a sinistra
- Scelta punto in alto a destra
- Scelta punto in basso a destra
- Costruzione della mesh dal primo triangolo
- Aggiunta del secondo triangolo



- Creazione della griglia

- Ascissa e ordinata min e max ( $O(n)$ )
- Griglia con  $\lceil \sqrt{n} \rceil$  righe e colonne
- Assegnazione dei punti ( $O(n)$ )

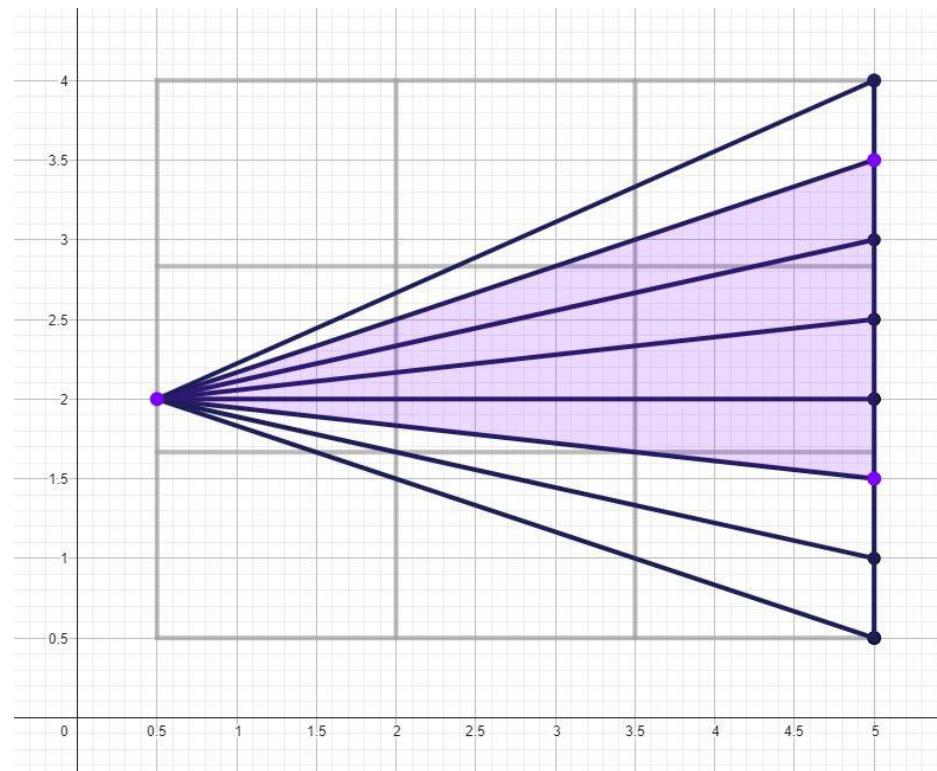
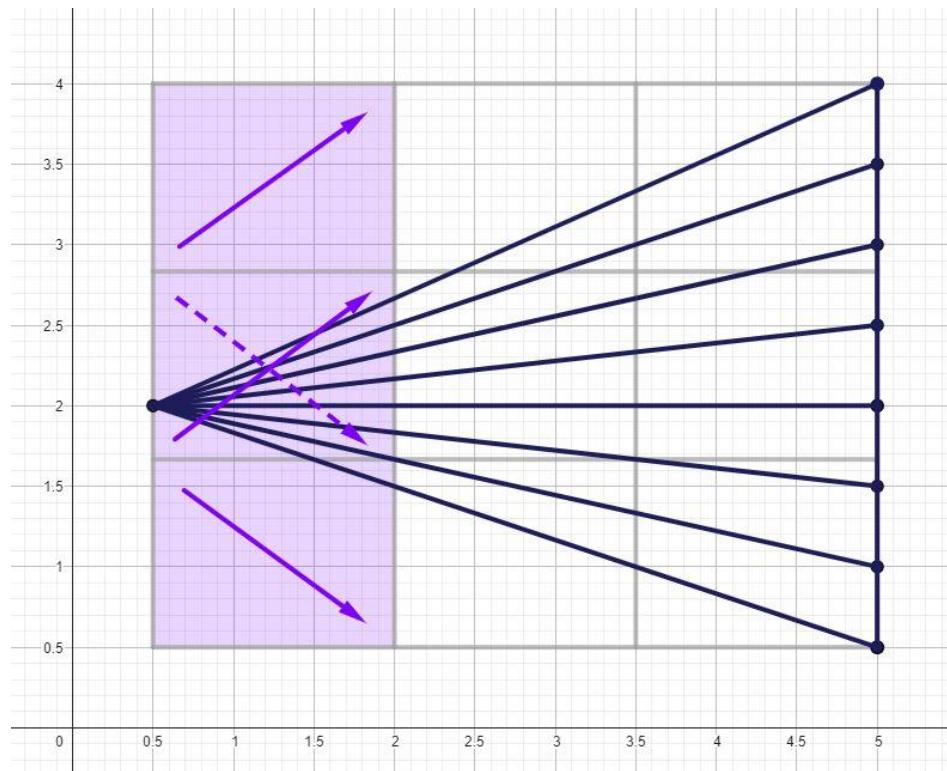
### **array<Point, 4> Grid::Snake()**

- Scelta punto in alto a sinistra
- Scelta punto in basso a sinistra
- Scelta punto in alto a destra
- Scelta punto in basso a destra
- Costruzione della mesh dal primo triangolo
- Aggiunta del secondo triangolo

- Se si raggiunge nuovamente un rettangolo già esplorato
- Se i primi 3 punti sono allineati



Selezione di 4 punti  
randomici non allineati



- 
- Selezione del triangolo iniziale e creazione della mesh
  - Aggiunta di punti alla mesh
    - Il ConvexHull
    - Metodo AddExternalPoint
    - Metodo AddInternalPoint
    - Metodo AddSidePoint
  - Verifica dell'ipotesi di Delaunay
  - Conclusioni e considerazioni

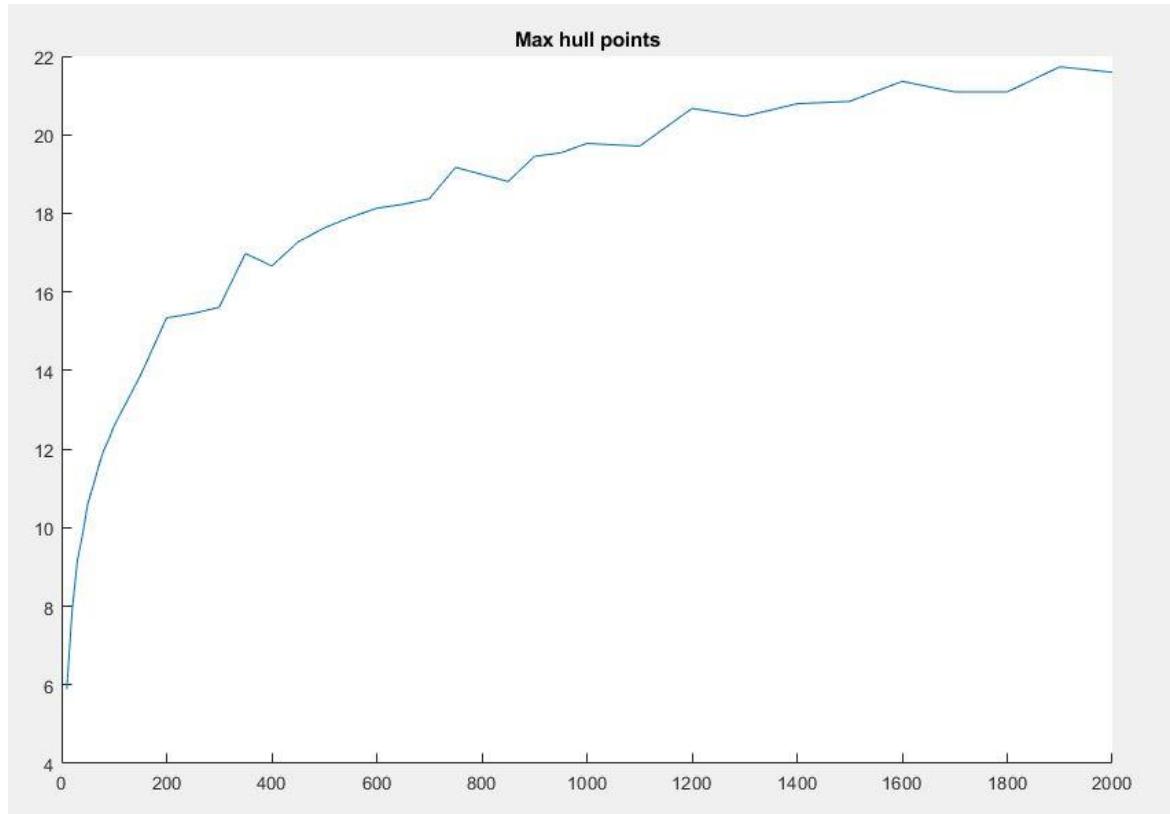
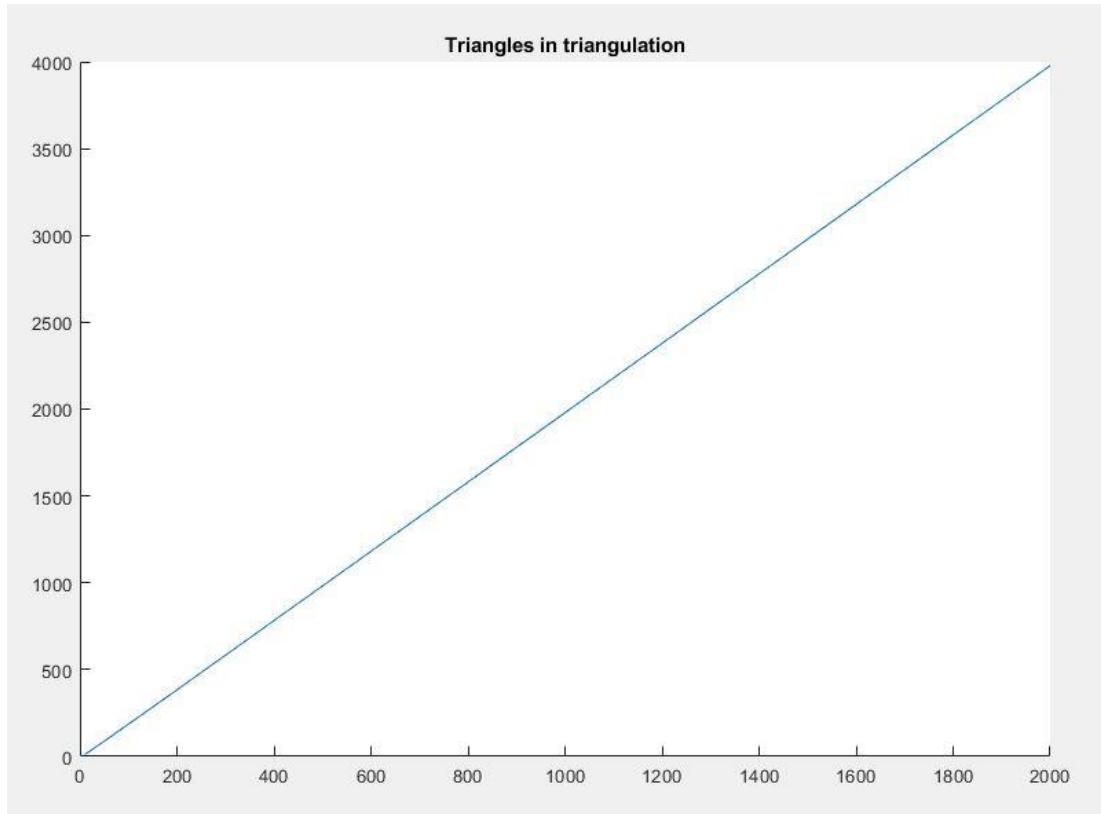
## DelaunayLibrary

C

Mesh

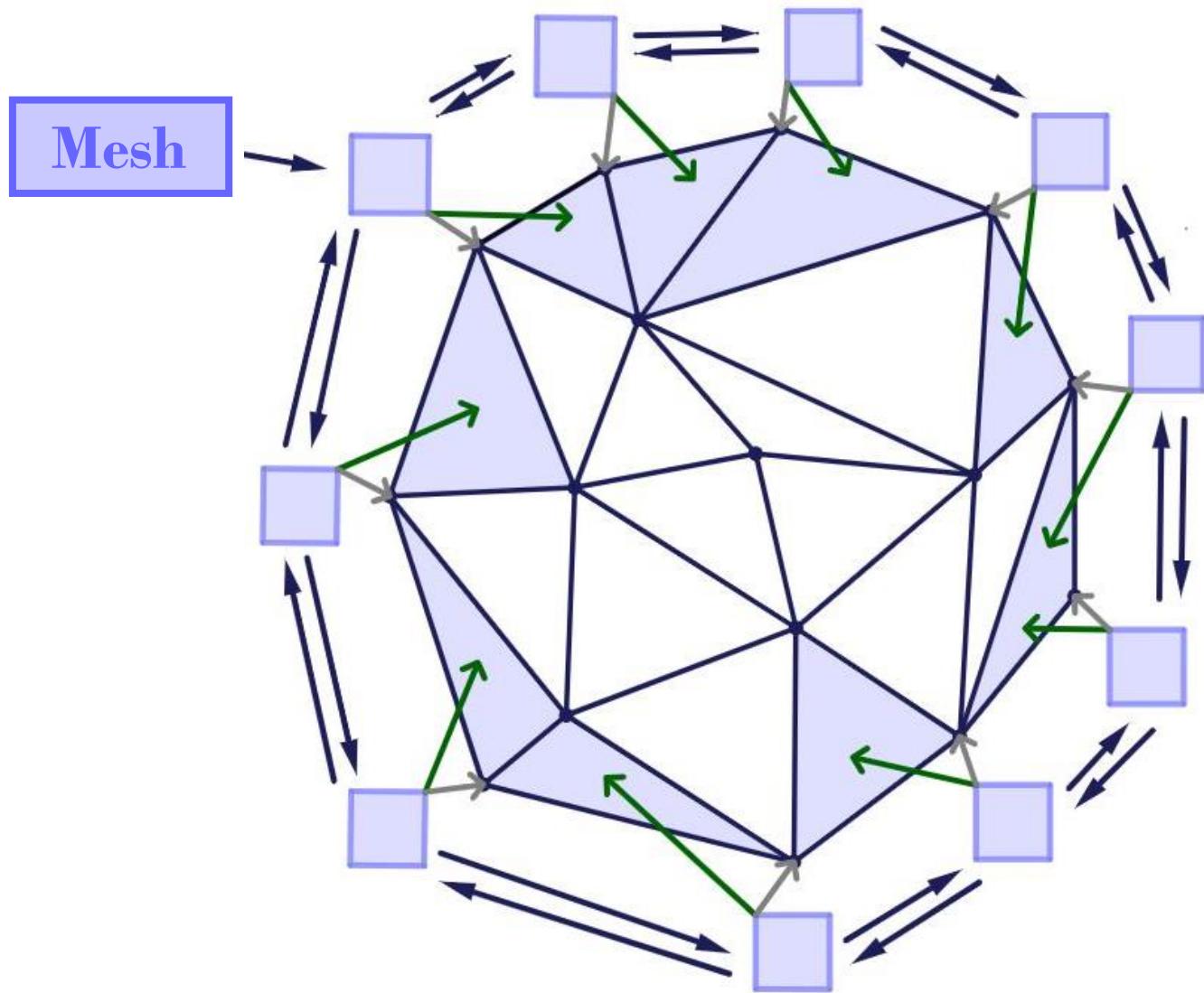
- `vector<Triangle*> guideTriangles`
- `convexHullElem* convexHull`
  
- `Mesh()`
- `Mesh(Triangle& triangle)`
- `void DelaunayPropagation(Triangle* startTriangle)`
- `void DelaunayNewTriangles(vector<Triangle*> new_triangles)`
- `void AddExternalPoint(Point& point)`
- `void AddInternalPoint(Point& point, Triangle* rootTriangle)`
- `void AddSidePoint(Point& point, Triangle* bigTriangle, int side)`
- `int CheckInside(Point point)`
- `void DeleteConvexHull()`

# La scelta del ConvexHull



## ConvexHullElem

```
Point* hullPoint;  
Triangle* externalTriangle;  
ConvexHullElem* prev;  
ConvexHullElem* next;
```



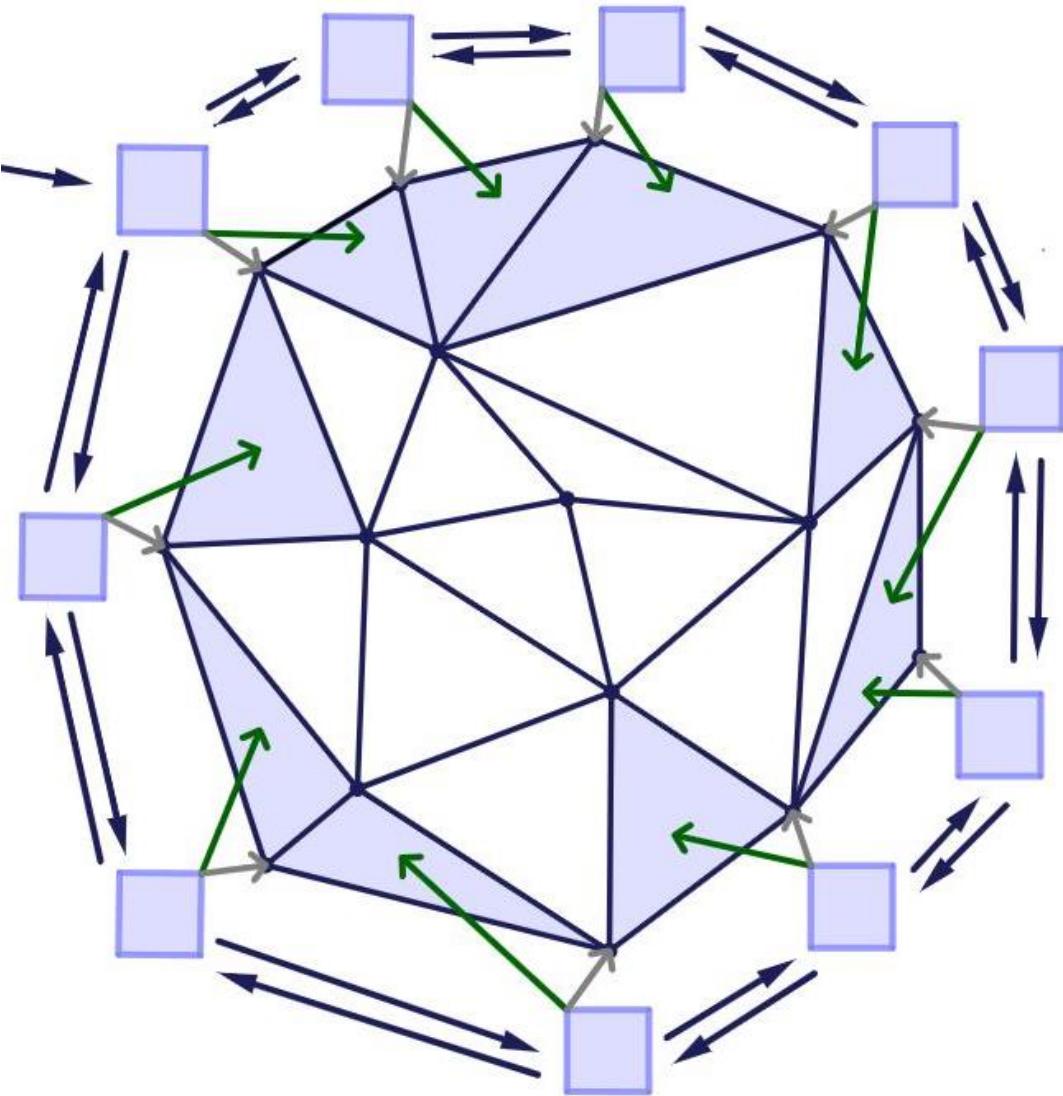
## ConvexHullElem

```
Point* hullPoint;  
Triangle* externalTriangle;  
ConvexHullElem* prev;  
ConvexHullElem* next;
```

## Triangle

```
array<Point,3> vertices;  
vector<Triangle*> pointedTriangles;  
array<Triangle*,3> adjacentTriangles;
```

## Mesh

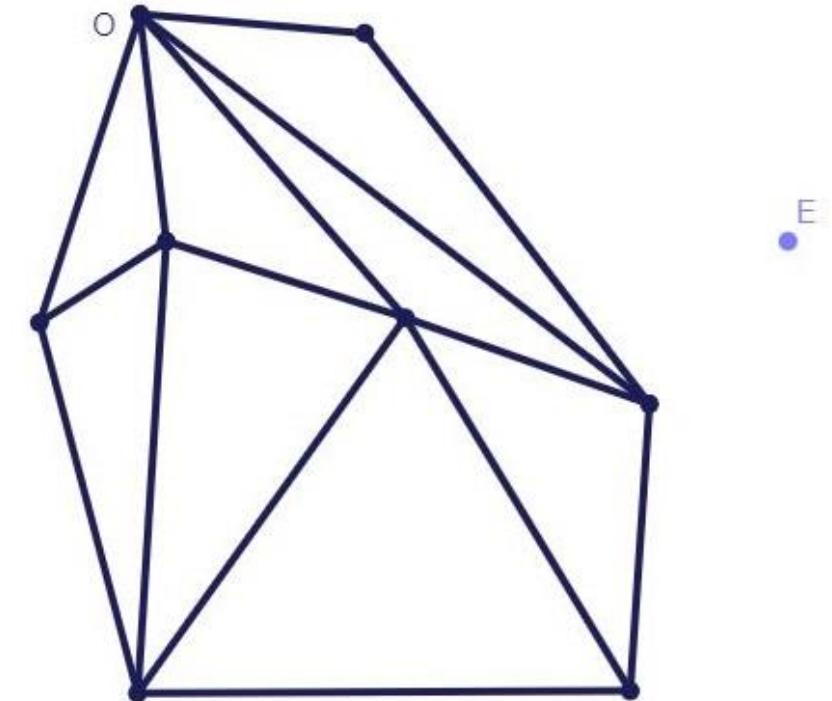


## **void Mesh::AddExternalPoint(Point& point)**

- Aggiunta di un punto dal lato opposto al primo elemento del ConvexHull
- Aggiunta di un punto dal lato del primo elemento del ConvexHull

**Obiettivo:** Minimizzare il numero di controlli sui lati del ConvexHull.

Aggiornamento del ConvexHull con l'aggiunta di un punto esterno.

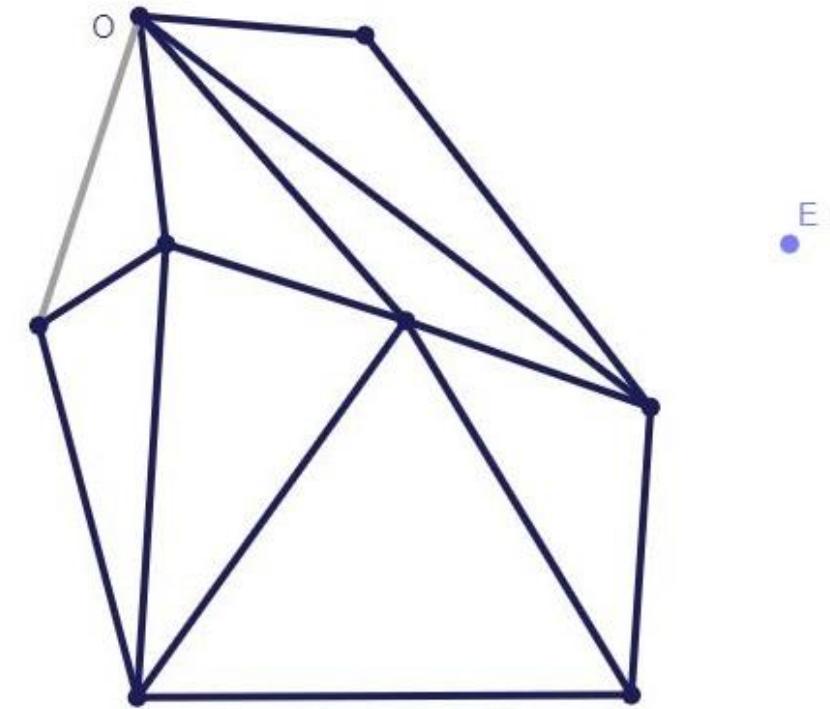


## **void Mesh::AddExternalPoint(Point& point)**

- Aggiunta di un punto dal lato opposto al primo elemento del ConvexHull
- Aggiunta di un punto dal lato del primo elemento del ConvexHull

**Obiettivo:** Minimizzare il numero di controlli sui lati del ConvexHull.

Aggiornamento del ConvexHull con l'aggiunta di un punto esterno.

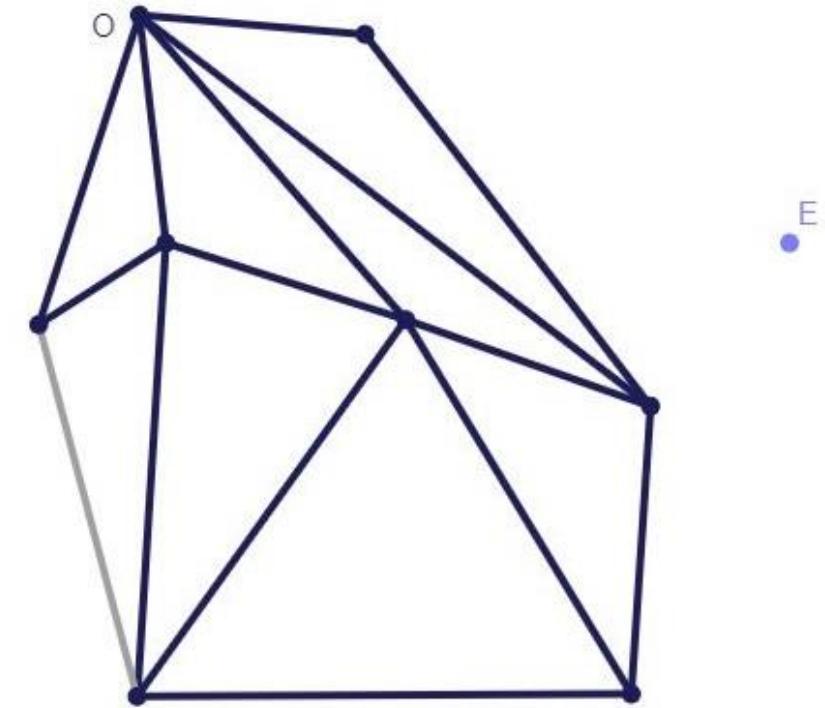


## **void Mesh::AddExternalPoint(Point& point)**

- Aggiunta di un punto dal lato opposto al primo elemento del ConvexHull
- Aggiunta di un punto dal lato del primo elemento del ConvexHull

**Obiettivo:** Minimizzare il numero di controlli sui lati del ConvexHull.

Aggiornamento del ConvexHull con l'aggiunta di un punto esterno.

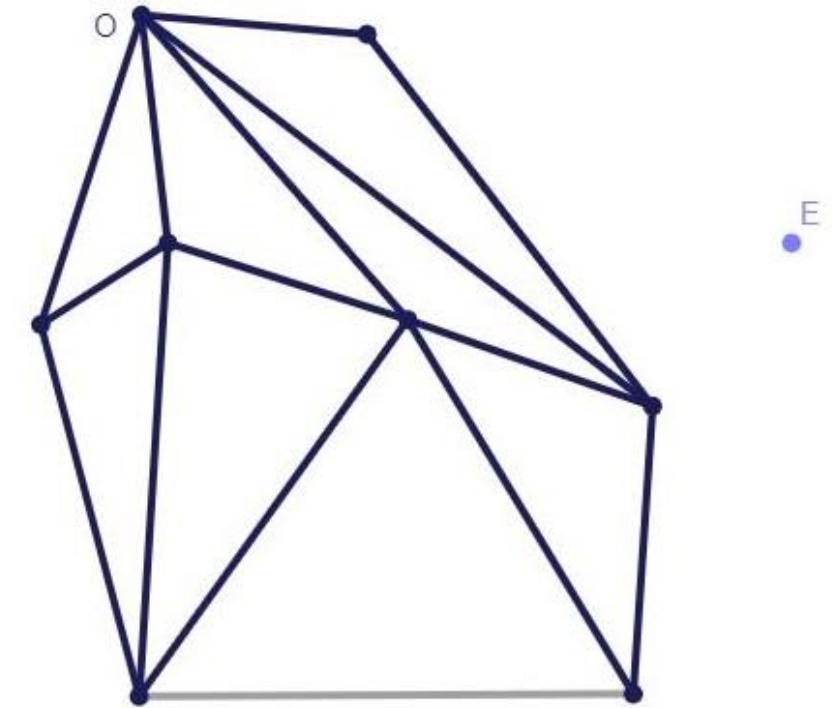


## **void Mesh::AddExternalPoint(Point& point)**

- Aggiunta di un punto dal lato opposto al primo elemento del ConvexHull
- Aggiunta di un punto dal lato del primo elemento del ConvexHull

**Obiettivo:** Minimizzare il numero di controlli sui lati del ConvexHull.

Aggiornamento del ConvexHull con l'aggiunta di un punto esterno.

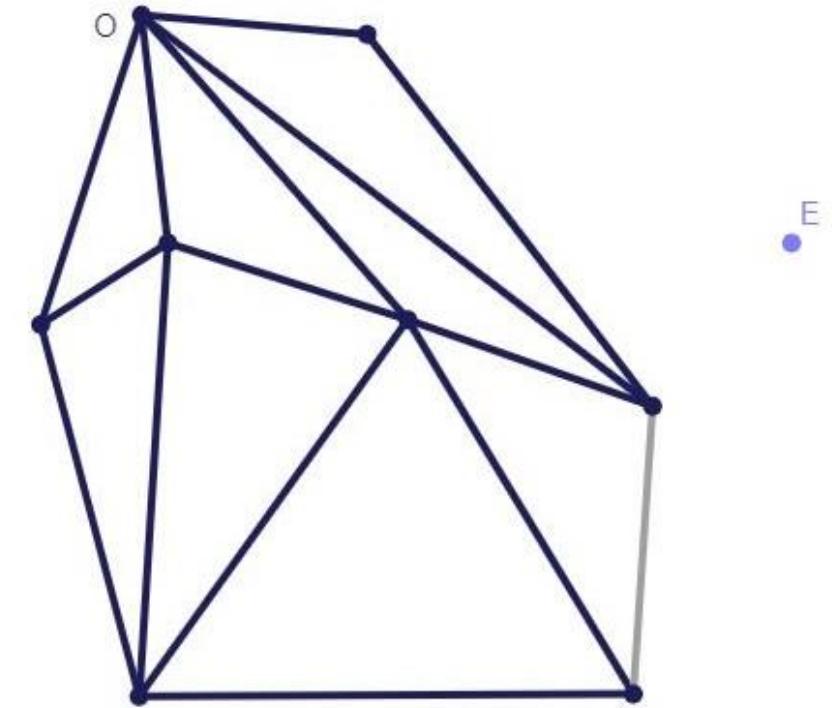


## **void Mesh::AddExternalPoint(Point& point)**

- Aggiunta di un punto dal lato opposto al primo elemento del ConvexHull
- Aggiunta di un punto dal lato del primo elemento del ConvexHull

**Obiettivo:** Minimizzare il numero di controlli sui lati del ConvexHull.

Aggiornamento del ConvexHull con l'aggiunta di un punto esterno.

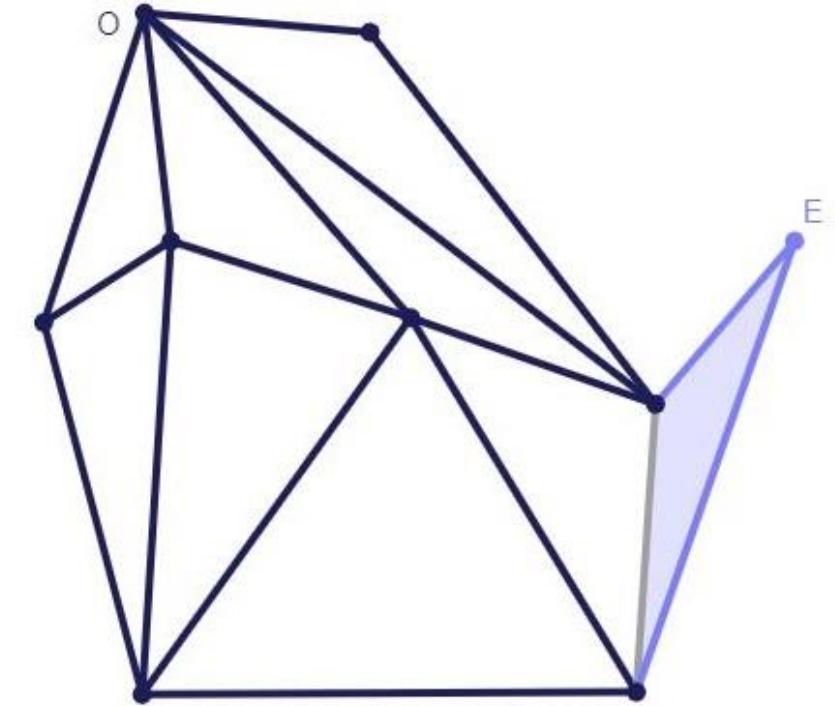


## **void Mesh::AddExternalPoint(Point& point)**

- Aggiunta di un punto dal lato opposto al primo elemento del ConvexHull
- Aggiunta di un punto dal lato del primo elemento del ConvexHull

**Obiettivo:** Minimizzare il numero di controlli sui lati del ConvexHull.

Aggiornamento del ConvexHull con l'aggiunta di un punto esterno.

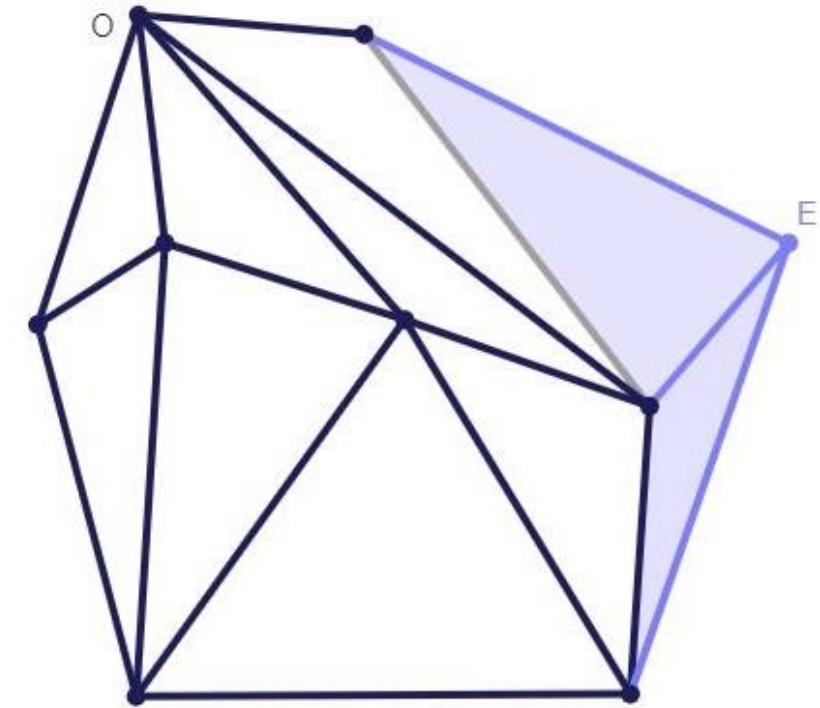


## **void Mesh::AddExternalPoint(Point& point)**

- Aggiunta di un punto dal lato opposto al primo elemento del ConvexHull
- Aggiunta di un punto dal lato del primo elemento del ConvexHull

**Obiettivo:** Minimizzare il numero di controlli sui lati del ConvexHull.

Aggiornamento del ConvexHull con l'aggiunta di un punto esterno.

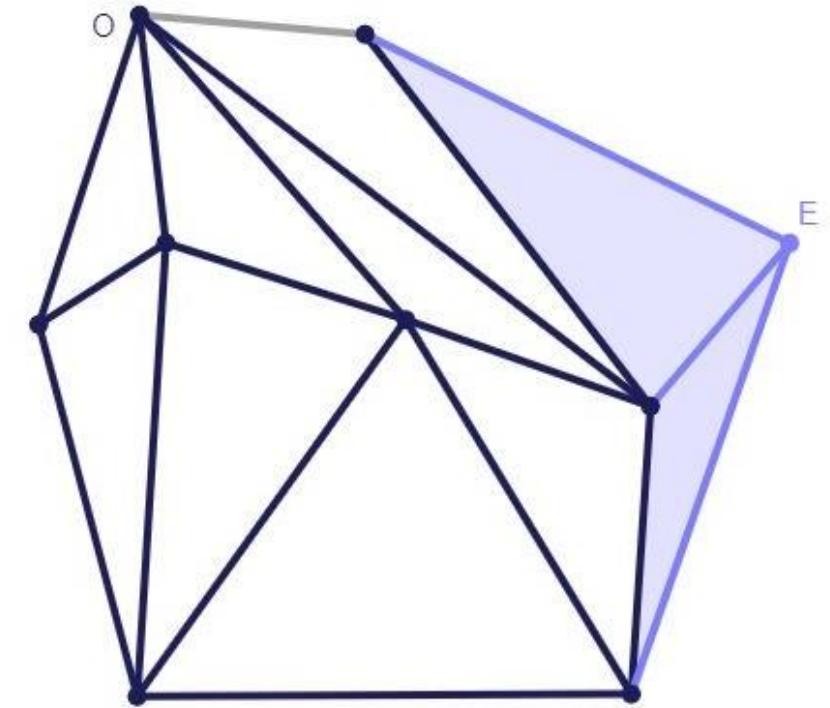


## **void Mesh::AddExternalPoint(Point& point)**

- Aggiunta di un punto dal lato opposto al primo elemento del ConvexHull
- Aggiunta di un punto dal lato del primo elemento del ConvexHull

**Obiettivo:** Minimizzare il numero di controlli sui lati del ConvexHull.

Aggiornamento del ConvexHull con l'aggiunta di un punto esterno.



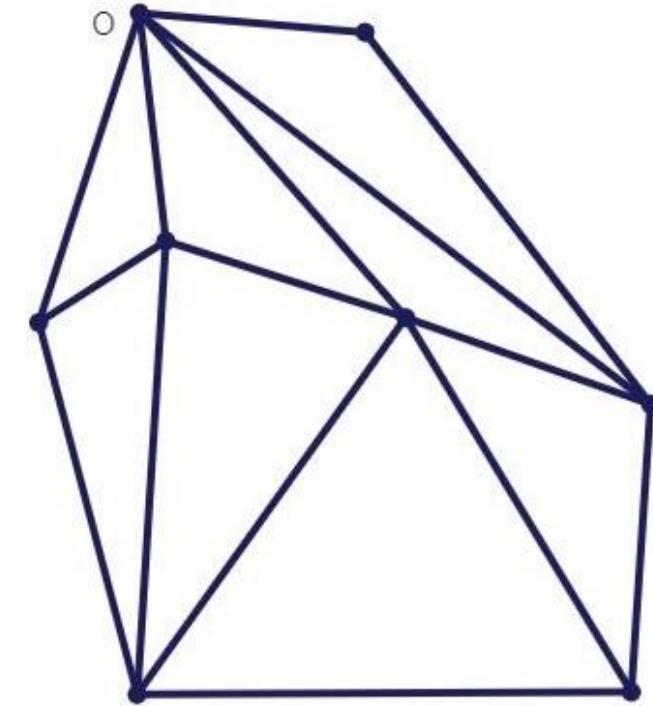
## **void Mesh::AddExternalPoint(Point& point)**

- Aggiunta di un punto dal lato opposto al primo elemento del ConvexHull
- Aggiunta di un punto dal lato del primo elemento del ConvexHull

**Obiettivo:** Minimizzare il numero di controlli sui lati del ConvexHull.

Aggiornamento del ConvexHull con l'aggiunta di un punto esterno.

E'



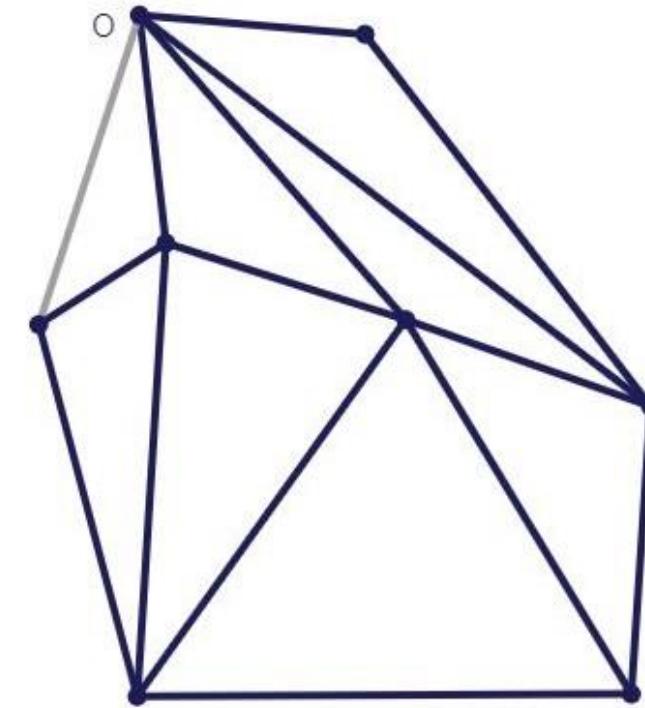
## **void Mesh::AddExternalPoint(Point& point)**

- Aggiunta di un punto dal lato opposto al primo elemento del ConvexHull
- Aggiunta di un punto dal lato del primo elemento del ConvexHull

**Obiettivo:** Minimizzare il numero di controlli sui lati del ConvexHull.

Aggiornamento del ConvexHull con l'aggiunta di un punto esterno.

E'

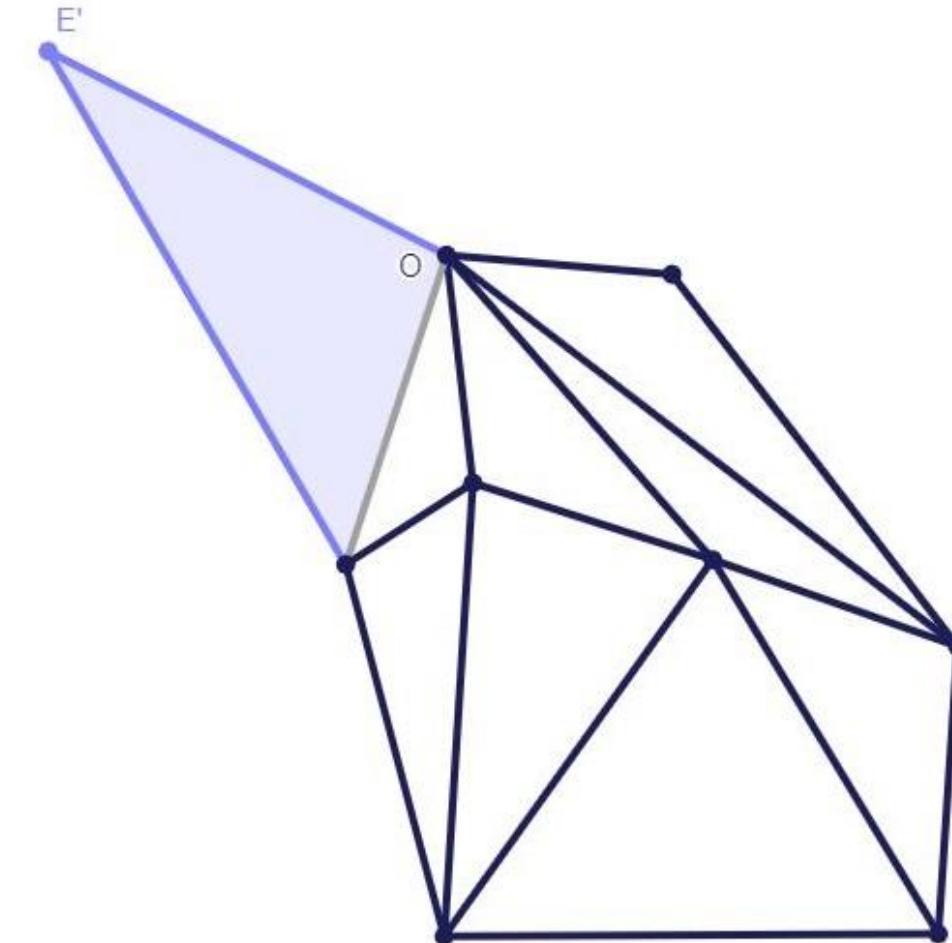


## **void Mesh::AddExternalPoint(Point& point)**

- Aggiunta di un punto dal lato opposto al primo elemento del ConvexHull
- Aggiunta di un punto dal lato del primo elemento del ConvexHull

**Obiettivo:** Minimizzare il numero di controlli sui lati del ConvexHull.

Aggiornamento del ConvexHull con l'aggiunta di un punto esterno.

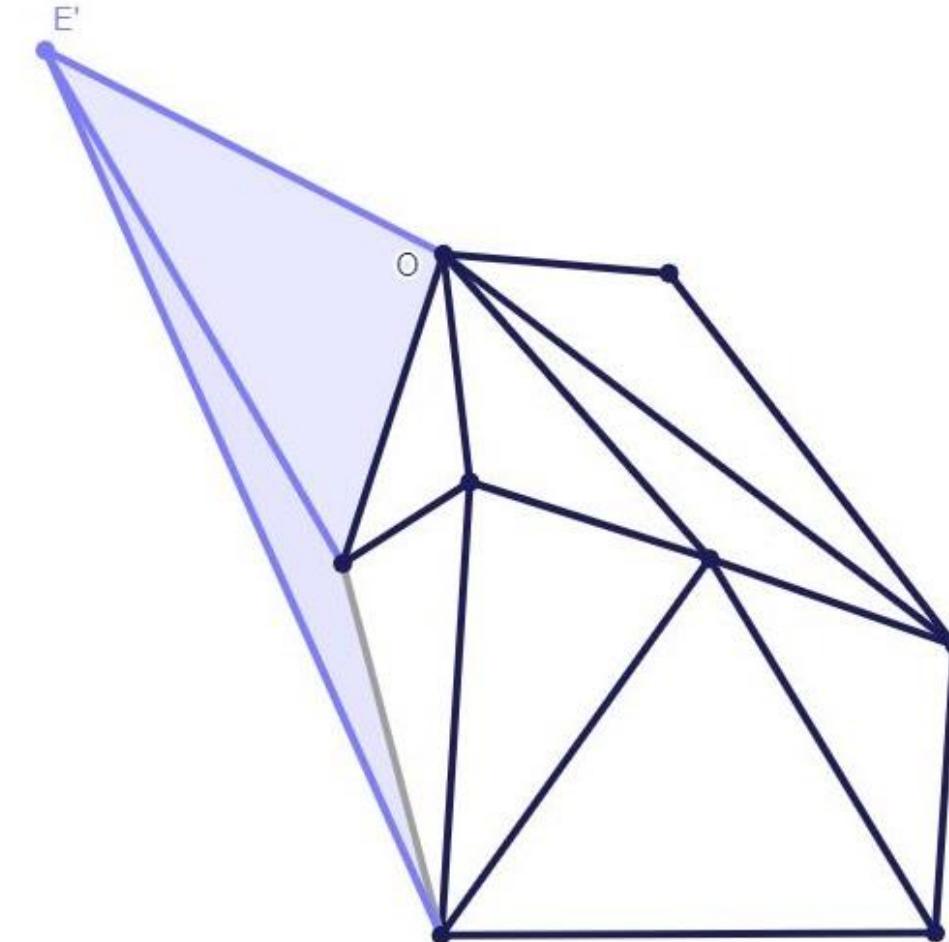


## **void Mesh::AddExternalPoint(Point& point)**

- Aggiunta di un punto dal lato opposto al primo elemento del ConvexHull
- Aggiunta di un punto dal lato del primo elemento del ConvexHull

**Obiettivo:** Minimizzare il numero di controlli sui lati del ConvexHull.

Aggiornamento del ConvexHull con l'aggiunta di un punto esterno.

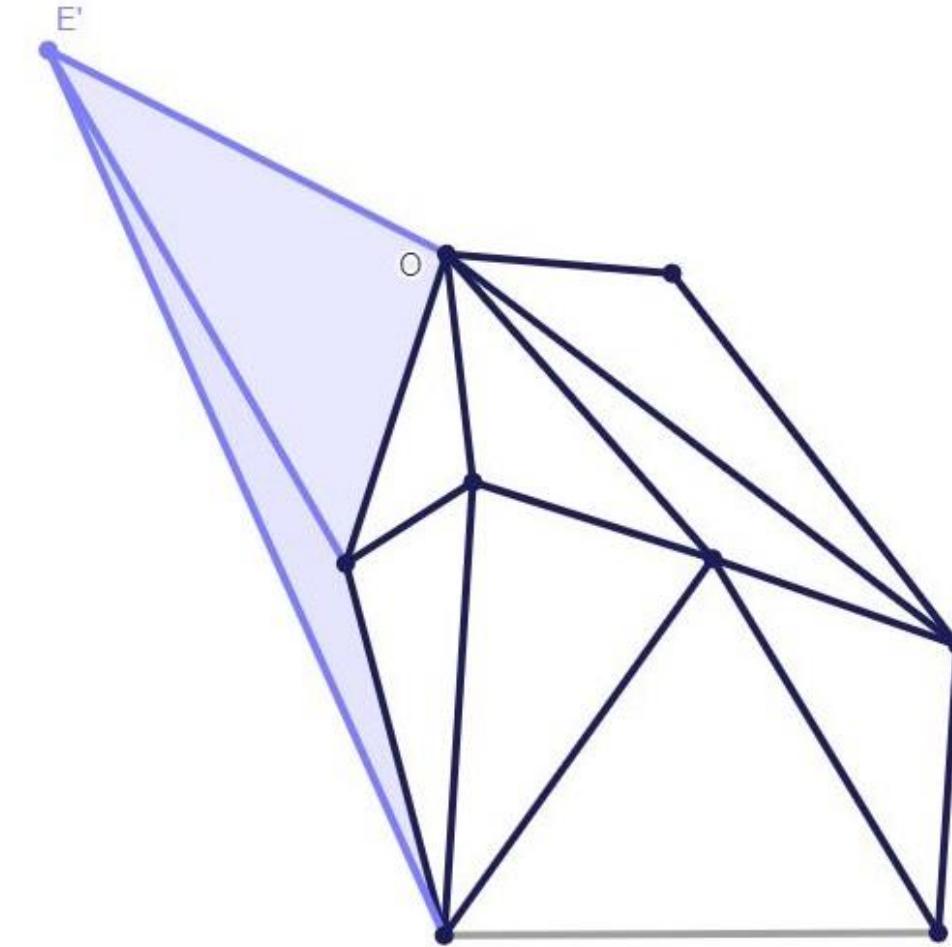


## **void Mesh::AddExternalPoint(Point& point)**

- Aggiunta di un punto dal lato opposto al primo elemento del ConvexHull
- Aggiunta di un punto dal lato del primo elemento del ConvexHull

**Obiettivo:** Minimizzare il numero di controlli sui lati del ConvexHull.

Aggiornamento del ConvexHull con l'aggiunta di un punto esterno.

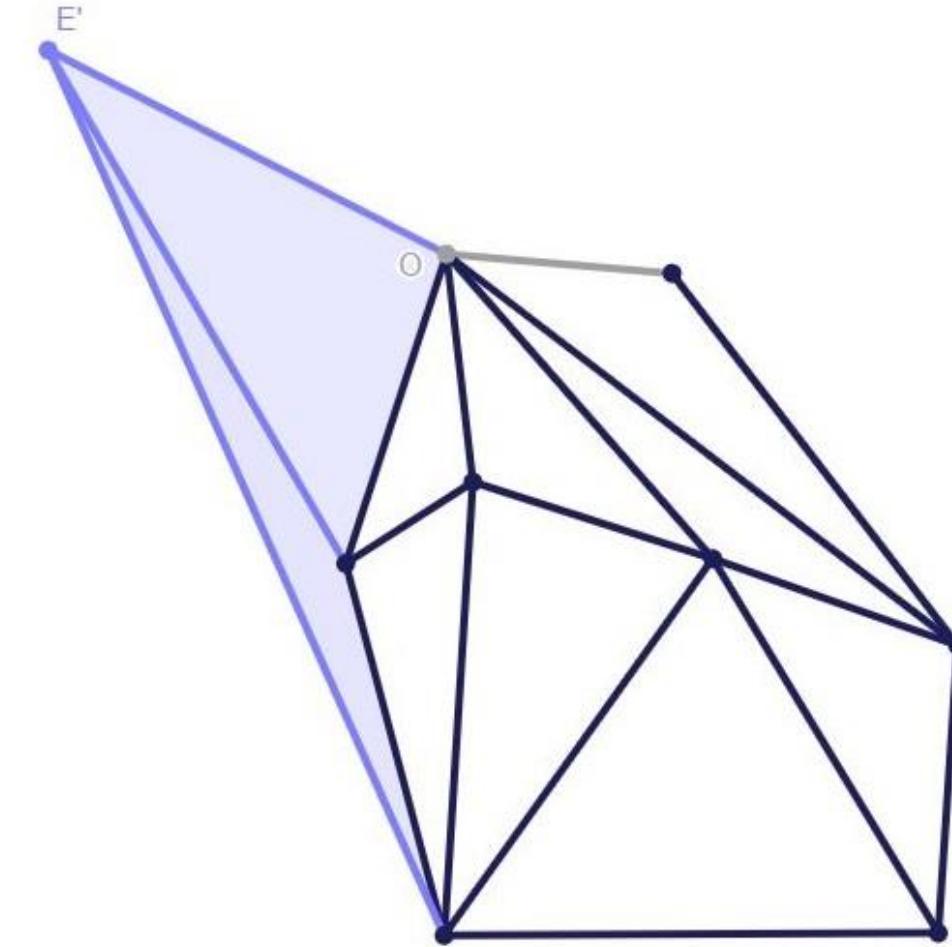


## **void Mesh::AddExternalPoint(Point& point)**

- Aggiunta di un punto dal lato opposto al primo elemento del ConvexHull
- Aggiunta di un punto dal lato del primo elemento del ConvexHull

**Obiettivo:** Minimizzare il numero di controlli sui lati del ConvexHull.

Aggiornamento del ConvexHull con l'aggiunta di un punto esterno.

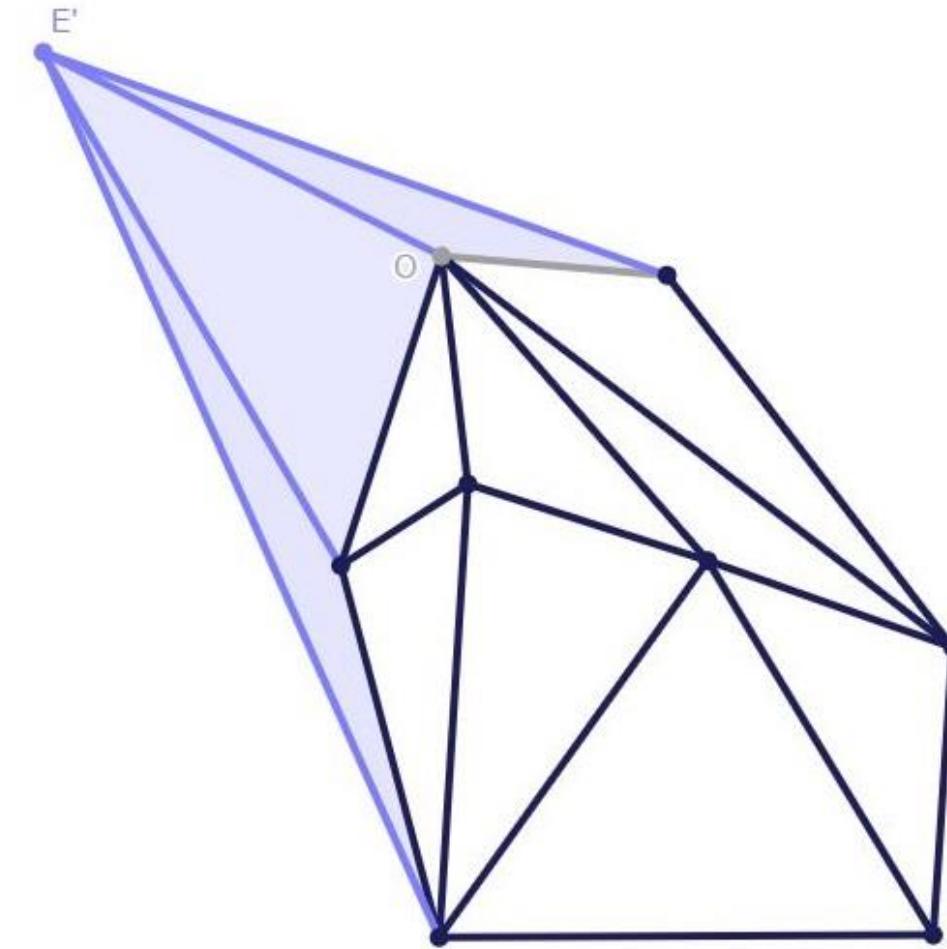


## **void Mesh::AddExternalPoint(Point& point)**

- Aggiunta di un punto dal lato opposto al primo elemento del ConvexHull
- Aggiunta di un punto dal lato del primo elemento del ConvexHull

**Obiettivo:** Minimizzare il numero di controlli sui lati del ConvexHull.

Aggiornamento del ConvexHull con l'aggiunta di un punto esterno.

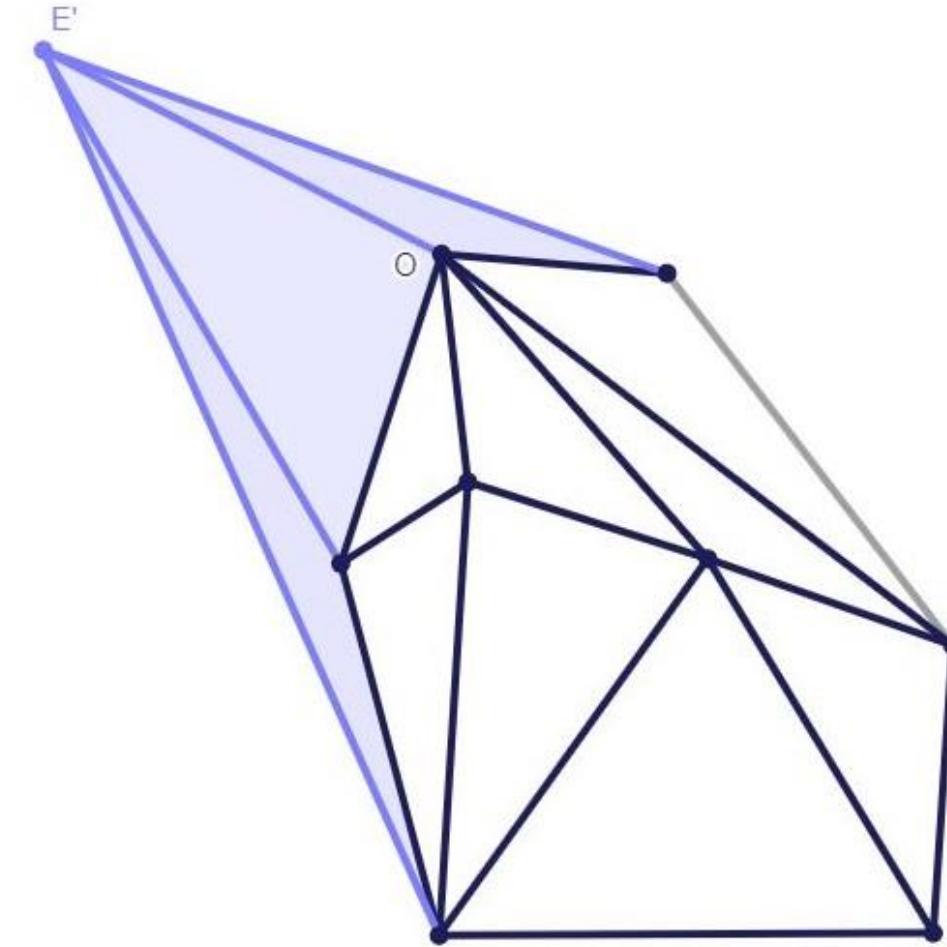


## **void Mesh::AddExternalPoint(Point& point)**

- Aggiunta di un punto dal lato opposto al primo elemento del ConvexHull
- Aggiunta di un punto dal lato del primo elemento del ConvexHull

**Obiettivo:** Minimizzare il numero di controlli sui lati del ConvexHull.

Aggiornamento del ConvexHull con l'aggiunta di un punto esterno.

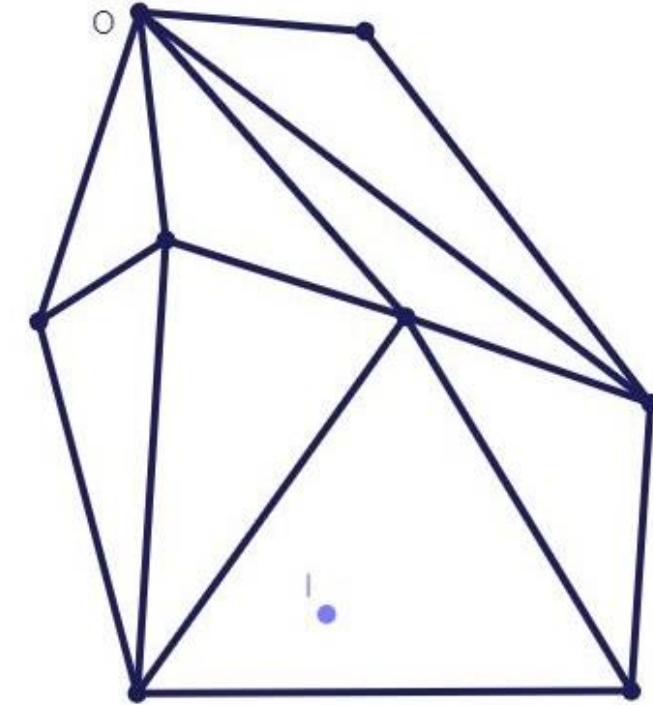


## void Mesh::AddInternalPoint(Point& point)

### Triangle

```
array<Point,3> vertices;  
vector<Triangle*> pointedTriangles;  
array<Triangle*,3> adjacentTriangles;
```

- Dai triangoli radice, creati aggiungendo punti esterni, si creano triangoli figli aggiungendo punti interni.
- Diventa fondamentale una struttura dei triangoli ad albero (grafo).

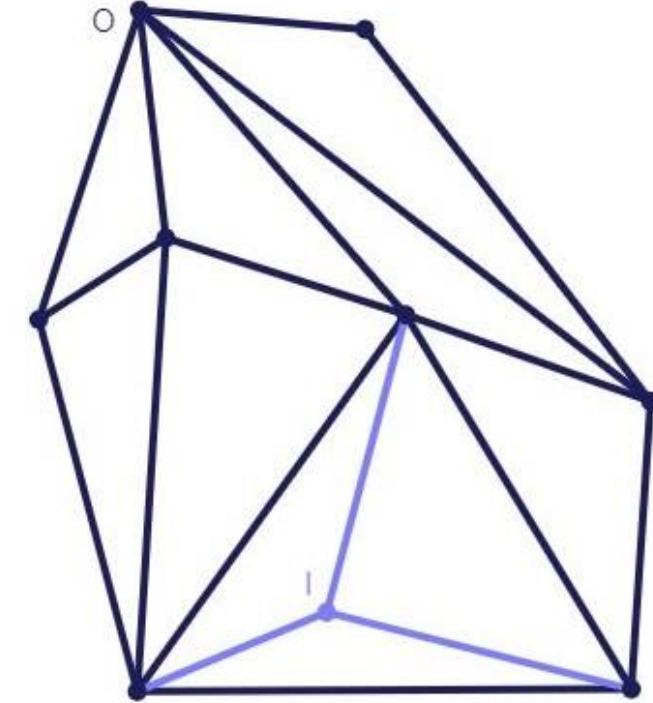


## void Mesh::AddInternalPoint(Point& point)

### Triangle

```
array<Point,3> vertices;  
vector<Triangle*> pointedTriangles;  
array<Triangle*,3> adjacentTriangles;
```

- Dai triangoli radice, creati aggiungendo punti esterni, si creano triangoli figli aggiungendo punti interni.
- Diventa fondamentale una struttura dei triangoli ad albero (grafo).

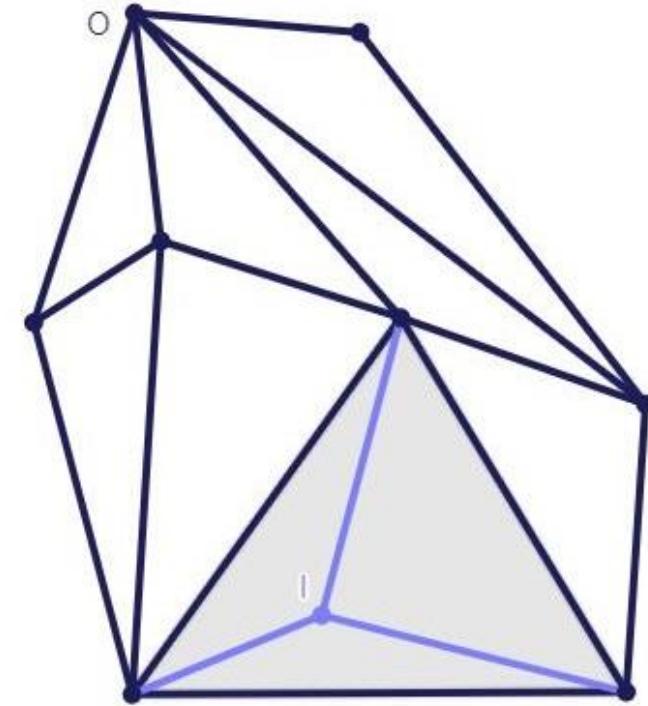


## void Mesh::AddInternalPoint(Point& point)

### Triangle

```
array<Point,3> vertices;  
vector<Triangle*> pointedTriangles;  
array<Triangle*,3> adjacentTriangles;
```

- Dai triangoli radice, creati aggiungendo punti esterni, si creano triangoli figli aggiungendo punti interni.
- Diventa fondamentale una struttura dei triangoli ad albero (grafo).

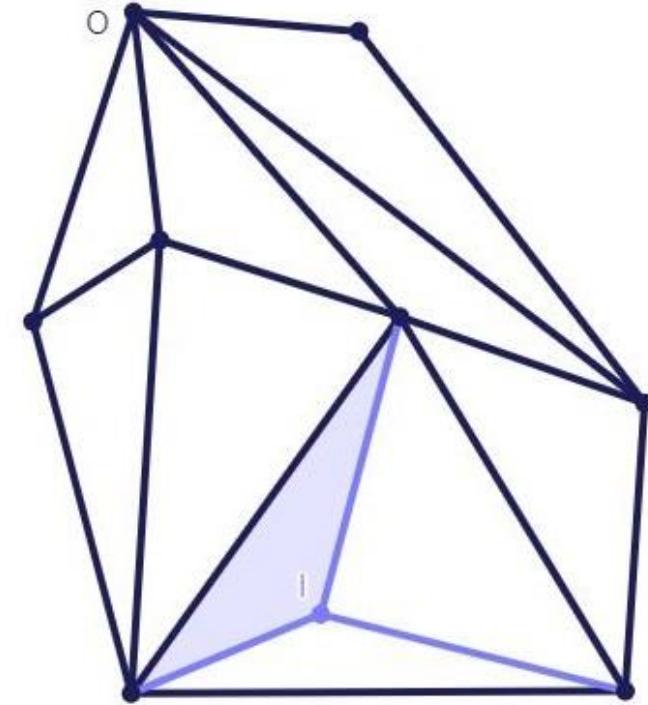


## void Mesh::AddInternalPoint(Point& point)

### Triangle

```
array<Point,3> vertices;  
vector<Triangle*> pointedTriangles;  
array<Triangle*,3> adjacentTriangles;
```

- Dai triangoli radice, creati aggiungendo punti esterni, si creano triangoli figli aggiungendo punti interni.
- Diventa fondamentale una struttura dei triangoli ad **albero** (grafo).

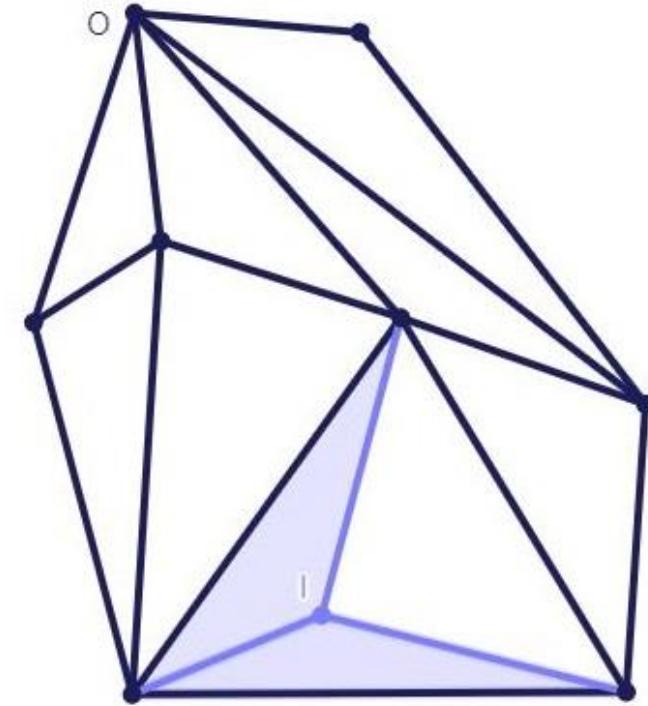


## void Mesh::AddInternalPoint(Point& point)

### Triangle

```
array<Point,3> vertices;  
vector<Triangle*> pointedTriangles;  
array<Triangle*,3> adjacentTriangles;
```

- Dai triangoli radice, creati aggiungendo punti esterni, si creano triangoli figli aggiungendo punti interni.
- Diventa fondamentale una struttura dei triangoli ad **albero** (grafo).

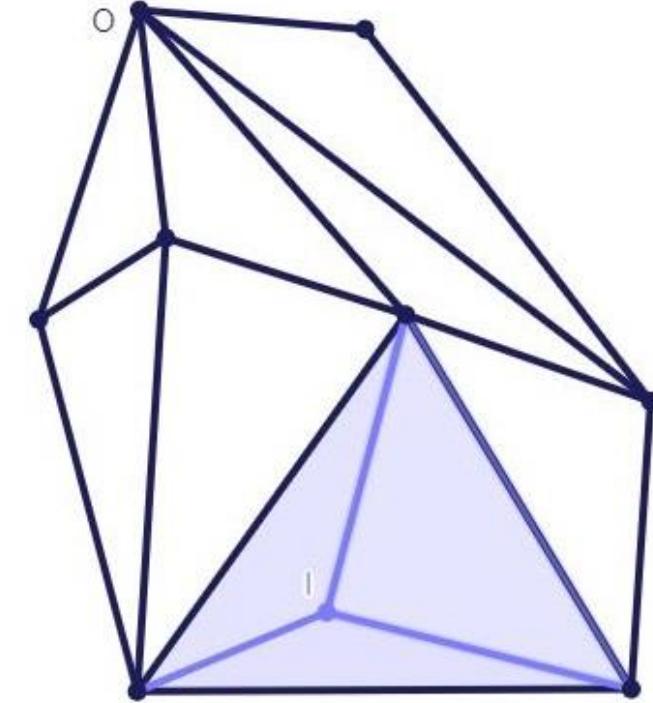


## void Mesh::AddInternalPoint(Point& point)

### Triangle

```
array<Point,3> vertices;  
vector<Triangle*> pointedTriangles;  
array<Triangle*,3> adjacentTriangles;
```

- Dai triangoli radice, creati aggiungendo punti esterni, si creano triangoli figli aggiungendo punti interni.
- Diventa fondamentale una struttura dei triangoli ad **albero** (grafo).

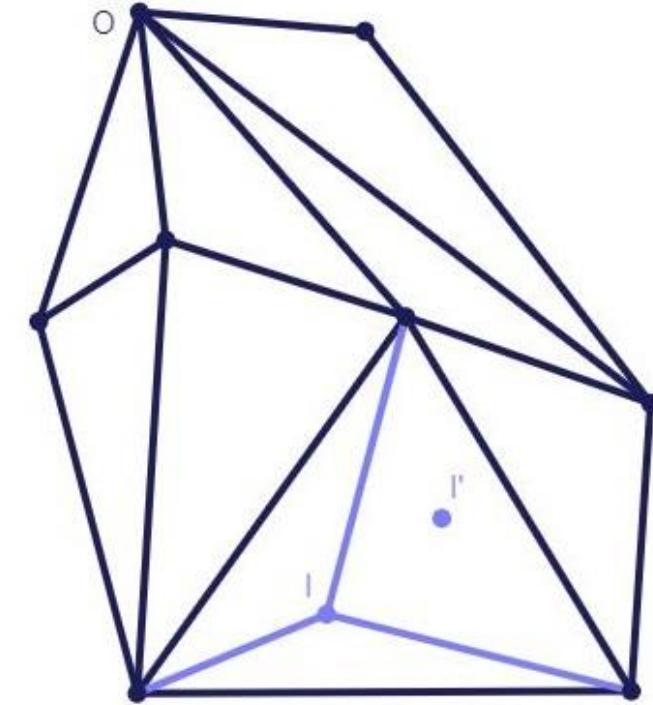


## void Mesh::AddInternalPoint(Point& point)

### Triangle

```
array<Point,3> vertices;  
vector<Triangle*> pointedTriangles;  
array<Triangle*,3> adjacentTriangles;
```

- Dai triangoli radice, creati aggiungendo punti esterni, si creano triangoli figli aggiungendo punti interni.
- Diventa fondamentale una struttura dei triangoli ad **albero** (grafo).

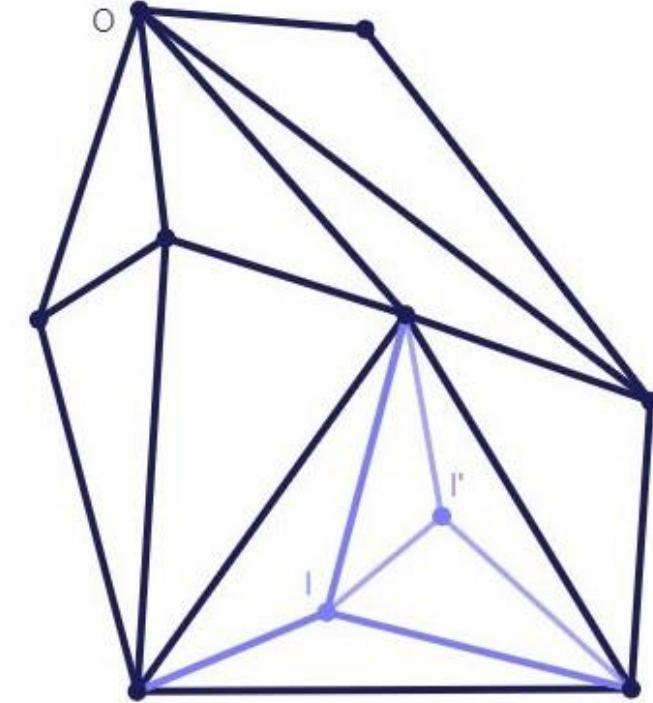


## void Mesh::AddInternalPoint(Point& point)

### Triangle

```
array<Point,3> vertices;  
vector<Triangle*> pointedTriangles;  
array<Triangle*,3> adjacentTriangles;
```

- Dai triangoli radice, creati aggiungendo punti esterni, si creano triangoli figli aggiungendo punti interni.
- Diventa fondamentale una struttura dei triangoli ad albero (grafo).

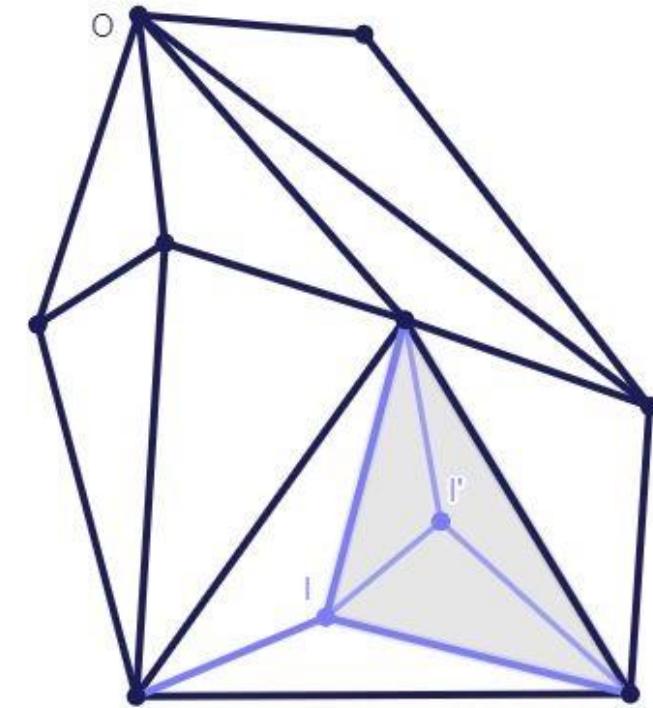


## void Mesh::AddInternalPoint(Point& point)

### Triangle

```
array<Point,3> vertices;  
vector<Triangle*> pointedTriangles;  
array<Triangle*,3> adjacentTriangles;
```

- Dai triangoli radice, creati aggiungendo punti esterni, si creano triangoli figli aggiungendo punti interni.
- Diventa fondamentale una struttura dei triangoli ad **albero** (grafo).

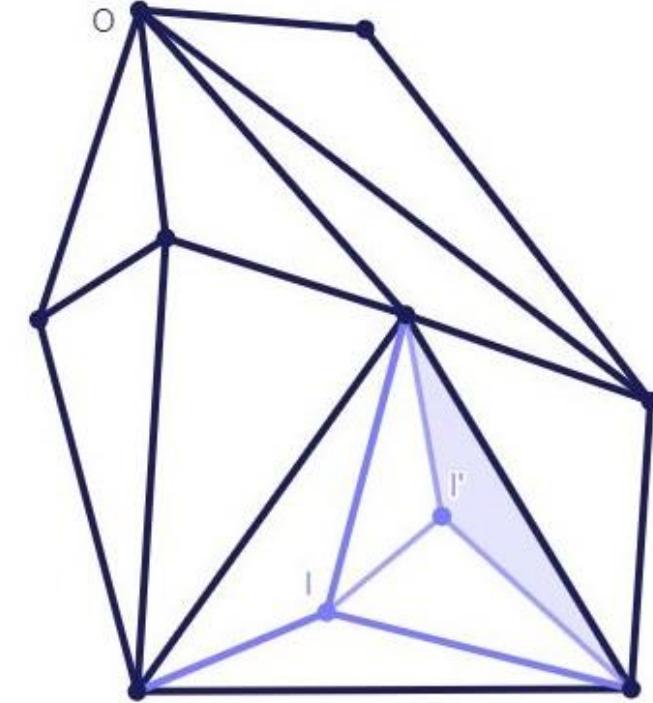


## void Mesh::AddInternalPoint(Point& point)

### Triangle

```
array<Point,3> vertices;  
vector<Triangle*> pointedTriangles;  
array<Triangle*,3> adjacentTriangles;
```

- Dai triangoli radice, creati aggiungendo punti esterni, si creano triangoli figli aggiungendo punti interni.
- Diventa fondamentale una struttura dei triangoli ad albero (grafo).

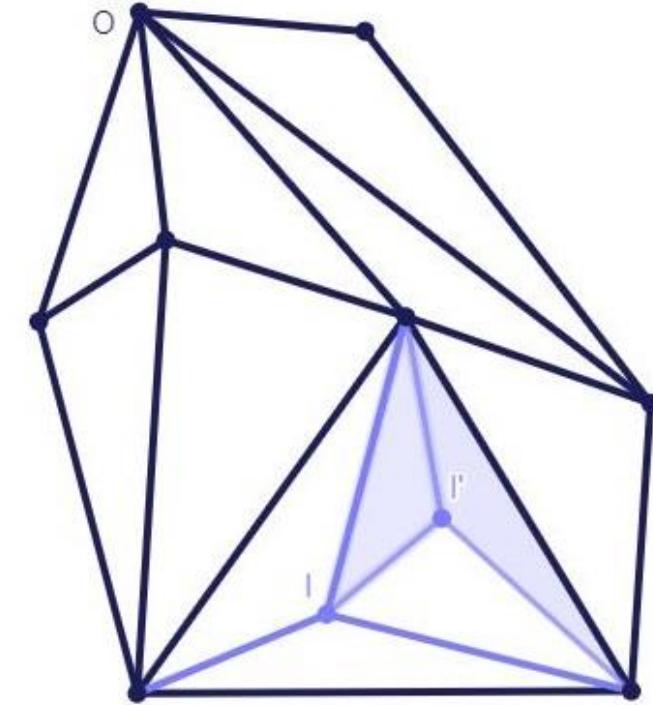


## void Mesh::AddInternalPoint(Point& point)

### Triangle

```
array<Point,3> vertices;  
vector<Triangle*> pointedTriangles;  
array<Triangle*,3> adjacentTriangles;
```

- Dai triangoli radice, creati aggiungendo punti esterni, si creano triangoli figli aggiungendo punti interni.
- Diventa fondamentale una struttura dei triangoli ad albero (grafo).

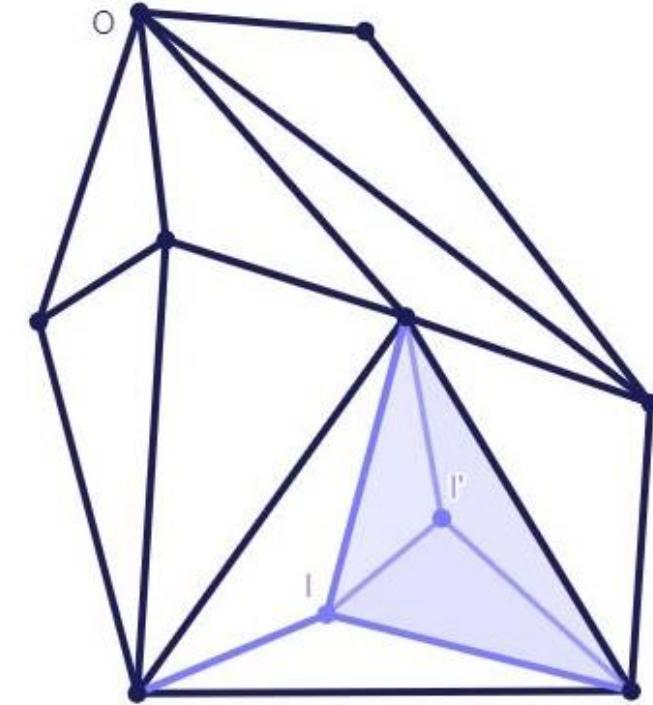


## void Mesh::AddInternalPoint(Point& point)

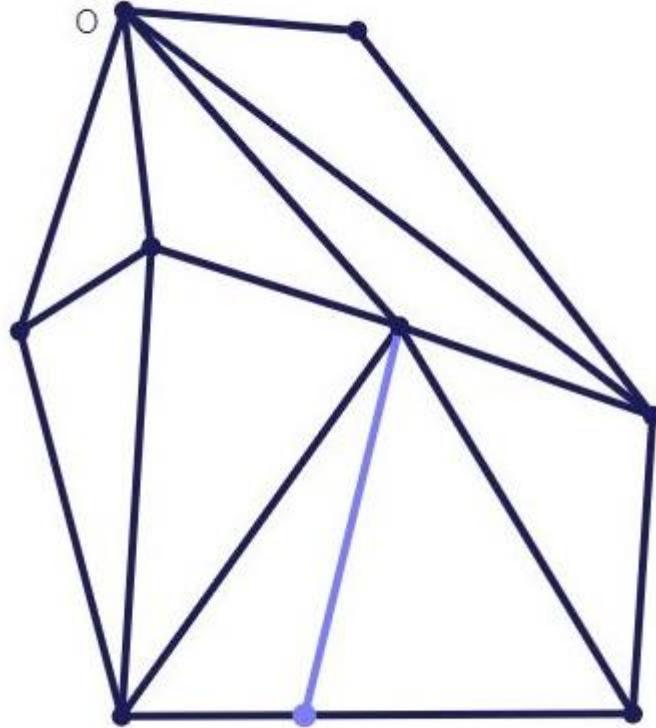
### Triangle

```
array<Point,3> vertices;  
vector<Triangle*> pointedTriangles;  
array<Triangle*,3> adjacentTriangles;
```

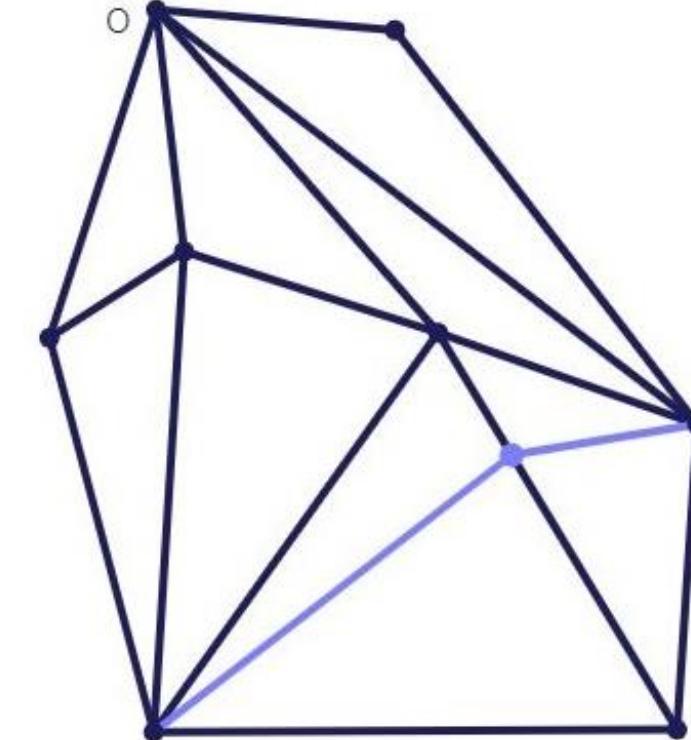
- Dai triangoli radice, creati aggiungendo punti esterni, si creano triangoli figli aggiungendo punti interni.
- Diventa fondamentale una struttura dei triangoli ad albero (grafo).



**void Mesh::AddSidePoint(Point& point)**



Intervento sul ConvexHull



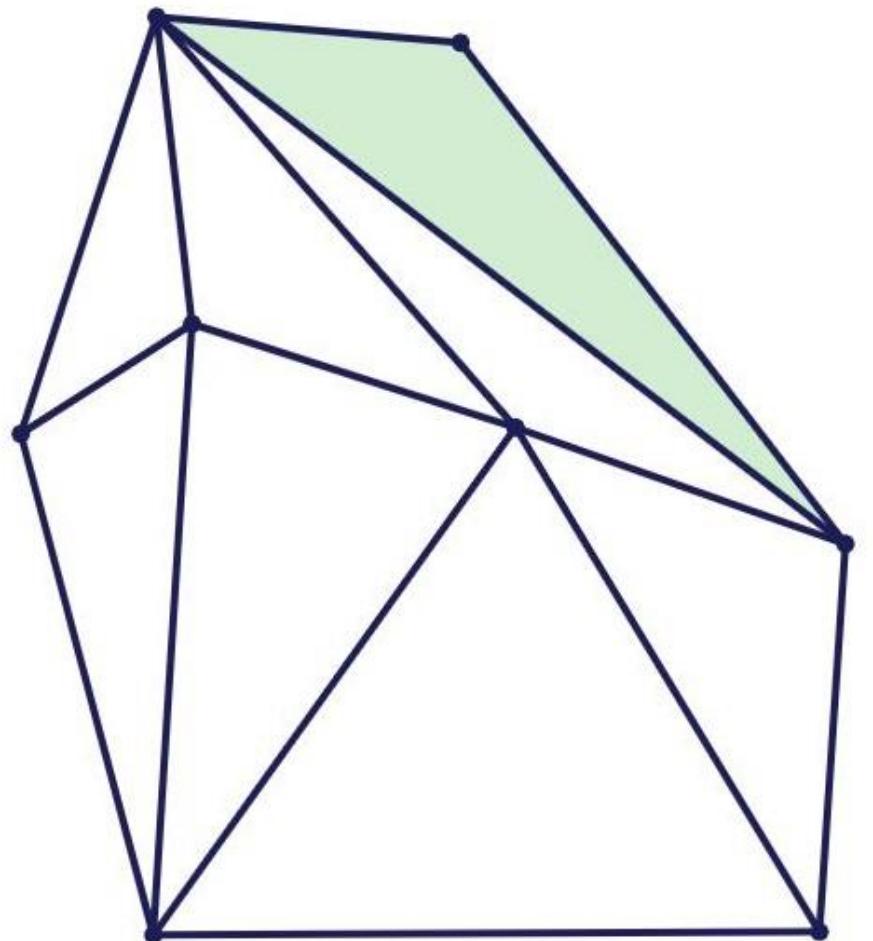
Intervento su figli di due triangoli adiacenti

- 
- Selezione del triangolo iniziale e creazione della mesh
  - Aggiunta di punti alla mesh
  - Soddisfacimento dell'ipotesi di Delaunay
    - Verifica della proprietà
    - Metodo Flip
    - Propagazione
  - Conclusioni e considerazioni

## DelaunayLibrary\

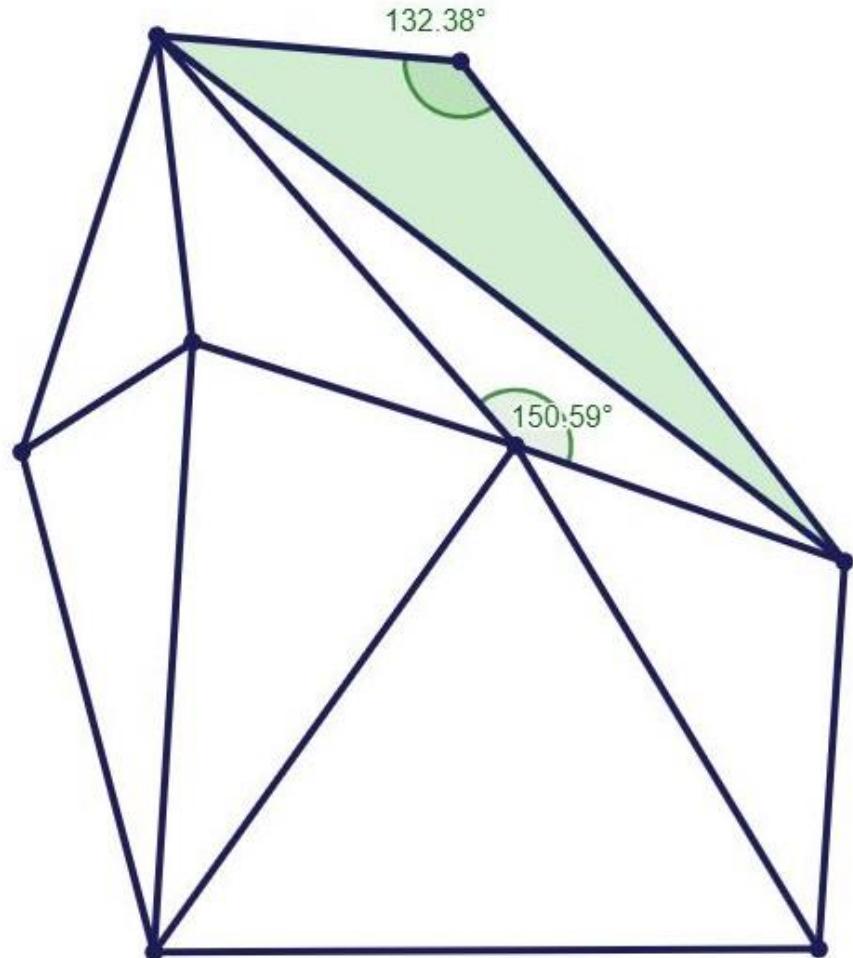
### C Triangle

- o array<Point,3> vertices
- o vector<Triangle\*> pointedTriangles
- o array<Triangle\*, 3> adjacentTriangles
- Triangle()
- Triangle(Point& a, Point& b, Point& c)
- array<Point,3> OrderVertices()
- int ContainsPoint(Point& point)
- Triangle\* FromRootToLeaf(Point& point);
- void SetAdjacentTriangle(Triangle& triangle1, Triangle\* triangle2, Point& tail, Point& head);
- array<Point\*,4> FindCommonEdge(Triangle& triangle1, Triangle& triangle2);
- **array<Triangle\*, 2> Flip(Triangle& triangle1, Triangle& triangle2);**
- void Adjourn(Triangle\* triangle new 1, Triangle\* triangolo1, Triangle\* triangolo2);
- bool DelaunayProperty(Triangle& triangle1, Triangle& triangle2)
- friend bool operator==(const Triangle& triangle1, const Triangle& triangle2)
- friend ostream& operator<<(ostream& os, const Triangle& triangle)



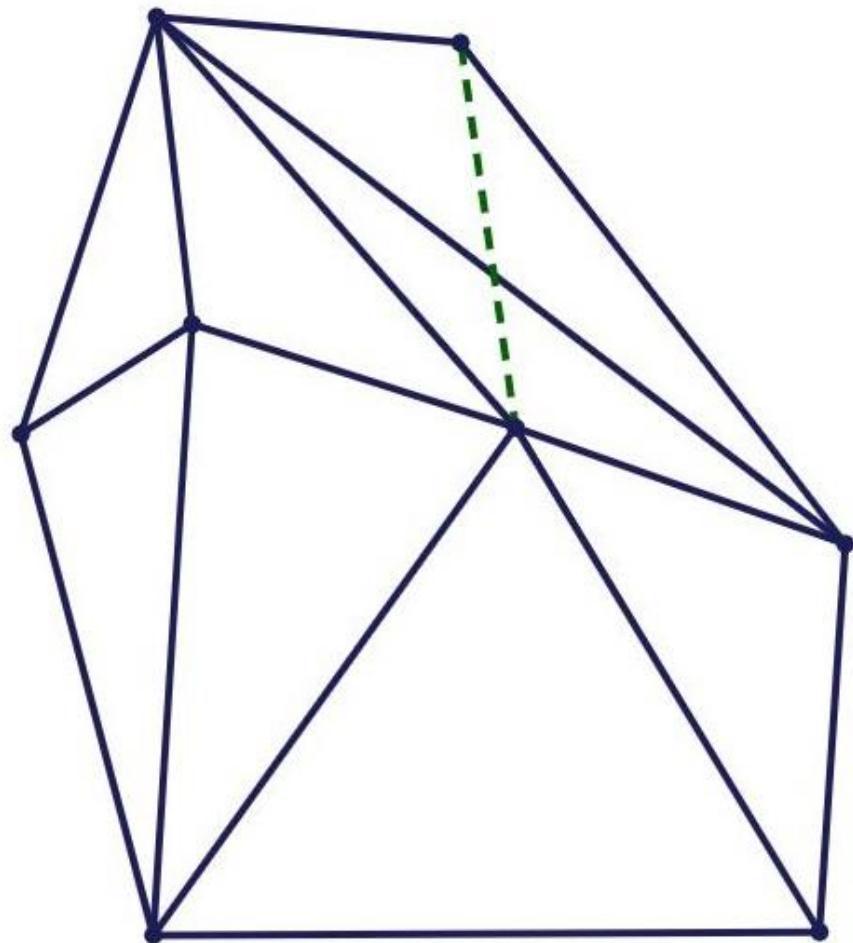
- Verifica dell’ipotesi di Delaunay tramite il calcolo del determinante.
- Triangoli possono non soddisfare l’ipotesi se condividono un lato con un nuovo triangolo o un triangolo flippato.
- Partenza da un vettore di puntatori a triangoli T e iterazione su ciascun T.
- Per ogni iterazione controllo l’ipotesi con ciascun triangolo adiacente a T.
- Se T flippa, ripeto ricorsivamente l’operazione sul vettore dei ‘triangoli’ adiacenti a T1 e poi T2, generati dal flip.

```
array<Triangle*, 2> Triangle::Flip(Triangle& T1, Triangle& T2);
```



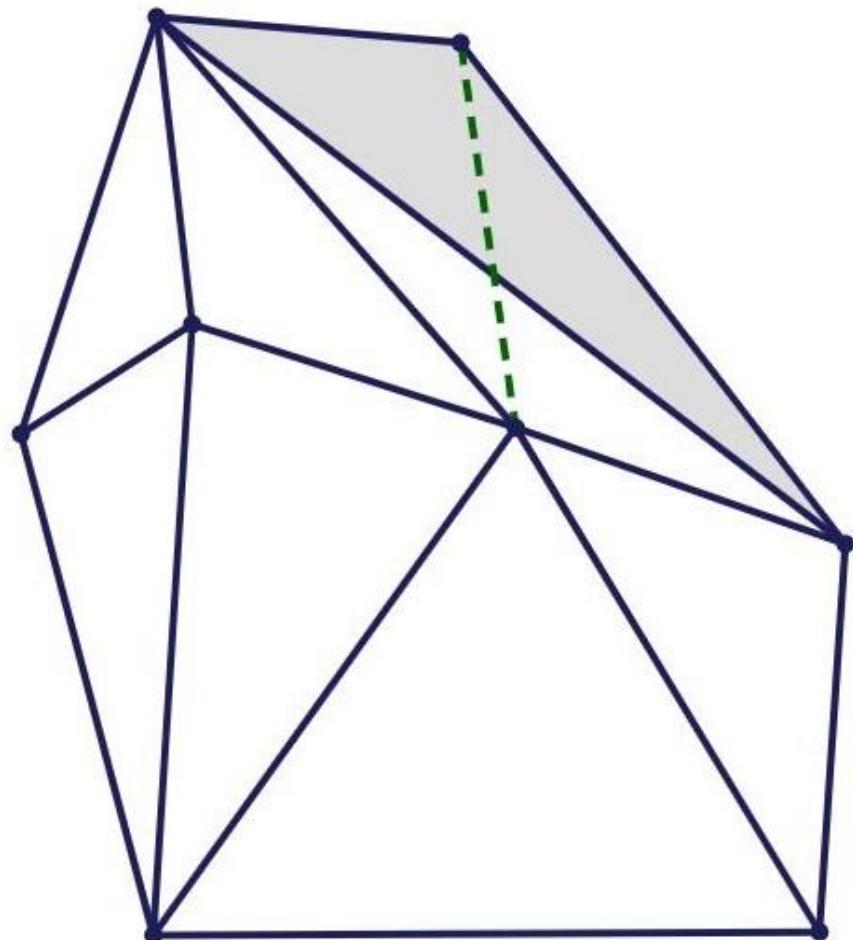
- Verifica dell'ipotesi di Delaunay tramite il calcolo del determinante.
- Triangoli possono non soddisfare l'ipotesi se condividono un lato con un nuovo triangolo o un triangolo flippato.
- Partenza da un vettore di puntatori a triangoli T e iterazione su ciascun T.
- Per ogni iterazione controllo l'ipotesi con ciascun triangolo adiacente a T.
- Se T flippa, ripeto ricorsivamente l'operazione sul vettore dei 'triangoli' adiacenti a T1 e poi T2, generati dal flip.

```
array<Triangle*, 2> Triangle::Flip(Triangle& T1, Triangle& T2);
```



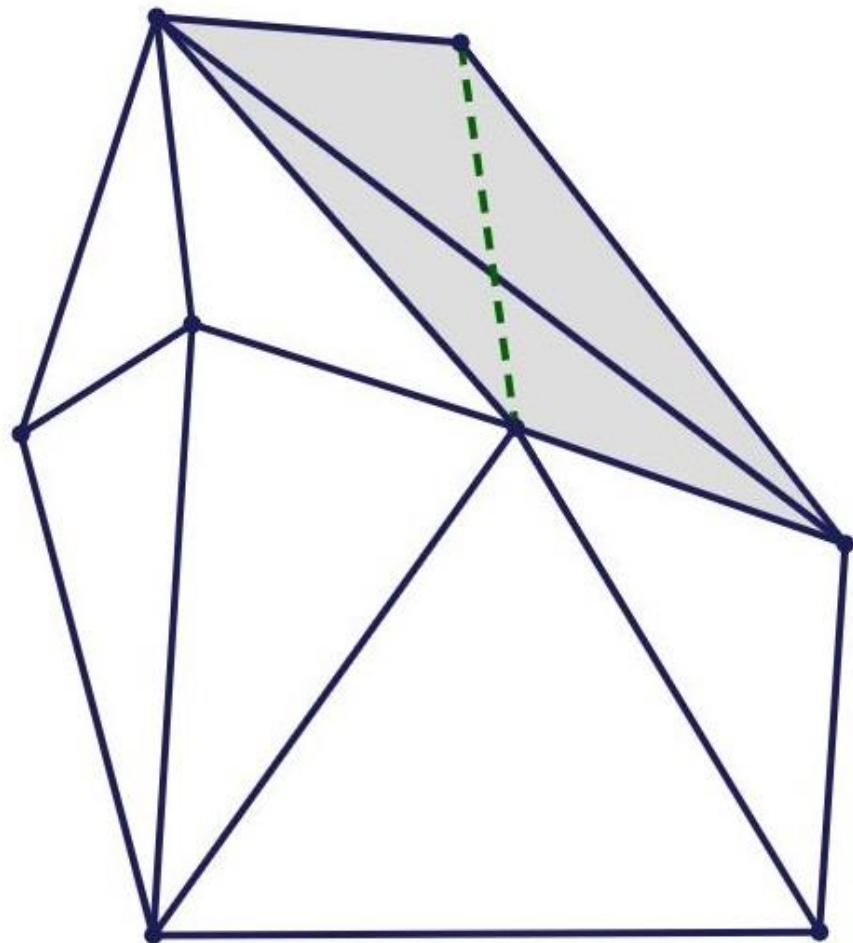
- Verifica dell'ipotesi di Delaunay tramite il calcolo del determinante.
- Triangoli possono non soddisfare l'ipotesi se condividono un lato con un nuovo triangolo o un triangolo flippato.
- Partenza da un vettore di puntatori a triangoli T e iterazione su ciascun T.
- Per ogni iterazione controllo l'ipotesi con ciascun triangolo adiacente a T.
- Se T flippa, ripeto ricorsivamente l'operazione sul vettore dei 'triangoli' adiacenti a T1 e poi T2, generati dal flip.

```
array<Triangle*, 2> Triangle::Flip(Triangle& T1, Triangle& T2);
```



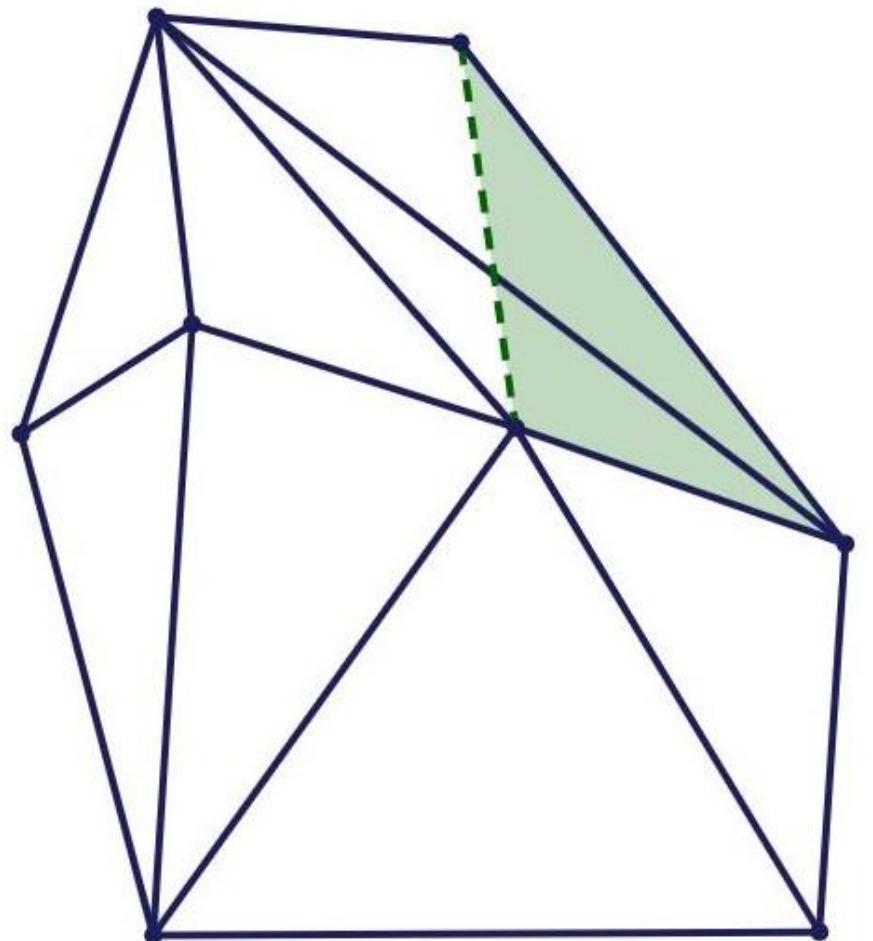
- Verifica dell'ipotesi di Delaunay tramite il calcolo del determinante.
- Triangoli possono non soddisfare l'ipotesi se condividono un lato con un nuovo triangolo o un triangolo flippato.
- Partenza da un vettore di puntatori a triangoli T e iterazione su ciascun T.
- Per ogni iterazione controllo l'ipotesi con ciascun triangolo adiacente a T.
- Se T flippa, ripeto ricorsivamente l'operazione sul vettore dei 'triangoli' adiacenti a T1 e poi T2, generati dal flip.

array<Triangle\*, 2> Triangle::Flip(Triangle& T1, Triangle& T2);



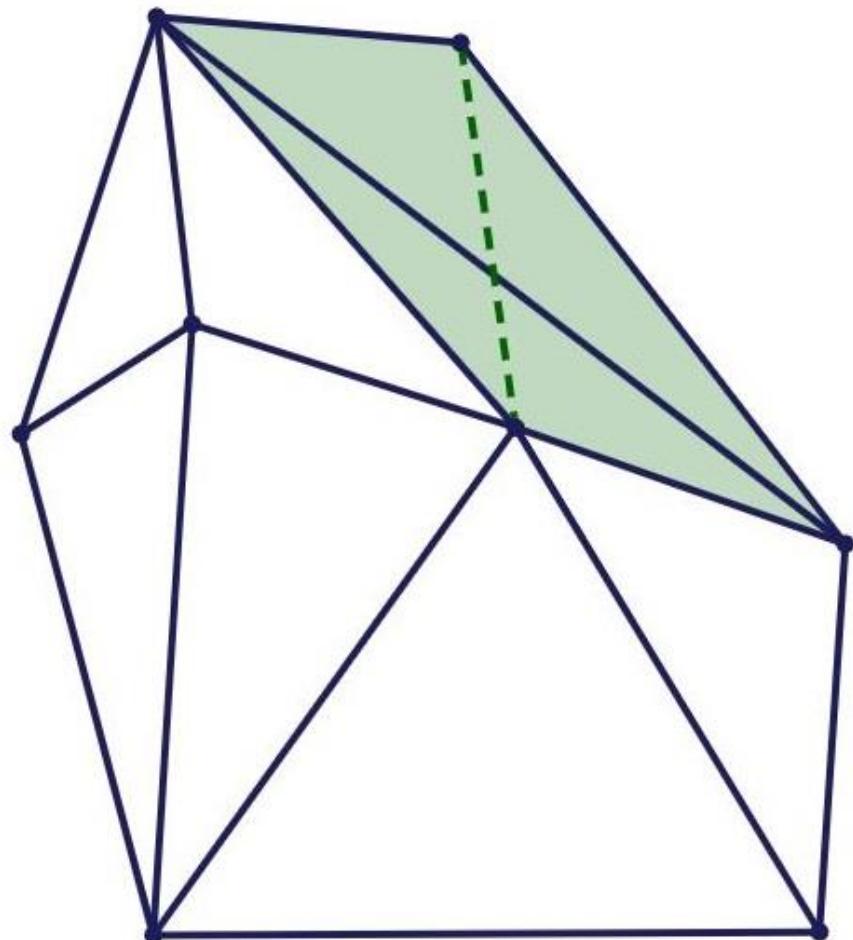
- Verifica dell'ipotesi di Delaunay tramite il calcolo del determinante.
- Triangoli possono non soddisfare l'ipotesi se condividono un lato con un nuovo triangolo o un triangolo flippato.
- Partenza da un vettore di puntatori a triangoli T e iterazione su ciascun T.
- Per ogni iterazione controllo l'ipotesi con ciascun triangolo adiacente a T.
- Se T flippa, ripeto ricorsivamente l'operazione sul vettore dei 'triangoli' adiacenti a T1 e poi T2, generati dal flip.

```
array<Triangle*, 2> Triangle::Flip(Triangle& T1, Triangle& T2);
```



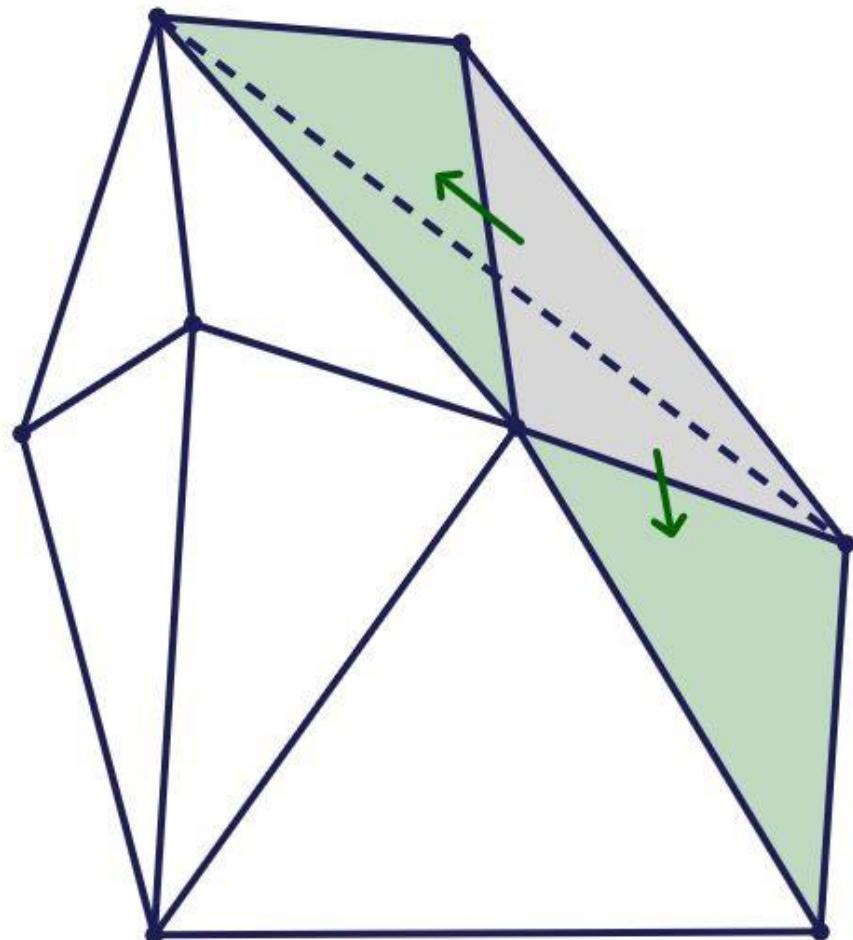
- Verifica dell'ipotesi di Delaunay tramite il calcolo del determinante.
- Triangoli possono non soddisfare l'ipotesi se condividono un lato con un nuovo triangolo o un triangolo flippato.
- Partenza da un vettore di puntatori a triangoli T e iterazione su ciascun T.
- Per ogni iterazione controllo l'ipotesi con ciascun triangolo adiacente a T.
- Se T flippa, ripeto ricorsivamente l'operazione sul vettore dei 'triangoli' adiacenti a T1 e poi T2, generati dal flip.

```
array<Triangle*, 2> Triangle::Flip(Triangle& T1, Triangle& T2);
```



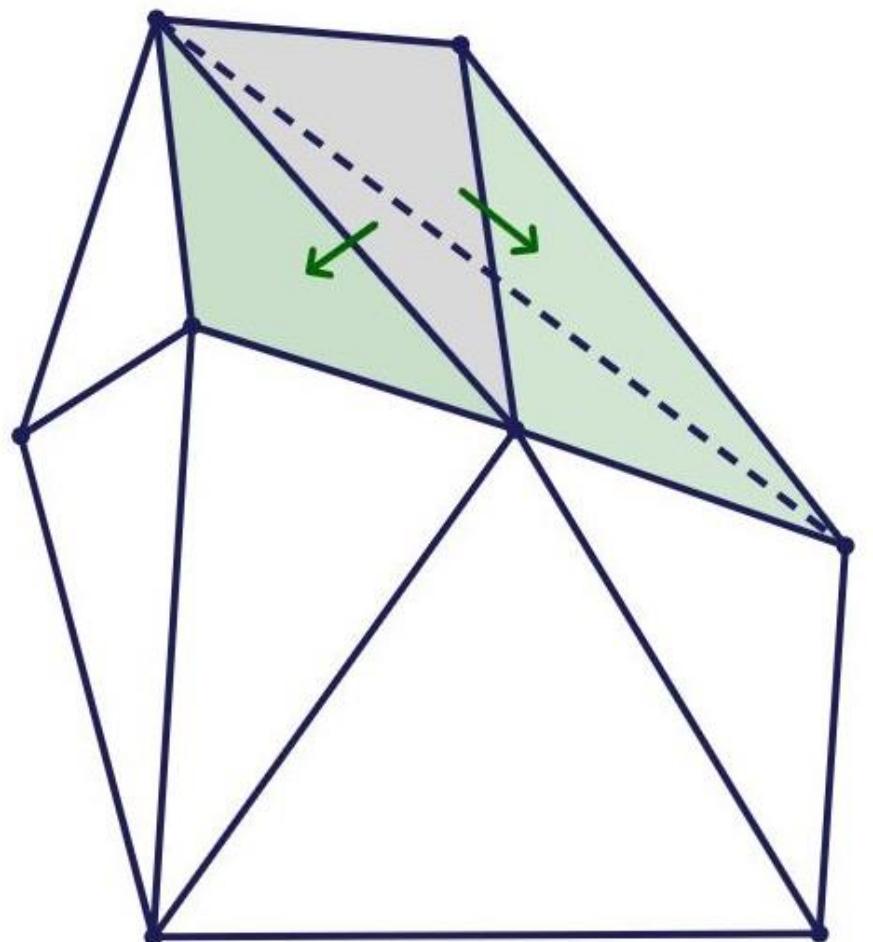
- Verifica dell'ipotesi di Delaunay tramite il calcolo del determinante.
- Triangoli possono non soddisfare l'ipotesi se condividono un lato con un nuovo triangolo o un triangolo flippato.
- Partenza da un vettore di puntatori a triangoli T e iterazione su ciascun T.
- Per ogni iterazione controllo l'ipotesi con ciascun triangolo adiacente a T.
- Se T flippa, ripeto ricorsivamente l'operazione sul vettore dei 'triangoli' adiacenti a T1 e poi T2, generati dal flip.

```
array<Triangle*, 2> Triangle::Flip(Triangle& T1, Triangle& T2);
```



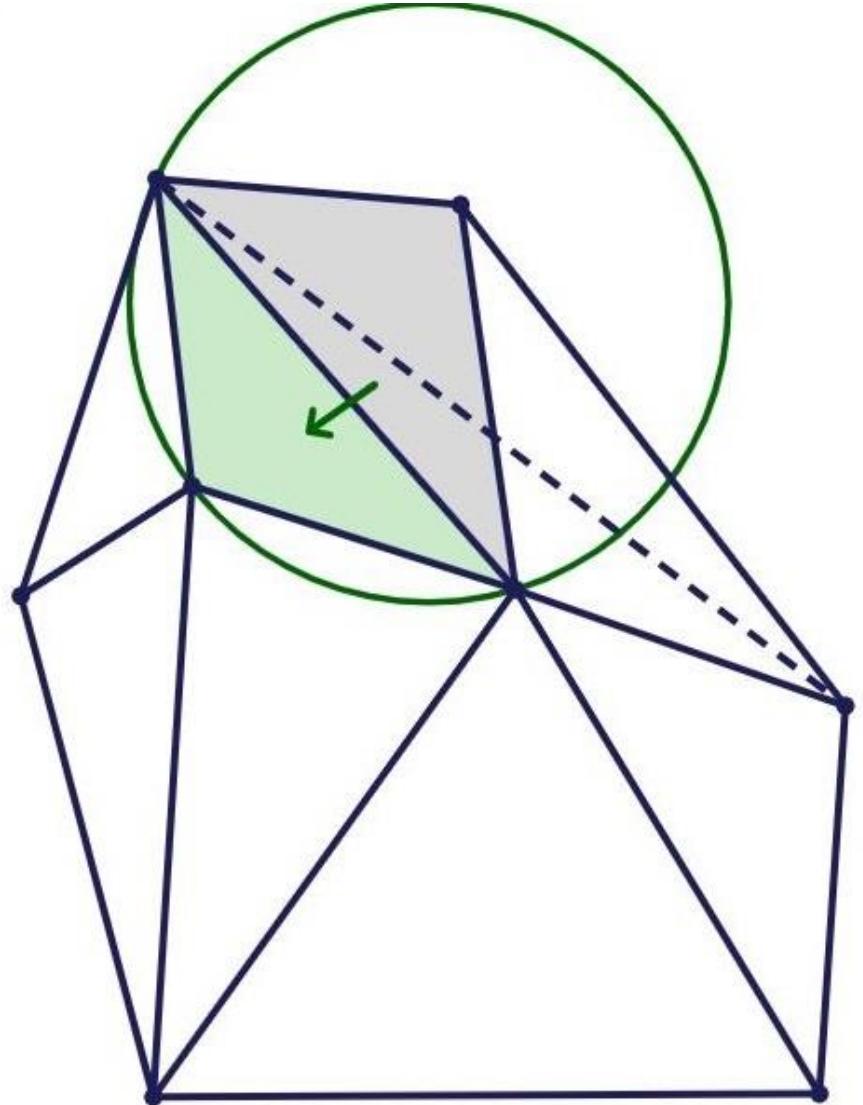
- Verifica dell'ipotesi di Delaunay tramite il calcolo del determinante.
- Triangoli possono non soddisfare l'ipotesi se condividono un lato con un nuovo triangolo o un triangolo flippato.
- Partenza da un vettore di puntatori a triangoli T e iterazione su ciascun T.
- Per ogni iterazione controllo l'ipotesi con ciascun triangolo adiacente a T.
- Se T flippa, ripeto ricorsivamente l'operazione sul vettore dei 'triangoli' adiacenti a T1 e poi T2, generati dal flip.

```
array<Triangle*, 2> Triangle::Flip(Triangle& T1, Triangle& T2);
```



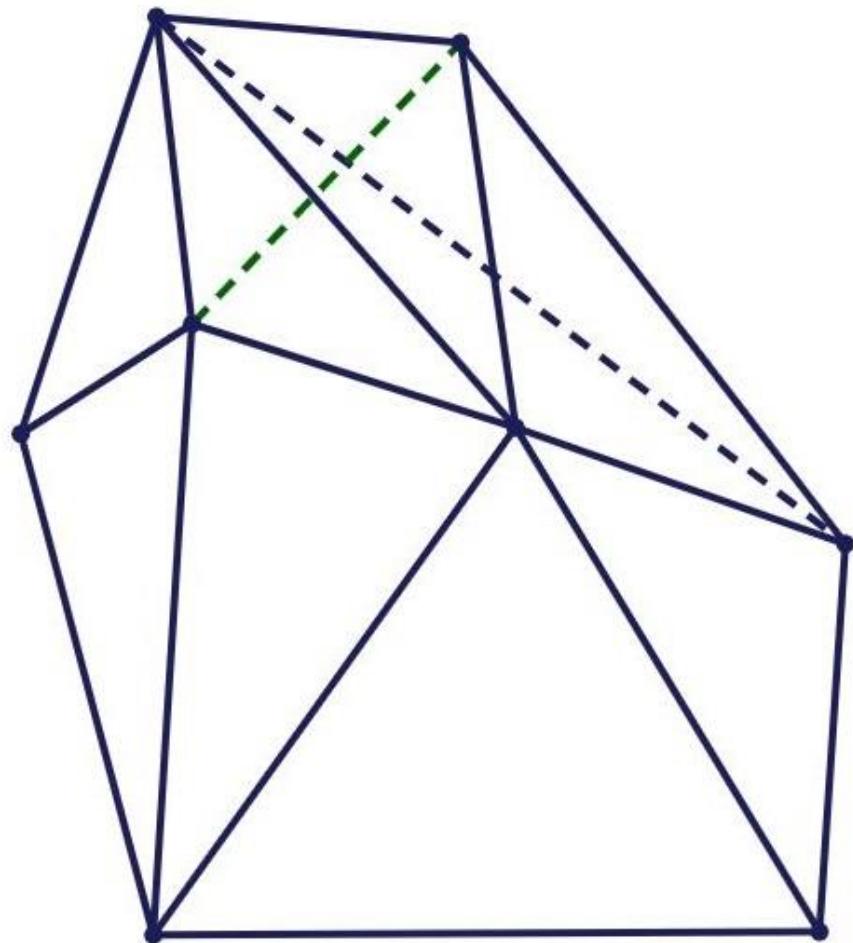
- Verifica dell'ipotesi di Delaunay tramite il calcolo del determinante.
- Triangoli possono non soddisfare l'ipotesi se condividono un lato con un nuovo triangolo o un triangolo flippato.
- Partenza da un vettore di puntatori a triangoli T e iterazione su ciascun T.
- Per ogni iterazione controllo l'ipotesi con ciascun triangolo adiacente a T.
- Se T flippa, ripeto ricorsivamente l'operazione sul vettore dei 'triangoli' adiacenti a T1 e poi T2, generati dal flip.

```
array<Triangle*, 2> Triangle::Flip(Triangle& T1, Triangle& T2);
```



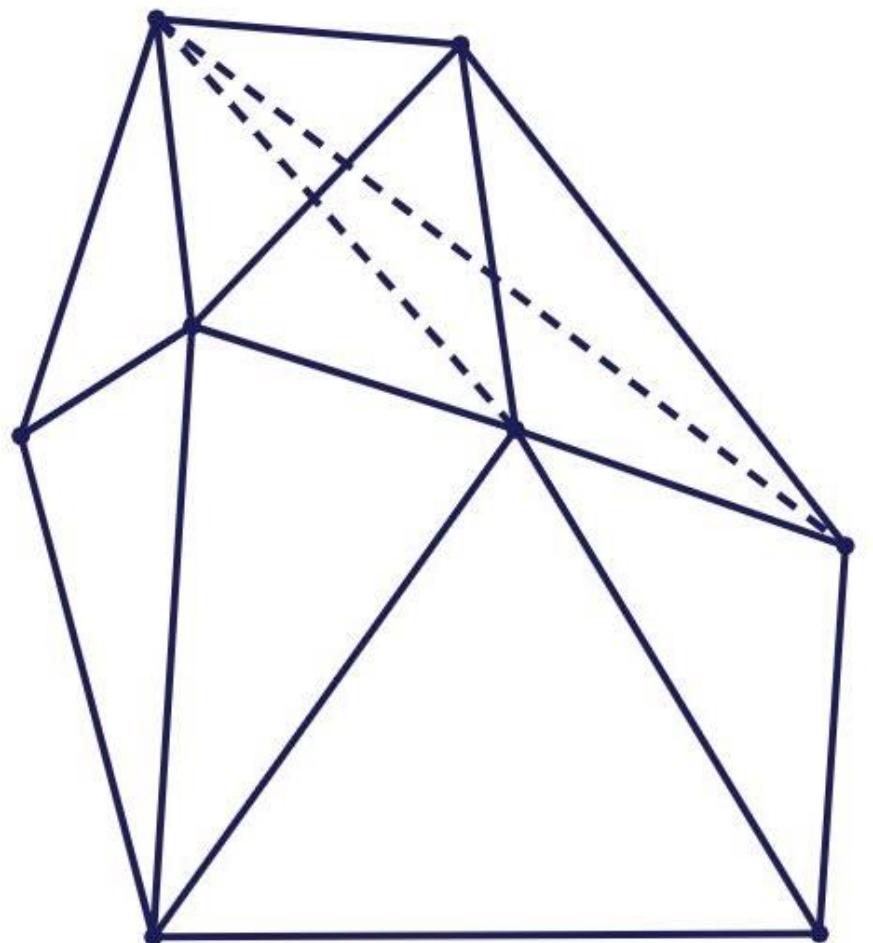
- Verifica dell'ipotesi di Delaunay tramite il calcolo del determinante.
- Triangoli possono non soddisfare l'ipotesi se condividono un lato con un nuovo triangolo o un triangolo flippato.
- Partenza da un vettore di puntatori a triangoli T e iterazione su ciascun T.
- Per ogni iterazione controllo l'ipotesi con ciascun triangolo adiacente a T.
- Se T flippa, ripeto ricorsivamente l'operazione sul vettore dei 'triangoli' adiacenti a T1 e poi T2, generati dal flip.

```
array<Triangle*, 2> Triangle::Flip(Triangle& T1, Triangle& T2);
```



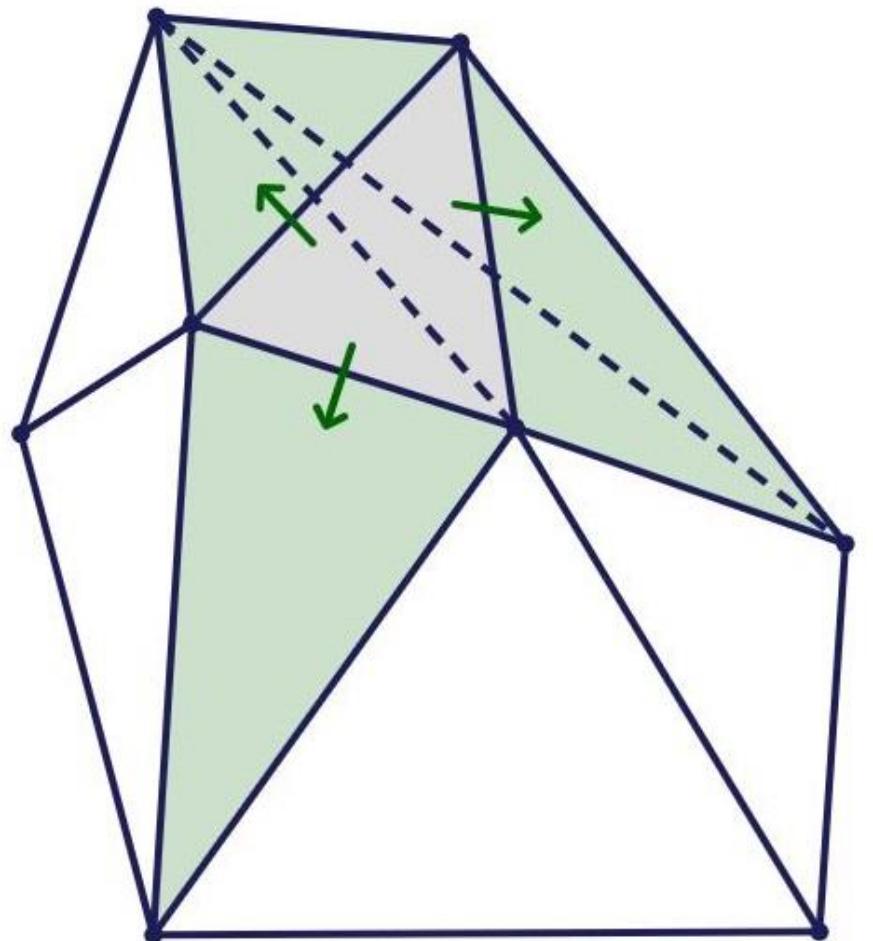
- Verifica dell'ipotesi di Delaunay tramite il calcolo del determinante.
- Triangoli possono non soddisfare l'ipotesi se condividono un lato con un nuovo triangolo o un triangolo flippato.
- Partenza da un vettore di puntatori a triangoli T e iterazione su ciascun T.
- Per ogni iterazione controllo l'ipotesi con ciascun triangolo adiacente a T.
- Se T flippa, ripeto ricorsivamente l'operazione sul vettore dei 'triangoli' adiacenti a T1 e poi T2, generati dal flip.

```
array<Triangle*, 2> Triangle::Flip(Triangle& T1, Triangle& T2);
```



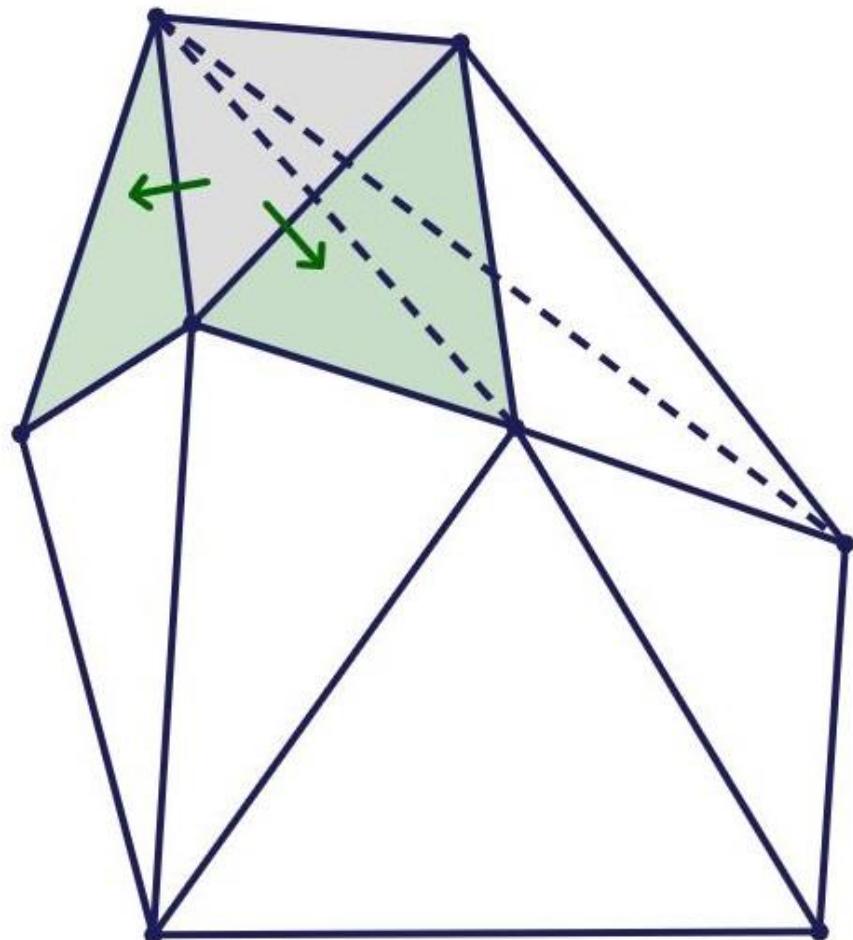
- Verifica dell’ipotesi di Delaunay tramite il calcolo del determinante.
- Triangoli possono non soddisfare l’ipotesi se condividono un lato con un nuovo triangolo o un triangolo flippato.
- Partenza da un vettore di puntatori a triangoli T e iterazione su ciascun T.
- Per ogni iterazione controllo l’ipotesi con ciascun triangolo adiacente a T.
- Se T flippa, ripeto ricorsivamente l’operazione sul vettore dei ‘triangoli’ adiacenti a T1 e poi T2, generati dal flip.

```
array<Triangle*, 2> Triangle::Flip(Triangle& T1, Triangle& T2);
```



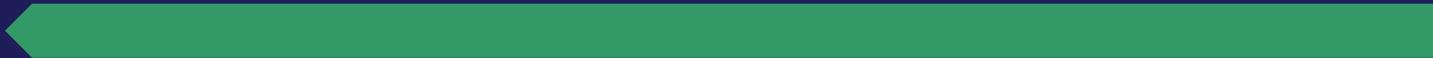
- Verifica dell'ipotesi di Delaunay tramite il calcolo del determinante.
- Triangoli possono non soddisfare l'ipotesi se condividono un lato con un nuovo triangolo o un triangolo flippato.
- Partenza da un vettore di puntatori a triangoli T e iterazione su ciascun T.
- Per ogni iterazione controllo l'ipotesi con ciascun triangolo adiacente a T.
- Se T flippa, ripeto ricorsivamente l'operazione sul vettore dei 'triangoli' adiacenti a T1 e poi T2, generati dal flip.

```
array<Triangle*, 2> Triangle::Flip(Triangle& T1, Triangle& T2);
```



- Verifica dell'ipotesi di Delaunay tramite il calcolo del determinante.
- Triangoli possono non soddisfare l'ipotesi se condividono un lato con un nuovo triangolo o un triangolo flippato.
- Partenza da un vettore di puntatori a triangoli T e iterazione su ciascun T.
- Per ogni iterazione controllo l'ipotesi con ciascun triangolo adiacente a T.
- Se T flippa, ripeto ricorsivamente l'operazione sul vettore dei 'triangoli' adiacenti a T1 e poi T2, generati dal flip.

```
array<Triangle*, 2> Triangle::Flip(Triangle& T1, Triangle& T2);
```

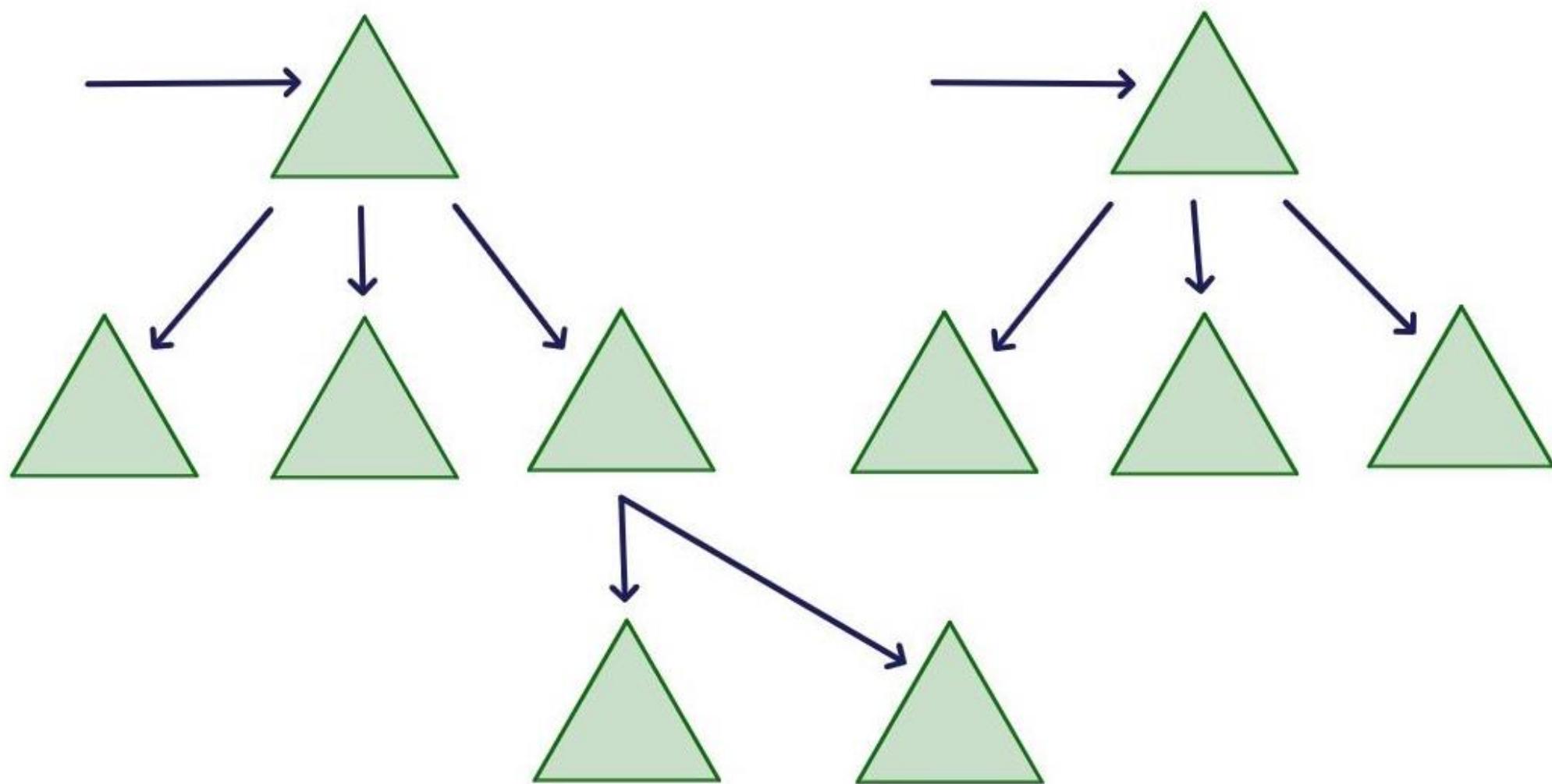
- 
- Costo computazionale **medio** dato dai flip generati a seguito dell'aggiunta di un punto interno:  $O(1)$
  - Costo **medio** per l'identificazione del triangolo foglia a cui un punto è interno:  $O(\log n)$



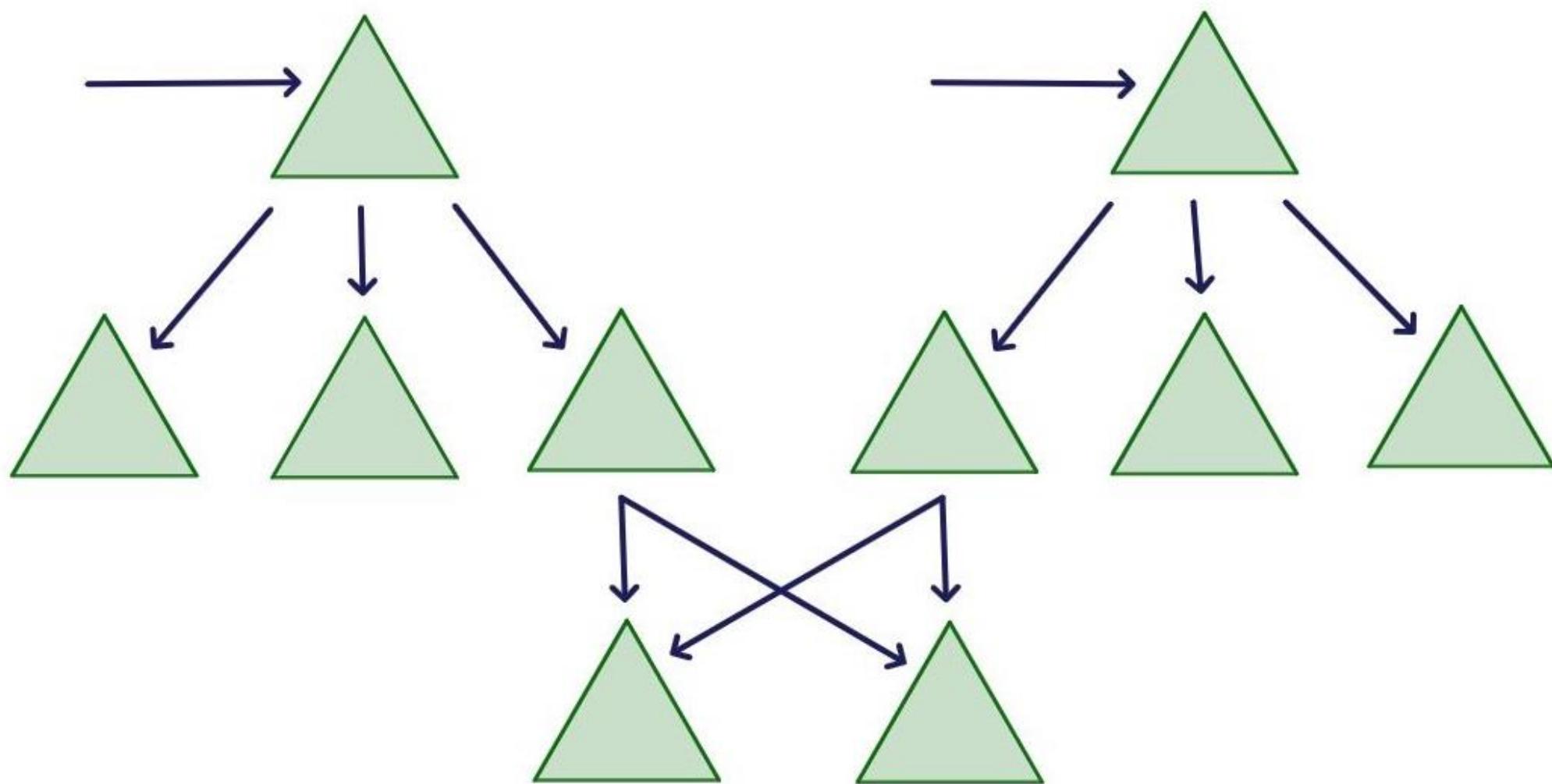
Se l'algoritmo prevedesse **SOLO** l'aggiunta di punti **INTERNI**  
il costo computazionale sarebbe in media:  $O(n \log n)$

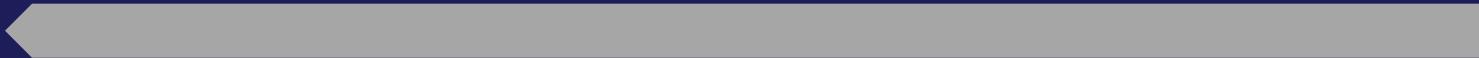
**Fonte:** Leonidas J. Guibas, Donald E. Knuth and Micha Sharir (1992)  
*Randomized Incremental Construction of Delaunay and Voronoi Diagrams*,  
pubblicato su *Algorithmica*, Springer

## SISTEMA DI PUNTATORI



## SISTEMA DI PUNTATORI





- Selezione del triangolo iniziale e creazione della mesh
- Aggiunta di punti alla mesh
- Soddisfacimento dell'ipotesi di Delaunay
- Conclusioni e considerazioni
  - Output
  - Risultati ottenuti
  - Costo computazionale complessivo
  - Confronto con algoritmi discostanti

## INPUT

Nome del file da inserire come  
argomento nella linea di comando

### Formattazione del file di input

```
Id X Y
0 0.00000000000000e+00 0.00000000000000e+00
1 1.00000000000000e+00 0.00000000000000e+00
2 1.00000000000000e+00 1.00000000000000e+00
3 0.00000000000000e+00 1.00000000000000e+00
4 5.00000000000000e-01 5.00000000000000e-01
5 0.00000000000000e+00 5.00000000000000e-01
6 5.00000000000000e-01 0.00000000000000e+00
7 1.00000000000000e+00 5.00000000000000e-01
8 5.00000000000000e-01 1.00000000000000e+00
9 7.50000000000000e-01 2.50000000000000e-01
10 2.50000000000000e-01 7.50000000000000e-01
11 7.50000000000000e-01 7.50000000000000e-01
12 5.00000000000000e-01 2.50000000000000e-01
13 7.50000000000000e-01 2.50000000000000e-01
```

# OUTPUT

## outputEdges

File di Output ordinato

LATI TRIANGOLAZIONE DI DELAUNAY

0. (0.074606,0.520781) (0.108849,0.116148)  
1. (0.074606,0.520781) (0.130653,0.688817)  
2. (0.074606,0.520781) (0.135233,0.618546)  
3. (0.074606,0.520781) (0.372270,0.523817)  
4. (0.074606,0.520781) (0.400688,0.209773)  
5. (0.108849,0.116148) (0.190221,0.087561)  
6. (0.108849,0.116148) (0.400688,0.209773)  
7. (0.130653,0.688817) (0.135233,0.618546)  
8. (0.130653,0.688817) (0.372270,0.523817)  
9. (0.130653,0.688817) (0.888792,0.870320)  
10. (0.135233,0.618546) (0.372270,0.523817)  
11. (0.190221,0.087561) (0.400688,0.209773)  
12. (0.190221,0.087561) (0.789261,0.403304)  
13. (0.372270,0.523817) (0.400688,0.209773)  
14. (0.372270,0.523817) (0.474522,0.280555)  
15. (0.372270,0.523817) (0.789261,0.403304)  
16. (0.372270,0.523817) (0.888792,0.870320)  
17. (0.400688,0.209773) (0.474522,0.280555)  
18. (0.400688,0.209773) (0.789261,0.403304)  
19. (0.474522,0.280555) (0.789261,0.403304)  
20. (0.789261,0.403304) (0.888792,0.870320)

## outputEdges\_Geogebra

File di Output per la visualizzazione su Geogebra

PUNTI

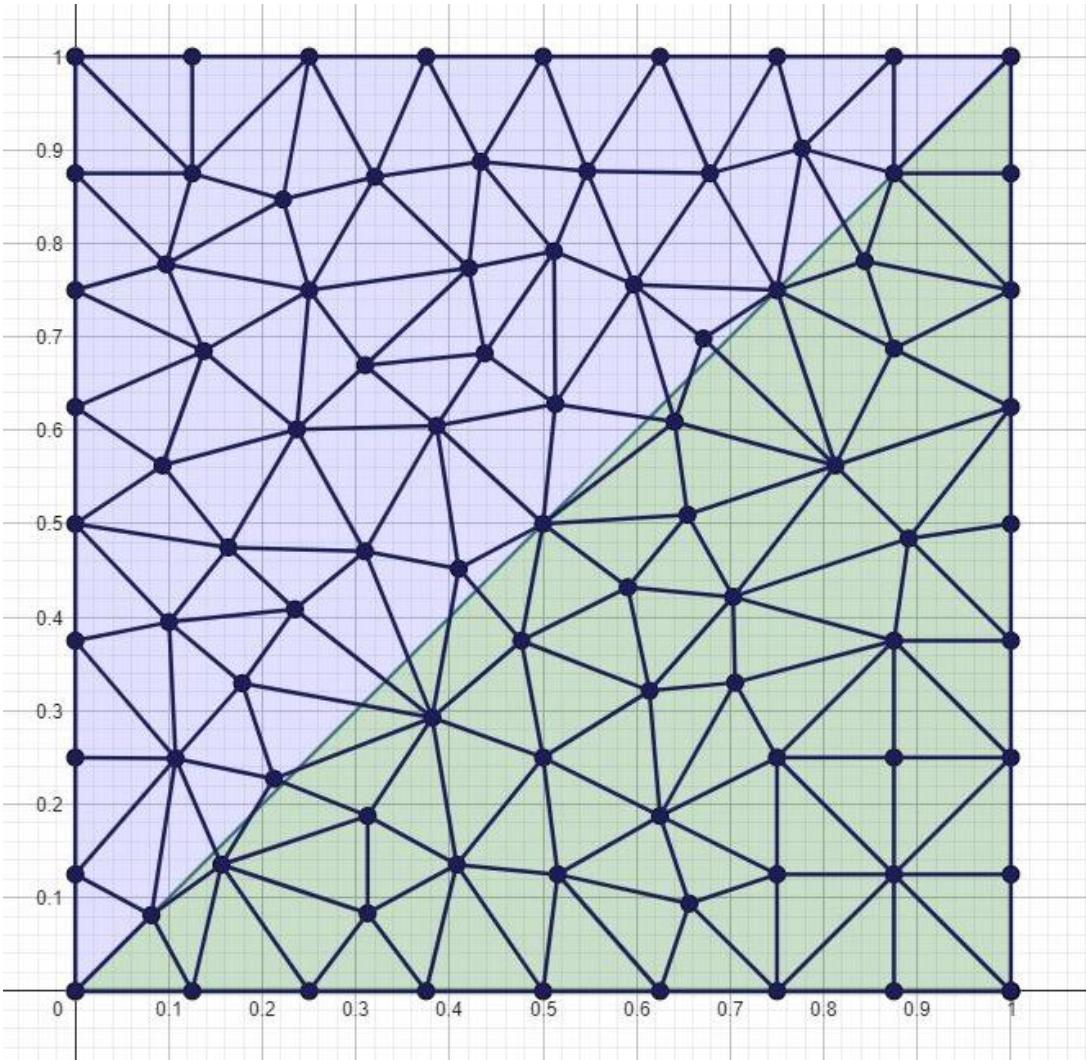
{(0.789261,0.403304),(0.888792,0.870320),(0.135233,0.618546),  
(0.074606,0.520781),(0.190221,0.087561),(0.789261,0.403304),  
(0.130653,0.688817),(0.474522,0.280555),(0.372270,0.523817),  
(0.108849,0.116148),(0.400688,0.209773)}

LATI

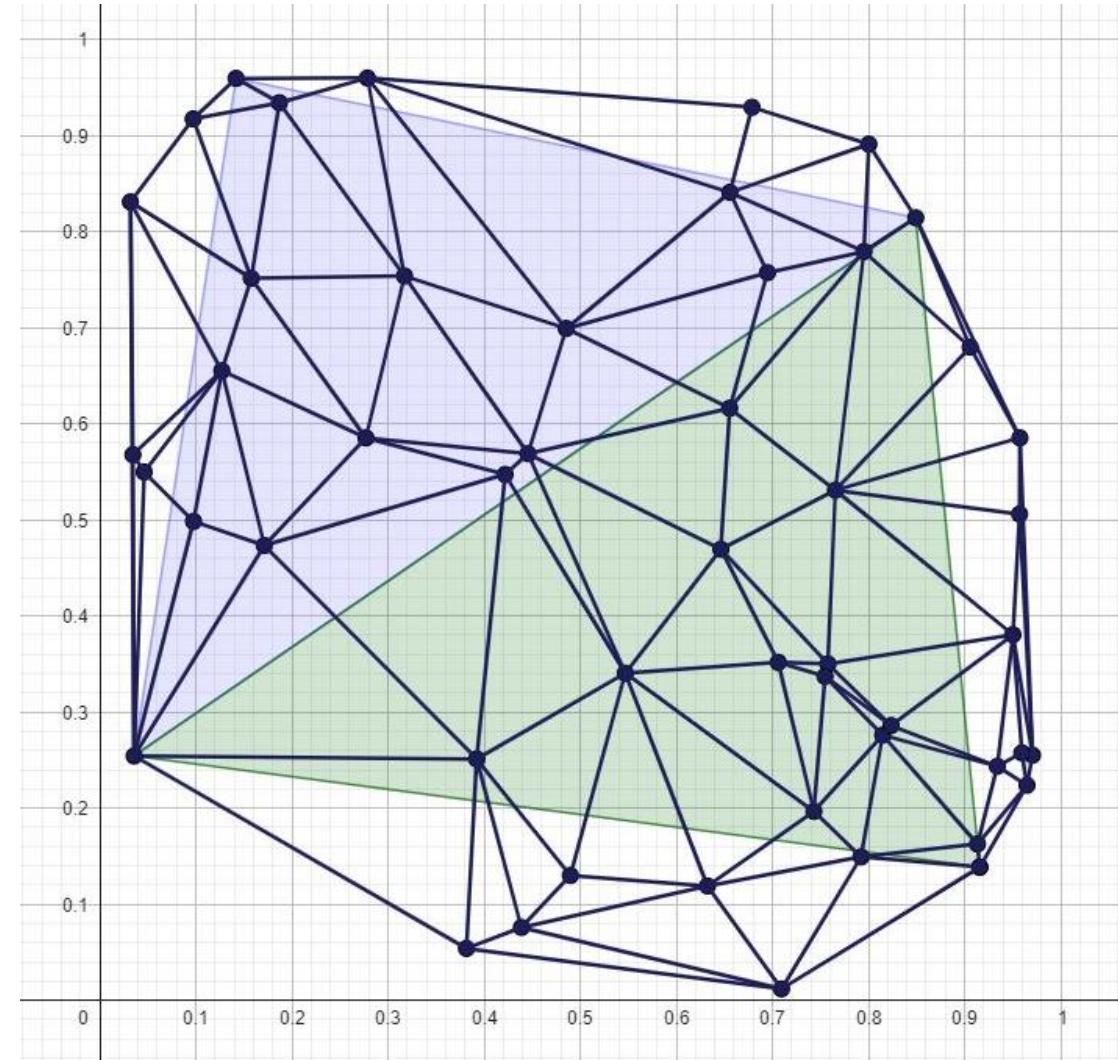
{Segmento((0.074606,0.520781),(0.108849,0.116148)),Segmento  
((0.074606,0.520781),(0.130653,0.688817)),Segmento((0.074606,0.520781),  
(0.135233,0.618546)),Segmento((0.074606,0.520781),(0.372270,0.523817)),  
Segmento((0.074606,0.520781),(0.400688,0.209773)),Segmento  
((0.108849,0.116148),(0.190221,0.087561)),Segmento((0.108849,0.116148),  
(0.400688,0.209773)),Segmento((0.130653,0.688817),(0.135233,0.618546)),  
Segmento((0.130653,0.688817),(0.372270,0.523817)),Segmento  
((0.130653,0.688817),(0.888792,0.870320)),Segmento((0.135233,0.618546),  
(0.372270,0.523817)),Segmento((0.190221,0.087561),(0.400688,0.209773)),  
Segmento((0.190221,0.087561),(0.789261,0.403304)),Segmento  
((0.372270,0.523817),(0.400688,0.209773)),Segmento((0.372270,0.523817),  
(0.474522,0.280555)),Segmento((0.372270,0.523817),(0.789261,0.403304)),  
Segmento((0.372270,0.523817),(0.888792,0.870320)),Segmento  
((0.400688,0.209773),(0.474522,0.280555)),Segmento((0.400688,0.209773),  
(0.789261,0.403304)),Segmento((0.474522,0.280555),(0.789261,0.403304)),  
Segmento((0.789261,0.403304),(0.888792,0.870320))}

# RISULTATI

Test1

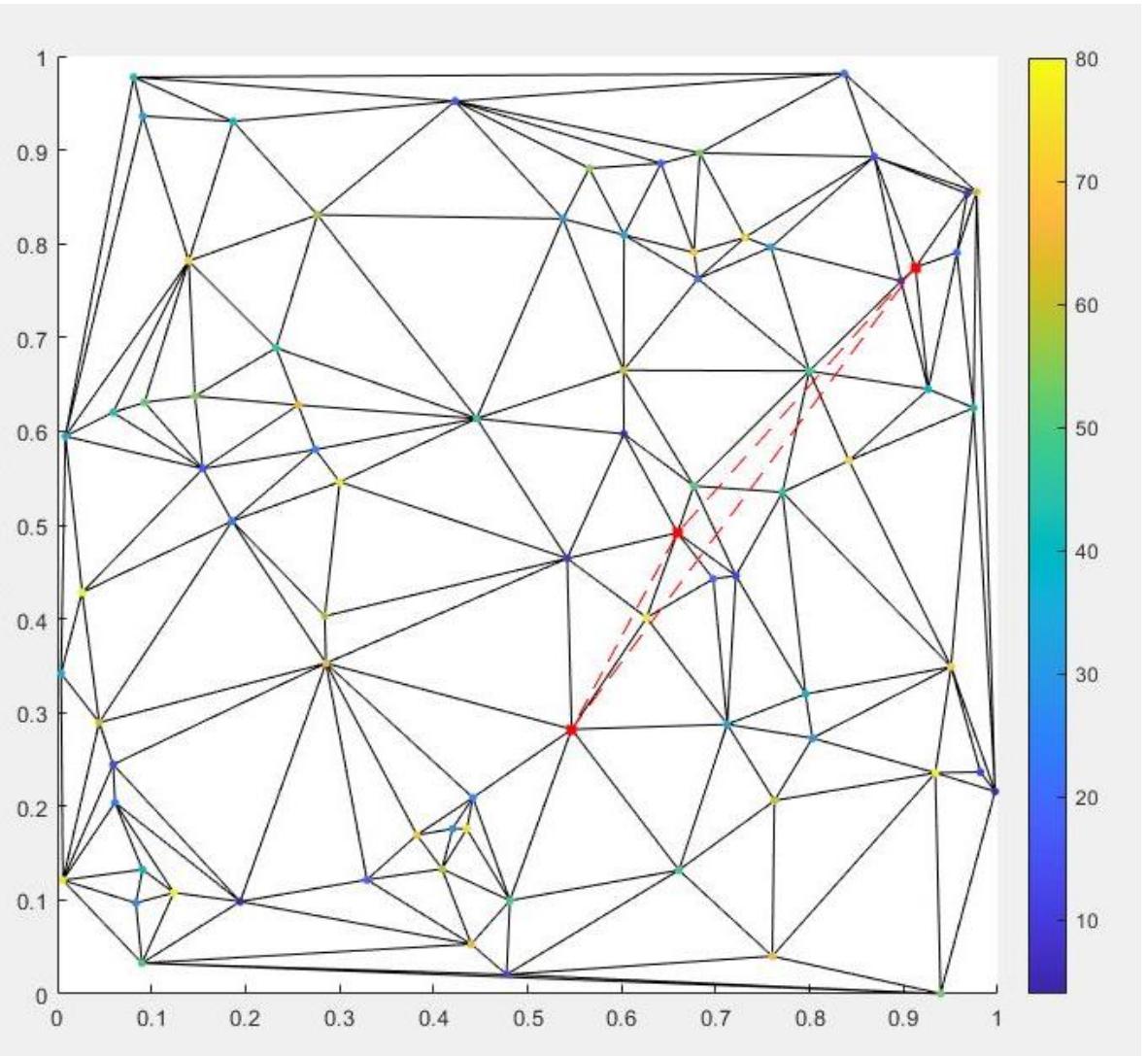
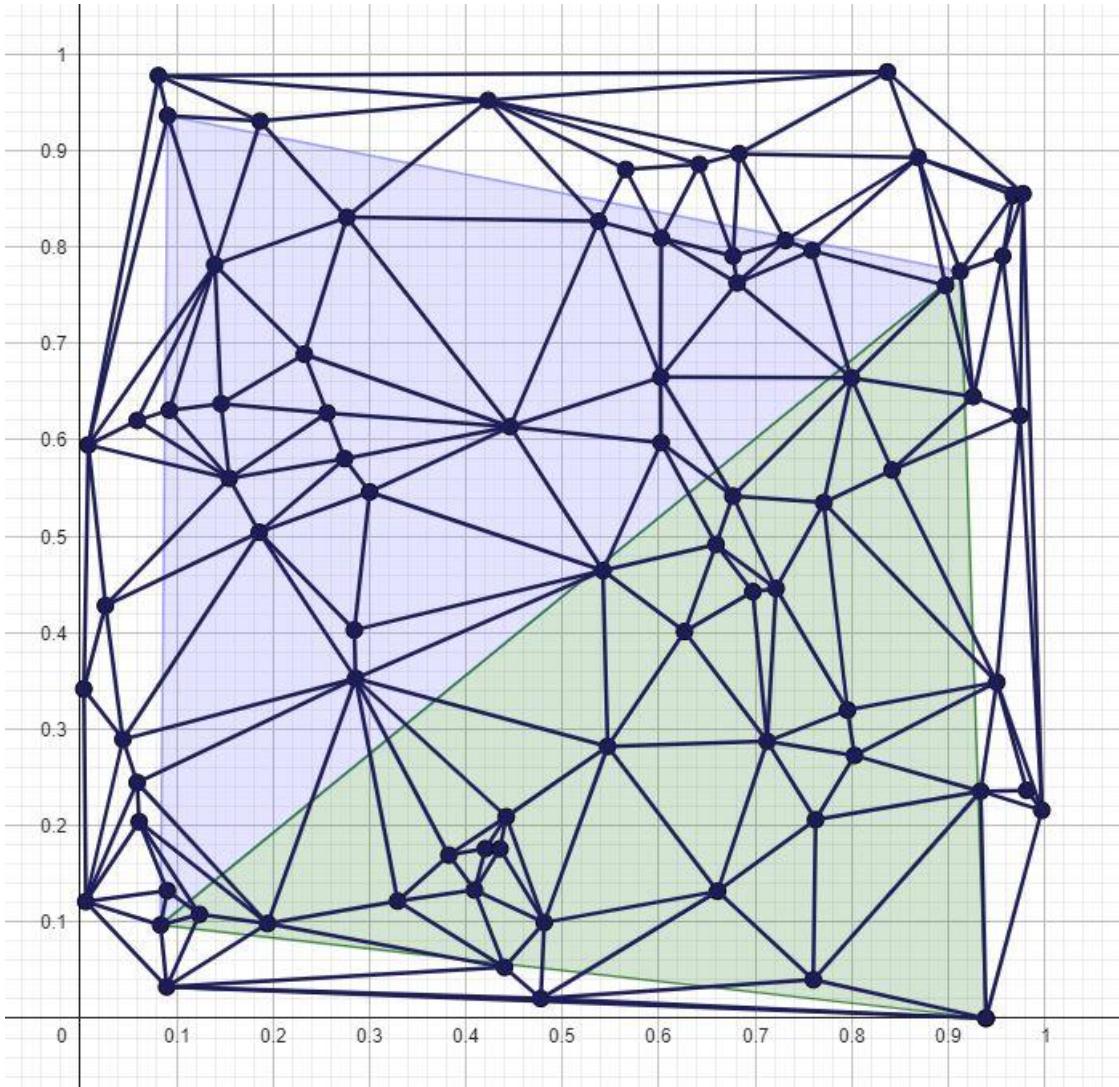


Test2



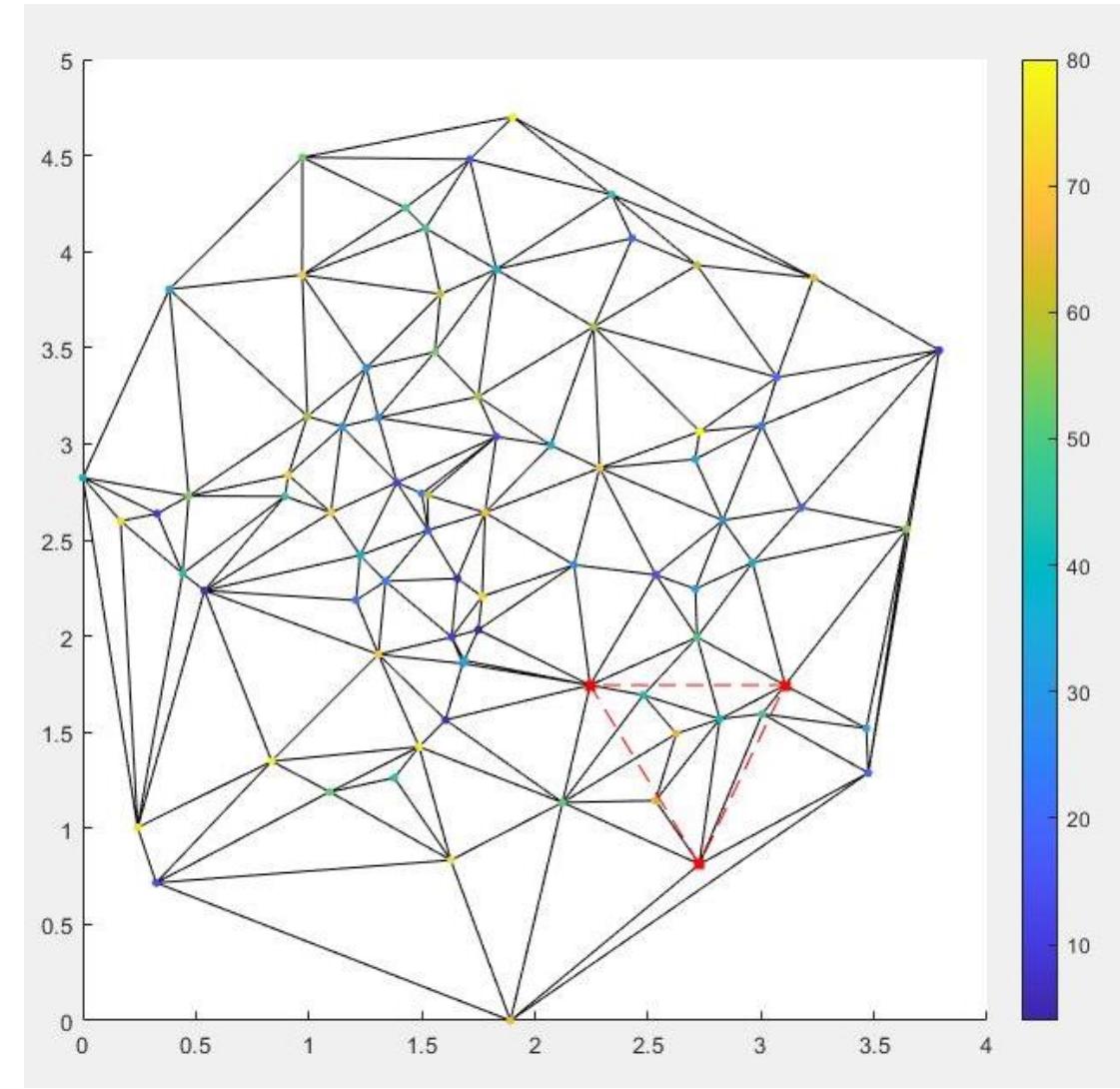
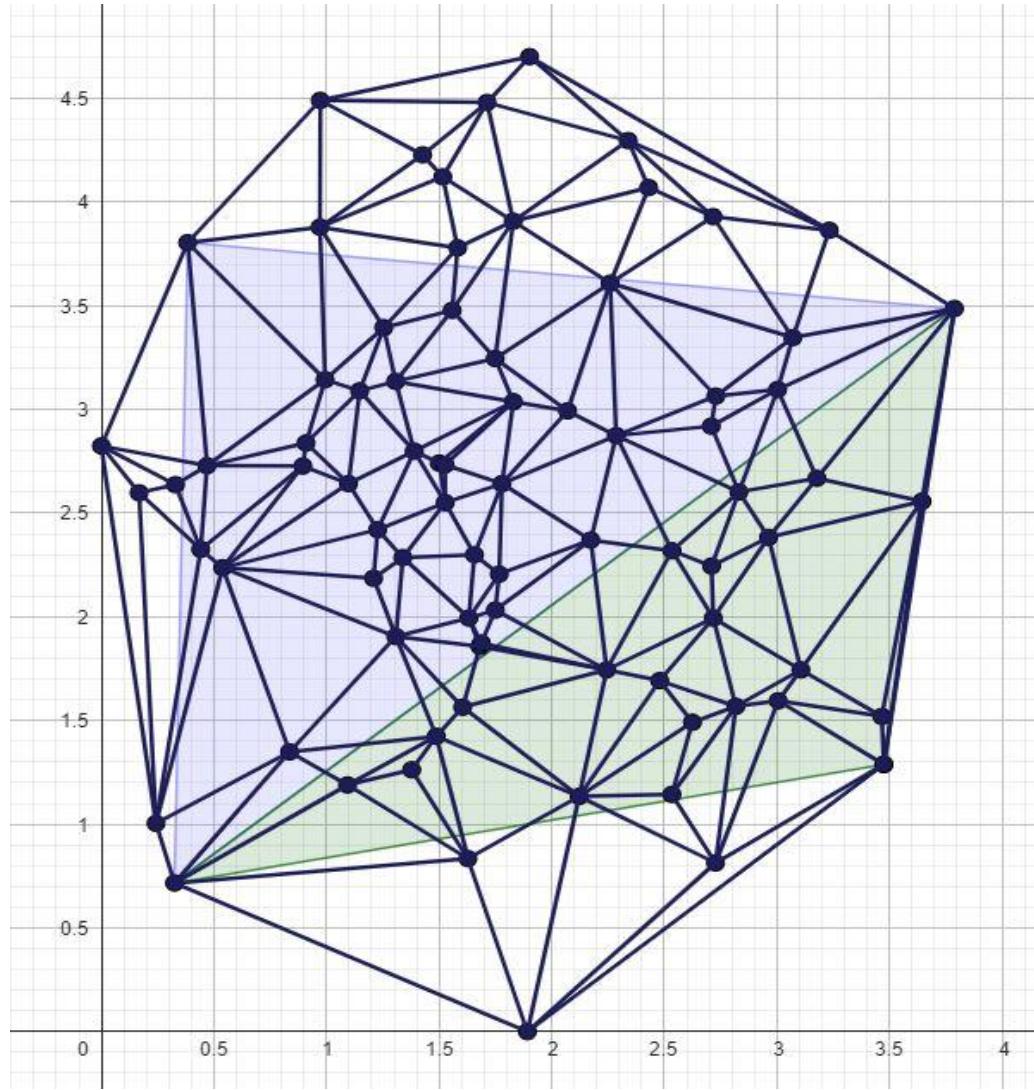
Numero di punti: 80  
Distribuzione: Uniforme

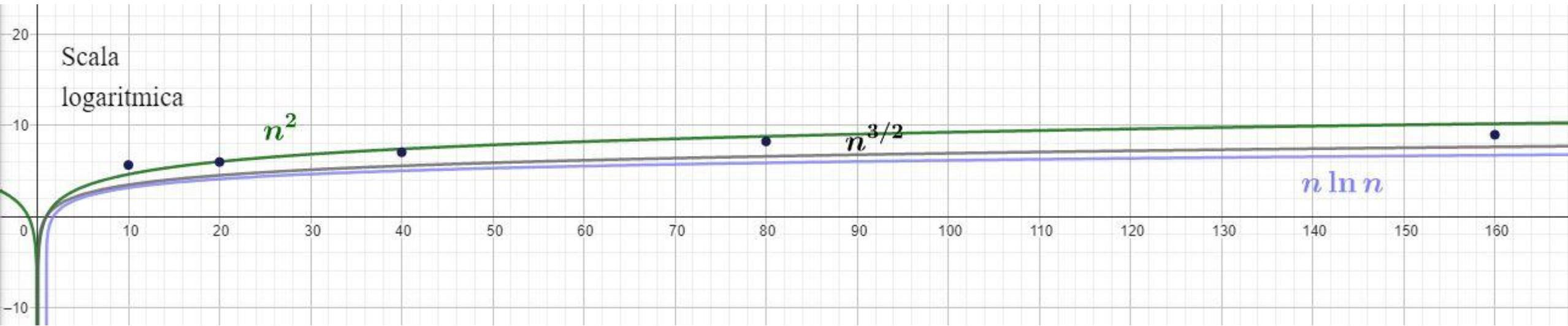
## RISULTATI



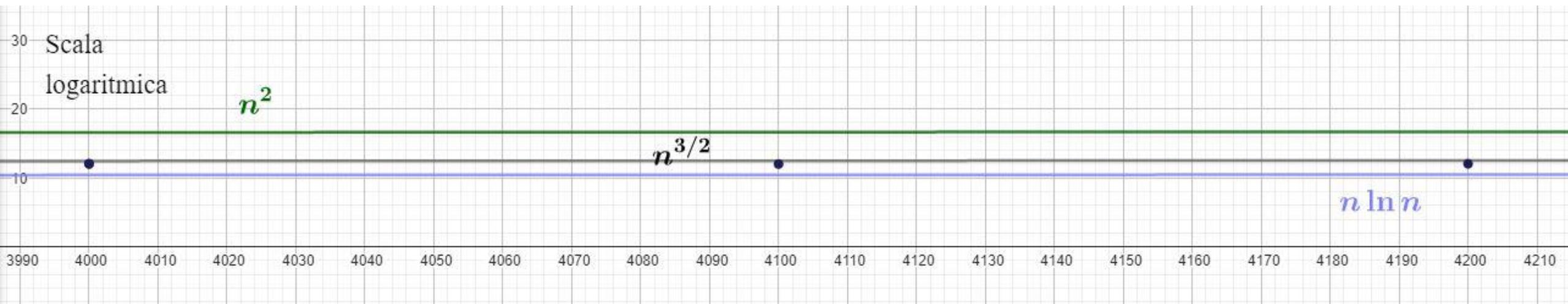
Numero di punti: 80  
Distribuzione: Gaussiana

## RISULTATI



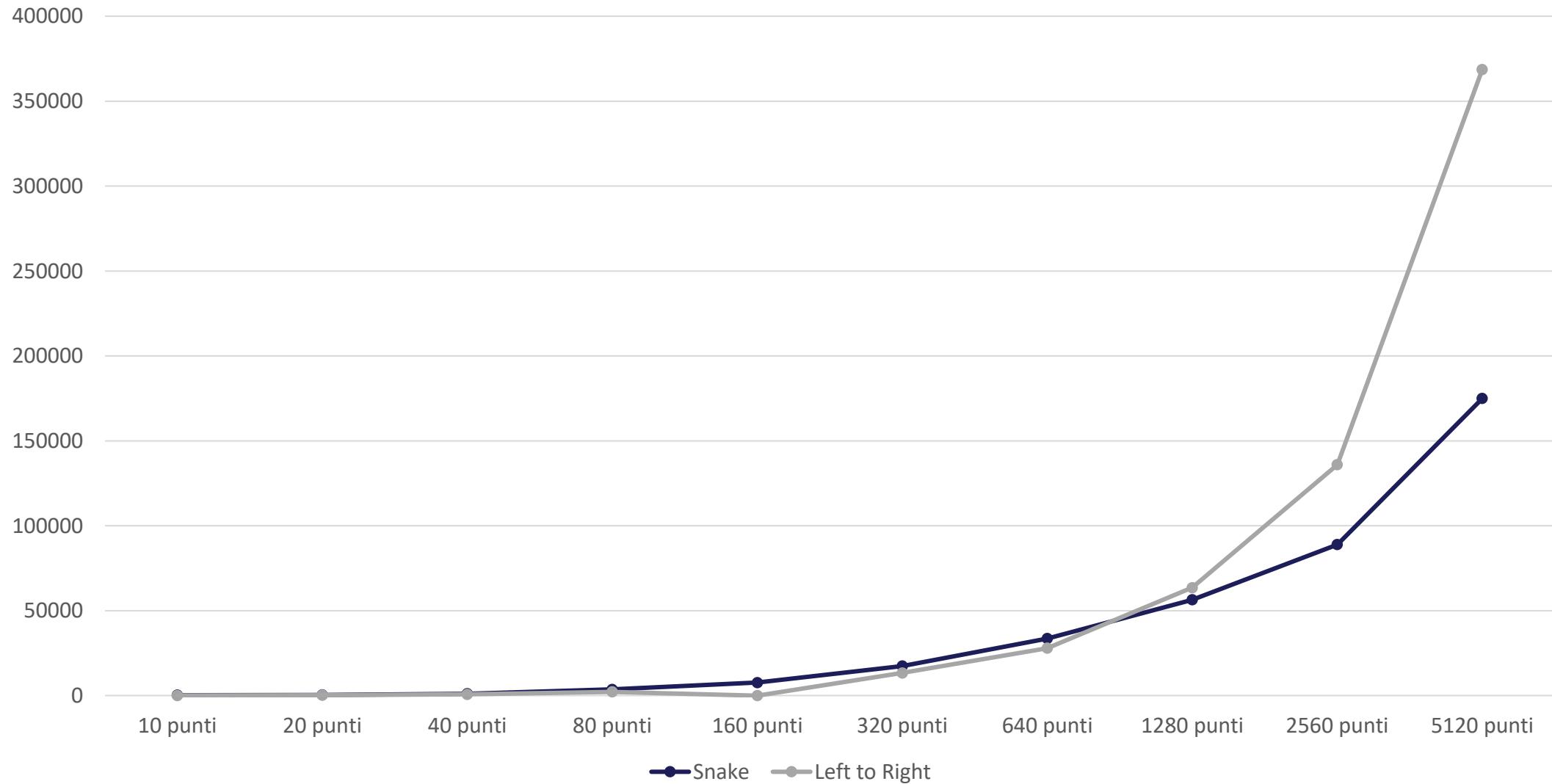


Costo Computazionale:  $O(n\sqrt{n})$



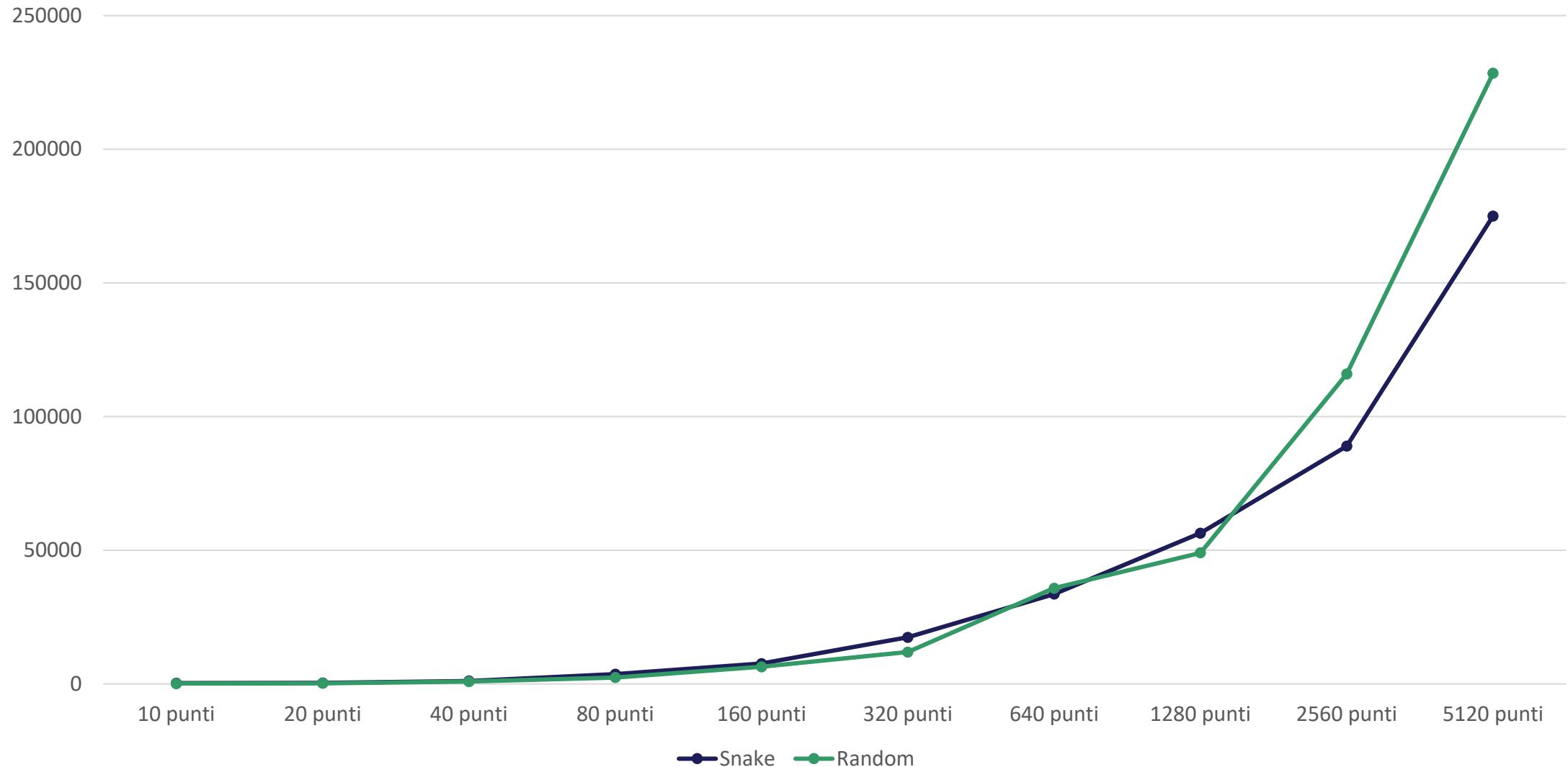
- Risultati approssimativi basati su tempi medi tra 10 iterazioni

## Confronto Snake - Left to Right



- Risultati approssimativi basati su tempi medi tra 10 iterazioni

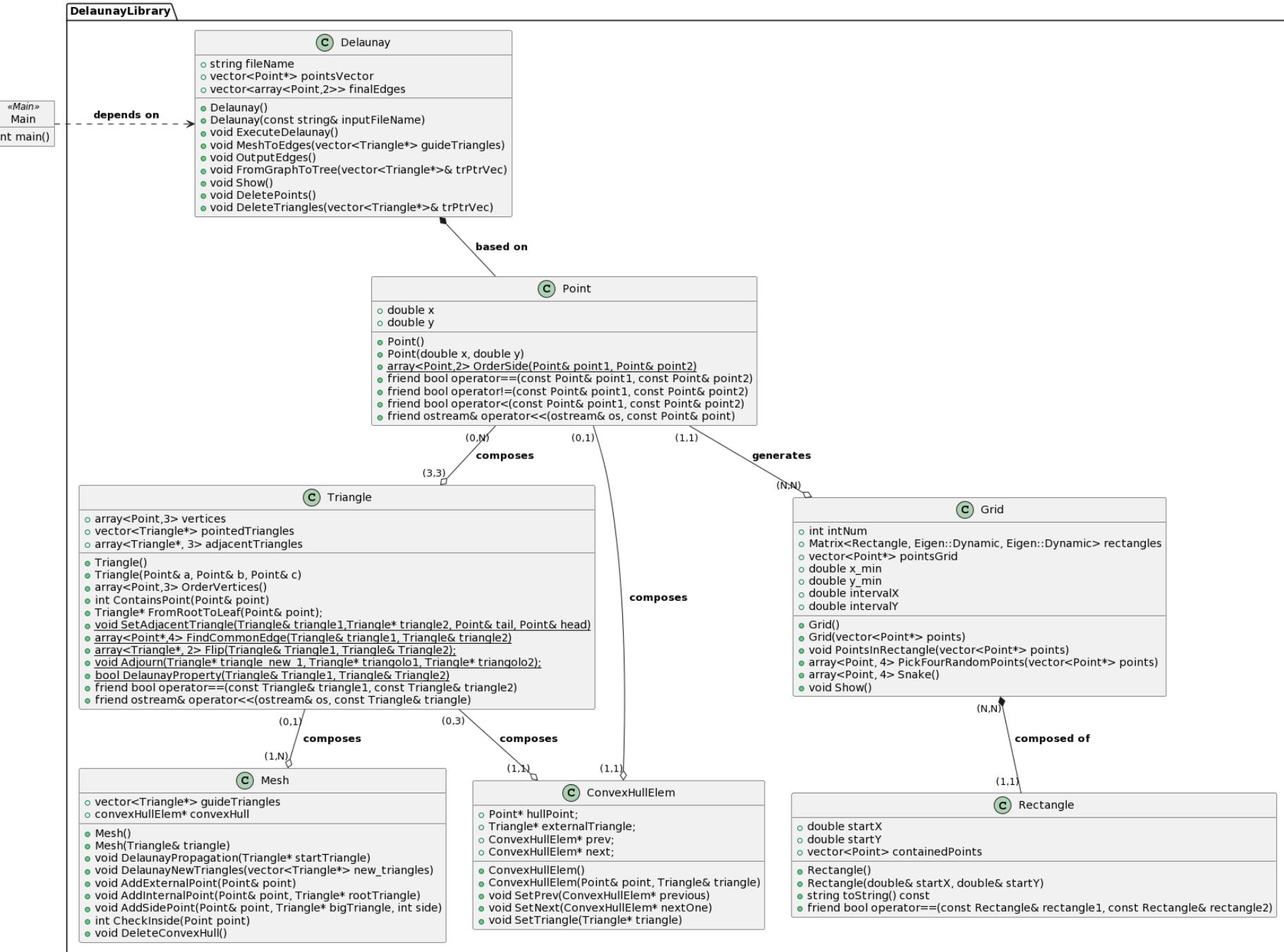
## Confronto Snake - Random

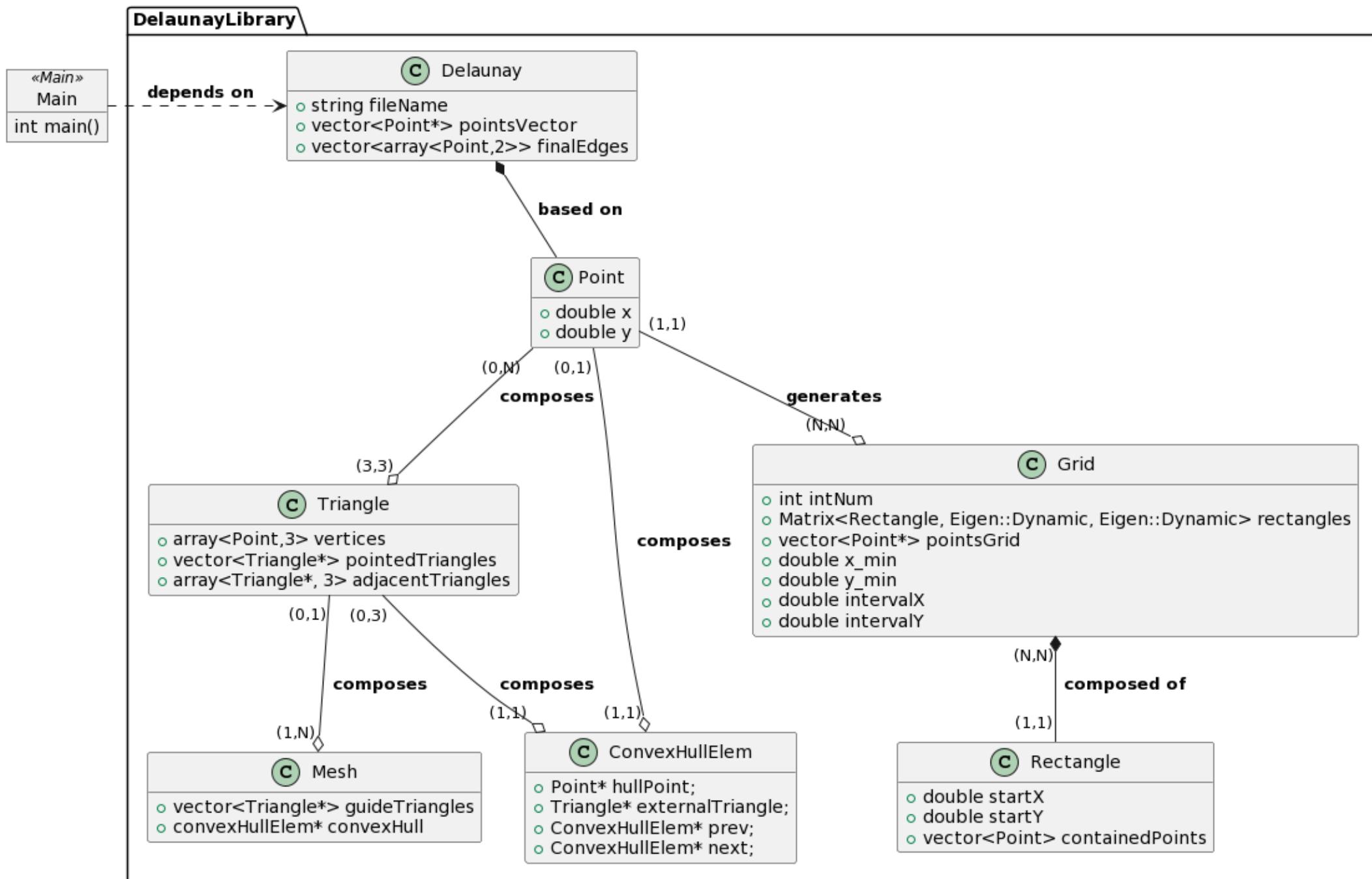


- Risultati approssimativi basati su tempi medi tra 10 iterazioni

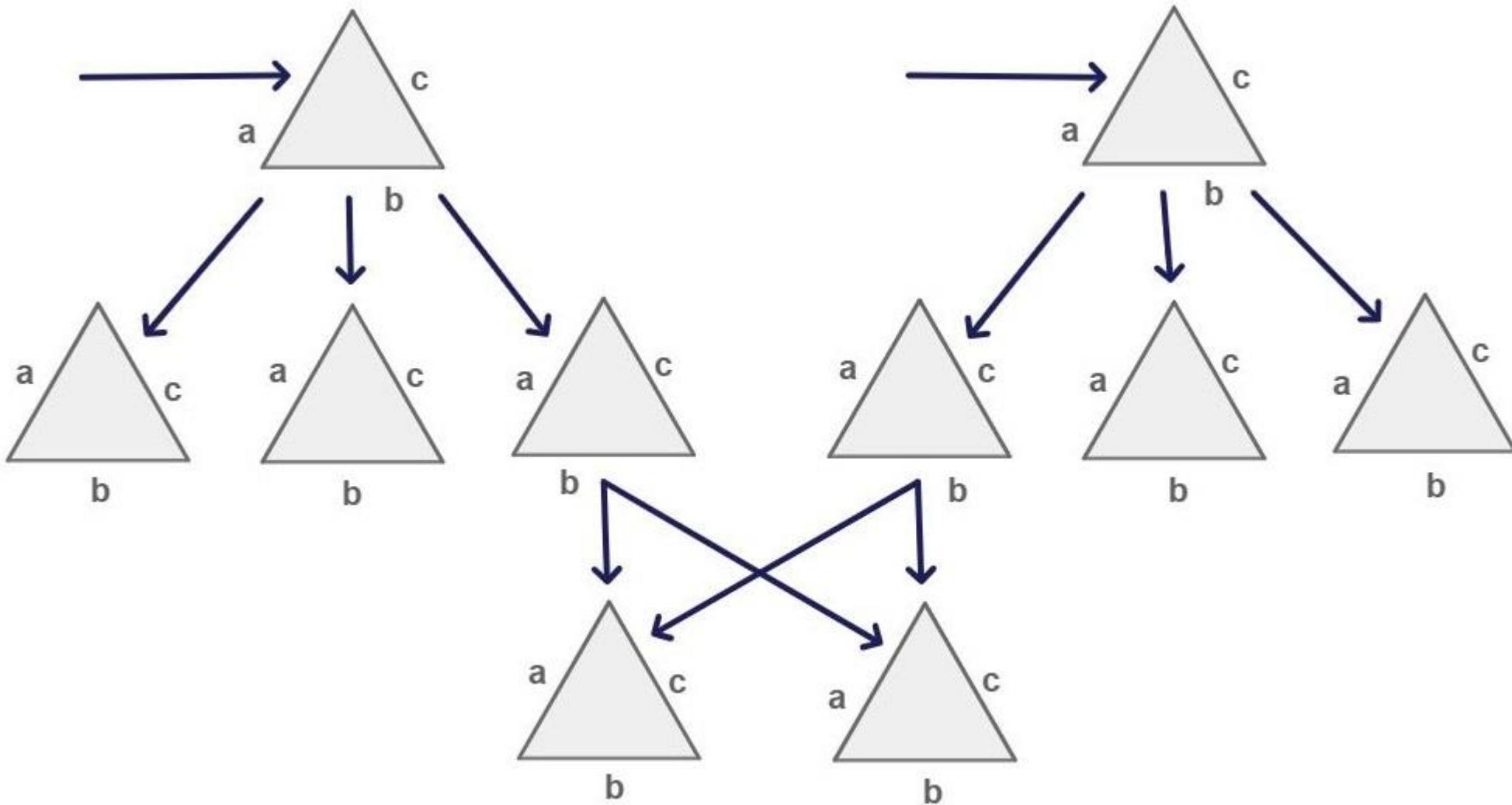


GRAZIE

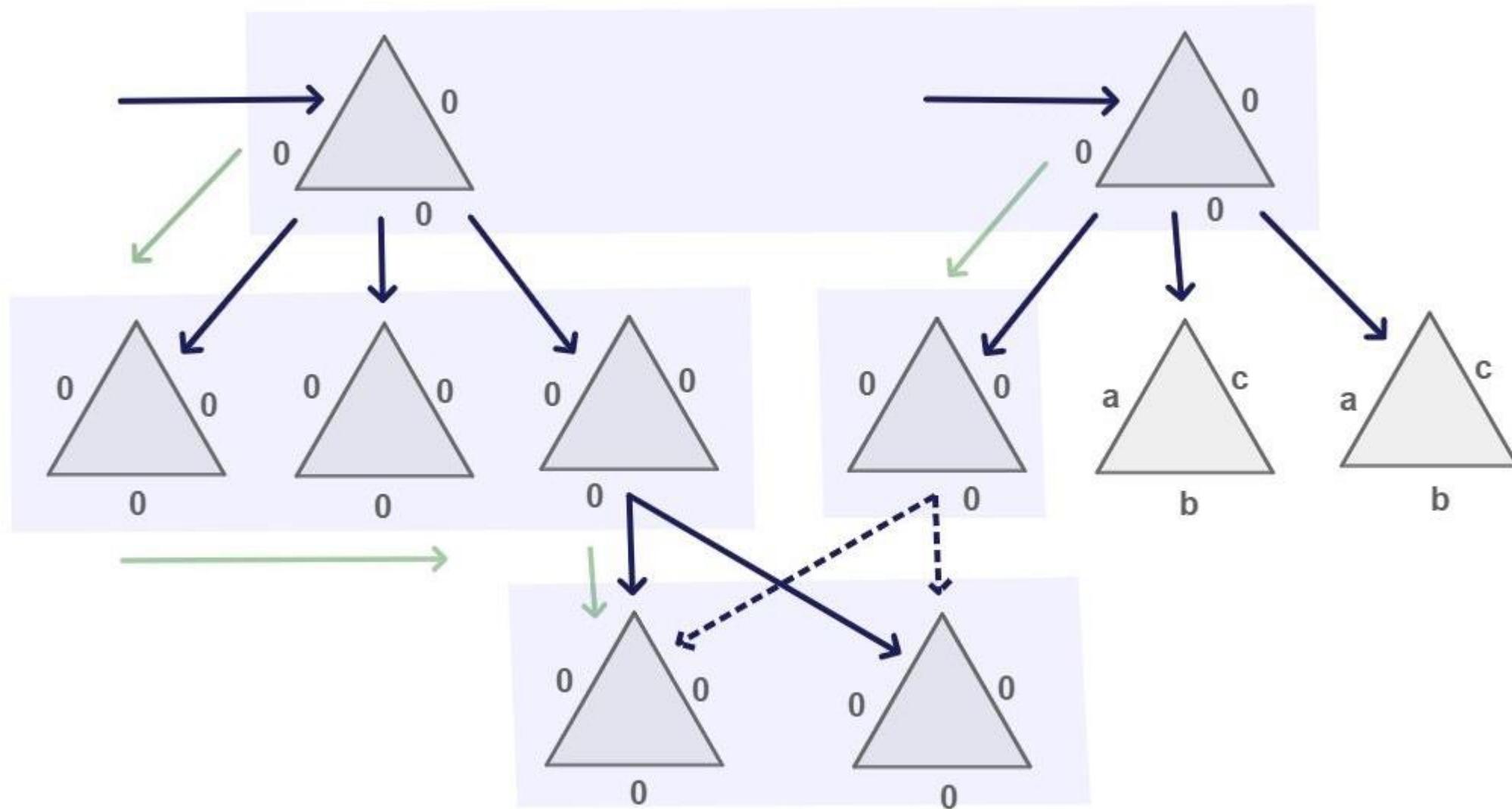




# DELETE DEI TRIANGOLI



# DELETE DEI TRIANGOLI



# DELETE DEI TRIANGOLI

