Computing the Delaunay Triangulation

Outline:

I. Edge legalization

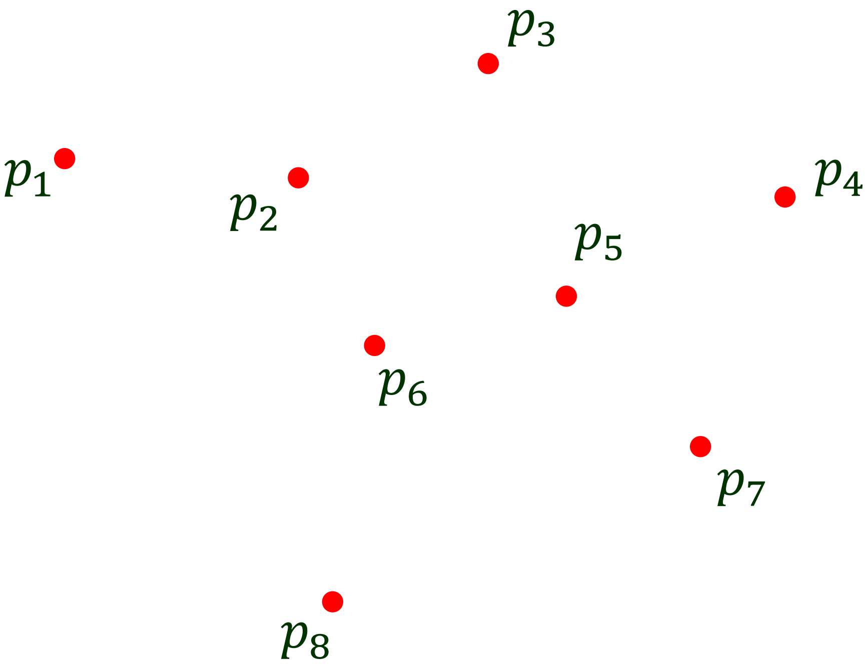
II. Correctness

III. Use of a trapezoidal map

IV. Analysis of storage and run time

I. The Construction Problem

Input: a set P of n points.

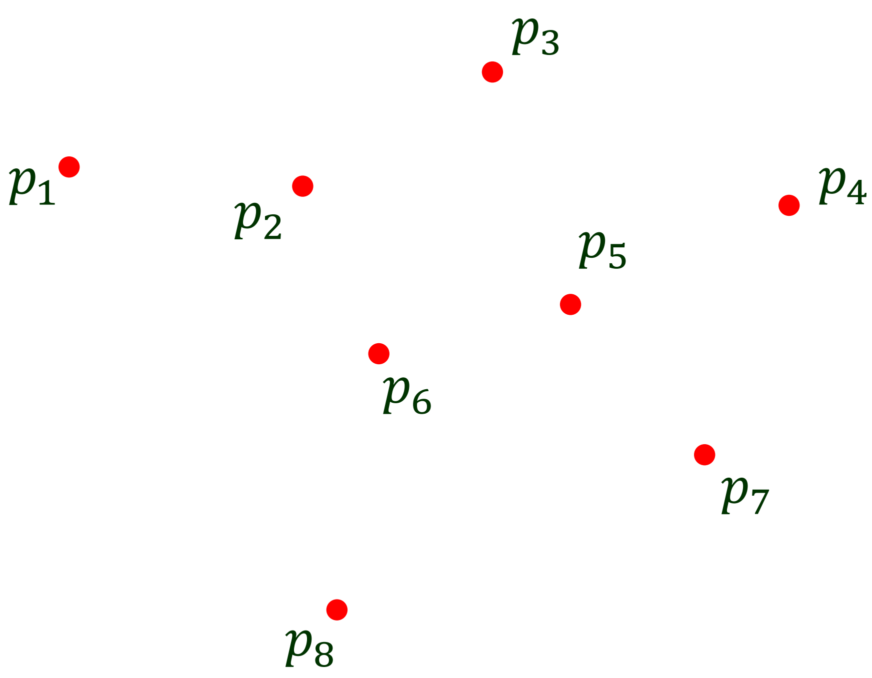


I. The Construction Problem

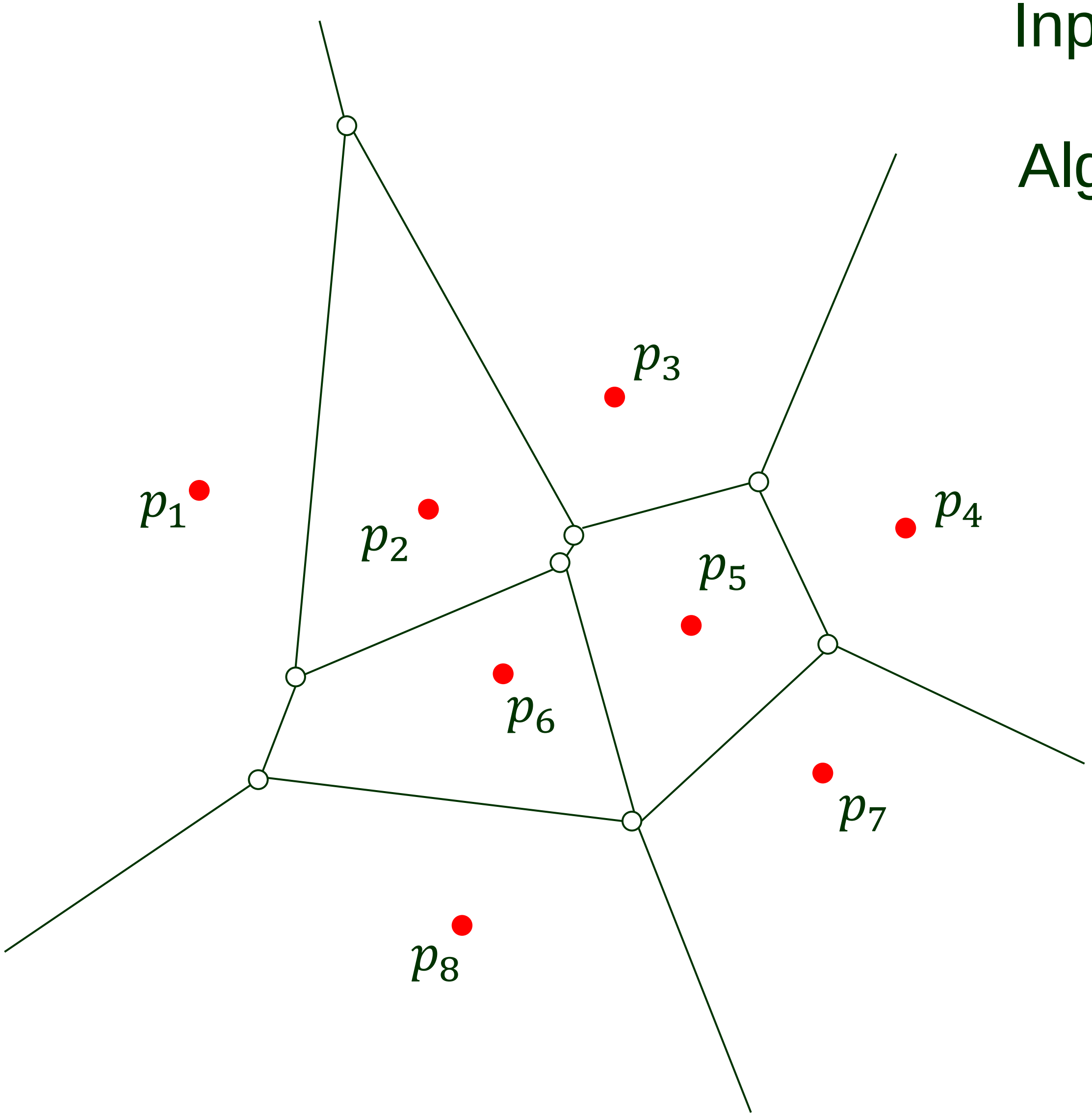
Input: a set P of n points.

Algorithm 1

1) Compute the Voronoi Diagram $\mathrm{Vor}(P)$.



I. The Construction Problem

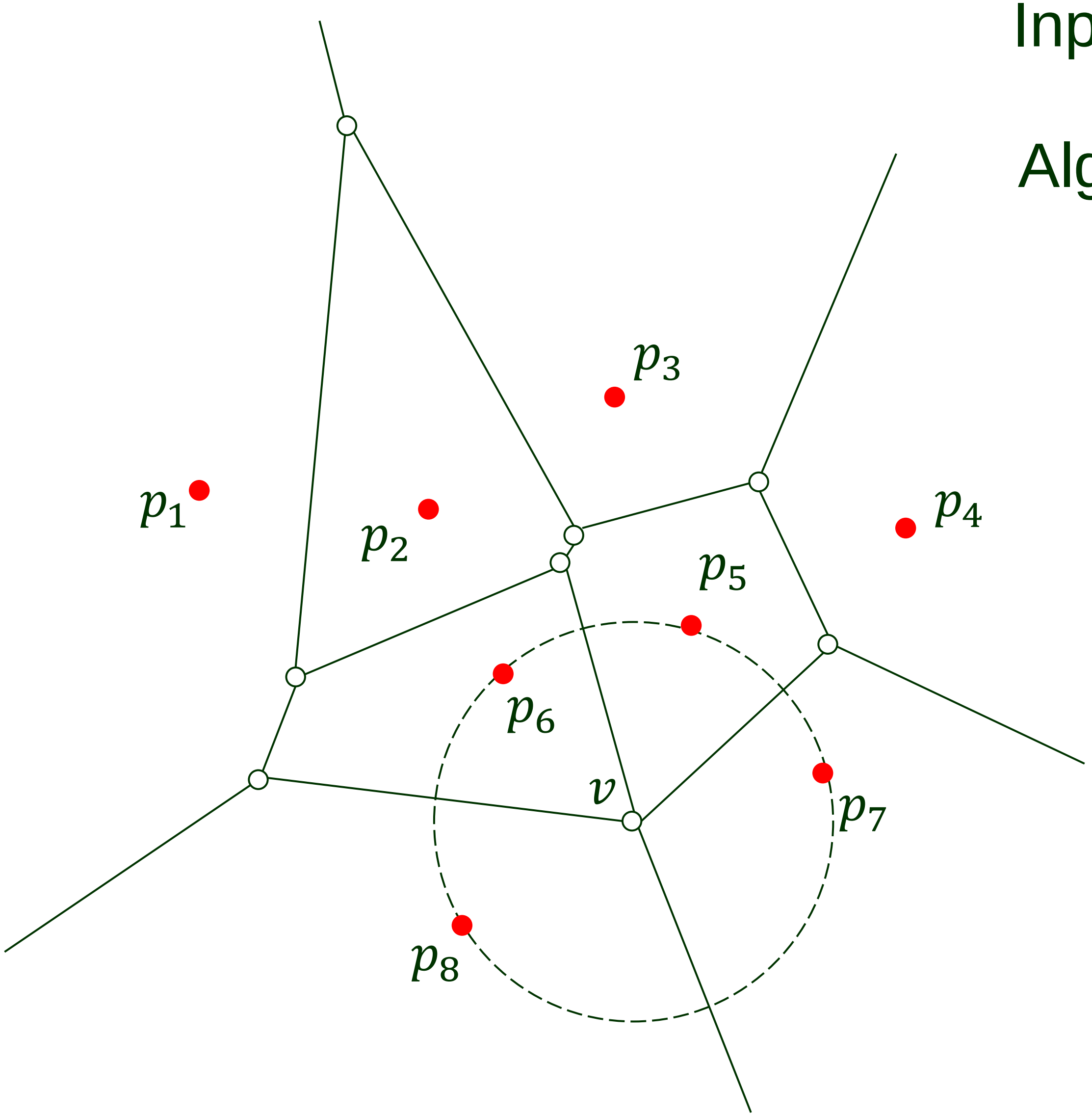


Input: a set P of n points.

Algorithm 1

1) Compute the Voronoi Diagram Vor(P).

I. The Construction Problem

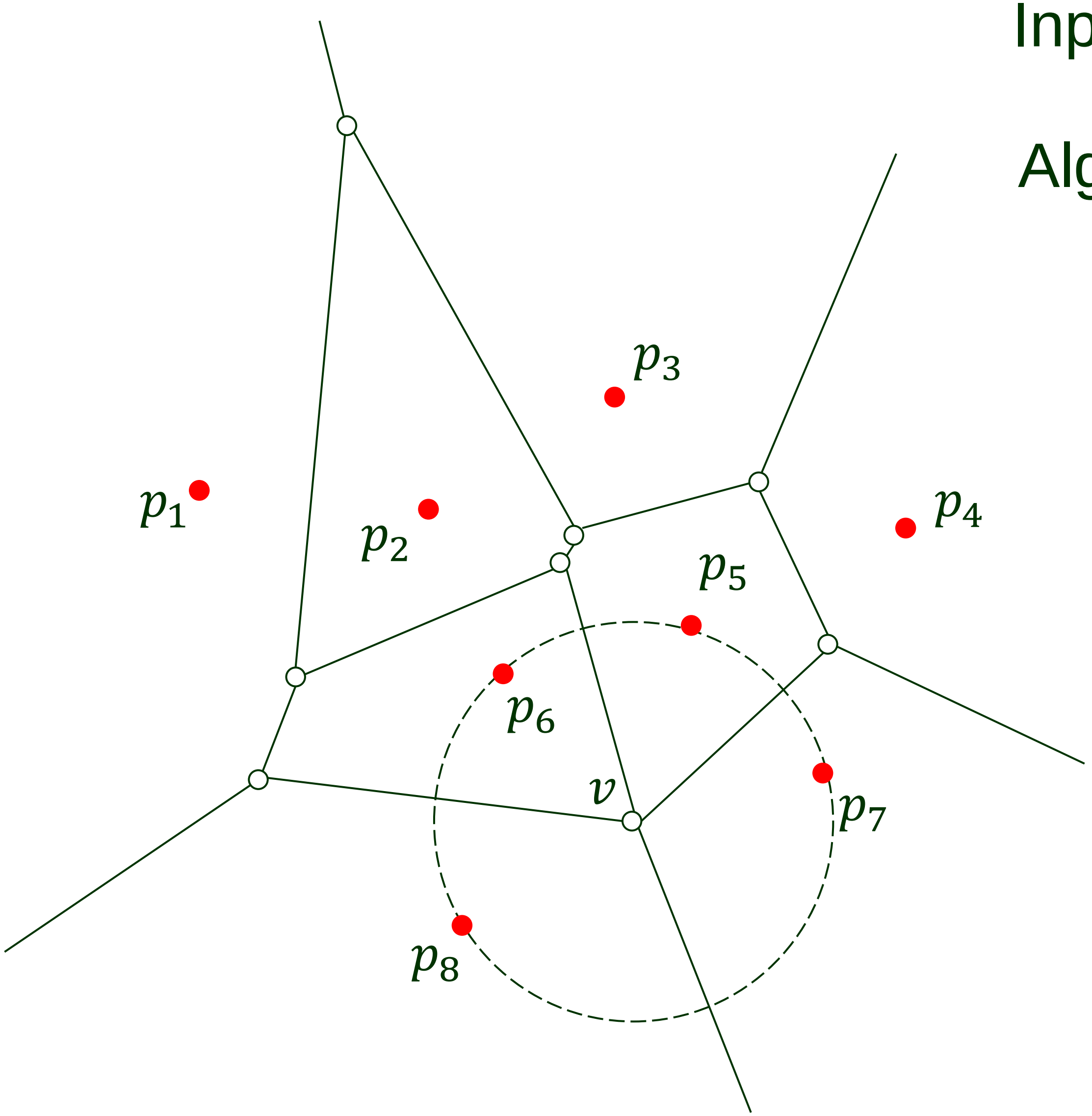


Input: a set P of n points.

Algorithm 1

1) Compute the Voronoi Diagram Vor(P).

I. The Construction Problem



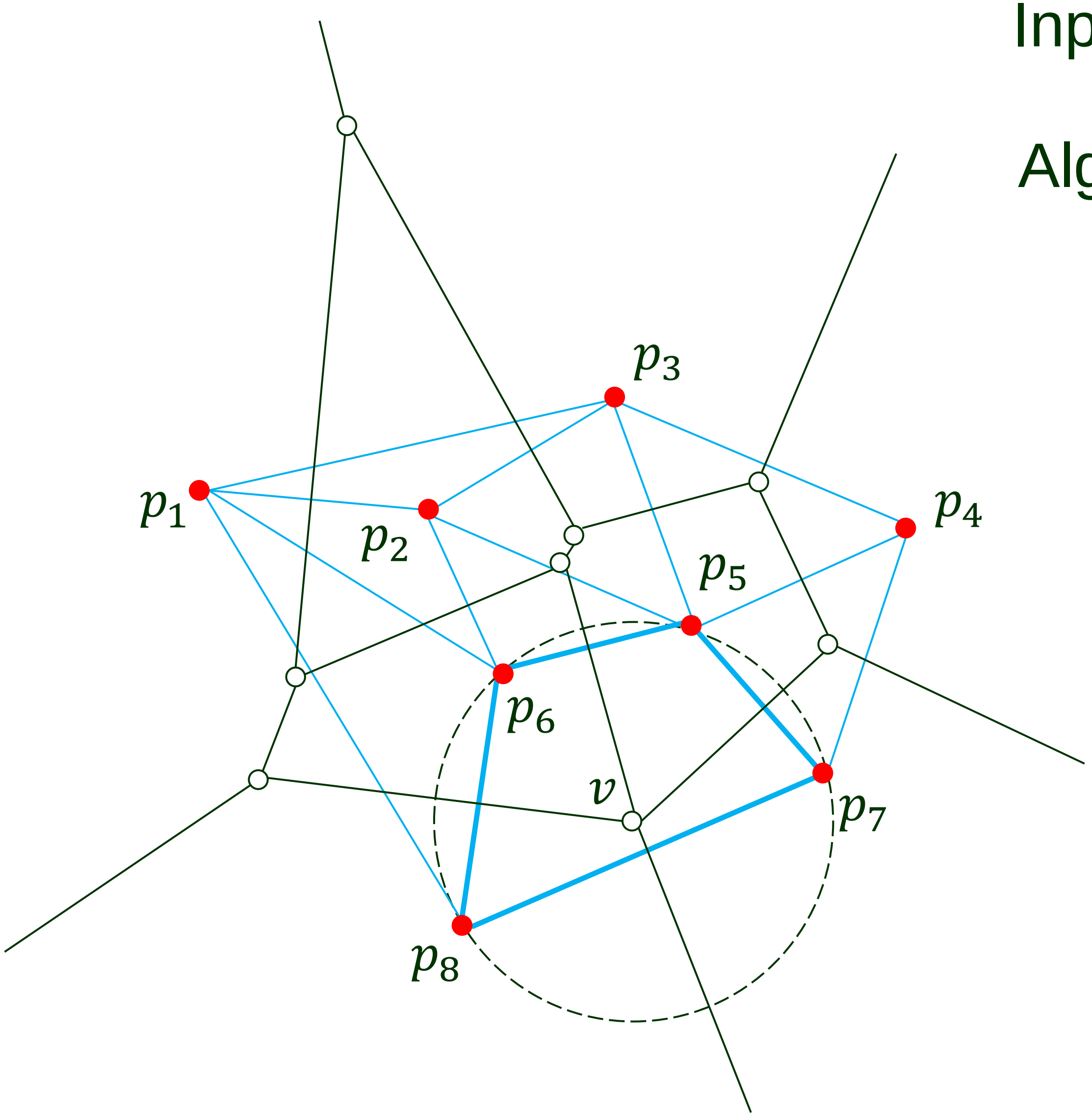
Input: a set P of n points.

Algorithm 1

1) Compute the Voronoi Diagram Vor(P).

2) Obtain DG(P).

I. The Construction Problem



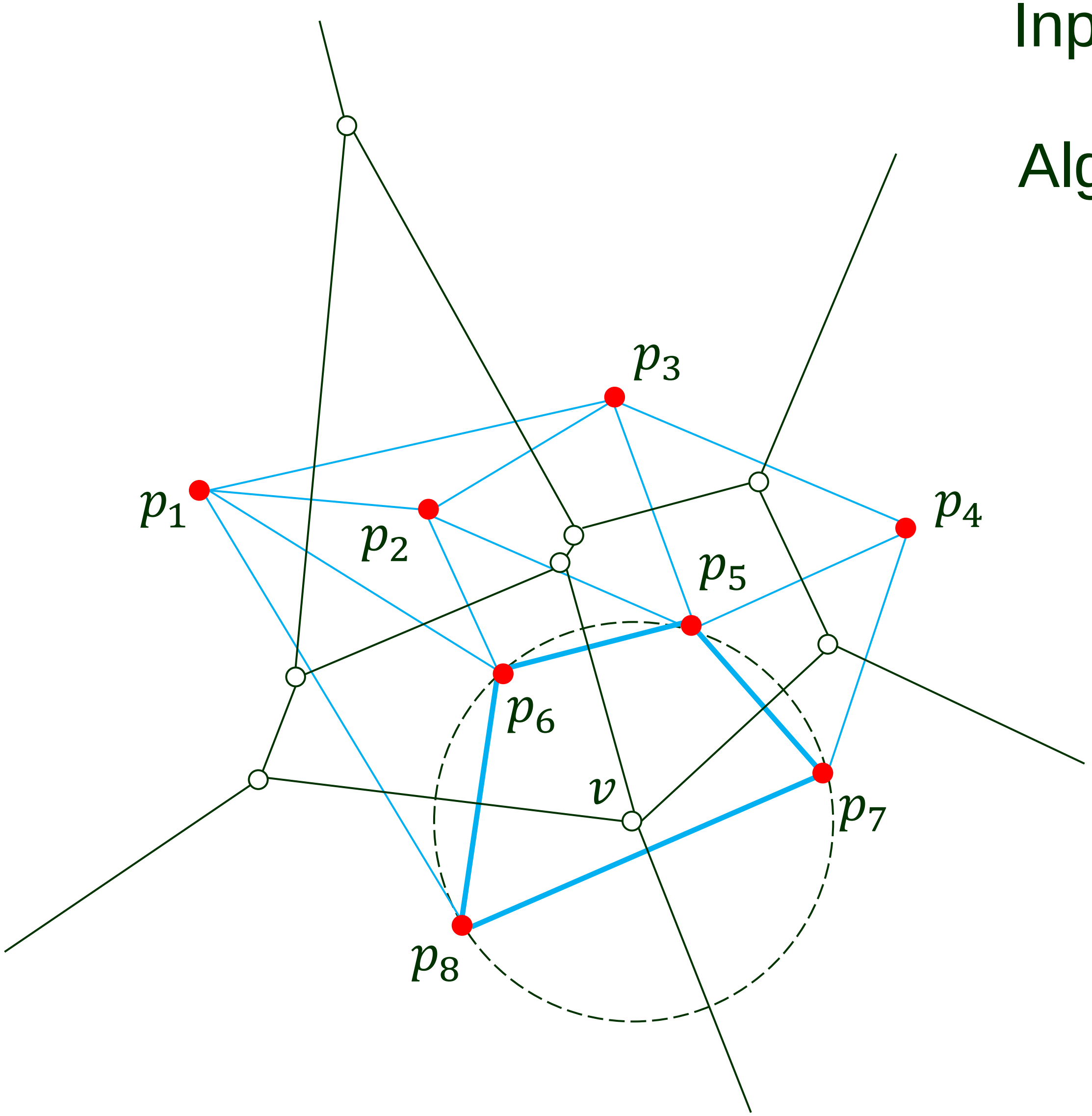
Input: a set P of n points.

Algorithm 1

1) Compute the Voronoi Diagram Vor(P).

2) Obtain DG(P).

I. The Construction Problem



Input: a set P of n points.

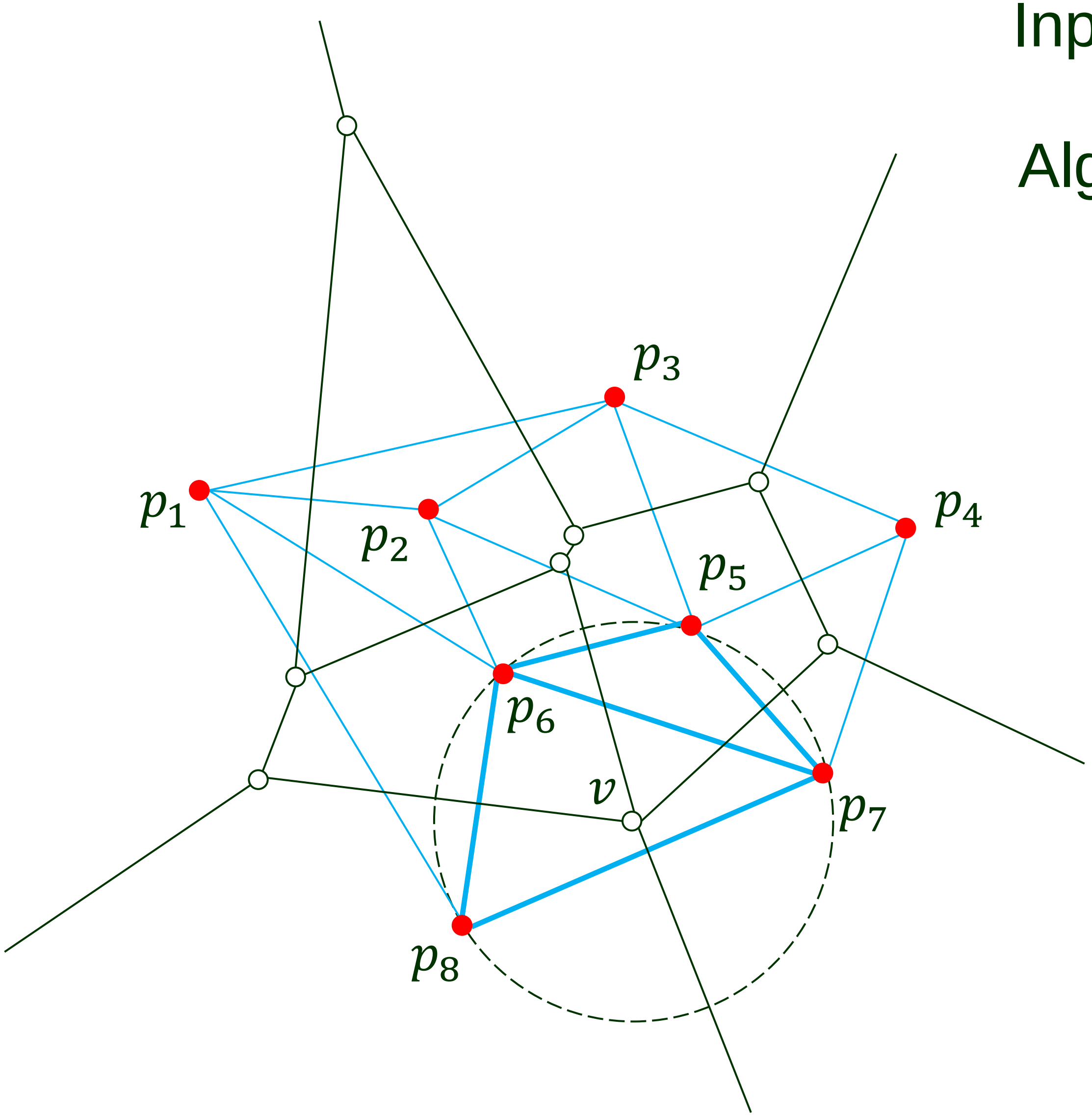
Algorithm 1

1) Compute the Voronoi Diagram Vor(P).

2) Obtain DG(P).

3) Triangulate faces with > 3 vertices.

I. The Construction Problem



Input: a set P of n points.

Algorithm 1

1) Compute the Voronoi Diagram Vor(P).

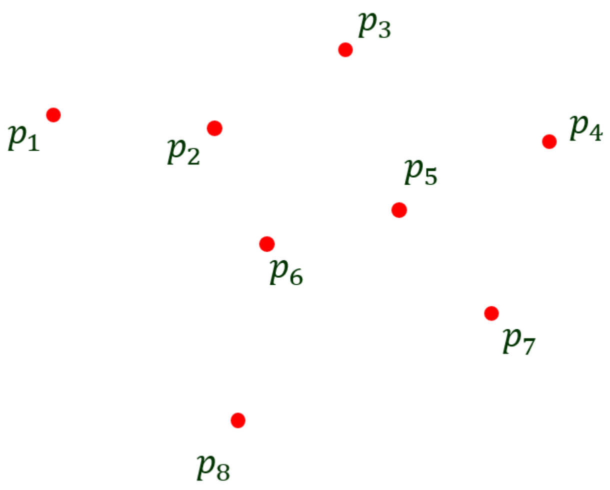
2) Obtain DG(P).

3) Triangulate faces with > 3 vertices.

Randomized Incremental Construction

Algorithm 2

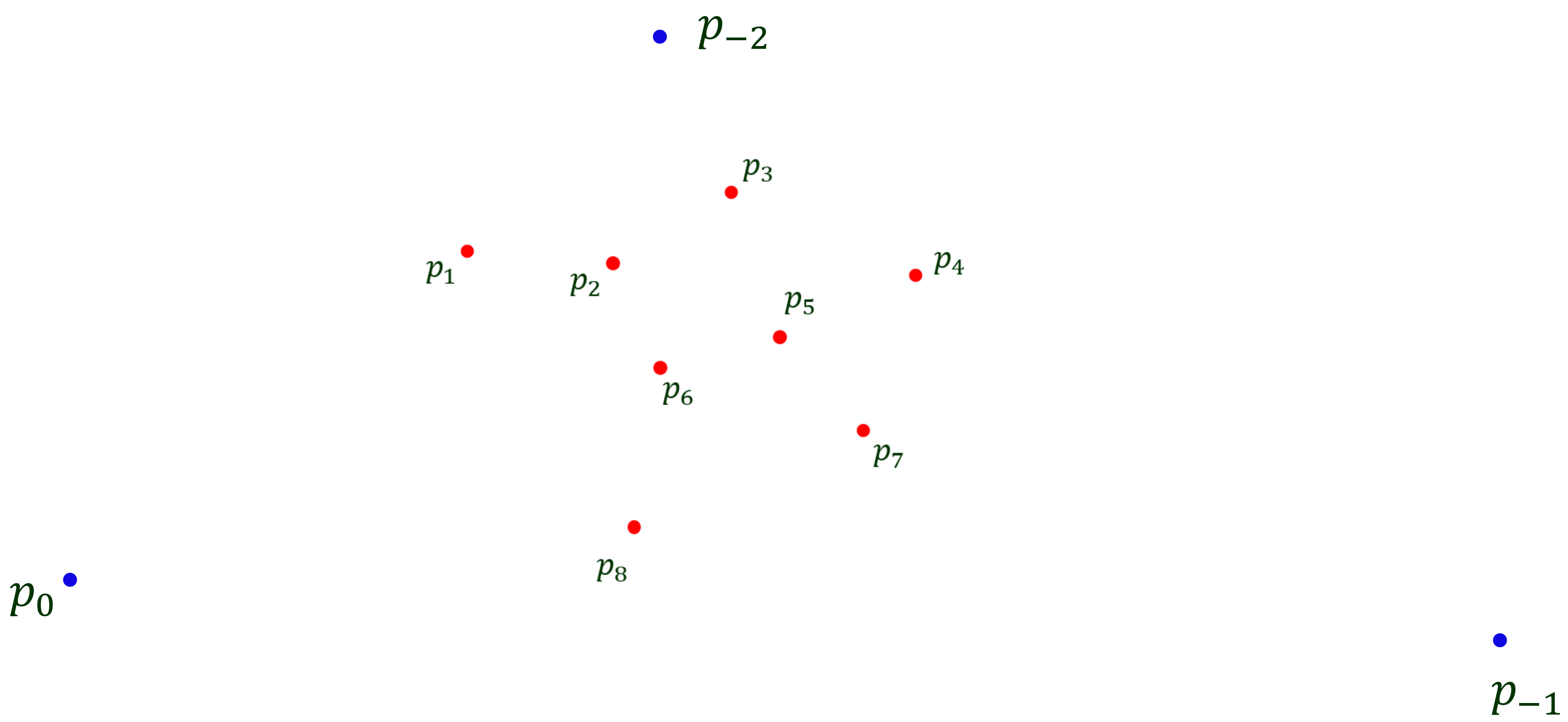
1) Introduce a set $\Omega = \{p_0, p_{-1}, p_{-2}\}$ of three auxiliary points.



Randomized Incremental Construction

Algorithm 2

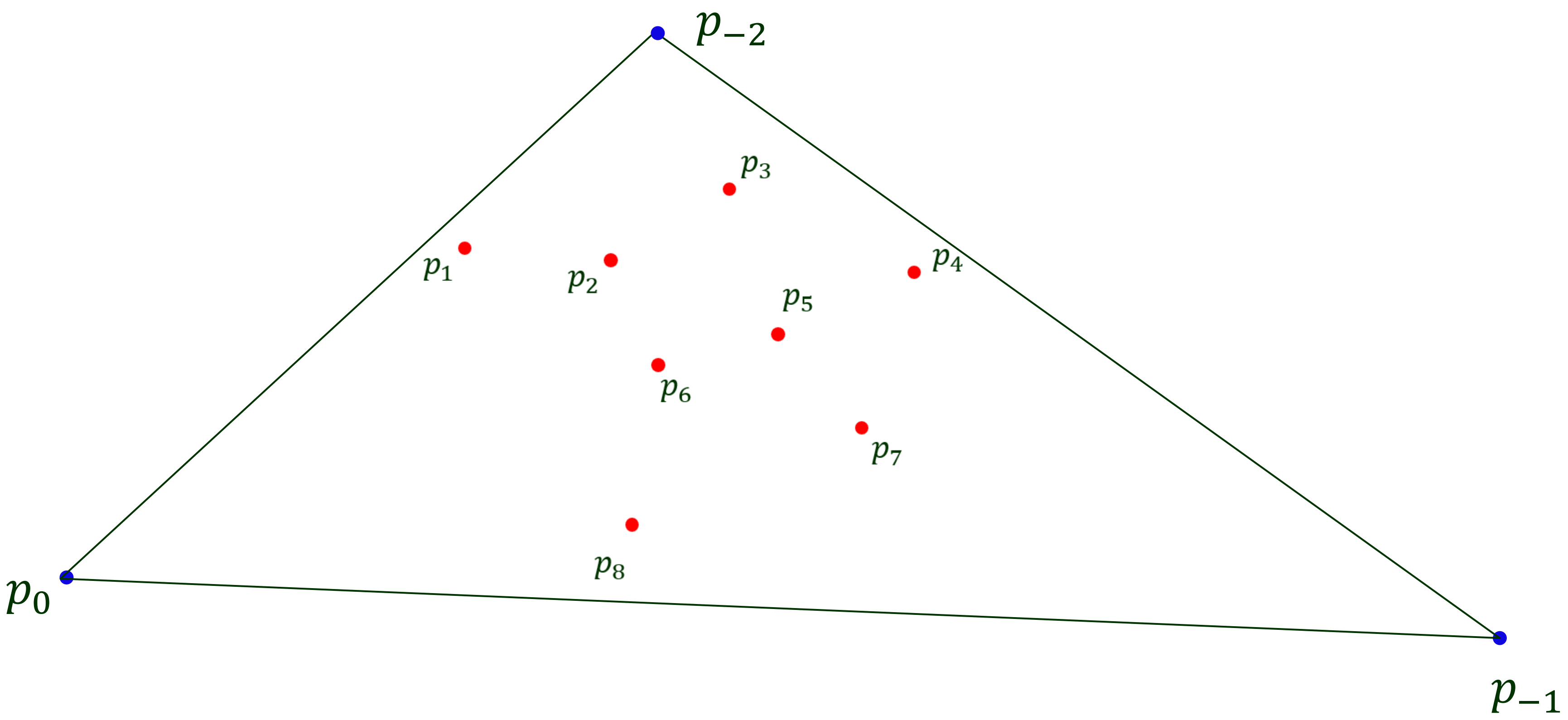
1) Introduce a set $\Omega = \{p_0, p_{-1}, p_{-2}\}$ of three auxiliary points.



Randomized Incremental Construction

Algorithm 2

1) Introduce a set $\Omega = \{p_0, p_{-1}, p_{-2}\}$ of three auxiliary points.
   Start with $\Delta\, p_0 p_{-1} p_{-2}$ containing all points from P in the interior.



Point Addition

2) Add points in random order, maintaining a Delaunay triangulation of the current set.

Point Addition

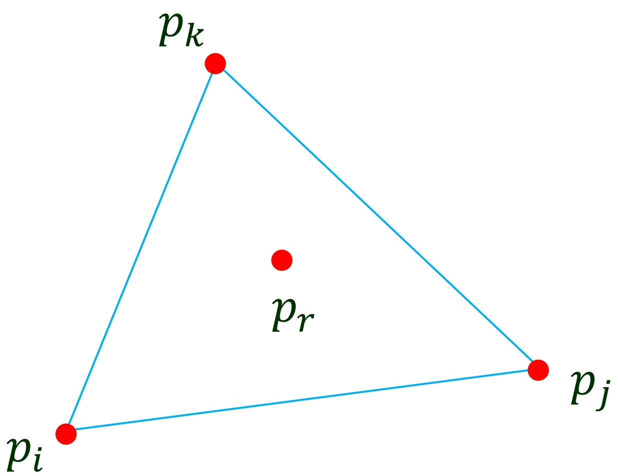
2) Add points in random order, maintaining a Delaunay triangulation of the current set.

- In step r, find the triangle $\Delta\, p_i p_j p_k$ that contains the newly added p_r.

Point Addition

2) Add points in random order, maintaining a Delaunay triangulation of the current set.

- In step r, find the triangle $\Delta\, p_i p_j p_k$ that contains the newly added p_r.

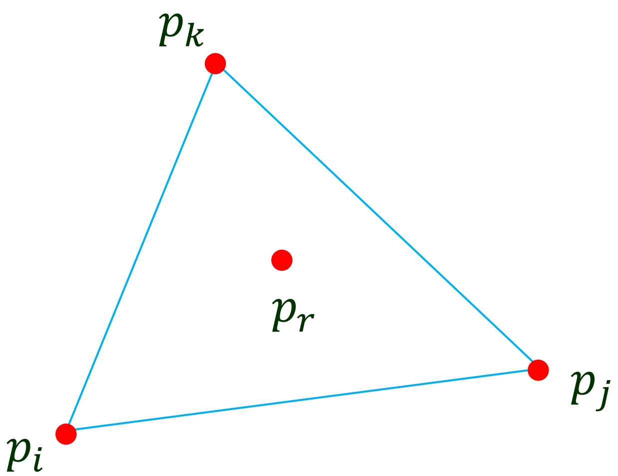


Case 1: p_r in the interior of $\Delta\, p_i p_j p_k$

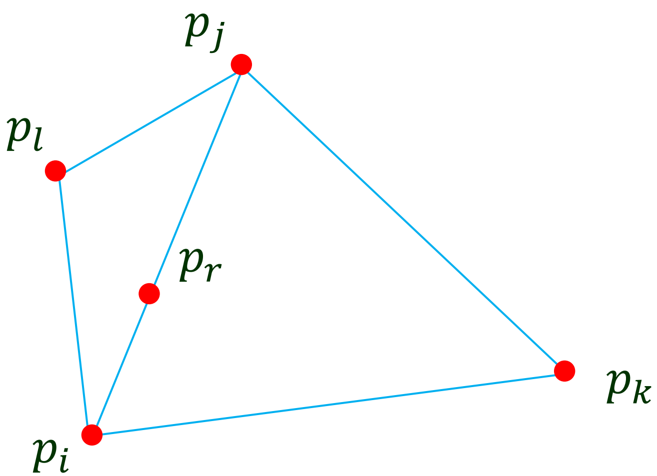
Point Addition

2) Add points in random order, maintaining a Delaunay triangulation of the current set.

- In step r, find the triangle $\Delta\, p_i p_j p_k$ that contains the newly added p_r.



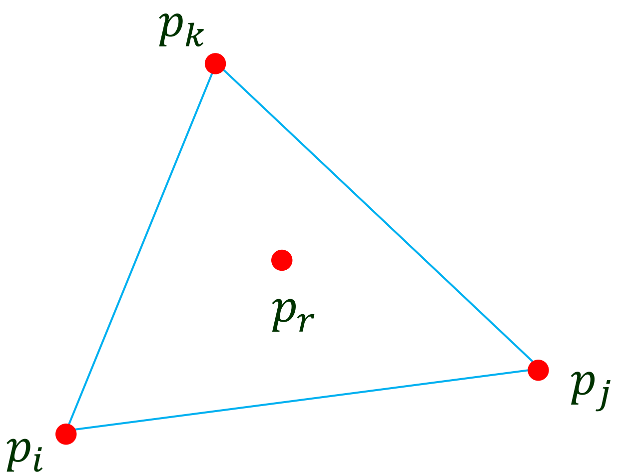
Case 1: p_r in the interior of $\Delta\, p_i p_j p_k$

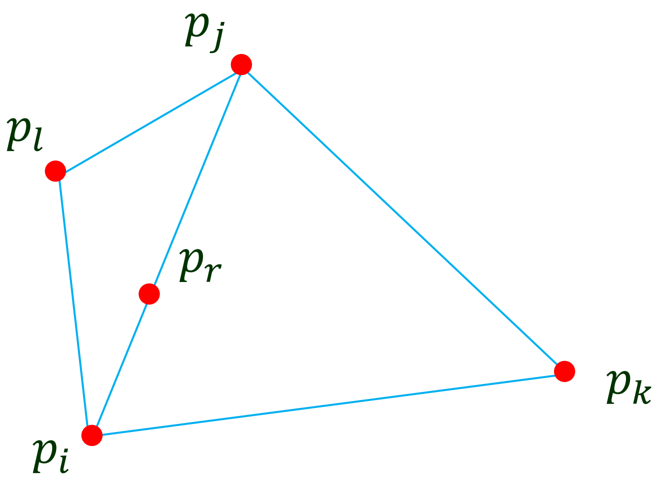


Case 2: p_r on an edge

Legal and Illegal Edges

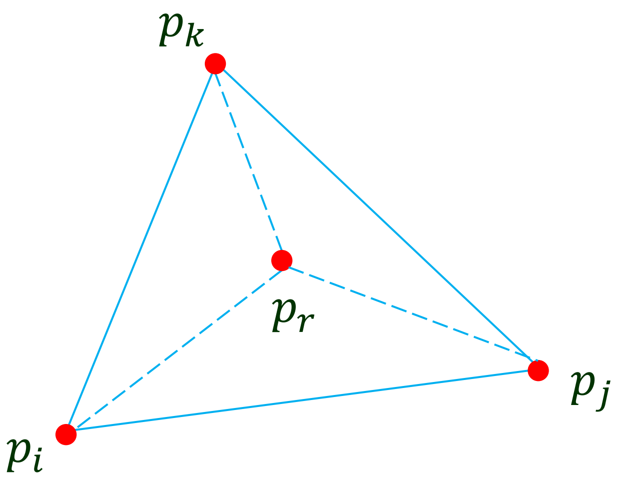
- Add edges from p_r to the vertices of the containing triangle(s).

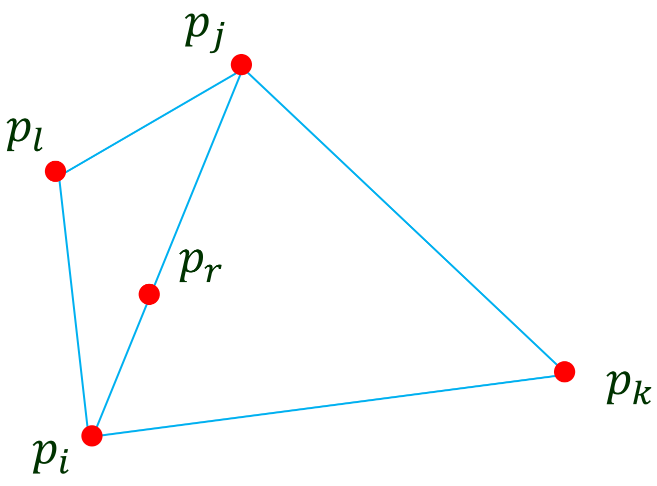




Legal and Illegal Edges

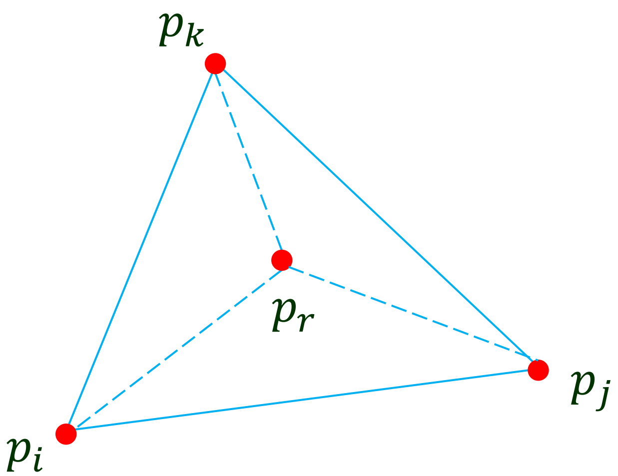
- Add edges from p_r to the vertices of the containing triangle(s).

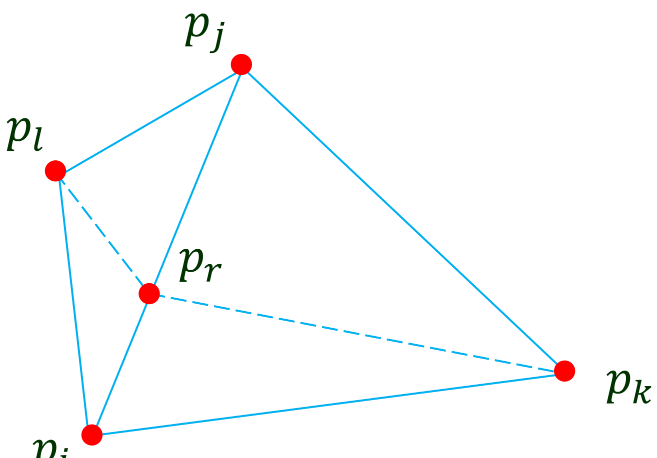




Legal and Illegal Edges

- Add edges from p_r to the vertices of the containing triangle(s).

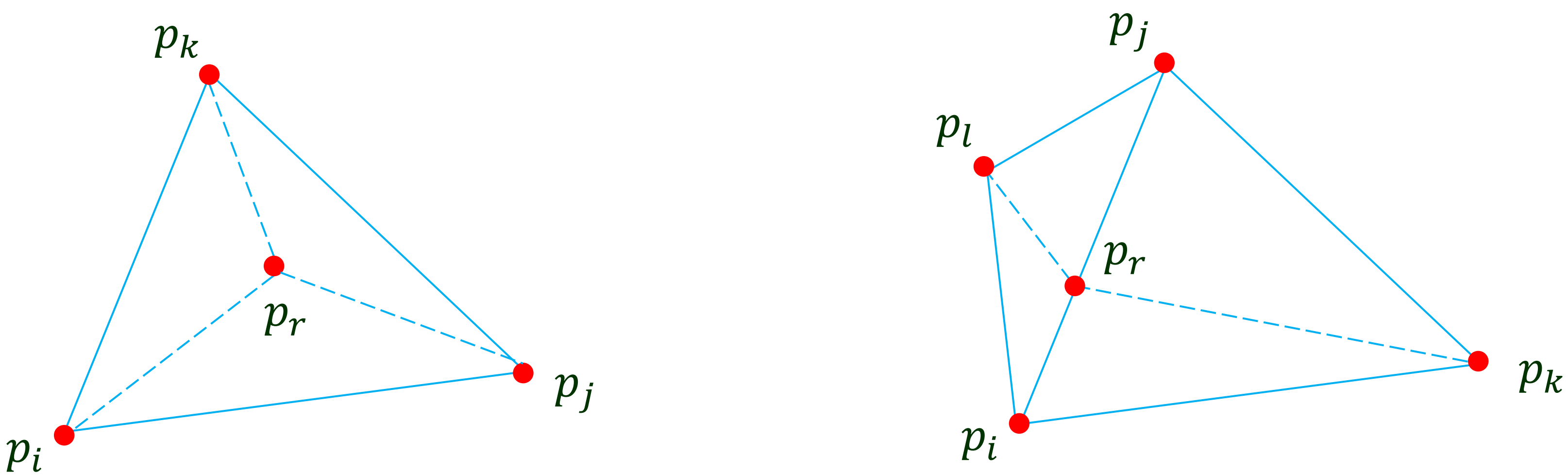




Legal and Illegal Edges

- Add edges from p_r to the vertices of the containing triangle(s).

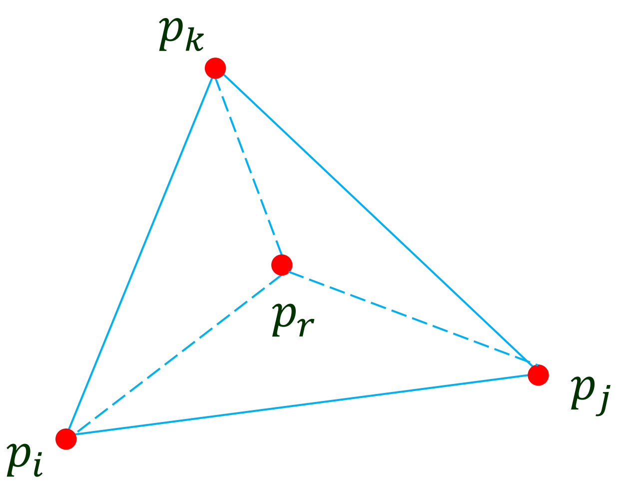
The new edges incident on p_r are legal (to be shown in Lemma 1).

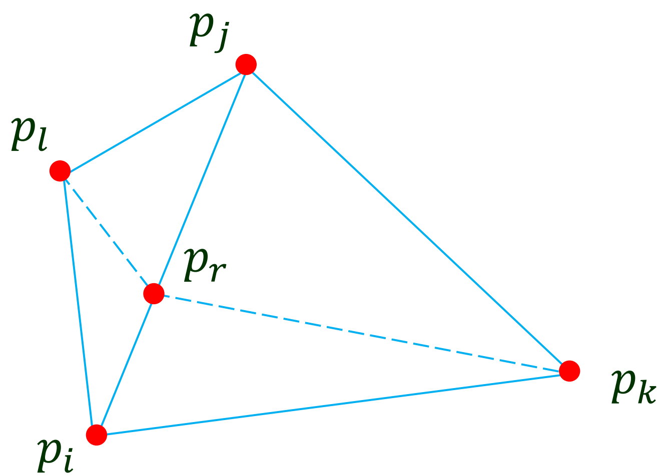


Legal and Illegal Edges

- Add edges from p_r to the vertices of the containing triangle(s).

  The new edges incident on p_r are legal (to be shown in Lemma 1).



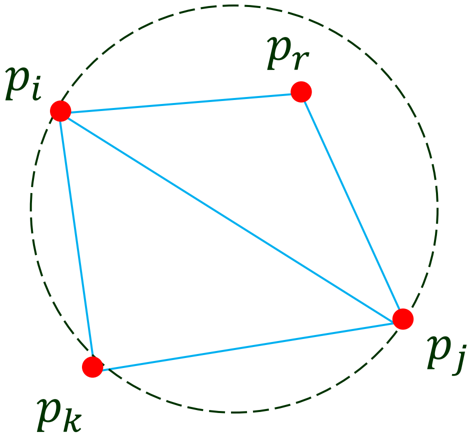


- Replace illegal edges by legal edges through edge flips.

Edge Legalization

LegalizeEdge($p_r, \overline{p_i p_j}, T$)

1. **if** $\overline{p_i p_j}$ is illegal
2. **then** let $\Delta\, p_i p_j p_k$ and $\Delta\, p_i p_j p_r$ be adjacent along $\overline{p_i p_j}$
3. replace $\overline{p_i p_j}$ with $\overline{p_r p_k}$
4. LegalizeEdge($p_r, \overline{p_i p_k}, T$)
5. LegalizeEdge($p_r, \overline{p_j p_k}, T$)



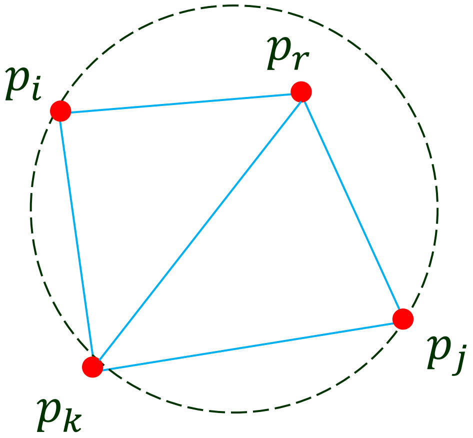
Edge Legalization

```
LegalizeEdge(p_r, p_i p_j, T)

1. if p_i p_j is illegal
2.     then let Δ p_i p_j p_k and Δ p_i p_j p_r be adjacent along p_i p_j
3.          replace p_i p_j with p_r p_k
4.          LegalizeEdge(p_r, p_i p_k, T)
5.          LegalizeEdge(p_r, p_j p_k, T)
```

LegalizeEdge(p_r, $\overline{p_i p_j}$, T)

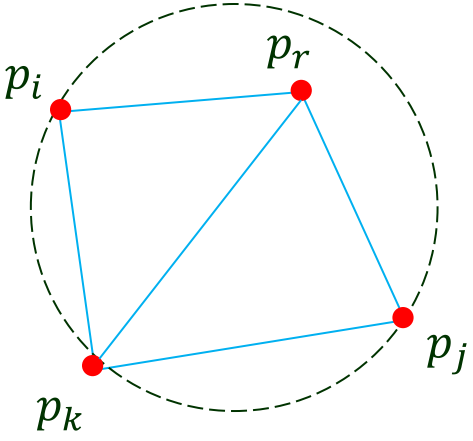
1. **if** $\overline{p_i p_j}$ is illegal
2. **then** let $\Delta\, p_i p_j p_k$ and $\Delta\, p_i p_j p_r$ be adjacent along $\overline{p_i p_j}$
3. replace $\overline{p_i p_j}$ with $\overline{p_r p_k}$
4. LegalizeEdge(p_r, $\overline{p_i p_k}$, T)
5. LegalizeEdge(p_r, $\overline{p_j p_k}$, T)



Edge Legalization

```
LegalizeEdge(p_r, p_i p_j, T)

1. if p_i p_j is illegal
2.    then let Δ p_i p_j p_k and Δ p_i p_j p_r be adjacent along p_i p_j
3.         replace p_i p_j with p_r p_k
4.         LegalizeEdge(p_r, p_i p_k, T)
5.         LegalizeEdge(p_r, p_j p_k, T)
```



- All recursive calls involve edges opposing p_r.

Edge Legalization

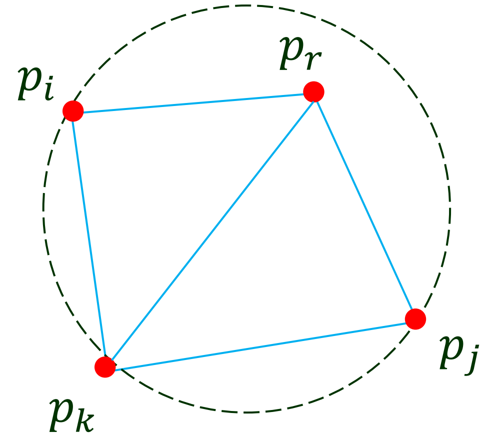
LegalizeEdge($p_r, \overline{p_i p_j}, T$)

if $\overline{p_i p_j}$ is illegal
    then let $\Delta\, p_i p_j p_k$ and $\Delta\, p_i p_j p_r$ be adjacent along $\overline{p_i p_j}$
        replace $\overline{p_i p_j}$ with $\overline{p_r p_k}$
        LegalizeEdge($p_r, \overline{p_i p_k}, T$)
        LegalizeEdge($p_r, \overline{p_j p_k}, T$)

- All recursive calls involve edges opposing p_r.

$$\Downarrow$$

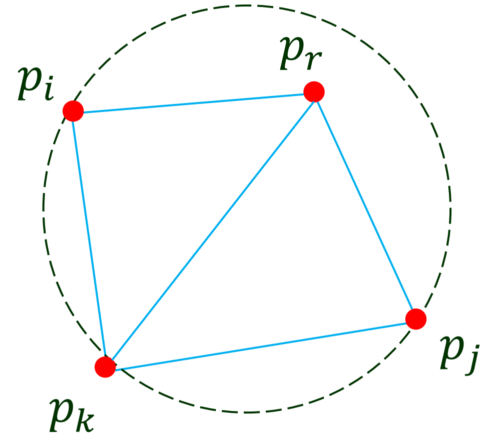
- All replacing edges during the edge flips are incident on p_r.



Edge Legalization

LegalizeEdge(p_r, $\overline{p_i p_j}$, T)

if $\overline{p_i p_j}$ is illegal
    then let $\Delta\, p_i p_j p_k$ and $\Delta\, p_i p_j p_r$ be adjacent along $\overline{p_i p_j}$
        replace $\overline{p_i p_j}$ with $\overline{p_r p_k}$
        LegalizeEdge(p_r, $\overline{p_i p_k}$, T)
        LegalizeEdge(p_r, $\overline{p_j p_k}$, T)



- All recursive calls involve edges opposing p_r.

$\Downarrow$

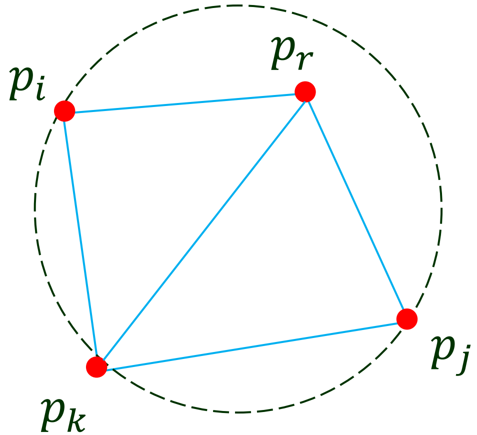
- All replacing edges during the edge flips are incident on p_r.

- An edge (which was legal before) can only become illegal if one of its two incident triangles has changed.

Edge Legalization

LegalizeEdge($p_r, \overline{p_i p_j}, T$)

if $\overline{p_i p_j}$ is illegal
    then let $\Delta\, p_i p_j p_k$ and $\Delta\, p_i p_j p_r$ be adjacent along $\overline{p_i p_j}$
        replace $\overline{p_i p_j}$ with $\overline{p_r p_k}$
        LegalizeEdge($p_r, \overline{p_i p_k}, T$)
        LegalizeEdge($p_r, \overline{p_j p_k}, T$)



- All recursive calls involve edges opposing p_r.

$\Downarrow$

- All replacing edges during the edge flips are incident on p_r.

- An edge (which was legal before) can only become illegal if one of its two incident triangles has changed.

$\Downarrow$

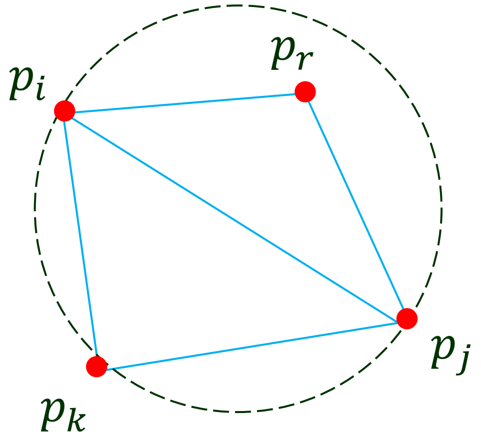
- Only the edges of the new triangles need to be checked.

More Observations

```
LegalizeEdge(p_r, p_i p_j, T)
```

$\texttt{LegalizeEdge}(p_r, \overline{p_i p_j}, T)$

1. **if** $\overline{p_i p_j}$ is illegal
2. **then** let $\Delta\, p_i p_j p_k$ and $\Delta\, p_i p_j p_r$ be adjacent along $\overline{p_i p_j}$
3. replace $\overline{p_i p_j}$ with $\overline{p_r p_k}$
4. $\texttt{LegalizeEdge}(p_r, \overline{p_i p_k}, T)$
5. $\texttt{LegalizeEdge}(p_r, \overline{p_j p_k}, T)$



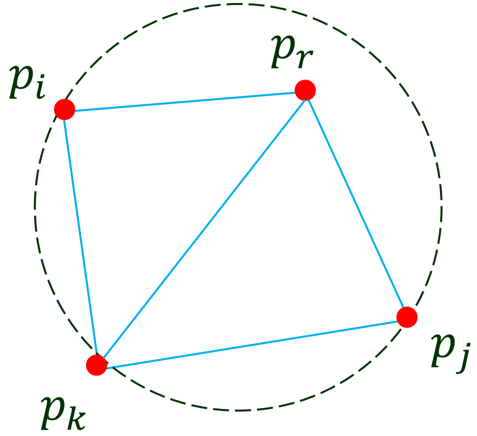
- Only the edges of the new triangles need to be checked.

More Observations

```
LegalizeEdge(p_r, p_i p_j, T)
```

$\texttt{LegalizeEdge}(p_r, \overline{p_i p_j}, T)$

1. **if** $\overline{p_i p_j}$ is illegal
2. **then** let $\Delta\, p_i p_j p_k$ and $\Delta\, p_i p_j p_r$ be adjacent along $\overline{p_i p_j}$
3. replace $\overline{p_i p_j}$ with $\overline{p_r p_k}$
4. $\texttt{LegalizeEdge}(p_r, \overline{p_i p_k}, T)$
5. $\texttt{LegalizeEdge}(p_r, \overline{p_j p_k}, T)$

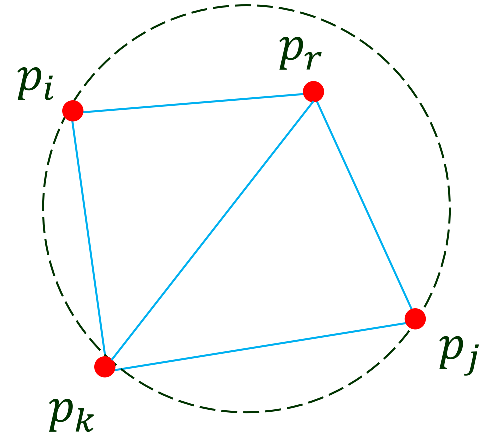


- Only the edges of the new triangles need to be checked.

More Observations

LegalizeEdge($p_r, \overline{p_i p_j}, T$)

if $\overline{p_i p_j}$ is illegal
    then let $\Delta\, p_i p_j p_k$ and $\Delta\, p_i p_j p_r$ be adjacent along $\overline{p_i p_j}$
        replace $\overline{p_i p_j}$ with $\overline{p_r p_k}$
        LegalizeEdge($p_r, \overline{p_i p_k}, T$)
        LegalizeEdge($p_r, \overline{p_j p_k}, T$)



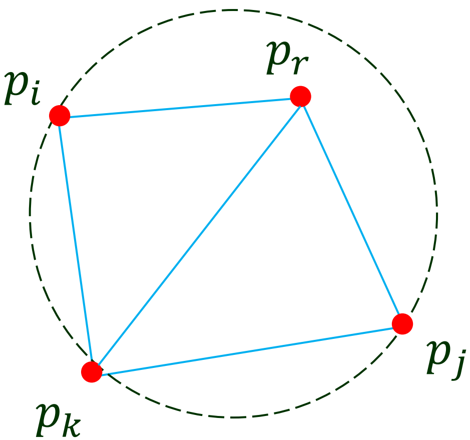
- Only the edges of the new triangles need to be checked.
- $\overline{p_r p_i}$ and $\overline{p_r p_j}$ are newly inserted and legal (to be shown).

More Observations

```
LegalizeEdge(p_r, p_i p_j, T)
```

$\texttt{LegalizeEdge}(p_r, \overline{p_i p_j}, T)$

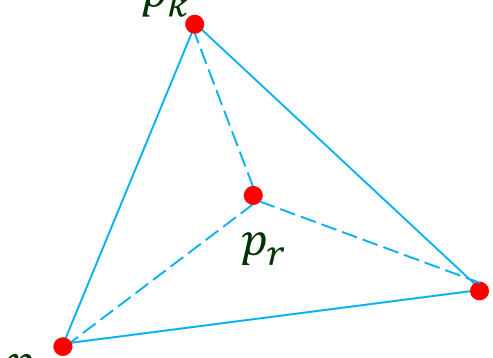
if $\overline{p_i p_j}$ is illegal
    then let $\Delta\, p_i p_j p_k$ and $\Delta\, p_i p_j p_r$ be adjacent along $\overline{p_i p_j}$
        replace $\overline{p_i p_j}$ with $\overline{p_r p_k}$
        $\texttt{LegalizeEdge}(p_r, \overline{p_i p_k}, T)$
        $\texttt{LegalizeEdge}(p_r, \overline{p_j p_k}, T)$



- Only the edges of the new triangles need to be checked.
- $\overline{p_r p_i}$ and $\overline{p_r p_j}$ are newly inserted and legal (to be shown).
- So check $\overline{p_i p_k}$ and $\overline{p_j p_k}$ opposing p_r, and recursively from there.

Iterations

```
Compute a random permutation p_1, p_2 ..., p_n
for r ← 1 to n
  do
    find Δ p_i p_j p_k ⊃ p_r in the current triangulation T
    if p_r lies in its interior
      then   // case 1
        add edges $\overline{p_r p_i}$, $\overline{p_r p_j}$, $\overline{p_r p_k}$
        LegalizeEdge(p_r, $\overline{p_i p_j}$, T)
        LegalizeEdge(p_r, $\overline{p_j p_k}$, T)
        LegalizeEdge(p_r, $\overline{p_k p_i}$, T
```



Iterations

Compute a random permutation $p_1, p_2 \ldots, p_n$

for $r \leftarrow 1$ to n

  do

    find $\Delta\, p_i p_j p_k \supset p_r$ in the current triangulation T

    if p_r lies in its interior

      then // case 1

        add edges $\overline{p_r p_i}$, $\overline{p_r p_j}$, $\overline{p_r p_k}$

        LegalizeEdge$(p_r, \overline{p_i p_j}, T)$

        LegalizeEdge$(p_r, \overline{p_j p_k}, T)$

        LegalizeEdge$(p_r, \overline{p_k p_i}, T$

      else // case 2

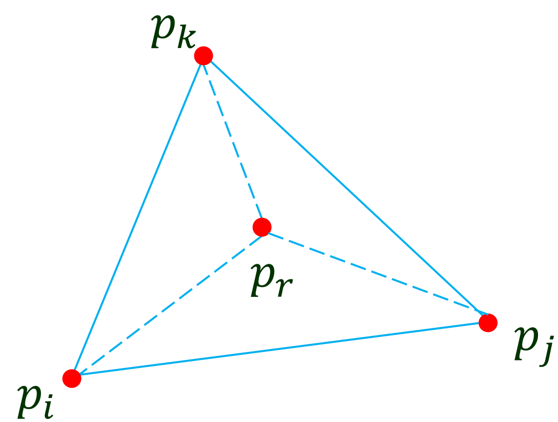
        add edges $\overline{p_r p_k}$, $\overline{p_r p_l}$

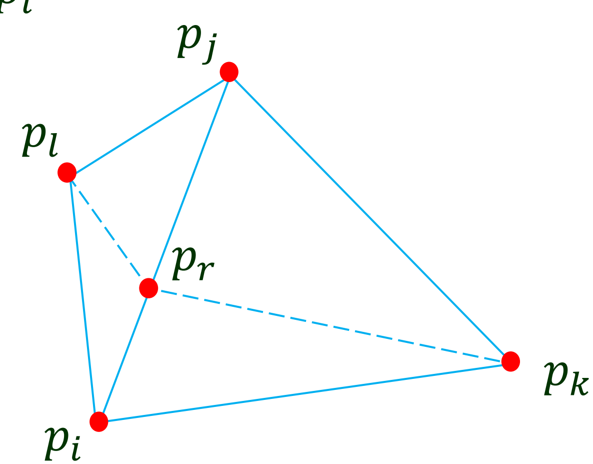
        LegalizeEdge$(p_r, \overline{p_i p_k}, T)$

        LegalizeEdge$(p_r, \overline{p_k p_j}, T)$

        LegalizeEdge$(p_r, \overline{p_j p_l}, T)$

        LegalizeEdge$(p_r, \overline{p_l p_i}, T)$





II. Correctness

Need to prove that no illegal edges remain after all calls to `LegalizeEdge`. Correctness is implied by the following:

II. Correctness

Need to prove that no illegal edges remain after all calls to `LegalizeEdge`. Correctness is implied by the following:

- Every new edge due to insertion of a point p_r is incident to p_r.

II. Correctness

Need to prove that no illegal edges remain after all calls to `LegalizeEdge`. Correctness is implied by the following:

- Every new edge due to insertion of a point p_r is incident to p_r.

  Ensured by the recursive calls.

II. Correctness

Need to prove that no illegal edges remain after all calls to `LegalizeEdge`. Correctness is implied by the following:

- Every new edge due to insertion of a point p_r is incident to p_r. ✓

Ensured by the recursive calls.

II. Correctness

Need to prove that no illegal edges remain after all calls to `LegalizeEdge`. Correctness is implied by the following:

- Every new edge due to insertion of a point p_r is incident to p_r.

  Ensured by the recursive calls.

- Every new edge is legal.

II. Correctness

Need to prove that no illegal edges remain after all calls to `LegalizeEdge`. Correctness is implied by the following:

- Every new edge due to insertion of a point p_r is incident to p_r.

  Ensured by the recursive calls.

- Every new edge is legal.

  To be shown in Lemma 1 next.

II. Correctness

Need to prove that no illegal edges remain after all calls to `LegalizeEdge`. Correctness is implied by the following:

- Every new edge due to insertion of a point p_r is incident to p_r. ✓

  Ensured by the recursive calls.

- Every new edge is legal.

  To be shown in Lemma 1 next.

- Any edge that may become illegal is tested.

II. Correctness

Need to prove that no illegal edges remain after all calls to `LegalizeEdge`. Correctness is implied by the following:

- Every new edge due to insertion of a point p_r is incident to p_r. ✓

    Ensured by the recursive calls.

- Every new edge is legal.

    To be shown in Lemma 1 next.

- Any edge that may become illegal is tested.

    Because an edge can only become illegal if one of its incident triangles changes.

II. Correctness

Need to prove that no illegal edges remain after all calls to `LegalizeEdge`. Correctness is implied by the following:

- Every new edge due to insertion of a point p_r is incident to p_r. ✓

  Ensured by the recursive calls.

- Every new edge is legal.

  To be shown in Lemma 1 next.

- Any edge that may become illegal is tested. ✓

  Because an edge can only become illegal if one of its incident triangles changes.

II. Correctness

Need to prove that no illegal edges remain after all calls to `LegalizeEdge`. Correctness is implied by the following:

- Every new edge due to insertion of a point p_r is incident to p_r. ✓

  Ensured by the recursive calls.

- Every new edge is legal.

  To be shown in Lemma 1 next.

- Any edge that may become illegal is tested. ✓

  Because an edge can only become illegal if one of its incident triangles changes.

- Algorithm terminates because every flips increases the angle vector of the triangulation.

II. Correctness

Need to prove that no illegal edges remain after all calls to `LegalizeEdge`. Correctness is implied by the following:

- Every new edge due to insertion of a point p_r is incident to p_r.

  Ensured by the recursive calls.

- Every new edge is legal.

  To be shown in Lemma 1 next.

- Any edge that may become illegal is tested.

  Because an edge can only become illegal if one of its incident triangles changes.

- Algorithm terminates because every flips increases the angle vector of the triangulation.

Legality of Every New Edge

Lemma 1 Every new edge created during the insertion of p_r is an edge of $DG(\{p_0, p_{-1}, p_{-2}, p_1, \ldots, p_r\})$.

Legality of Every New Edge

Lemma 1 Every new edge created during the insertion of p_r is an edge of $DG(\{p_0, p_{-1}, p_{-2}, p_1, \ldots, p_r\})$.

Proof Examine two types of edges.

Legality of Every New Edge

Lemma 1 Every new edge created during the insertion of p_r is an edge of $DG(\{p_0, p_{-1}, p_{-2}, p_1, \ldots, p_r\})$.

Proof Examine two types of edges.

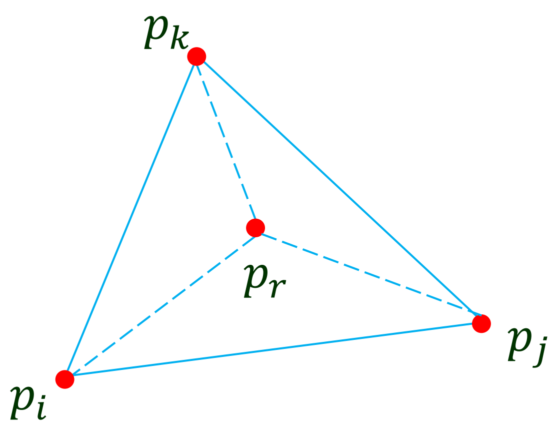
- The 1st type of edges (cases 1 & 2) are added right after insertion of p_r.

Legality of Every New Edge

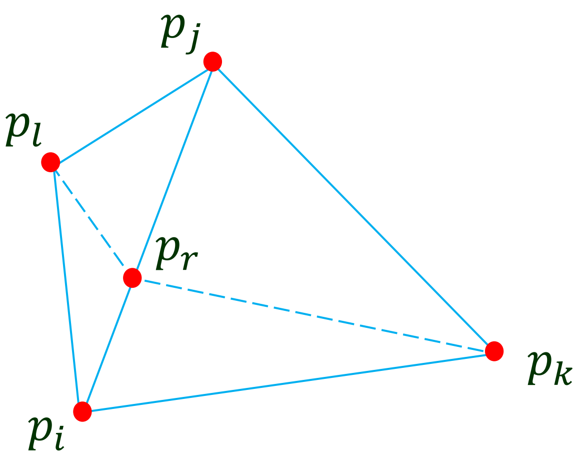
Lemma 1 Every new edge created during the insertion of p_r is an edge of $DG(\{p_0, p_{-1}, p_{-2}, p_1, \ldots, p_r\})$.

Proof Examine two types of edges.

- The 1st type of edges (cases 1 & 2) are added right after insertion of p_r.



Case 1

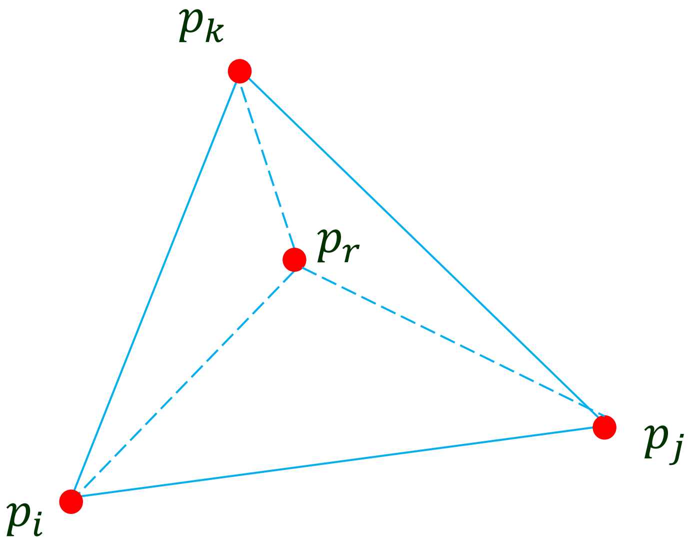


Case 2

Immediately Added Edges

- Case 1

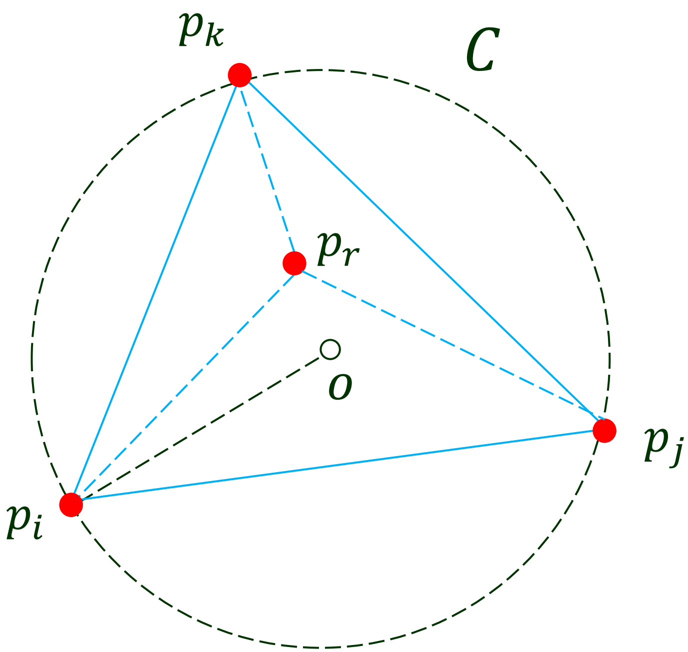
  $\Delta\, p_i p_j p_k$ is a triangle in $DG(\Omega \cup \{p_1, \ldots, p_{r-1}\})$.



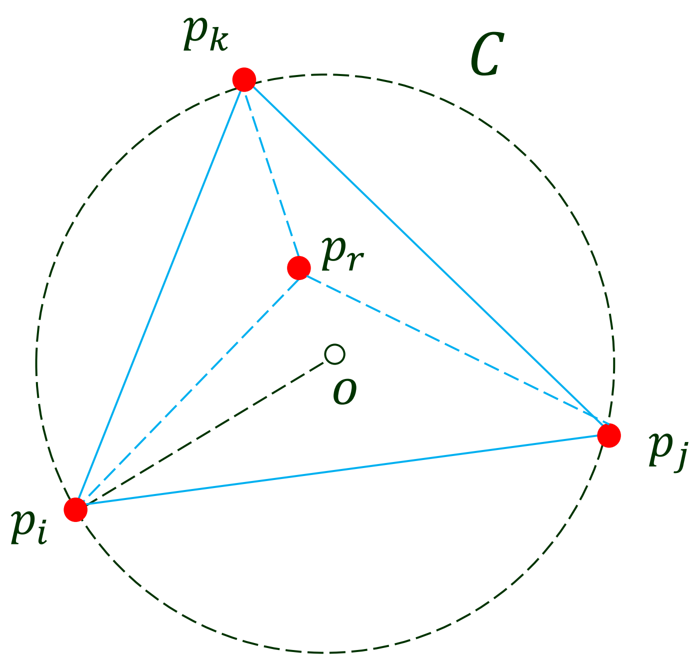
Immediately Added Edges

- Case 1

$\Delta\, p_i p_j p_k$ is a triangle in $DG(\Omega \cup \{p_1, \ldots, p_{r-1}\})$.



Immediately Added Edges



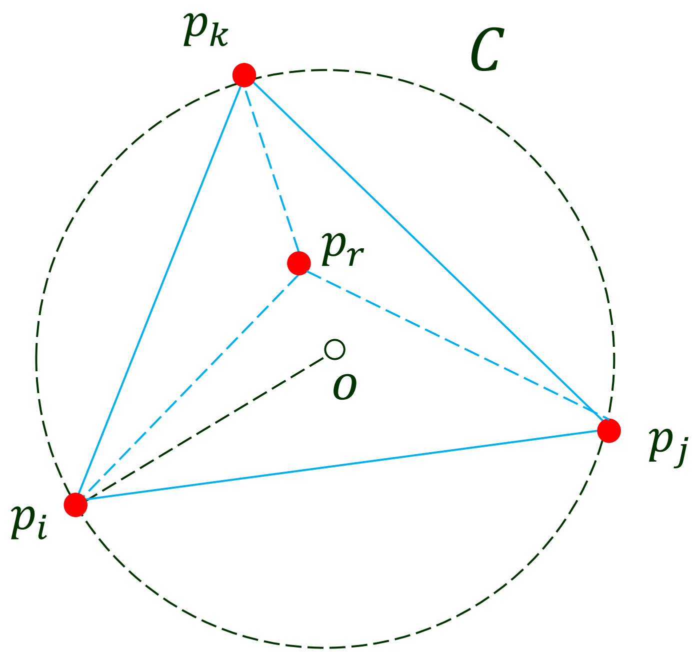
- Case 1

$\Delta\, p_i p_j p_k$ is a triangle in $DG(\Omega \cup \{p_1, \ldots, p_{r-1}\})$.

$\Downarrow$

The circumcircle C contains no point p_l, $l < r$, in its interior.

Immediately Added Edges



- Case 1

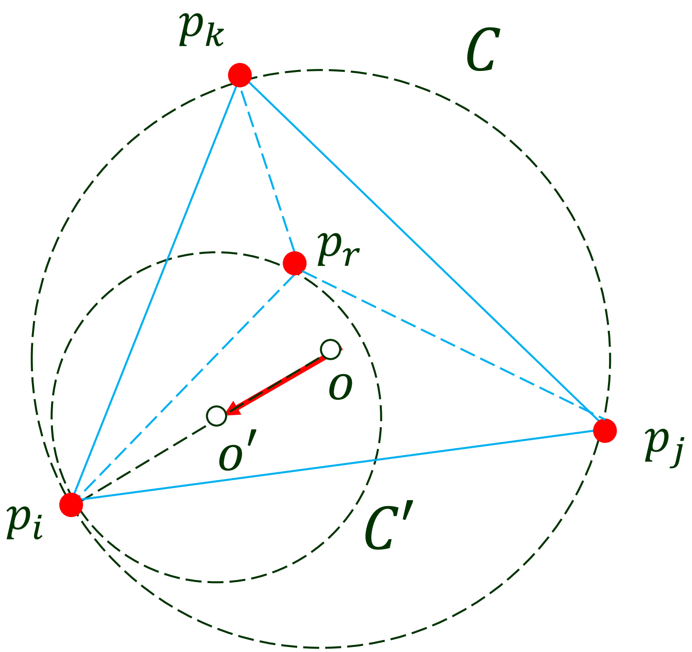
$\Delta\, p_i p_j p_k$ is a triangle in $DG(\Omega \cup \{p_1, \ldots, p_{r-1}\})$.

$\Downarrow$

The circumcircle C contains no point p_l, $l < r$, in its interior.

Shrink C (centered at o) to a circle C' (centered at o') through p_i and p_r.

Immediately Added Edges



- Case 1

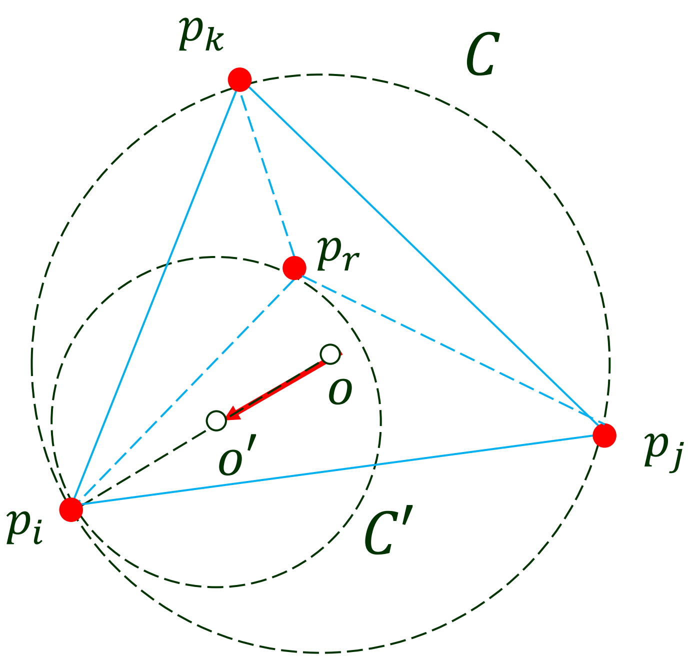
$\Delta\, p_i p_j p_k$ is a triangle in $DG(\Omega \cup \{p_1, \dots, p_{r-1}\})$.

$\Downarrow$

The circumcircle C contains no point p_l, $l < r$, in its interior.

Shrink C (centered at o) to a circle C' (centered at o') through p_i and p_r.

Immediately Added Edges



- Case 1

$\Delta\, p_i p_j p_k$ is a triangle in $DG(\Omega \cup \{p_1, \ldots, p_{r-1}\})$.

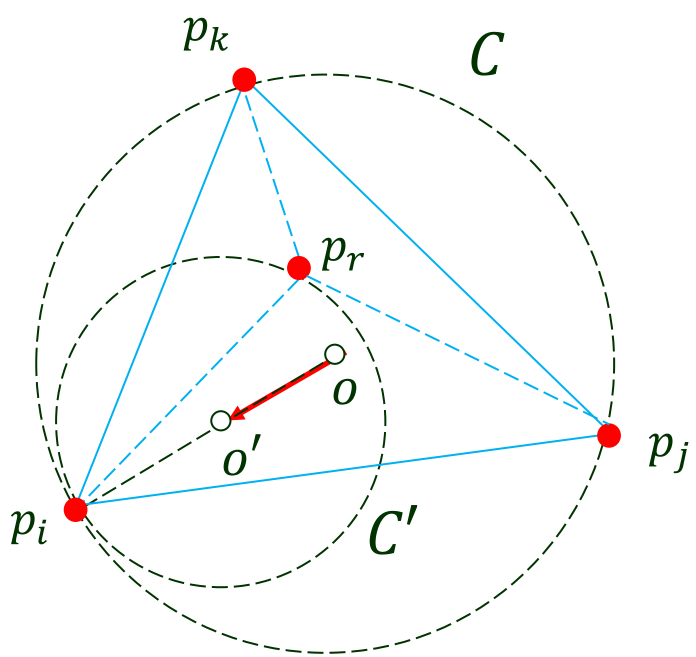
$\Downarrow$

The circumcircle C contains no point p_l, $l < r$, in its interior.

Shrink C (centered at o) to a circle C' (centered at o') through p_i and p_r.

C' is empty.

Immediately Added Edges



- Case 1

$\Delta\, p_i p_j p_k$ is a triangle in $DG(\Omega \cup \{p_1, \ldots, p_{r-1}\})$.

$\Downarrow$

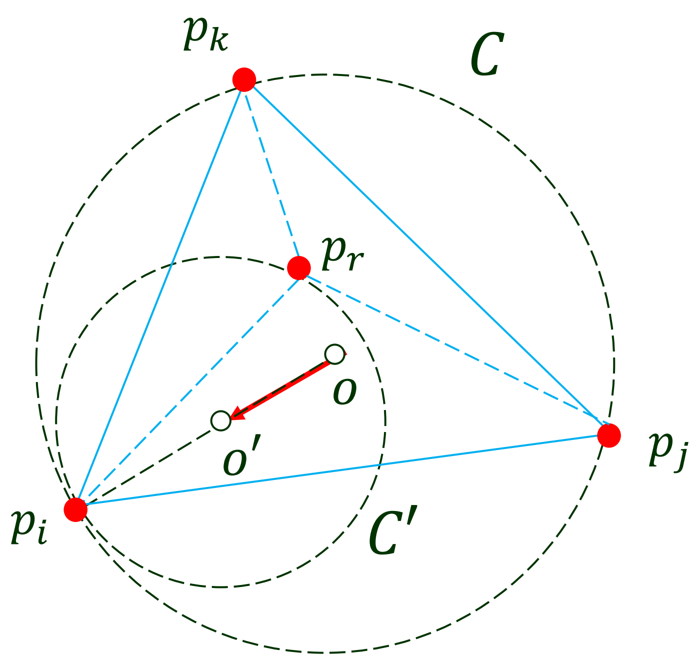
The circumcircle C contains no point p_l, $l < r$, in its interior.

Shrink C (centered at o) to a circle C' (centered at o') through p_i and p_r.

C' is empty. $\Rightarrow$ $\overline{p_r p_i}$ an edge of the DG after addition of p_r.

Immediately Added Edges

- Case 1



$\Delta\, p_i p_j p_k$ is a triangle in $DG(\Omega \cup \{p_1, \ldots, p_{r-1}\})$.

$\Downarrow$

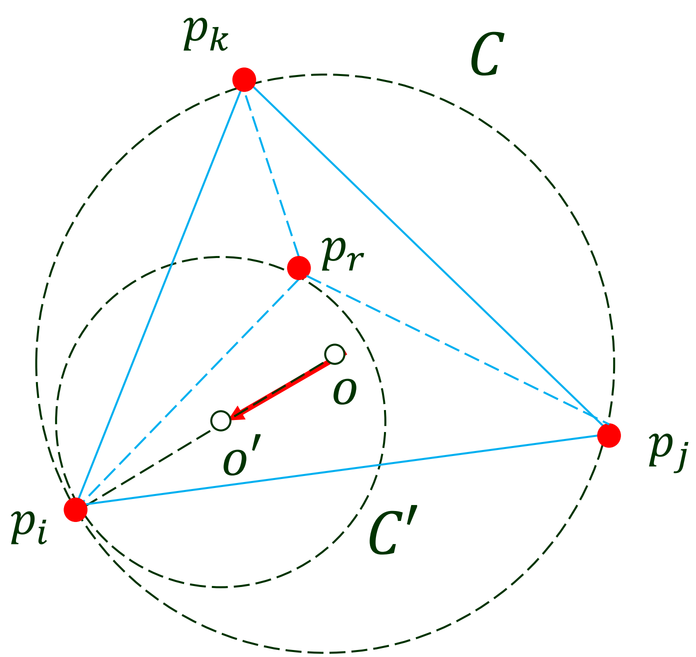
The circumcircle C contains no point p_l, $l < r$, in its interior.

Shrink C (centered at o) to a circle C' (centered at o') through p_i and p_r.

C' is empty. $\Rightarrow$ $\overline{p_r p_i}$ an edge of the DG after addition of p_r.

Similarly, $\overline{p_r p_j}$ and $\overline{p_r p_k}$ are edges too.

Immediately Added Edges



- Case 1

$\Delta\, p_i p_j p_k$ is a triangle in $DG(\Omega \cup \{p_1, \ldots, p_{r-1}\})$.

$\Downarrow$

The circumcircle C contains no point p_l, $l < r$, in its interior.

Shrink C (centered at o) to a circle C' (centered at o') through p_i and p_r.

C' is empty. $\Rightarrow$ $\overline{p_r p_i}$ an edge of the DG after addition of p_r.

Similarly, $\overline{p_r p_j}$ and $\overline{p_r p_k}$ are edges too.

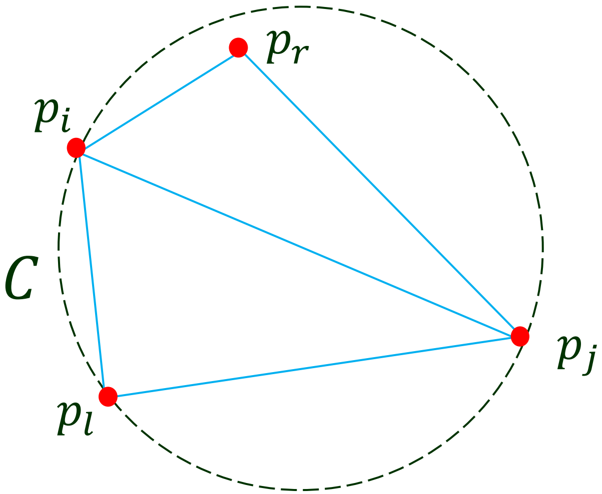
- Case 2

Similar to Case 1.

Edges Added Due to Flipping

- The 2nd type of edges are added due to flipping by `LegalizeEdge`.

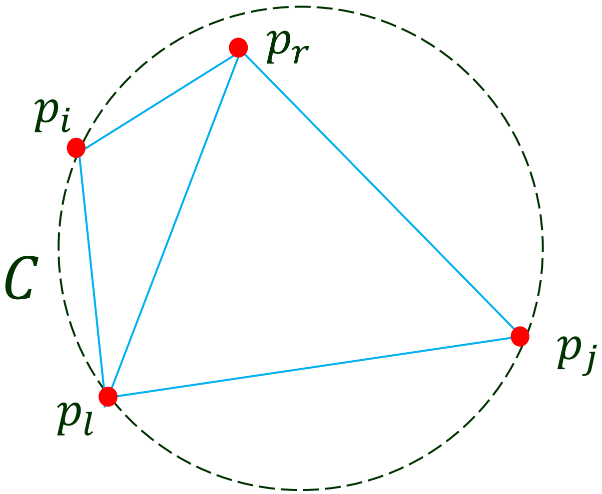
Suppose $\overline{p_i p_j}$ of $\Delta\, p_i p_j p_l$ is replaced by $\overline{p_r p_l}$.



Edges Added Due to Flipping

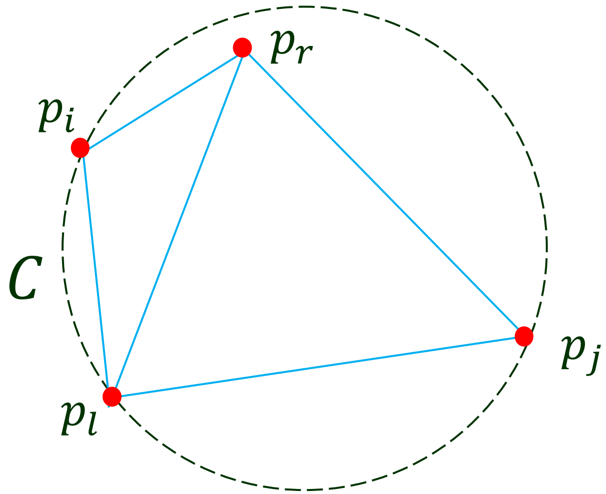
- The 2nd type of edges are added due to flipping by `LegalizeEdge`.

Suppose $\overline{p_i p_j}$ of $\Delta\, p_i p_j p_l$ is replaced by $\overline{p_r p_l}$.



Edges Added Due to Flipping

- The 2nd type of edges are added due to flipping by `LegalizeEdge`.

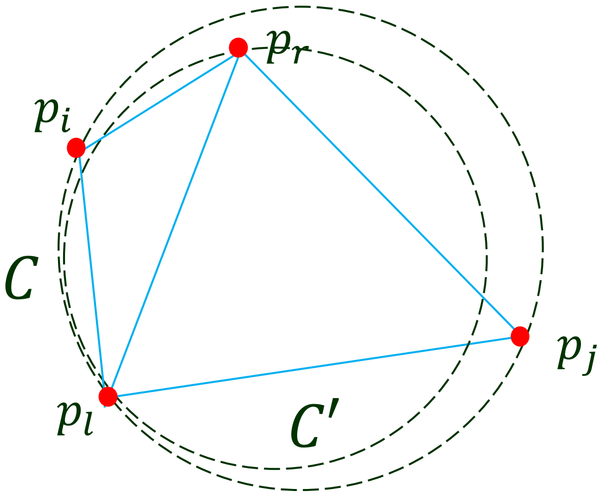


Suppose $\overline{p_i p_j}$ of $\Delta\, p_i p_j p_l$ is replaced by $\overline{p_r p_l}$.

$\Delta\, p_i p_j p_l$ was a Delaunay triangle and its circumcircle C contains p_r only.

Edges Added Due to Flipping

- The 2nd type of edges are added due to flipping by `LegalizeEdge`.



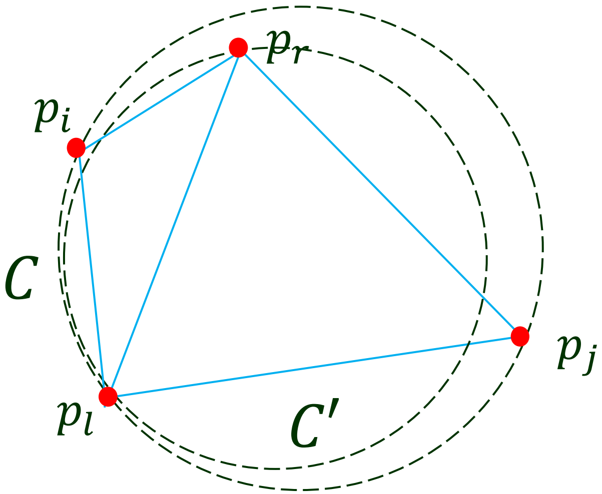
Suppose $\overline{p_i p_j}$ of $\Delta\, p_i p_j p_l$ is replaced by $\overline{p_r p_l}$.

$\Delta\, p_i p_j p_l$ was a Delaunay triangle and its circumcircle C contains p_r only.

Shrink C to a circle C' with only p_r and p_l on its boundary.

Edges Added Due to Flipping

- The 2nd type of edges are added due to flipping by `LegalizeEdge`.



Suppose $\overline{p_i p_j}$ of $\Delta\, p_i p_j p_l$ is replaced by $\overline{p_r p_l}$.

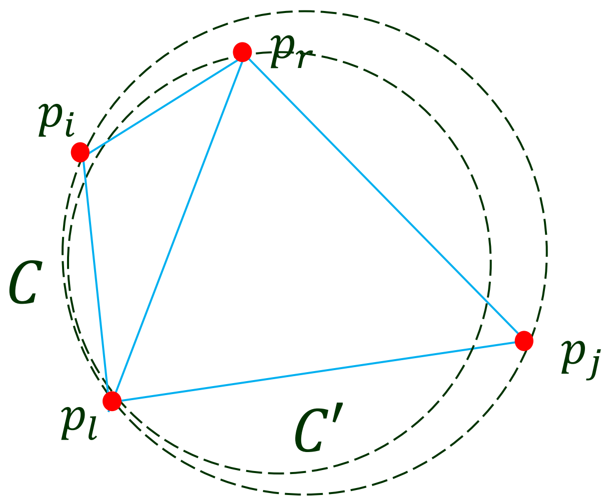
$\Delta\, p_i p_j p_l$ was a Delaunay triangle and its circumcircle C contains p_r only.

Shrink C to a circle C' with only p_r and p_l on its boundary.

C' is empty.

Edges Added Due to Flipping

- The 2nd type of edges are added due to flipping by `LegalizeEdge`.



Suppose $\overline{p_i p_j}$ of $\Delta\, p_i p_j p_l$ is replaced by $\overline{p_r p_l}$.

$\Delta\, p_i p_j p_l$ was a Delaunay triangle and its circumcircle C contains p_r only.

Shrink C to a circle C' with only p_r and p_l on its boundary.

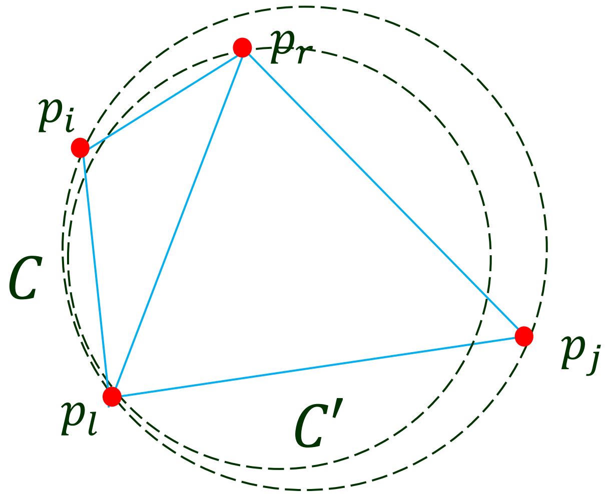
C' is empty.

$\Downarrow$

$\overline{p_r p_l}$ is a Delaunay edge after the addition.

Edges Added Due to Flipping

- The 2nd type of edges are added due to flipping by `LegalizeEdge`.



Suppose $\overline{p_i p_j}$ of $\Delta\, p_i p_j p_l$ is replaced by $\overline{p_r p_l}$.

$\Delta\, p_i p_j p_l$ was a Delaunay triangle and its circumcircle C contains p_r only.

Shrink C to a circle C' with only p_r and p_l on its boundary.

C' is empty.

$\Downarrow$

$\overline{p_r p_l}$ is a Delaunay edge after the addition.

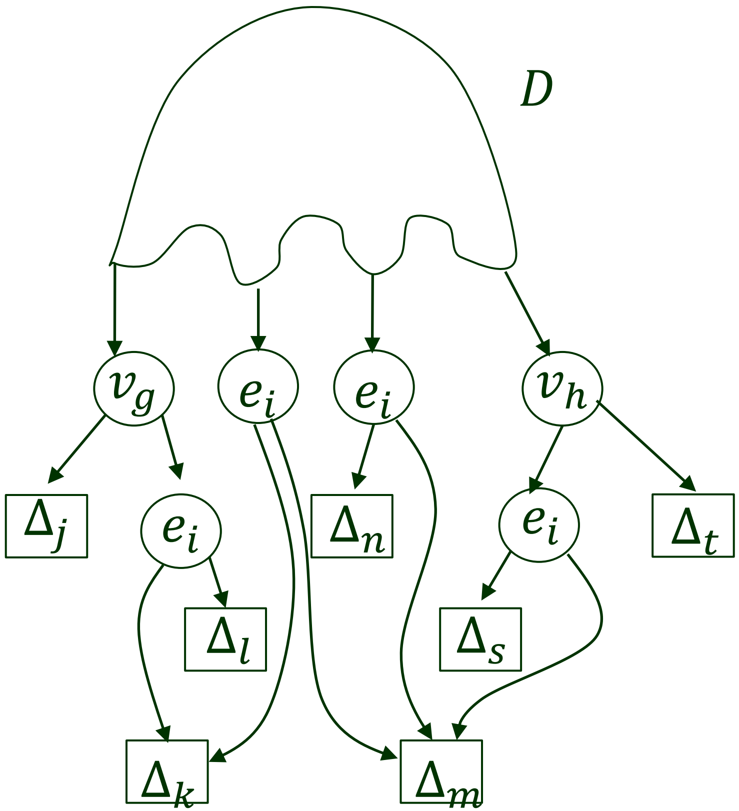
III. Locating the Containing Triangle

Build a point location structure D as a directed acyclic graph.

III. Locating the Containing Triangle

Build a point location structure D as a directed acyclic graph.

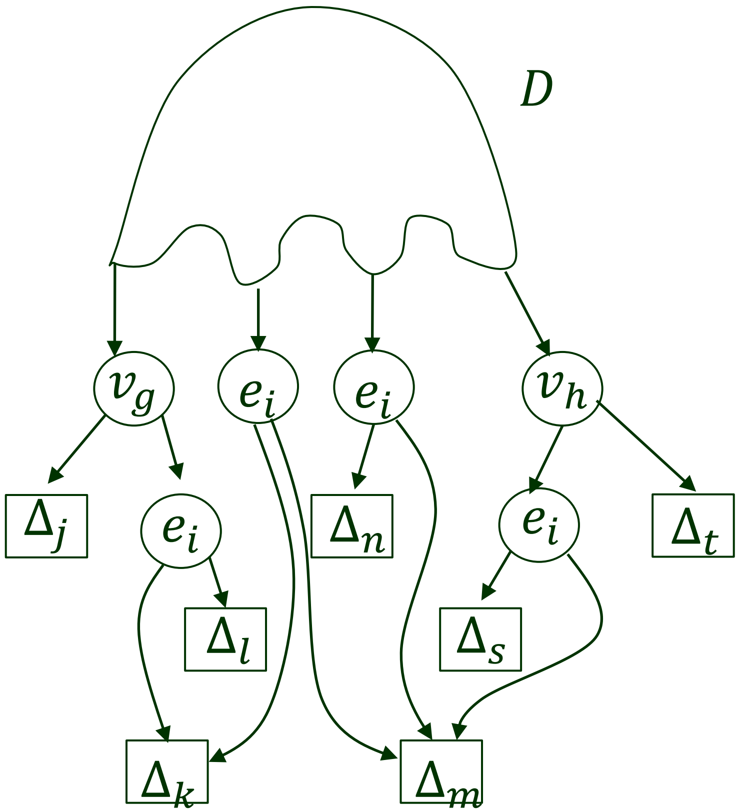
Trapezoidal map with only triangles no trapezoids.



III. Locating the Containing Triangle

Build a point location structure D as a directed acyclic graph.

Trapezoidal map with only triangles no trapezoids.

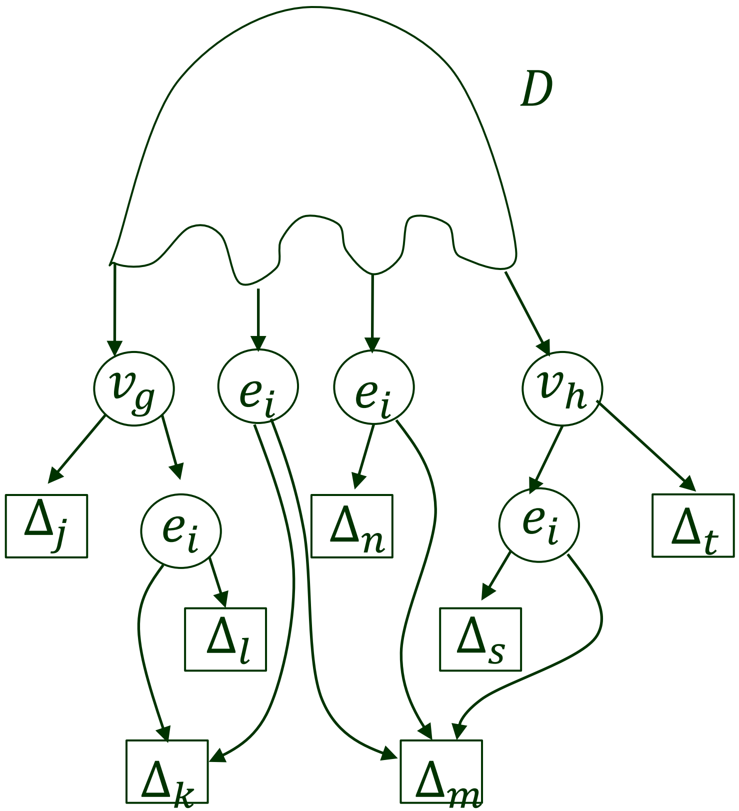


- Leaves: triangles of the current triangulation T.

III. Locating the Containing Triangle

Build a point location structure D as a directed acyclic graph.

Trapezoidal map with only triangles no trapezoids.

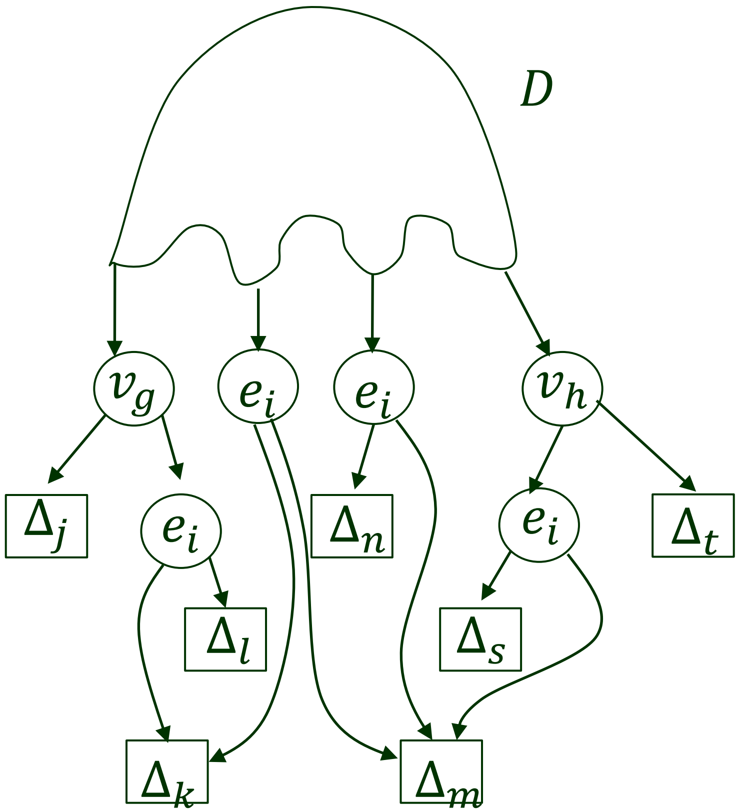


- Leaves: triangles of the current triangulation T.
- Internal nodes: triangles that existed before but have been destroyed.

III. Locating the Containing Triangle

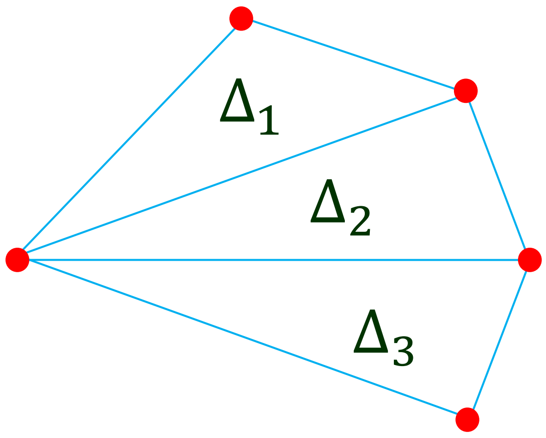
Build a point location structure D as a directed acyclic graph.

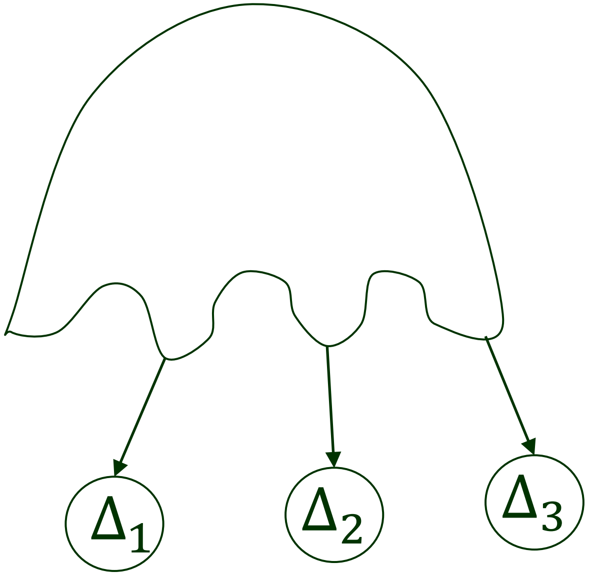
Trapezoidal map with only triangles no trapezoids.



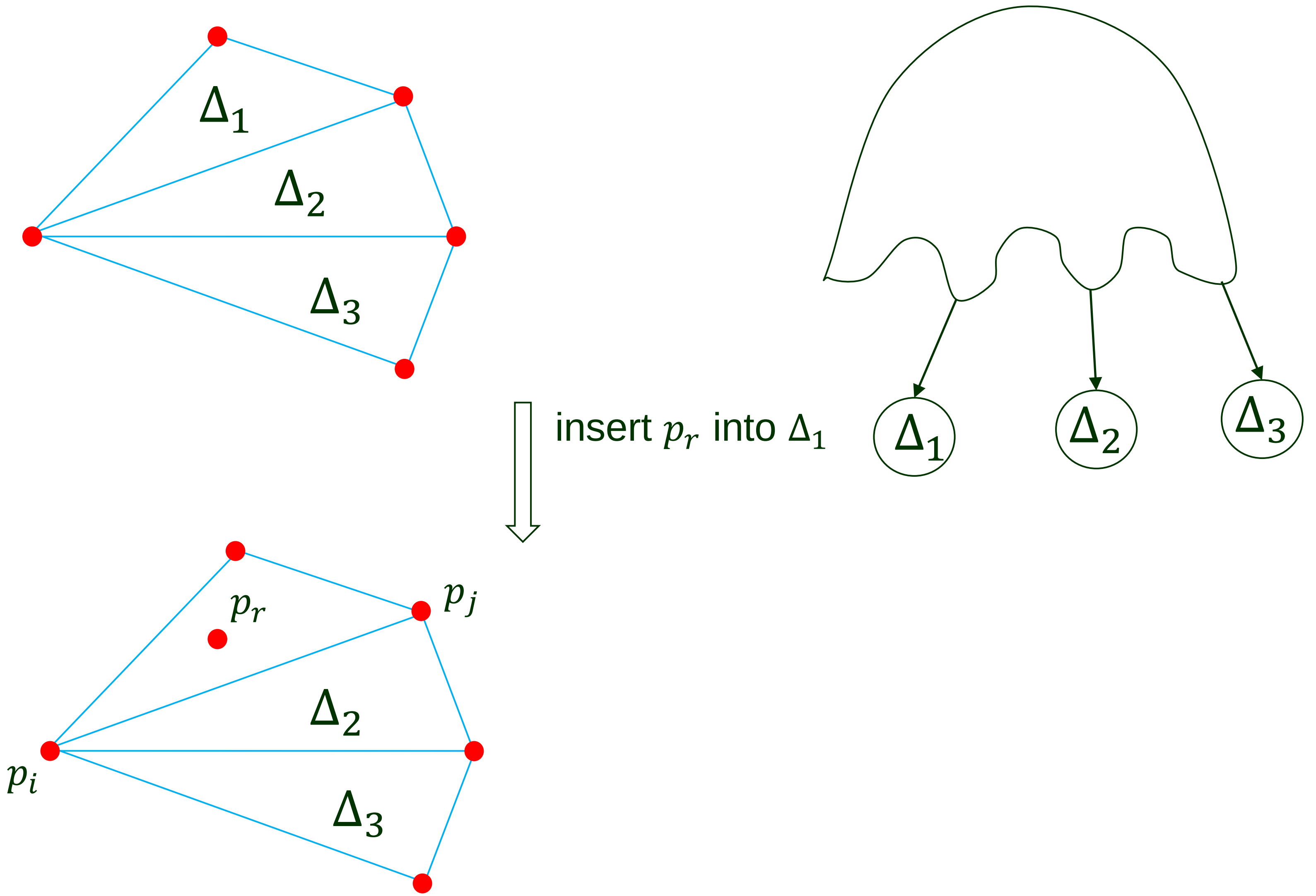
- Leaves: triangles of the current triangulation T.
- Internal nodes: triangles that existed before but have been destroyed.
- Initialized as a DAG with one node ($\Delta\, p_0 p_{-1} p_{-2}$).

Example



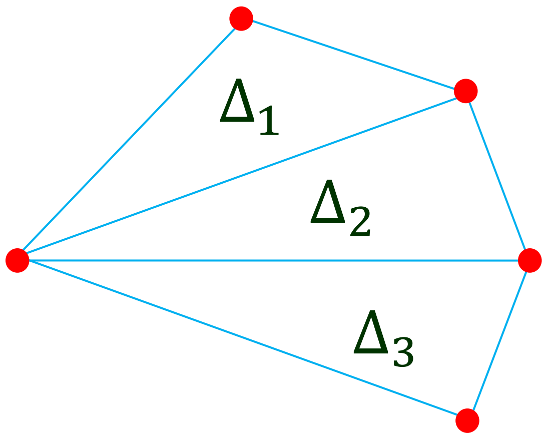


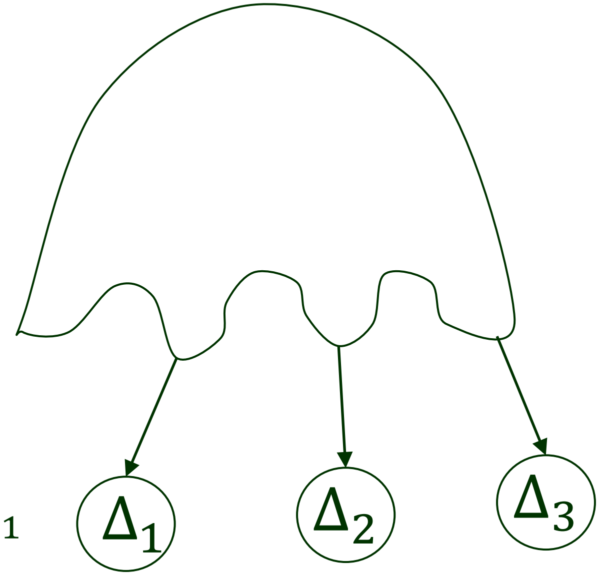
Example



Δ_1
Δ_2
Δ_3
insert p_r into Δ_1
Δ_1
Δ_2
Δ_3
p_r
p_j
Δ_2
p_i
Δ_3

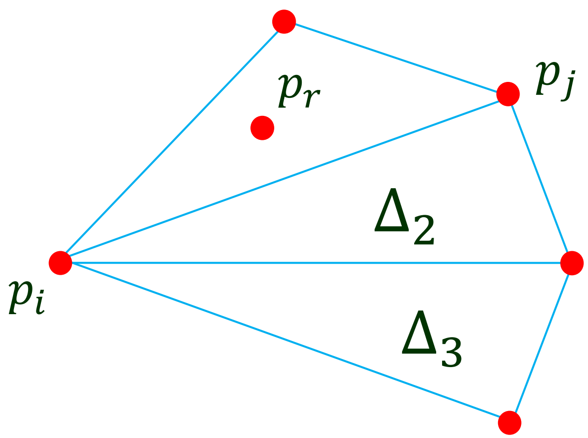
Example



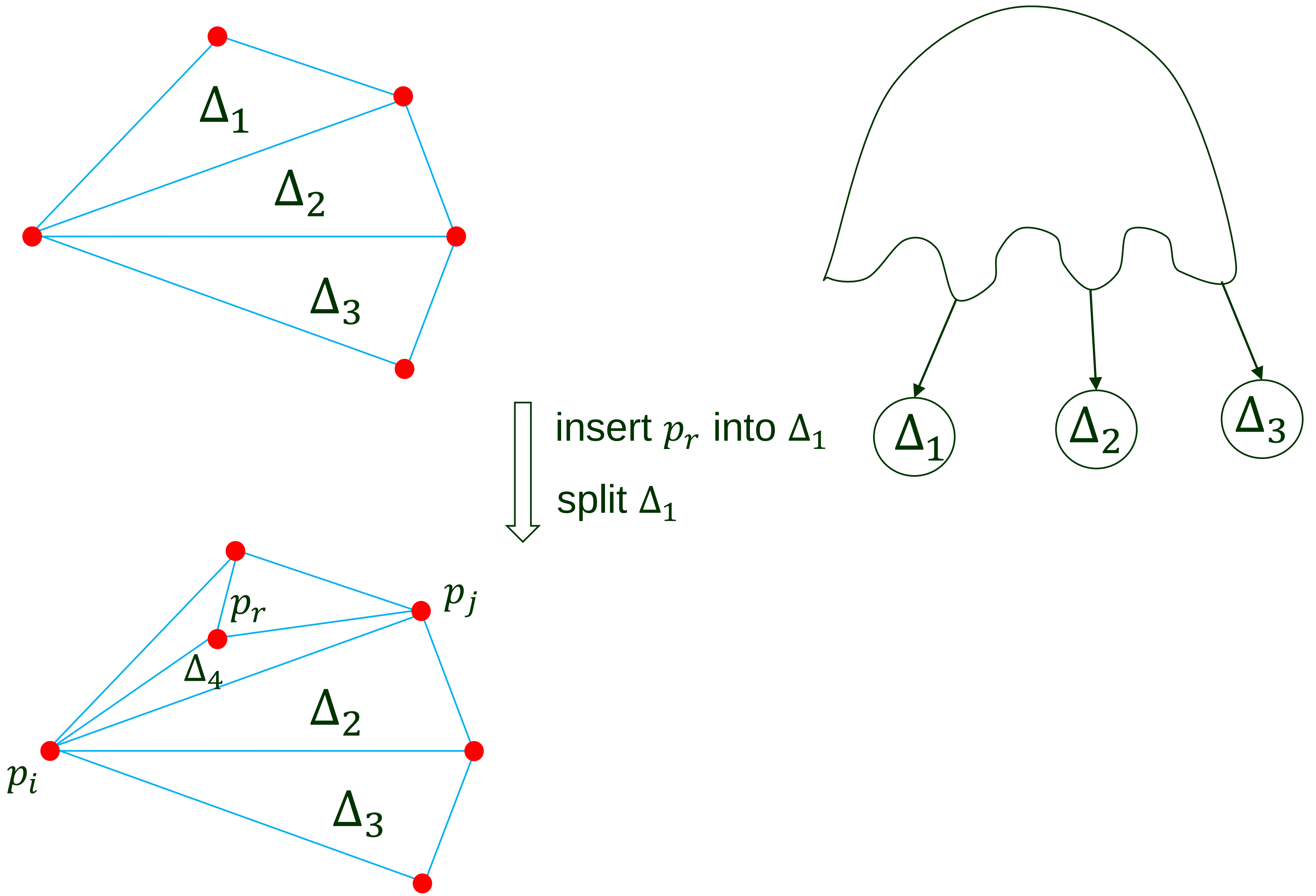


insert p_r into Δ_1

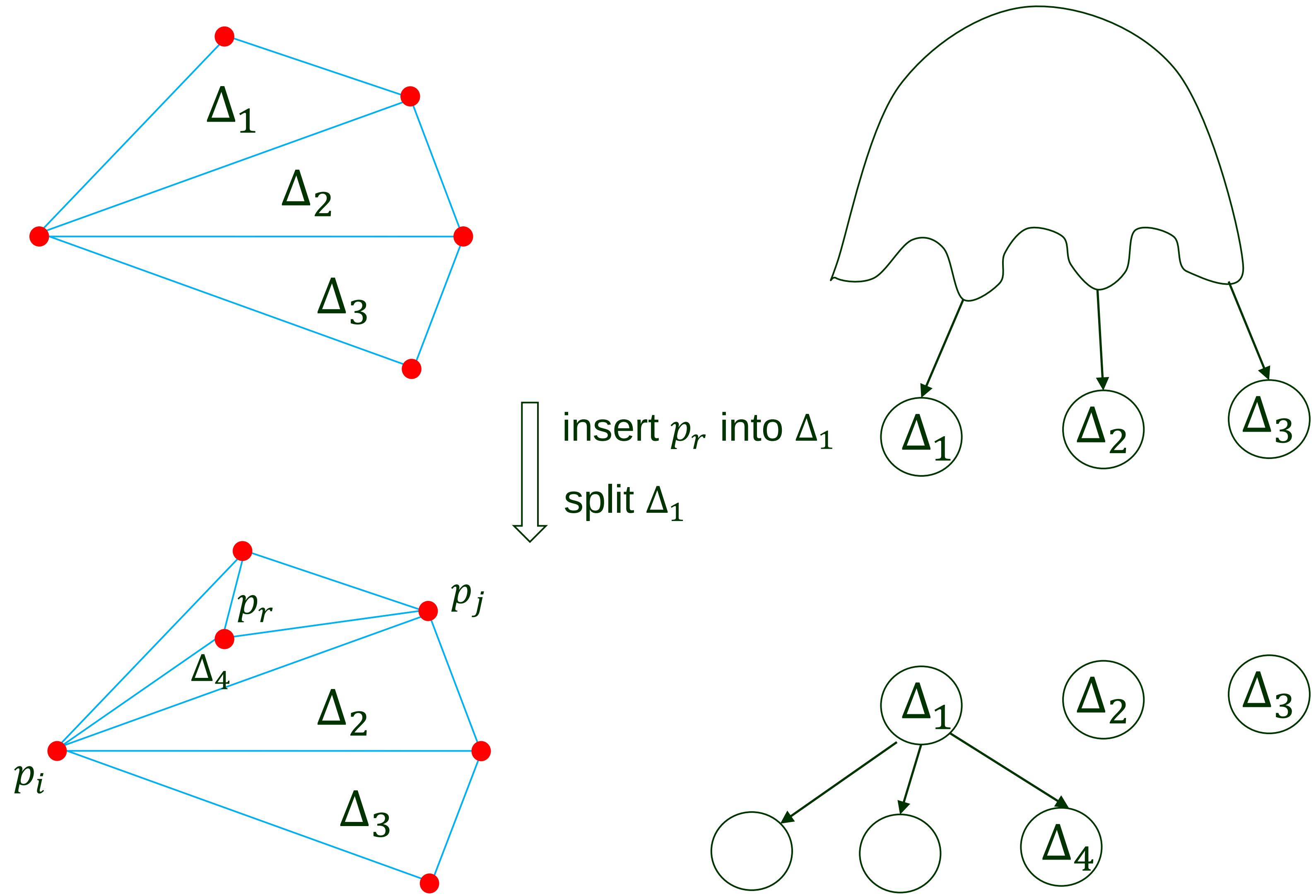
split Δ_1



Example

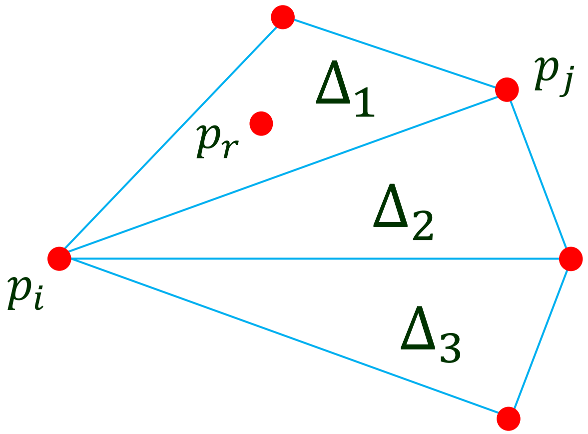


Example



Insertion

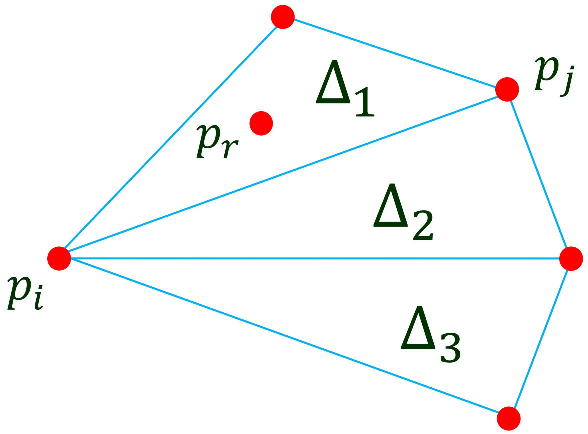
Locate p_r in $DG(\{p_{-2}, p_{-1}, p_0, p_1, \ldots, p_{r-1}\})$



- Start at the root ($\Delta\, p_0 p_{-1} p_{-2}$) of D.

Insertion

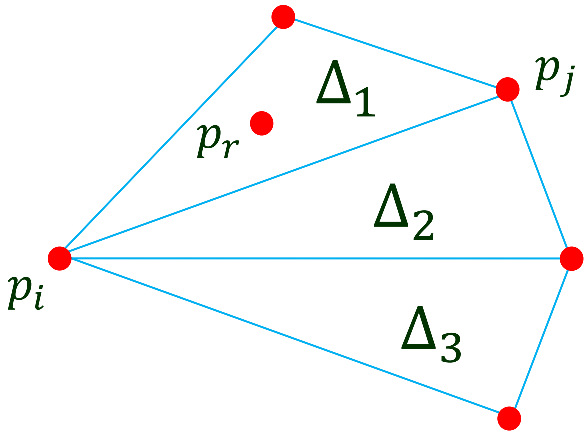
Locate p_r in $DG(\{p_{-2}, p_{-1}, p_0, p_1, \dots, p_{r-1}\})$



- Start at the root ($\Delta\, p_0 p_{-1} p_{-2}$) of D.
- Check its three children to see which one contains p_r.

Insertion

Locate p_r in $DG(\{p_{-2}, p_{-1}, p_0, p_1, \ldots, p_{r-1}\})$



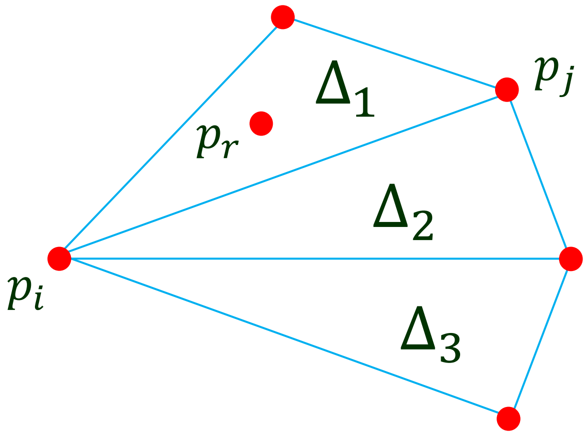
- Start at the root ($\Delta\ p_0 p_{-1} p_{-2}$) of D.

- Check its three children to see which one contains p_r.

created from addition of p_1 (which is in the interior of $\Delta\ p_0 p_{-1} p_{-2}$)

Insertion

Locate p_r in $DG(\{p_{-2}, p_{-1}, p_0, p_1, \dots, p_{r-1}\})$

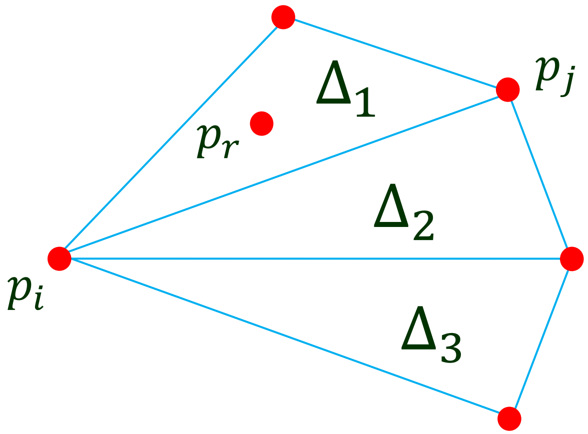


- Start at the root ($\Delta\, p_0 p_{-1} p_{-2}$) of D.
- Check its three children to see which one contains p_r.

  created from addition of p_1 (which is in the interior of $\Delta\, p_0 p_{-1} p_{-2}$)
- Descends to this child.

Insertion

Locate p_r in $DG(\{p_{-2}, p_{-1}, p_0, p_1, \ldots, p_{r-1}\})$



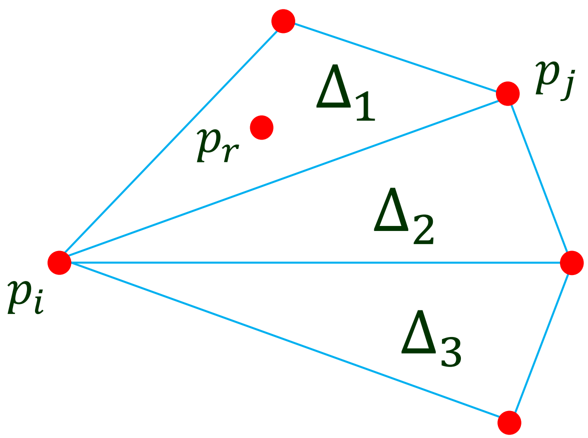
- Start at the root ($\Delta\, p_0 p_{-1} p_{-2}$) of D.
- Check its three children to see which one contains p_r.

  created from addition of p_1 (which is in the interior of $\Delta\, p_0 p_{-1} p_{-2}$)

- Descends to this child.
- Repeat the above two steps to reach a leaf.

Insertion

Locate p_r in $DG(\{p_{-2}, p_{-1}, p_0, p_1, \dots, p_{r-1}\})$



- Start at the root ($\Delta\, p_0 p_{-1} p_{-2}$) of D.
- Check its three children to see which one contains p_r.

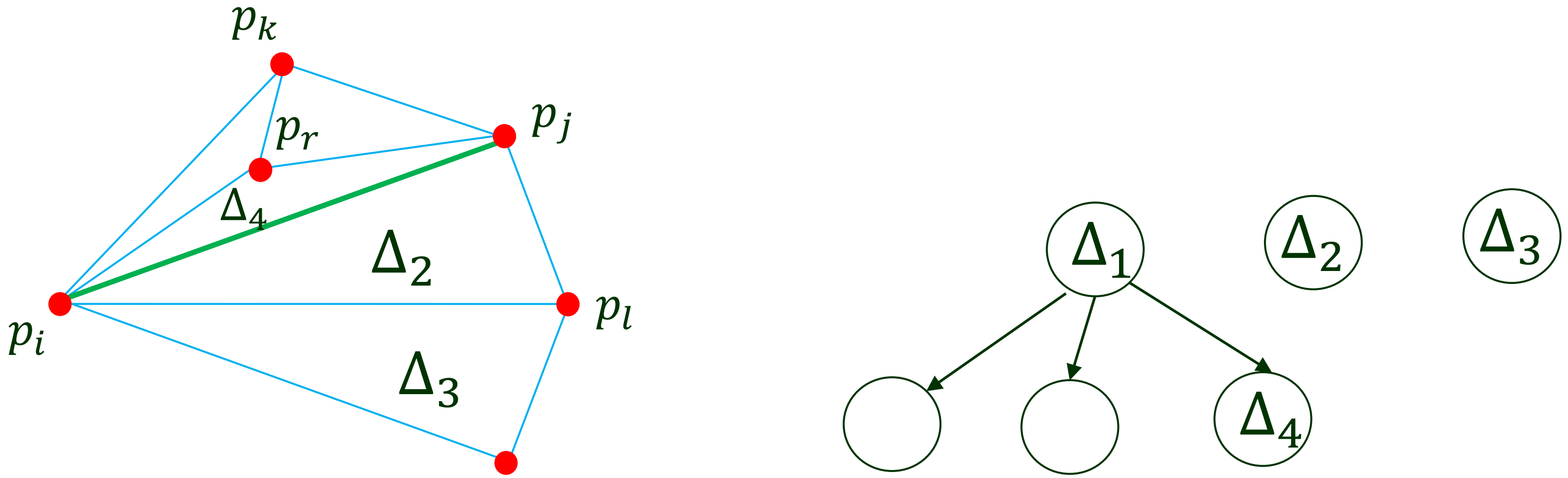
  created from addition of p_1 (which is in the interior of $\Delta\, p_0 p_{-1} p_{-2}$)

- Descends to this child.
- Repeat the above two steps to reach a leaf.

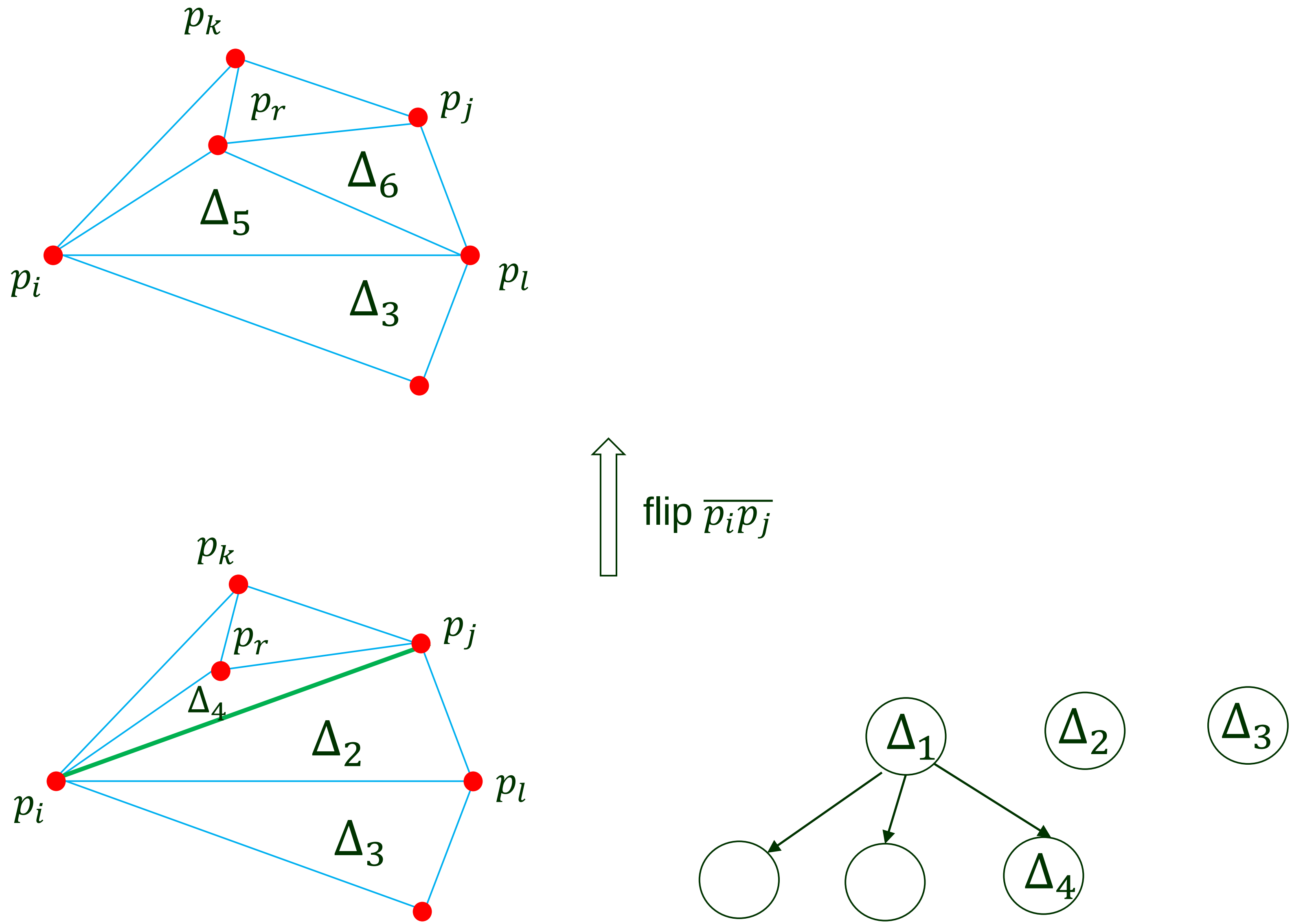
Time linear in

#nodes on the search path $=$ #triangles stored in D that contains p_r

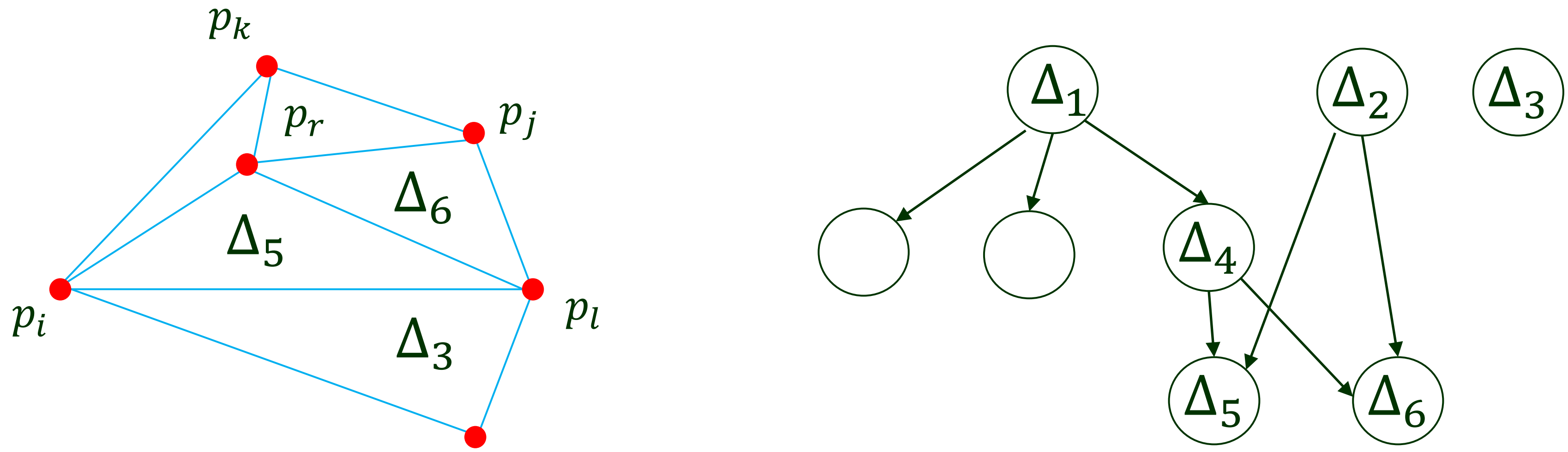
Example (cont'd)



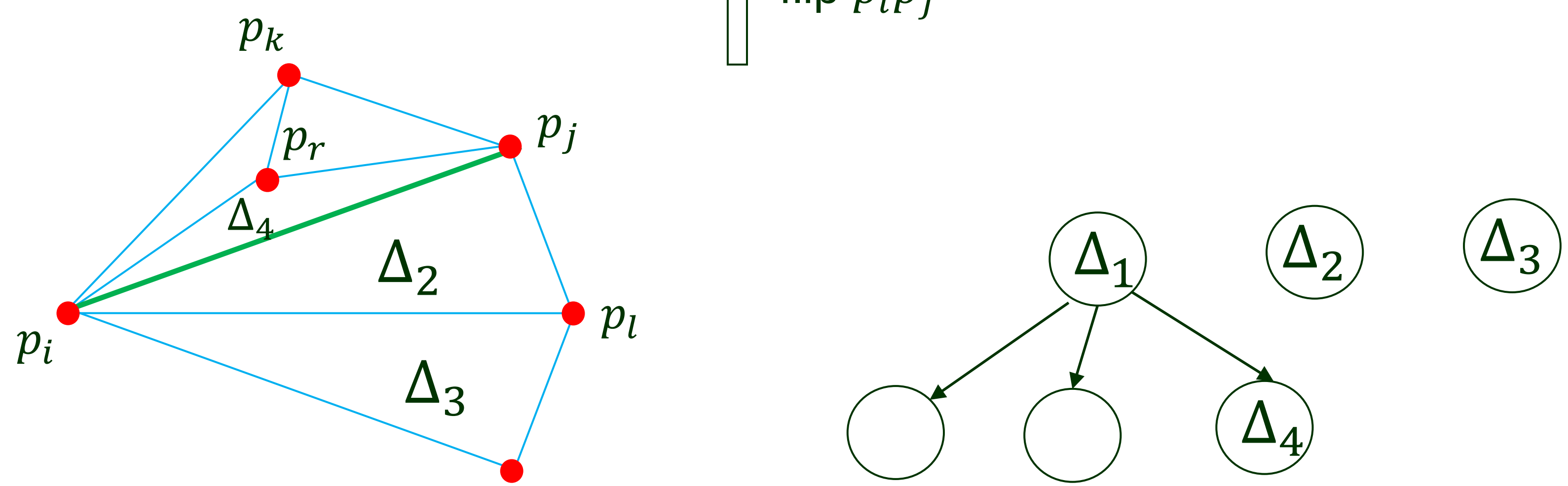
Example (cont'd)



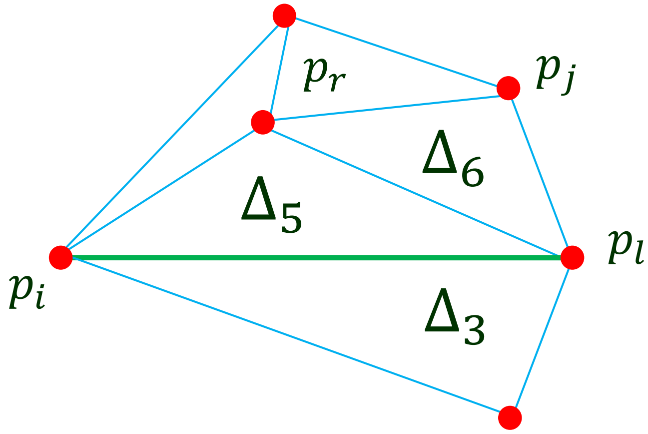
Example (cont'd)

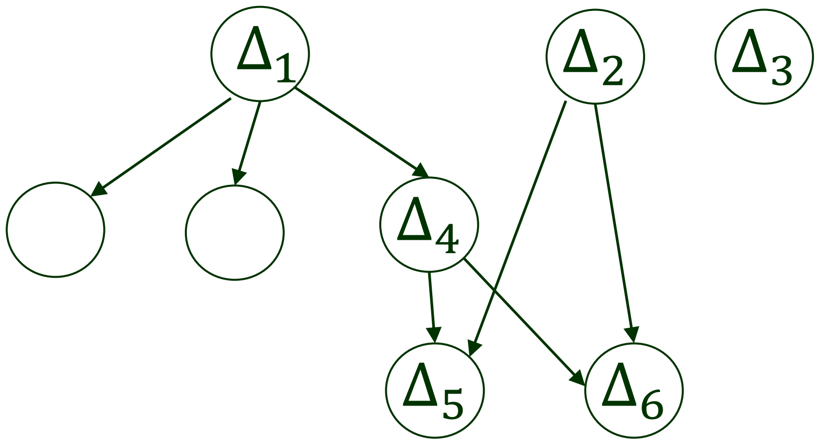


flip $\overline{p_i p_j}$

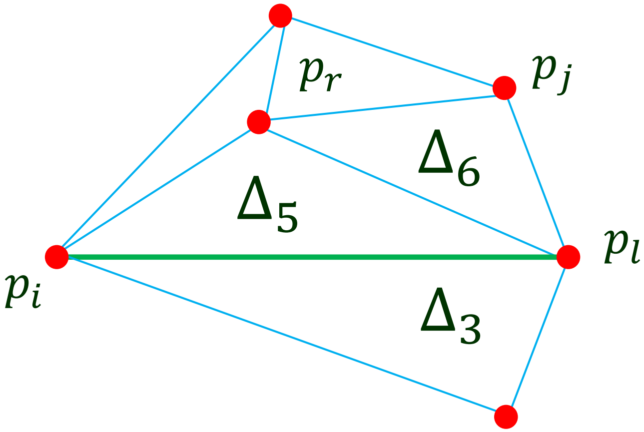


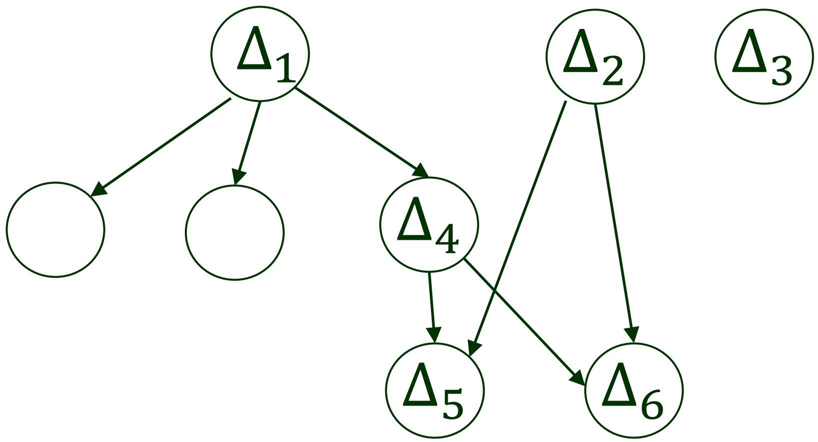
Example (finish)



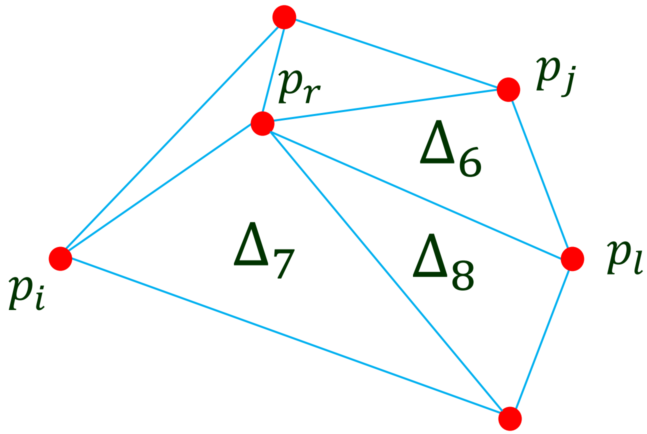


Example (finish)

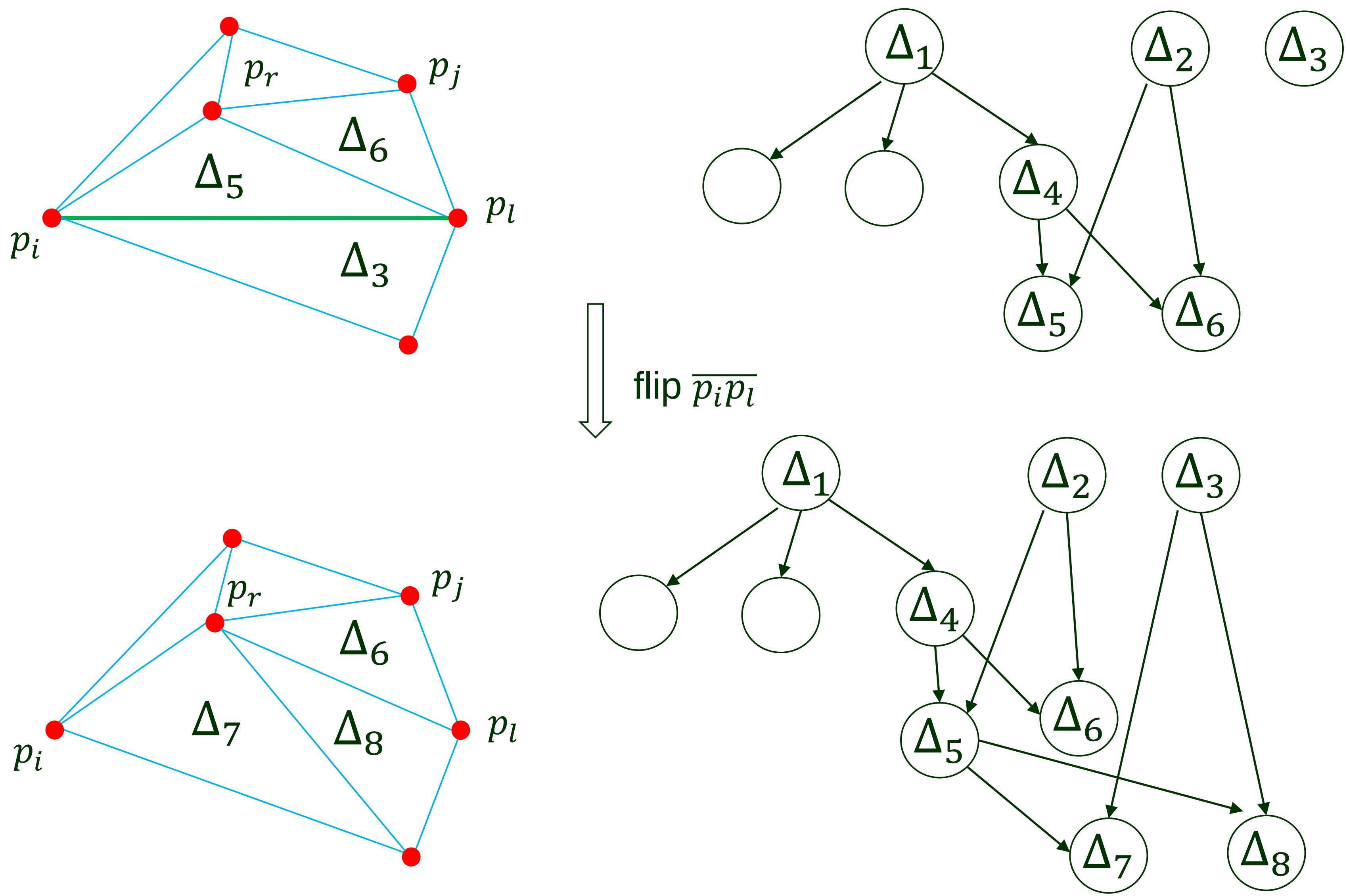




flip $\overline{p_i p_l}$

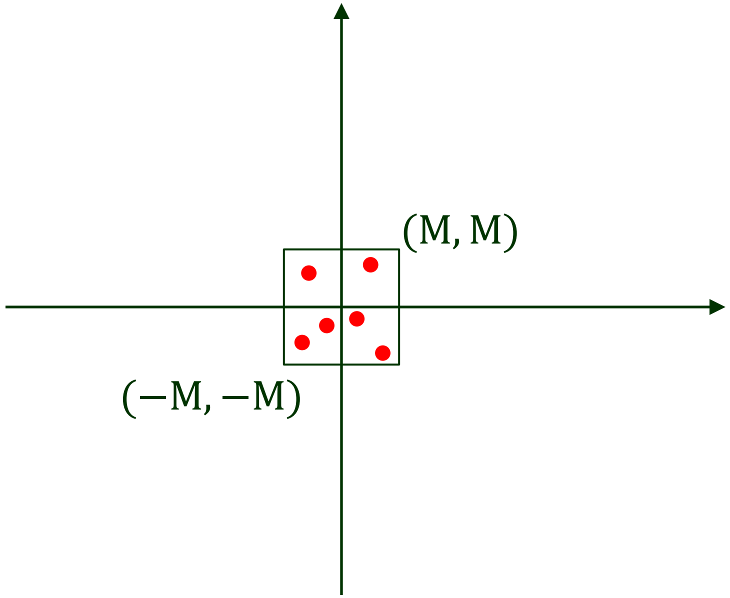


Example (finish)

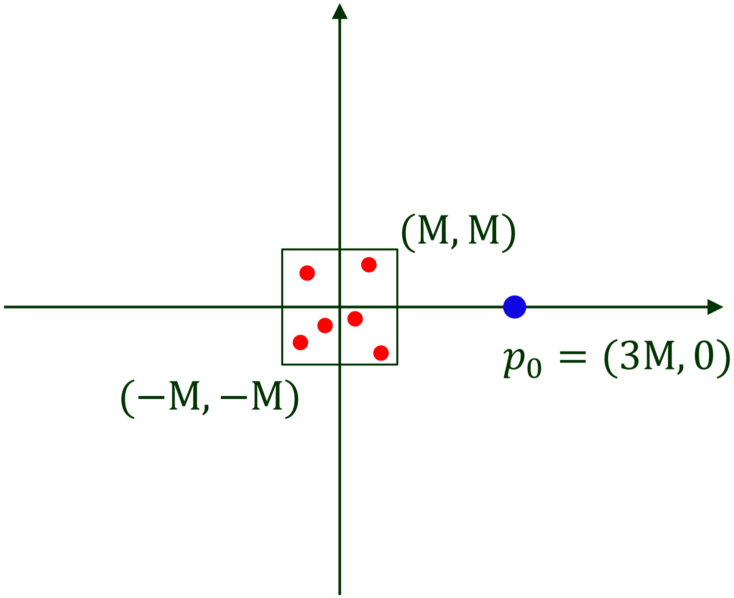


Selecting p_{-2}, p_{-1}, p_0

$$M = \max_{\substack{p_i=(x_i,y_i)\\ 1\le i\le n}} \{|x_i|, |y_i|\}$$



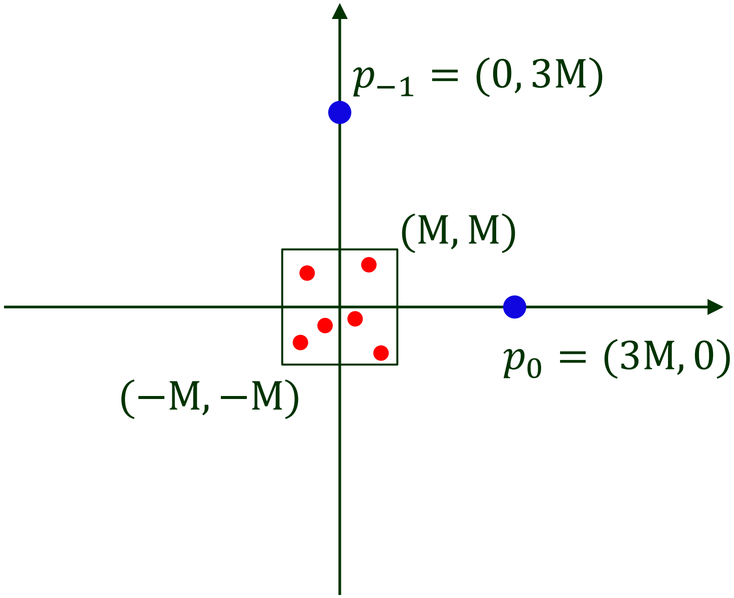
Selecting p_{-2}, p_{-1}, p_0



$$M = \max_{\substack{p_i=(x_i,y_i)\\ 1\le i\le n}} \{|x_i|, |y_i|\}$$

- p_0 lies outside circles defined by any three points in P.

Selecting p_{-2}, p_{-1}, p_0

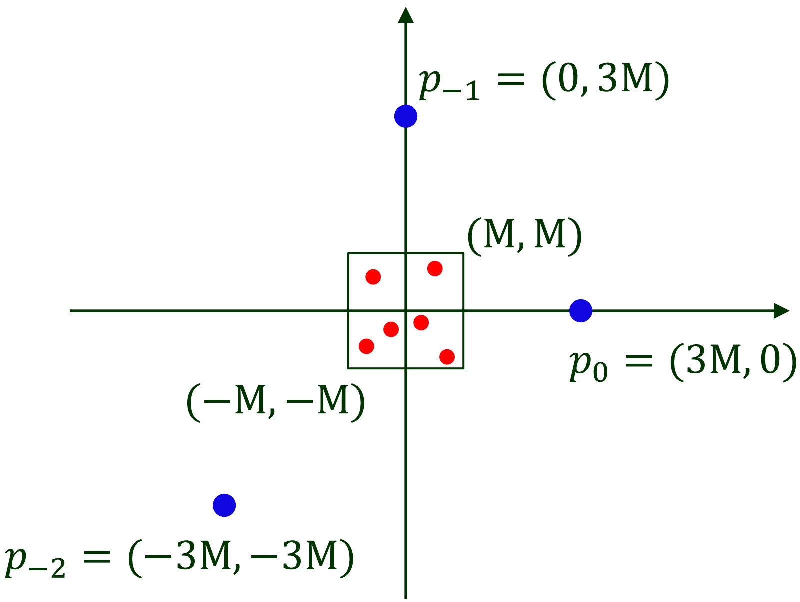


$$M = \max_{\substack{p_i=(x_i,y_i) \\ 1\le i\le n}} \{|x_i|, |y_i|\}$$

- p_0 lies outside circles defined by any three points in P.
- p_{-1} lies outside circles defined by any three points in $P \cup \{p_0\}$.

Selecting p_{-2}, p_{-1}, p_0

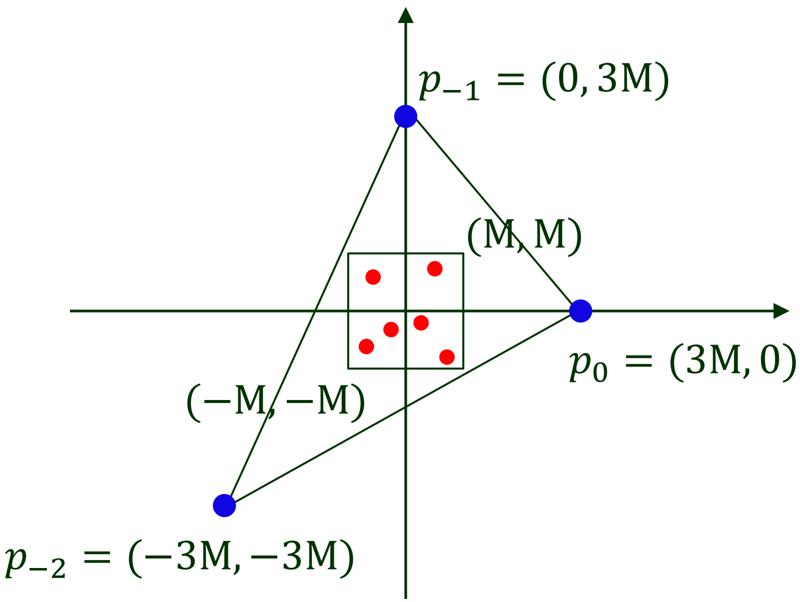
$$M = \max_{\substack{p_i=(x_i,y_i)\\ 1\le i\le n}} \{|x_i|, |y_i|\}$$



- p_0 lies outside circles defined by any three points in P.
- p_{-1} lies outside circles defined by any three points in $P \cup \{p_0\}$.
- p_{-2} lies outside circles defined by any three points in $P \cup \{p_{0,} p_{-1}\}$.

Selecting p_{-2}, p_{-1}, p_0

$$M = \max_{\substack{p_i=(x_i,y_i)\\ 1\le i\le n}} \{|x_i|, |y_i|\}$$



- p_0 lies outside circles defined by any three points in P.
- p_{-1} lies outside circles defined by any three points in $P \cup \{p_0\}$.
- p_{-2} lies outside circles defined by any three points in $P \cup \{p_0, p_{-1}\}$.

IV. Analysis

$P_r = \{p_1, p_2, \ldots, p_r\}$ $\quad\quad$ $DG_r = DG(\{p_{-2}, p_{-1}, p_0, p_1, \ldots, p_r\})$

Lemma 2 Expected number of triangles created (and deleted) by the algorithm is $\leq 9n + 1$.

IV. Analysis

$P_r = \{p_1, p_2, \dots, p_r\}$ $DG_r = DG(\{p_{-2}, p_{-1}, p_0, p_1, \dots, p_r\})$

Lemma 2 Expected number of triangles created (and deleted) by the algorithm is $\leq 9n + 1$.

Proof One triangle $\Delta\, p_0 p_{-1} p_{-2}$ at the start.

IV. Analysis

$P_r = \{p_1, p_2, \ldots, p_r\}$ $\qquad$ $DG_r = DG(\{p_{-2}, p_{-1}, p_0, p_1, \ldots, p_r\})$

Lemma 2 Expected number of triangles created (and deleted) by the algorithm is $\leq 9n + 1$.

Proof One triangle $\Delta\, p_0 p_{-1} p_{-2}$ at the start.

Iteration r inserts p_r :

IV. Analysis

$P_r = \{p_1, p_2, \ldots, p_r\}$ $DG_r = DG(\{p_{-2}, p_{-1}, p_0, p_1, \ldots, p_r\})$

Lemma 2 Expected number of triangles created (and deleted) by the algorithm is $\leq 9n + 1$.

Proof One triangle $\Delta\, p_0 p_{-1} p_{-2}$ at the start.

Iteration r inserts p_r :

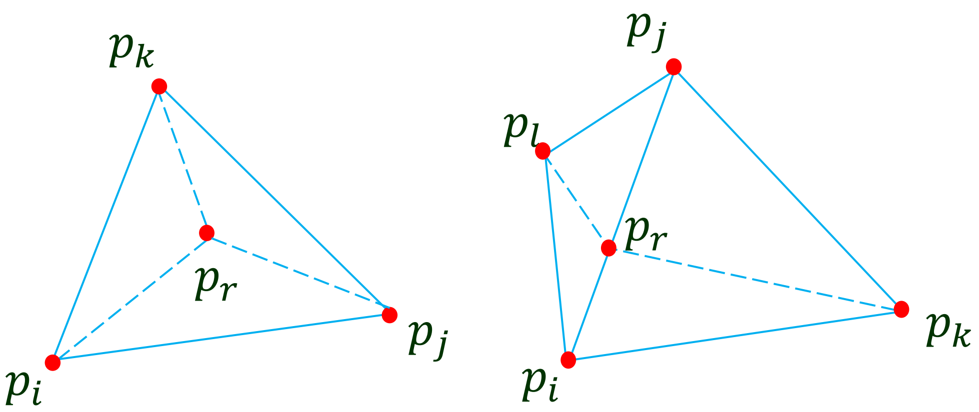
- Split 1 or 2 triangles, creating 3 or 4 new ones, and the same number of edges.

IV. Analysis

$P_r = \{p_1, p_2, \ldots, p_r\}$ $\qquad DG_r = DG(\{p_{-2}, p_{-1}, p_0, p_1, \ldots, p_r\})$

Lemma 2 Expected number of triangles created (and deleted) by the algorithm is $\leq 9n + 1$.

Proof One triangle $\Delta\, p_0 p_{-1} p_{-2}$ at the start.



Iteration r inserts p_r :

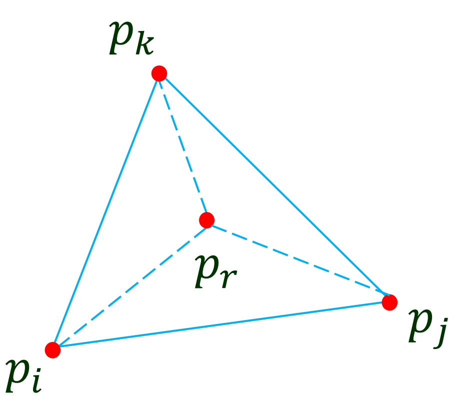
- Split 1 or 2 triangles, creating 3 or 4 new ones, and the same number of edges.

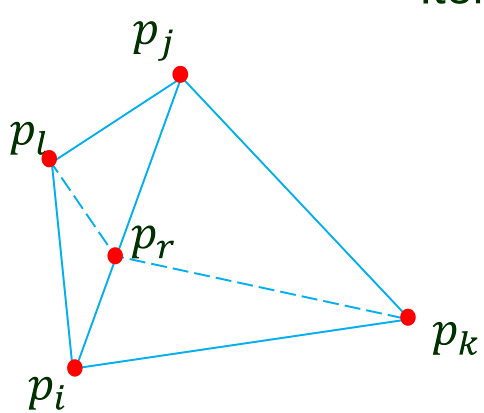
IV. Analysis

$P_r = \{p_1, p_2, \dots, p_r\}$ $\quad\quad$ $DG_r = DG(\{p_{-2}, p_{-1}, p_0, p_1, \dots, p_r\})$

Lemma 2 Expected number of triangles created (and deleted) by the algorithm is $\leq 9n + 1$.

Proof One triangle $\Delta\, p_0 p_{-1} p_{-2}$ at the start.





Iteration r inserts p_r :

- Split 1 or 2 triangles, creating 3 or 4 new ones, and the same number of edges.
- Every edge flipped in the subsequent `LegalizeEdge` results in creation of an edge adjacent to p_r and 2 new triangles bordering this edge.

Proof of Lemma 2 (cont'd)

Suppose k edges in DG_r are incident to p_r at the end of the iteration.

Proof of Lemma 2 (cont'd)

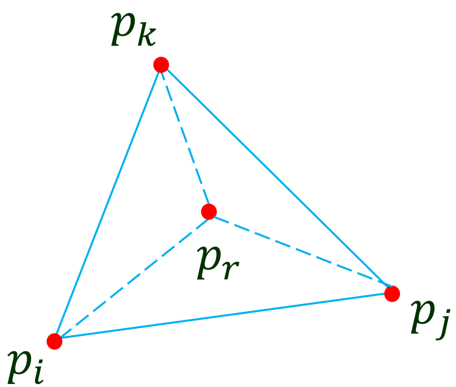
Suppose k edges in DG_r are incident to p_r at the end of the iteration.

Iteration r starts with (right after inserting p_r) one of two cases below:

Proof of Lemma 2 (cont'd)

Suppose k edges in DG_r are incident to p_r at the end of the iteration.

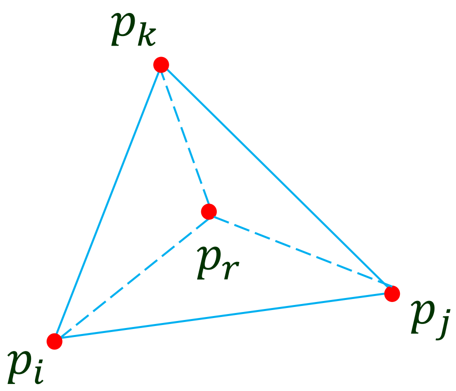
Iteration r starts with (right after inserting p_r) one of two cases below:



Proof of Lemma 2 (cont'd)

Suppose k edges in DG_r are incident to p_r at the end of the iteration.

Iteration r starts with (right after inserting p_r) one of two cases below:

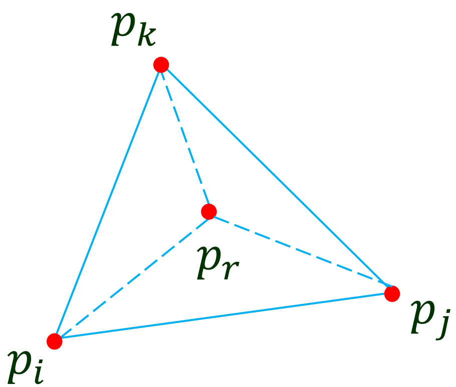


- 3 new edges

Proof of Lemma 2 (cont'd)

Suppose k edges in DG_r are incident to p_r at the end of the iteration.

Iteration r starts with (right after inserting p_r) one of two cases below:

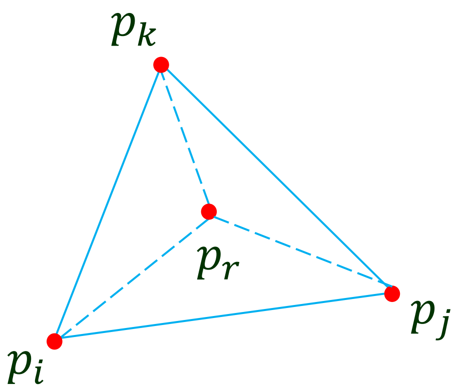


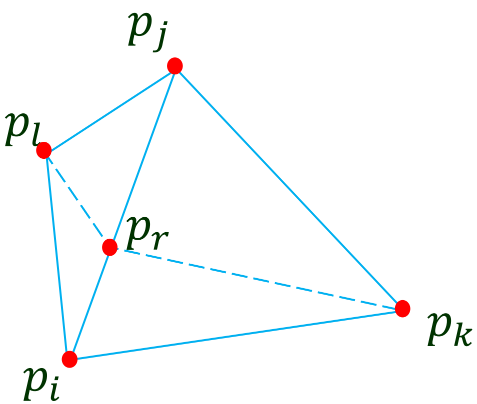
- 3 new edges
- 3 new triangles

Proof of Lemma 2 (cont'd)

Suppose k edges in DG_r are incident to p_r at the end of the iteration.

Iteration r starts with (right after inserting p_r) one of two cases below:



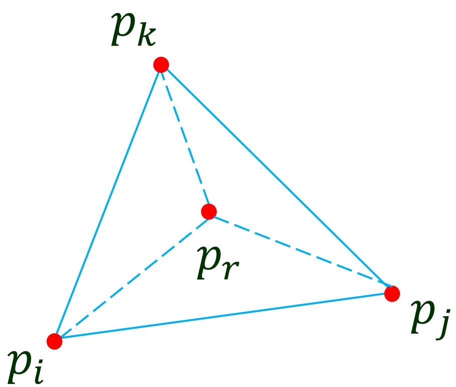


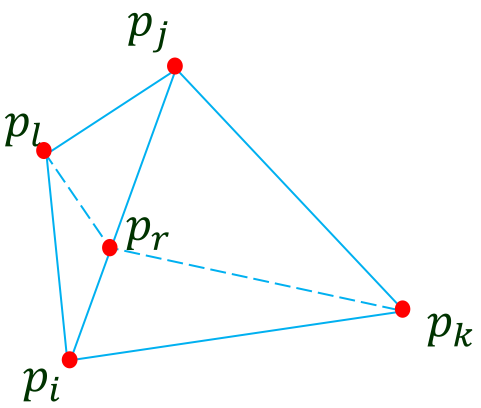
- 3 new edges
- 3 new triangles

Proof of Lemma 2 (cont'd)

Suppose k edges in DG_r are incident to p_r at the end of the iteration.

Iteration r starts with (right after inserting p_r) one of two cases below:





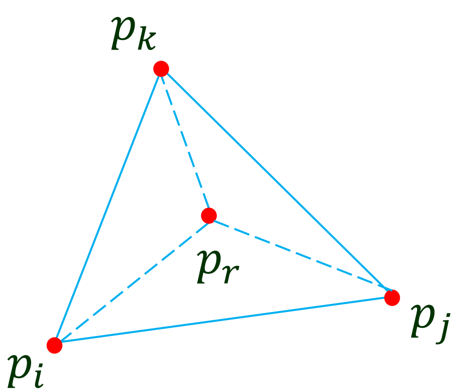
- 3 new edges
- 3 new triangles

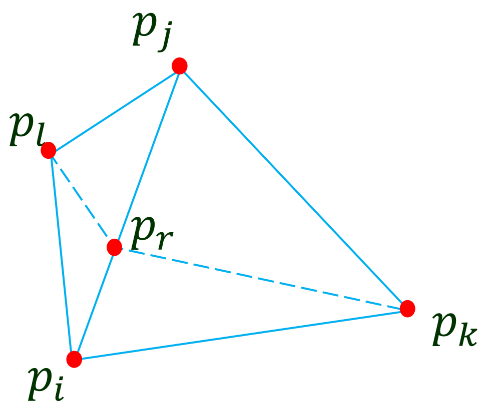
- 4 new edges

Proof of Lemma 2 (cont'd)

Suppose k edges in DG_r are incident to p_r at the end of the iteration.

Iteration r starts with (right after inserting p_r) one of two cases below:





- 3 new edges
- 3 new triangles

- 4 new edges
- 4 new triangles

Triangles Generated in One Iteration

Iteration r ends with

Triangles Generated in One Iteration

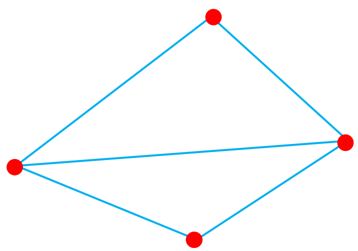
Iteration r ends with

- $k-3$ more new edges, each from a flip

Triangles Generated in One Iteration

Iteration r ends with

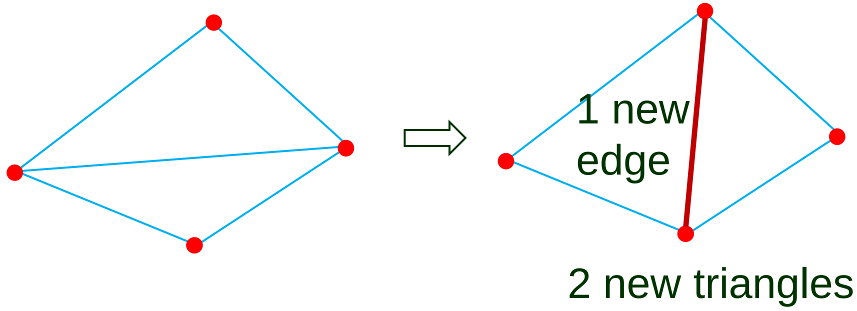
- $k - 3$ more new edges, each from a flip



Triangles Generated in One Iteration

Iteration r ends with

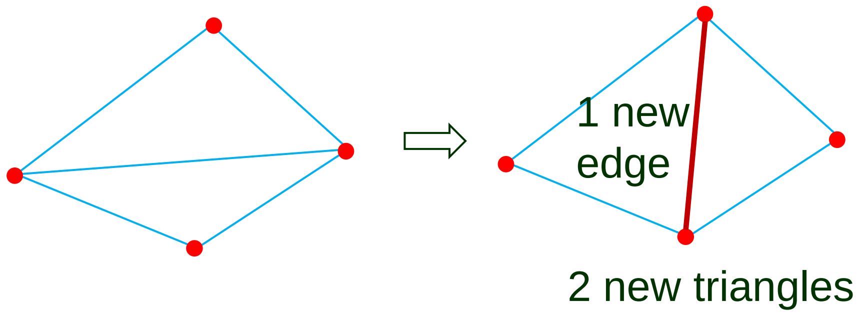
- $k - 3$ more new edges, each from a flip



Triangles Generated in One Iteration

Iteration r ends with

- $k - 3$ more new edges, each from a flip

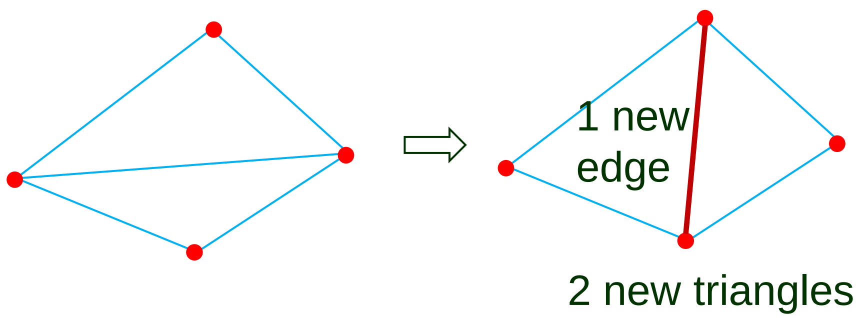


- $2(k - 3)$ more new triangles

Triangles Generated in One Iteration

Iteration r ends with

- $k-3$ more new edges, each from a flip



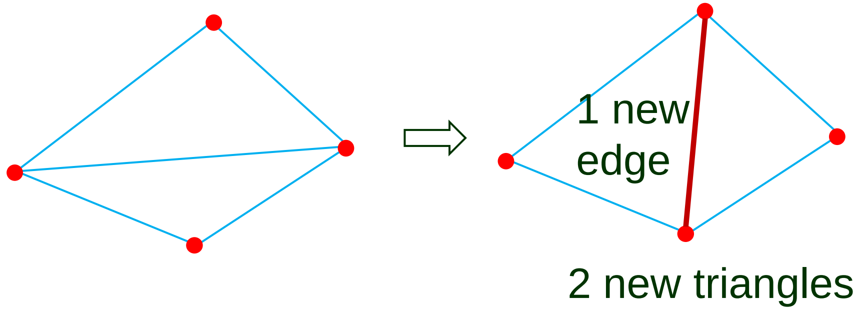
- $2(k-3)$ more new triangles

- $k-4$ more new edges due to flips
- $2(k-4)$ more new triangles

Triangles Generated in One Iteration

Iteration r ends with

- $k - 3$ more new edges, each from a flip



- $2(k-3)$ more new triangles

- $k - 4$ more new edges due to flips
- $2(k-4)$ more new triangles

#new triangles due to iteration r $\leq \max\{2(k-3)+3, 2(k-4)+4\}$

$$= 2k - 3$$

Backward Analysis

Let $\deg(p_r, DG_r) = k$ be the degree of p_r in DG_r.

Apply *backward analysis* to determine its expected value.

Backward Analysis

Let $\deg(p_r, DG_r) = k$ be the degree of p_r in DG_r.

Apply *backward analysis* to determine its expected value.

- Fix the set $P_r = \{p_1, p_2, \dots, p_r\}$ but view p_r as a *random* element from the set.

Backward Analysis

Let $\deg(p_r, DG_r) = k$ be the degree of p_r in DG_r.

Apply *backward analysis* to determine its expected value.

- Fix the set $P_r = \{p_1, p_2, \ldots, p_r\}$ but view p_r as a *random* element from the set.
- DG_r has the same number ($\leq 3(r+3) - 6$) of edges as the Voronoi diagram Vor$((\{p_{-2}, p_{-1}, p_0, p_1, \ldots, p_r\})$.

Backward Analysis

Let $\deg(p_r, DG_r) = k$ be the degree of p_r in DG_r.

Apply *backward analysis* to determine its expected value.

- Fix the set $P_r = \{p_1, p_2, \ldots, p_r\}$ but view p_r as a *random* element from the set.

- DG_r has the same number ($\leq 3(r+3) - 6$) of edges as the Voronoi diagram Vor(($\{p_{-2}, p_{-1}, p_0, p_1, \ldots, p_r\}$).

$\Downarrow$

Three of the edges are $\overline{p_{-1}p_{-2}}$, $\overline{p_{-1}p_0}$, $\overline{p_{-2}p_0}$.

Total degree of vertices from P_r is $\leq 2(3(r+3) - 9) = 6r$.

Backward Analysis

Let $\deg(p_r, DG_r) = k$ be the degree of p_r in DG_r.

Apply *backward analysis* to determine its expected value.

- Fix the set $P_r = \{p_1, p_2, \dots, p_r\}$ but view p_r as a *random* element from the set.

- DG_r has the same number ($\leq 3(r+3) - 6$) of edges as the Voronoi diagram Vor$((\{p_{-2}, p_{-1}, p_0, p_1, \dots, p_r\})$.

$\Downarrow$

Three of the edges are $\overline{p_{-1}p_{-2}}$, $\overline{p_{-1}p_0}$, $\overline{p_{-2}p_0}$.

Total degree of vertices from P_r is $\leq 2(3(r+3) - 9) = 6r$.

$\Downarrow$

Expected degree of such a vertex is ≤ 6.

Proof of Lemma 2 (finish)

Expected # triangles created in step r

Proof of Lemma 2 (finish)

Expected # triangles created in step r

$$\leq E(2\deg(p_r, DG_r) - 3)$$

Proof of Lemma 2 (finish)

Expected # triangles created in step r

$\leq E(2\deg(p_r, DG_r) - 3)$

i.e., $\leq 2k - 3$ shown two slides earlier

Proof of Lemma 2 (finish)

Expected # triangles created in step r

$\leq E(2\deg(p_r, DG_r) - 3)$ i.e., $\leq 2k - 3$ shown two slides earlier

$= 2E(\deg(p_r, DG_r)) - 3$

Proof of Lemma 2 (finish)

Expected # triangles created in step r

$\leq E(2\deg(p_r, DG_r) - 3)$ — i.e., $\leq 2k - 3$ shown two slides earlier

$= 2E(\deg(p_r, DG_r)) - 3$

$= 2 \cdot 6 - 3$ — Expected degree ≤ 6 from previous slide

Proof of Lemma 2 (finish)

Expected # triangles created in step r

$\leq E(2\deg(p_r, DG_r) - 3)$

i.e., $\leq 2k - 3$ shown two slides earlier

$= 2E(\deg(p_r, DG_r)) - 3$

$= 2 \cdot 6 - 3$

Expected degree ≤ 6 from previous slide

$= 9$

Proof of Lemma 2 (finish)

Expected # triangles created in step r

$\leq E(2\deg(p_r, DG_r) - 3)$

i.e., $\leq 2k - 3$ shown two slides earlier

$= 2E(\deg(p_r, DG_r)) - 3$

$= 2 \cdot 6 - 3$

Expected degree ≤ 6 from previous slide

$= 9$

$\Downarrow$ n insertion steps .

$\leq 9n$ triangles created

Proof of Lemma 2 (finish)

Expected # triangles created in step r

$\leq E(2\deg(p_r, DG_r) - 3)$

i.e., $\leq 2k - 3$ shown two slides earlier

$= 2E(\deg(p_r, DG_r)) - 3$

$= 2 \cdot 6 - 3$

Expected degree ≤ 6 from previous slide

$= 9$

$\Downarrow$ n insertion steps .

$\leq 9n$ triangles created

$\Downarrow$ Include $\Delta\, p_0 p_{-1} p_{-2}$

$\leq 9n + 1$ triangles.

Storage and Run Time

Theorem 3 $DG(P)$ can be computed in $O(n \log n)$ expected time using $O(n)$ expected storage.

Storage and Run Time

Theorem 3 $DG(P)$ can be computed in $O(n \log n)$ expected time using $O(n)$ expected storage.

Sketch of Proof

Storage and Run Time

Theorem 3 $DG(P)$ can be computed in $O(n \log n)$ expected time using $O(n)$ expected storage.

Sketch of Proof

(Storage) Every node of the search structure corresponds to a triangle.

By Lemma 2, expected number of triangles is $O(n)$.

Storage and Run Time

Theorem 3 $DG(P)$ can be computed in $O(n \log n)$ expected time using $O(n)$ expected storage.

Sketch of Proof

(Storage) Every node of the search structure corresponds to a triangle.

By Lemma 2, expected number of triangles is $O(n)$.

(Time) Time cost is attributed to two types of operations:

- all point location steps
- remaining portion $\backsim$ # created triangles

Storage and Run Time

Theorem 3 $DG(P)$ can be computed in $O(n \log n)$ expected time using $O(n)$ expected storage.

Sketch of Proof

(Storage) Every node of the search structure corresponds to a triangle.

By Lemma 2, expected number of triangles is $O(n)$.

(Time) Time cost is attributed to two types of operations:

- all point location steps
- remaining portion $\backsim$ # created triangles $O(n)$

Expected Time to Locate a Point

Time to locate $p_r \sim$ # nodes visited in the search structure

Expected Time to Locate a Point

Time to locate $p_r \sim$ # nodes visited in the search structure

$\sim$ # triangles that were present at some earlier stage and containing p_r but have been destroyed

Expected Time to Locate a Point

Time to locate $p_r \sim$ # nodes visited in the search structure

$\sim$ # triangles that were present at some earlier stage and containing p_r but have been destroyed

One triangle may be charged multiple times, each time for locating a different point.

Expected Time to Locate a Point

Time to locate $p_r \sim$ # nodes visited in the search structure

$\sim$ # triangles that were present at some earlier stage and containing p_r but have been destroyed

One triangle may be charged multiple times, each time for locating a different point.

S: set of all triangles created by the algorithm.

n_Δ: number of points from P that lie within the triangle Δ

Expected Time to Locate a Point

Time to locate $p_r \sim$ # nodes visited in the search structure

$\sim$ # triangles that were present at some earlier stage and containing p_r but have been destroyed

One triangle may be charged multiple times, each time for locating a different point.

S: set of all triangles created by the algorithm.

n_Δ: number of points from P that lie within the triangle Δ

Total time for all point location steps is

Expected Time to Locate a Point

Time to locate $p_r \sim$ # nodes visited in the search structure

$\sim$ # triangles that were present at some earlier stage and containing p_r but have been destroyed

One triangle may be charged multiple times, each time for locating a different point.

S: set of all triangles created by the algorithm.

n_Δ: number of points from P that lie within the triangle Δ

Total time for all point location steps is

$$O(n + \sum_{\Delta \in S} n_\Delta) = O(n \log n)$$

Expected Time to Locate a Point

Time to locate $p_r \sim$ # nodes visited in the search structure

$\sim$ # triangles that were present at some earlier stage and containing p_r but have been destroyed

One triangle may be charged multiple times, each time for locating a different point.

S: set of all triangles created by the algorithm.

n_Δ: number of points from P that lie within the triangle Δ

Total time for all point location steps is

$$O\left(n + \boxed{\sum_{\Delta \in S} n_\Delta}\right) = O(n \log n)$$ (for proof see Lemma 9.13)