Computing the Delaunay Triangulation

Outline:

I. Edge legalization

II. Correctness

III. Use of a trapezoidal map

IV. Analyses 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 $\text{Vor}(P)$. 

$p_1$, $p_2$, $p_3$, $p_4$, $p_5$, $p_6$, $p_7$, $p_8$
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 $\text{Vor}(P)$. 
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

Algorithm 1

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

2) Obtain $\text{DG}(P)$.

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 $\text{Vor}(P)$.

2) Obtain $\text{DG}(P)$.

3) Triangulate faces with $> 3$ vertices.
I. The Construction Problem

Algorithm 1

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

2) Obtain $\text{DG}(P)$.

3) Triangulate faces with > 3 vertices.

Input: a set $P$ of $n$ points.
Randomized Incremental Construction

Algorithm 2

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

Algorithm 2

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

1) Introduce a set $\Omega = \{p_0, p_{-1}, p_{-2}\}$ of three auxiliary points such that $\Delta p_0p_{-1}p_{-2}$ contains all points from $P$ in the interior. Start with $\Delta p_0p_{-1}p_{-2}$. 

![Diagram showing points $p_0$, $p_{-1}$, $p_{-2}$, and other points $p_1$ to $p_8$ within the triangle $p_0p_{-1}p_{-2}$]
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 $\Delta p_ip_jp_k$ that contains the newly added $p_r$. 
2) Add points in random order, maintaining a Delaunay triangulation of the current set.

♦ In step $r$, find the triangle $\Delta p_ip_jp_k$ that contains the newly added $p_r$.

Case 1: $p_r$ in the interior of $\Delta p_ip_jp_k$
2) Add points in random order, maintaining a Delaunay triangulation of the current set.

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

Case 1: $p_r$ in the interior of $\Delta p_ip_jp_k$

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($p_r, \overline{p_ip_j}, T$)

1. if $\overline{p_ip_j}$ is illegal
2. then let $\Delta p_ip_jp_k$ and $\Delta p_ip_jp_r$ be adjacent along $\overline{p_ip_j}$
3. replace $\overline{p_ip_j}$ with $\overline{p_rp_k}$
4. LegalizeEdge($p_r, \overline{p_ip_k}, T$)
5. LegalizeEdge($p_r, \overline{p_jp_k}, T$)
Edge Legalization

LegalizeEdge($p_r, \overline{p_ip_j}, T$)

1. if $\overline{p_ip_j}$ is illegal
2. then let $\Delta \overline{p_ip_jp_k}$ and $\Delta \overline{p_ip_jp_r}$ be adjacent along $\overline{p_ip_j}$
3. replace $\overline{p_ip_j}$ with $\overline{p_ip_k}$
4. LegalizeEdge($p_r, \overline{p_ip_k}, T$)
5. LegalizeEdge($p_r, \overline{p_ip_k}, T$)
LegalizeEdge($p_r, \overline{p_ip_j}, T$)

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

- All recursive calls involve edges opposing $p_r$. 
Edge Legalization

LegalizeEdge($p_r, \overline{p_ip_j}, T$)

1. if $\overline{p_ip_j}$ is illegal
2. then let $\Delta p_ip_jp_k$ and $\Delta p_ip_jp_r$ be adjacent along $\overline{p_ip_j}$
3. replace $\overline{p_ip_j}$ with $\overline{p_rp_k}$
4. LegalizeEdge($p_r, \overline{p_ip_k}, T$)
5. LegalizeEdge($p_r, \overline{p_jp_k}, T$)

- All recursive calls involve edges opposing $p_r$.
- All replacing edges during the edge flips are incident on $p_r$. 
Edge Legalization

LegalizeEdge($p_r, \overline{pipj}, T$)

1. **if** $\overline{pipj}$ is illegal
2. **then** let $\Delta pipjk$ and $\Delta pipjr$ be adjacent along $\overline{pipj}$
3. **replace** $\overline{pipj}$ with $\overline{prpk}$
4. **LegalizeEdge**($p_r, \overline{pipk}, T$)
5. **LegalizeEdge**($p_r, \overline{pjpk}, T$)

- All recursive calls involve edges opposing $p_r$.

- 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\((p_r, \overline{p_ip_j}, T)\)

1. if \(\overline{p_ip_j}\) is illegal
2. then let \(\Delta p_ip_jp_k\) and \(\Delta p_ip_jp_r\) be adjacent along \(\overline{p_ip_j}\)
3. replace \(\overline{p_ip_j}\) with \(\overline{p_ip_k}\)
4. LegalizeEdge\((p_r, \overline{p_ip_k}, T)\)
5. LegalizeEdge\((p_r, \overline{p_jp_k}, T)\)

- All recursive calls involve edges opposing \(p_r\).
- 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.
- Only the edges of the new triangles need to be checked.
More Observations

LegalizeEdge\((p_r, \overline{p_ip_j}, T)\)

1. if \(p_ip_j\) is illegal
2. then let \(\Delta p_ip_jp_k\) and \(\Delta p_ip_jp_r\) be adjacent along \(\overline{p_ip_j}\)
3. replace \(\overline{p_ip_j}\) with \(\overline{p_rp_k}\)
4. LegalizeEdge\((p_r, \overline{p_ip_k}, T)\)
5. LegalizeEdge\((p_r, \overline{p_jp_k}, T)\)

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

LegalizeEdge\((p_r, \overline{pi}p_j, T)\)

1. if \(\overline{pi}p_j\) is illegal
2. then let \(\Delta p_ip_jp_k\) and \(\Delta p_ip_jp_r\) be adjacent along \(\overline{pi}p_j\)
3. replace \(\overline{pi}p_j\) with \(\overline{pr}p_k\)
4. LegalizeEdge\((p_r, \overline{pi}p_k, T)\)
5. LegalizeEdge\((p_r, \overline{pj}p_k, T)\)

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

LegalizeEdge\((p_r, \overline{pi}p_j, T)\)

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

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

1. if $\overline{p_ip_j}$ is illegal
2. then let $\Delta p_ip_jp_k$ and $\Delta p_ip_jp_r$ be adjacent along $\overline{p_ip_j}$
3. replace $\overline{p_ip_j}$ with $\overline{p_rp_k}$
4. LegalizeEdge($p_r, \overline{p_ip_k}, T$)
5. LegalizeEdge($p_r, \overline{p_jp_k}, T$)

- Only the edges of the new triangles need to be checked.
- $\overline{p_rp_i}$ and $\overline{p_rp_j}$ are newly inserted and legal (to be shown).
- So check $\overline{p_ip_k}$ and $\overline{p_jp_k}$ opposing $p_r$, and recursively from there.
1. Compute a random permutation $p_1, p_2, ..., p_n$
2. for $r \leftarrow 1$ to $n$
3. \hspace{1em} do
4. \hspace{2em} find $\Delta p_ip_jp_k \supset p_r$ in the current triangulation $T$
5. \hspace{2em} if $p_r$ lies in its interior
6. \hspace{3em} then // case 1
7. \hspace{4em} add edges $p_mp_i$, $p_mp_j$, $p_mp_k$
8. \hspace{2em} LegalizeEdge($p_r, p_ip_j, T$)
9. \hspace{2em} LegalizeEdge($p_r, p_jp_k, T$)
10. \hspace{2em} LegalizeEdge($p_r, p_kp_i, T$)
1. Compute a random permutation $p_1, p_2, ..., p_n$
2. for $r \leftarrow 1$ to $n$
3. do
4. find $\Delta p_ip_jp_k \supset p_r$ in the current triangulation $T$
5. if $p_r$ lies in its interior
6. then  // case 1
7. add edges $\overline{prp_i}$, $\overline{prp_j}$, $\overline{prp_k}$
8. LegalizeEdge($p_r, \overline{ip_j}, T$)
9. LegalizeEdge($p_r, \overline{jp_k}, T$)
10. LegalizeEdge($p_r, \overline{kp_i}, T$
11. else  // case 2
12. add edges $\overline{prp_k}$, $\overline{prp_l}$
13. LegalizeEdge($p_r, \overline{ip_k}, T$)
14. LegalizeEdge($p_r, \overline{kp_j}, T$)
15. LegalizeEdge($p_r, \overline{jp_l}, T$)
16. LegalizeEdge($p_r, \overline{lp_i}, T$)
II. Correctness

Need to prove that no illegal edges remain after all calls to \texttt{LegalizeEdge}. Correctness is implied by the following:
II. Correctness

Need to prove that no illegal edges remain after all calls to \texttt{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 \texttt{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 \texttt{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 \texttt{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 flip increases the angle vector of the triangulation.
Lemma 1 Every new edge created during the insertion of \( p_r \) is an edge of \( DG(\{p_0, p_{-1}, p_{-2}, p_1, \ldots, p_r\}) \).
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.
Lemma 1  Every new edge created during the insertion of $p_r$ is an edge of $\mathcal{DG} \{p_0, p_{-1}, p_{-2}, p_1, \ldots, p_r \}$.

Proof  Examine two types of edges.

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

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

Proof  Examine two types of edges.

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

Case 1

[Diagram showing Case 1]

Case 2

[Diagram showing Case 2]
Immediately Added Edges

- Case 1

$\Delta p_ip_jp_k$ 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

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

The circumcircle $C$ of $p_i, p_j, p_k$ 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 \( DG(\Omega \cup \{p_1, ..., p_{r-1}\}) \).

The circumcircle \( C \) of \( p_i, p_j, p_k \) contains no point \( p_l, l < r \), in its interior.

Shrink \( C \) (centered at \( o \)) to a circle \( C' \) centered at \( o' \) on \( \overline{op_i} \) and passing through \( p_i \) and \( p_r \).
Immediately Added Edges

- Case 1

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

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

Shrink $C$ (centered at $o$) to a circle $C'$ centered at $o'$ on $\overline{op_i}$ and passing through $p_i$ and $p_r$. 
Immediately Added Edges

- Case 1

\[ \Delta p_ip_jp_k \text{ is a triangle in } DG(\Omega \cup \{p_1, ..., p_{r-1}\}) \].

\[ \Downarrow \]

The circumcircle \( C \) of \( p_i, p_j, p_k \) contains no point \( p_l, l < r \), in its interior.

Shrink \( C \) (centered at \( o \)) to a circle \( C' \) centered at \( o' \) on \( \overline{op_i} \) and passing through \( p_i \) and \( p_r \).

\( C' \) is empty.
Immediately Added Edges

- **Case 1**

\[ \Delta p_ip_jp_k \] is a triangle in \( DG(\Omega \cup \{p_1, ..., p_{r-1}\}) \).

\[
\Downarrow
\]

The circumcircle \( C \) of \( p_i, p_j, p_k \) contains no point \( p_l, l < r \), in its interior.

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

\( C' \) is empty. \( \Rightarrow \) \( p_rp_i \) an edge of the DG after addition of \( p_r \).
Immediately Added Edges

- Case 1

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

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

Shrink $C$ (centered at $o$) to a circle $C'$ centered at $o'$ on $op_i$ and passing 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 \( C \) of \( p_i, p_j, p_k \) contains no point \( p_l, l < r \), in its interior.

Shrink \( C \) (centered at \( o \)) to a circle \( C' \) centered at \( o' \) on \( \overline{op_i} \) and passing through \( p_i \) and \( p_r \).

\( C' \) is empty. \( \iff \) \( p_r p_i \) an edge of the DG after addition of \( p_r \).

Similarly, \( p_r p_j \) and \( p_r p_k \) are edges too.

• Case 2

Similar to Case 1.
Edges Added Due to Flipping

- The 2\textsuperscript{nd} type of edges are added due to flipping by \texttt{LegalizeEdge}.

Suppose $\overline{p_ip_j}$ of $\Delta p_ip_jp_l$ is replaced by $\overline{p_rp_l}$. 

![Diagram showing the points $p_i$, $p_r$, $o$, $p_j$, and $p_l$ connected to form a triangle and additional edges added due to flipping.](image)
The 2\textsuperscript{nd} type of edges are added due to flipping by LegalizeEdge.

Suppose $\overline{p_i p_j}$ of $\triangle p_i p_j p_l$ is replaced by $\overline{p_r p_l}$. 
Edges Added Due to Flipping

- The 2\textsuperscript{nd} type of edges are added due to flipping by LegalizeEdge.

Suppose $\overline{p_ip_j}$ of $\Delta p_ip_jp_l$ is replaced by $\overline{p_rp_l}$.

$\Delta p_ip_jp_l$ was a Delaunay triangle and its circumcircle $C$ (centered at $o$) contains $p_r$ in its interior only.
Edges Added Due to Flipping

- The 2\textsuperscript{nd} type of edges are added due to flipping by LegalizeEdge.

Suppose $\overline{p_ip_j}$ of $\Delta p_ip_jpl$ is replaced by $\overline{prpl}$.

$\Delta p_ip_jpl$ was a Delaunay triangle and its circumcircle $C$ (centered at $o$) contains $p_r$ in its interior only.

Shrink $C$ to a circle $C'$ with only $p_r$ and $pl$ on its boundary.
The 2\textsuperscript{nd} type of edges are added due to flipping by \texttt{LegalizeEdge}.

Suppose $p_ip_j$ of $\Delta p_ip_jp_l$ is replaced by $p_rp_l$.

$\Delta p_ip_jp_l$ was a Delaunay triangle and its circumcircle $C$ (centered at $o$) contains $p_r$ in its interior only.

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

- $o$ is closer to $p_r$ than to $p_l$.
- Move on $op_l$ from $o$ toward $p_l$ until reaching a location $o'$ that is equidistant to both $p_r$ and $p_l$. 
The 2\textsuperscript{nd} type of edges are added due to flipping by \texttt{LegalizeEdge}.

Suppose $\overline{p_ip_j}$ of $\Delta p_ip_jp_l$ is replaced by $\overline{p_rp_l}$.

$\Delta p_ip_jp_l$ was a Delaunay triangle and its circumcircle $C$ (centered at $o$) contains $p_r$ in its interior only.

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

- $o$ is closer to $p_r$ than to $p_l$.
- Move on $\overline{op_l}$ from $o$ toward $p_l$ until reaching a location $o'$ that is equidistant to both $p_r$ and $p_l$. 
The 2\textsuperscript{nd} type of edges are added due to flipping by LegalizeEdge.

Suppose $p_ip_j$ of $\triangle p_ip_jpl$ is replaced by $prpl$.

$\triangle p_ip_jpl$ was a Delaunay triangle and its circumcircle $C$ (centered at $o$) contains $pr$ in its interior only.

Shrink $C$ to a circle $C'$ with only $pr$ and $pl$ on its boundary.

- $o$ is closer to $pr$ than to $pl$.
- Move on $opl$ from $o$ toward $pl$ until reaching a location $o'$ that is equidistant to both $pr$ and $pl$.

$C'$ (centered at $o'$ and through $pr$ and $pl$) is empty.
The 2\textsuperscript{nd} type of edges are added due to flipping by \texttt{LegalizeEdge}.

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

$\Delta p_ip_jp_l$ was a Delaunay triangle and its circumcircle $C$ (centered at $o$) contains $p_r$ in its interior only.

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

- $o$ is closer to $p_r$ than to $p_l$.
- Move on $\overline{op_l}$ from $o$ toward $p_l$ until reaching a location $o'$ that is equidistant to both $p_r$ and $p_l$.

$C'$ (centered at $o'$ and through $p_r$ and $p_l$) is empty.

$\overline{p_r p_l}$ is a Delaunay edge after the addition.
Edges Added Due to Flipping

- The 2nd type of edges are added due to flipping by \texttt{LegalizeEdge}.

Suppose \( \overline{p_ip_j} \) of \( \Delta p_ip_jp_l \) is replaced by \( \overline{p_rp_l} \).

\( \Delta p_ip_jp_l \) was a Delaunay triangle and its circumcircle \( C \) (centered at \( o \)) contains \( p_r \) in its interior only.

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

- \( o \) is closer to \( p_r \) than to \( p_l \).

- Move on \( \overline{op_l} \) from \( o \) toward \( p_l \) until reaching a location \( o' \) that is equidistant to both \( p_r \) and \( p_l \).

\( C' \) (centered at \( o' \) and through \( p_r \) and \( p_l \)) is empty.

\( \overline{p_rp_l} \) 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** ($\Delta p_0p_{-1}p_{-2}$).
Example
Example

insert $p_r$ into $\Delta_1$
Example

\[ \Delta_1 \]
\[ \Delta_2 \]
\[ \Delta_3 \]

insert \( p_r \) into \( \Delta_1 \)
split \( \Delta_1 \)

\[ p_i \]
\[ \Delta_2 \]
\[ \Delta_3 \]

\[ p_j \]
Example

\[ \Delta_1 \]
\[ \Delta_2 \]
\[ \Delta_3 \]

insert \( p_r \) into \( \Delta_1 \)
split \( \Delta_1 \)

\[ p_i \]
\[ p_j \]
Example

\[ \Delta_1 \]
\[ \Delta_2 \]
\[ \Delta_3 \]

\[ p_i \]
\[ p_j \]
\[ p_r \]

- Insert \( p_r \) into \( \Delta_1 \)
- Split \( \Delta_1 \)
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 $D$. 

![Diagram showing a tree with nodes labeled $p_i$, $p_r$, $p_j$, $\Delta_1$, $\Delta_2$, $\Delta_3$.](image-url)
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 $p_r$. 

Insertion

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

- Start at the root ($\Delta p_0p_{-1}p_{-2}$) 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 $\Delta p_0p_{-1}p_{-2}$)
Insertion

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

- Start at the root ($\Delta p_0p_{-1}p_{-2}$) 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 $\Delta p_0p_{-1}p_{-2}$)
- 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 \( (\Delta p_0p_{-1}p_{-2}) \) 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 \( \Delta p_0p_{-1}p_{-2} \)).
- Descends to this child.
- Repeat the above two steps to reach a leaf.
Insertion

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

- 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.

Time linear in

$$\text{#nodes on the search path} = \text{#triangles stored in } D \text{ that contains } p_r$$
Example (cont’d)
Example (cont’d)
Example (cont’d)

\[
\Delta^2 \Delta^3 \Delta^4 \Delta^6
\]

\[
\Delta^5 \Delta^3 \Delta^4 \Delta^6
\]

\[
\text{flip } p_i p_j
\]
Example (finish)
Example (finish)
Example (finish)

\[ \Delta_1 \rightarrow \Delta_4 \rightarrow \Delta_5 \rightarrow \Delta_6 \]

flip \( \overline{p_ip_l} \)

\[ \Delta_1 \rightarrow \Delta_4 \rightarrow \Delta_5 \rightarrow \Delta_6 \]

\[ \Delta_7 \rightarrow \Delta_6 \rightarrow \Delta_8 \]
Selecting $p_{-2}, p_{-1}, p_0$

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

Point $M = \max_{i=1}^{n} (x_i, y_i) = (x, y)$
Selecting \( p_{-2}, p_{-1}, p_0 \)

\[
M = \max_{p_i=(x_i,y_i)} \{ |x_i|, |y_i| \}  \\
1 \leq i \leq n
\]

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

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

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

$p_0 = (3M, 0)$

$p_{-1} = (0, 3M)$

$(M, M)$

$(-M, -M)$

$p_0 = (3M, 0)$
Selecting $p_{-2}, p_{-1}, p_0$

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

- $p_0$ 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 $P \cup \{p_0, p_{-1}\}$. 

$p_{-1} = (0, 3M)$

$p_{-2} = (-3M, -3M)$

$p_0 = (3M, 0)$
Selecting $p_{-2}, p_{-1}, p_0$

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

- $p_0$ 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

\[ P_r = \{p_1, p_2, ..., p_r\} \quad DG_r = DG(\{p_{-2}, p_{-1}, p_0, p_1, ..., p_r\}) \]

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

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

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

**Proof** One triangle \( \Delta p_0p_{-1}p_{-2} \) at the start.
IV. Analysis

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

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

**Proof** One triangle \( \Delta p_0p_{-1}p_{-2} \) at the start.

Iteration \( r \) inserts \( p_r \):
IV. Analysis

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

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

Proof  One triangle \( \Delta p_0p_{-1}p_{-2} \) 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

\[ P_r = \{p_1, p_2, \ldots, p_r\} \quad \text{D}_G_r = D_G(\{p_{-2}, p_{-1}, p_0, p_1, \ldots, p_r\}) \]

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

**Proof** One triangle \( \Delta p_0p_{-1}p_{-2} \) 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 $D_G$ 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
Iteration $r$ ends with

- $k - 3$ more new edges, each from a flip
Iteration $r$ ends with

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

2 new triangles
Triangles Generated in One Iteration

Iteration \( r \) ends with

- \( k - 3 \) more new edges, each from a flip
- 1 new edge
- 2 new triangles
- \( 2(k - 3) \) more new triangles
Triangles Generated in One Iteration

Iteration $r$ ends with

- $k - 3$ more new edges, each from a flip
- $k - 4$ more new edges due to flips
- $2(k - 4)$ more new triangles
- $2(k - 3)$ more new triangles
Triangles Generated in One Iteration

Iteration $r$ ends with

- $k - 3$ more new edges, each from a flip
- $k - 4$ more new edges due to flips
- $2(k - 3)$ more new triangles
- $2(k - 4)$ more new triangles

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

$= 2k - 3$
Backward Analysis

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

Apply \textit{backward analysis} to determine its expected value.
Let \( \deg(p_r, DG_r) = k \) be the degree of \( p_r \) in \( DG_r \).

Apply \textit{backward analysis} to determine its expected value.

- Fix the set \( P_r = \{p_1, p_2, ..., p_r\} \) but view \( p_r \) as a \textit{random} element from the set.
Backward Analysis

Let \( \text{deg}(p_r, DG_r) = k \) 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 \( \text{Vor}((\{p_{-2}, p_{-1}, p_0, p_1, \ldots, p_r\})) \).
Let $\text{deg}(p_r, DG_r) = k$ be the degree of $p_r$ in $DG_r$.

Apply \textit{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 \textit{random} element from the set.

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

  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 $\leq 2(3(r + 3) - 9) = 6r$. 
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 $\text{Vor}((\{p_{-2}, p_{-1}, p_0, p_1, \ldots, p_r\}))$.

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

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

Expected degree of such a vertex is $\leq 6$. 


Proof of Lemma 2 (finish)

Expected # triangles created in step $r$
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 $r$

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

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 $r$

\[ \leq E(2 \deg(p_r, DG_r) - 3) \quad \text{i.e., } \leq 2k - 3 \text{ shown two slides earlier} \]

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

\[ = 2 \cdot 6 - 3 \quad \text{Expected degree } \leq 6 \text{ from previous slide} \]
Proof of Lemma 2 (finish)

Expected # triangles created in step \( r \)

\[ \leq E(2 \deg(p_r, DG_r) - 3) \quad \text{i.e.,} \quad \leq 2k - 3 \text{ shown two slides earlier} \]

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

\[ = 2 \cdot 6 - 3 \quad \text{Expected degree } \leq 6 \text{ from previous slide} \]

\[ = 9 \]
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

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

\[ = 2 \cdot 6 - 3 \]

\[ = 9 \]

Expected degree $\leq 6$
from previous slide

\[ \leq 9n \quad \text{triangles created} \]
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

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

\[
= 2 \cdot 6 - 3
\]

\[
= 9
\]

$n$ insertion steps

\[
\leq 9n\ \text{triangles created}
\]

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

\[
\leq 9n + 1\ \text{triangles.}
\]
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

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

\[ = 2 \cdot 6 - 3 \]

\[ = 9 \]

$n$ insertion steps

\[ \leq 9n \] triangles created

Include $\Delta p_0p_{-1}p_{-2}$

\[ \leq 9n + 1 \] triangles.
Theorem 3 \( DG(P) \) can be computed in \( O(n \log n) \) expected time using \( O(n) \) expected storage.
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 \( \sim \) # 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 \( \sim \) # created triangles \( O(n) \)
Expected Time to Locate a Point

Time to locate $p_r \sim \# \text{ nodes visited in the search structure}$
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.

$n_\Delta$: number of points from $P$ that lie within the triangle $\Delta$
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.

$n_{\Delta}$: number of points from $P$ that lie within the triangle $\Delta$

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.

$n_\Delta$: number of points from $P$ that lie within the triangle $\Delta$

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 $p_r \sim \# \text{ nodes visited in the search structure}$

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

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

$S$: set of all triangles created by the algorithm.

$n_\Delta$: number of points from $P$ that lie within the triangle $\Delta$

Total time for all point location steps is

$$O(n + \sum_{\Delta \in S} n_\Delta) = O(n \log n)$$ (for proof see Lemma 9.13)
Expected Time to Locate a Point

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

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

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

$S$: set of all triangles created by the algorithm.

$n_\Delta$: number of points from $P$ that lie within the triangle $\Delta$

Total time for all point location steps is

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