CS633 Lecture 10
Delaunay Triangulations

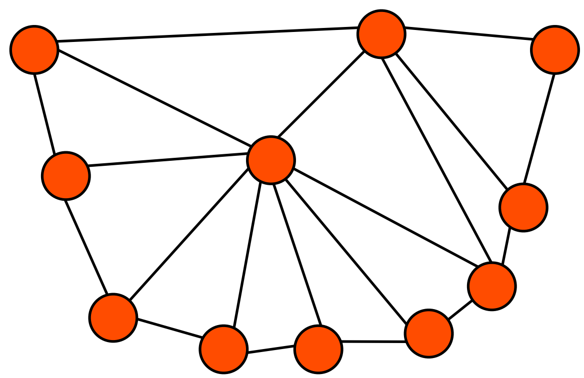
Jyh-Ming Lien

Department of Computer Science
George Mason University

Based on Glenn Eguchi's slides

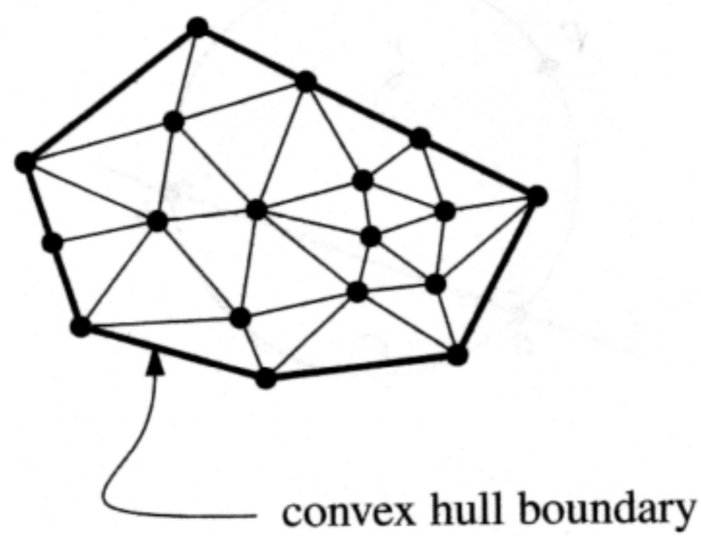
Triangulation

- *Triangulation*: Given a set of points P, triangulation of P is a planar subdivision whose bounded faces are triangles with vertices from P



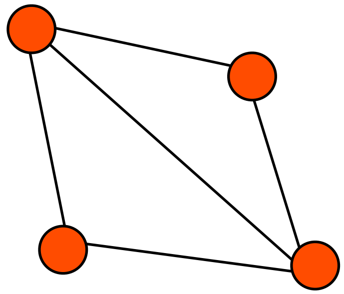
Triangulation is made of triangles

- Internal faces must be triangles, otherwise they could be triangulated further
- Outer polygon must be convex hull



Triangulation: Properties

- *triangulation* of set of points P: a maximal planar subdivision whose vertices are elements of P

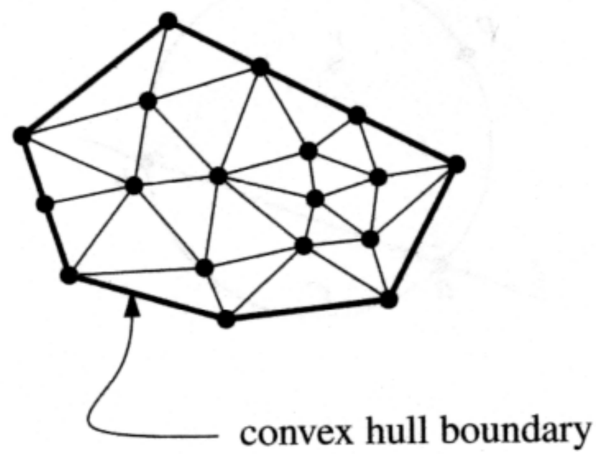


- *maximal planar subdivision*: a subdivision *S* such that no edge connecting two vertices can be added to *S* without destroying its planarity

Triangulation: Properties

For P consisting of n points, all triangulations contain 2n-2-k triangles, 3n-3 edges

- n = number of points in P
- k = number of points on convex hull of P



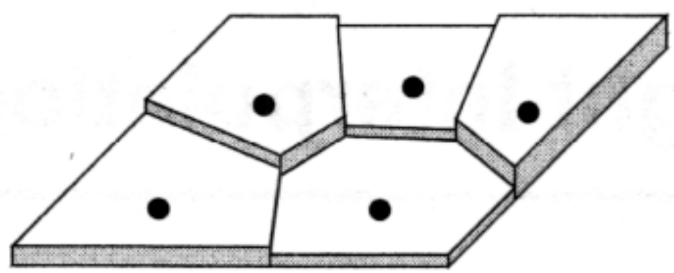
Application: Terrains

- Set of data points $P \subset R^2$
- Height $f(p)$ defined at each point p in P
- How can we most naturally approximate height of points not in P?

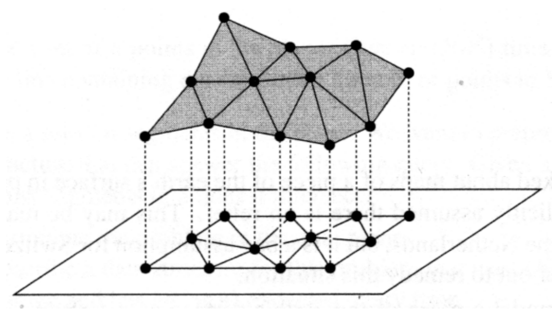
Option: Discretize

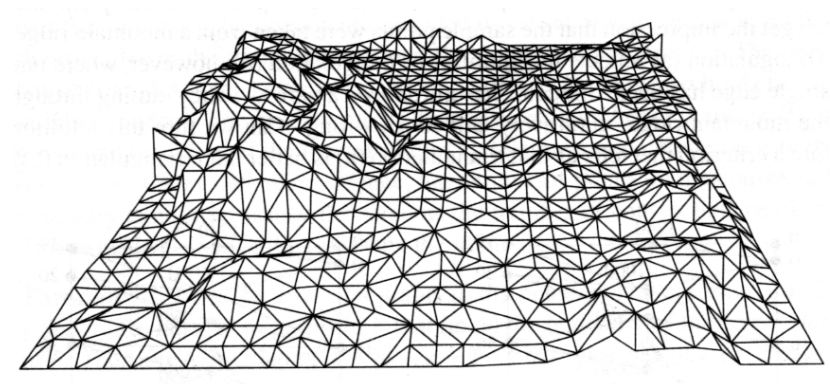
- Let f(p) = height of nearest point for points not in P

- Does not look natural

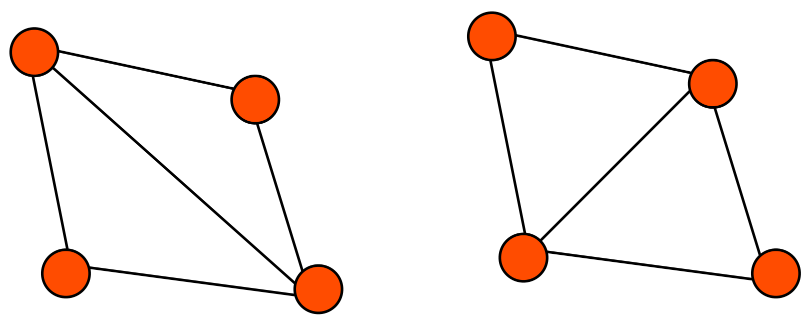


Better Option: Triangulation



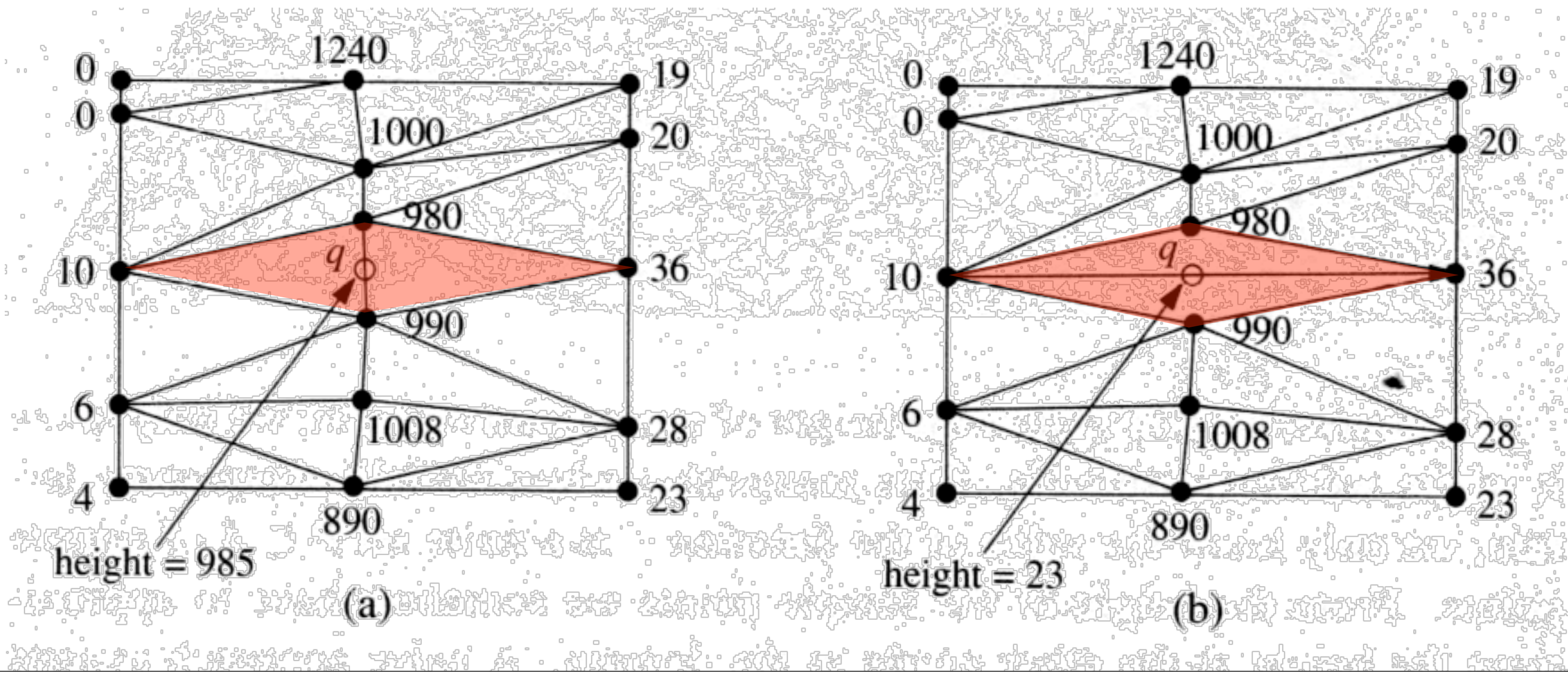


However given a point set, there can be many triangulations



Terrain Problem

- Some triangulations are “better” than others
- Avoid skinny triangles, i.e. maximize minimum angle of triangulation



Best Triangulations

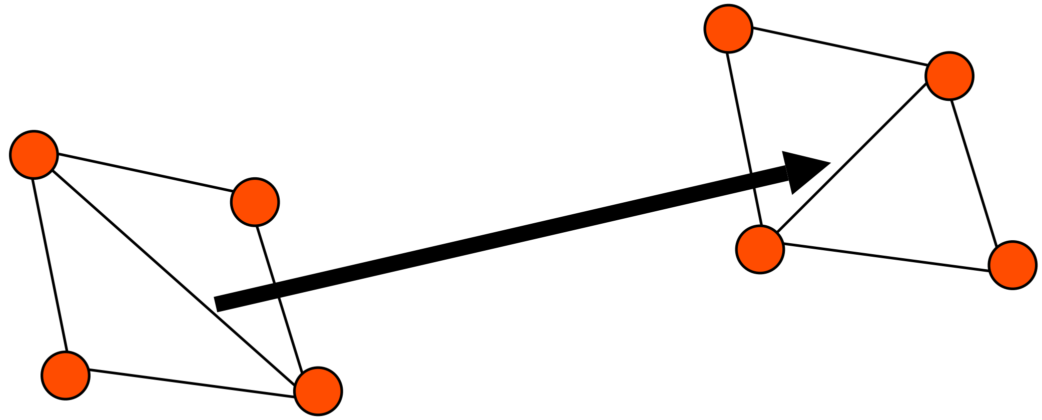
- Best triangulation is triangulation that is *angle optimal,* i.e. has the largest angle vector. Maximizes minimum angle.

- Create *angle vector* of the sorted angles of triangulation *T*, A(*T*) = (α_1, α_2, α_3, ... α_{3m}) with α_1 being the smallest angle

- A(*T*) is larger than A(*T'*) iff there exists an i such that $\alpha_j = \alpha'_j$ for all j < i and $\alpha_i > \alpha'_I$

Transform Triangulations

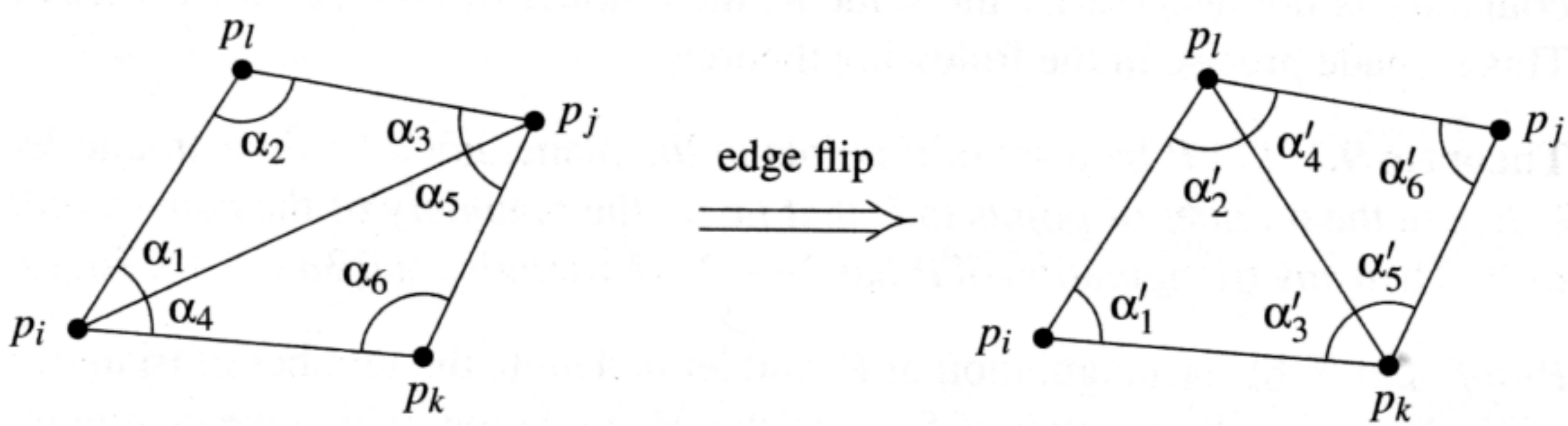
- Given two triangulations, one can always transform to the other one by flipping edges



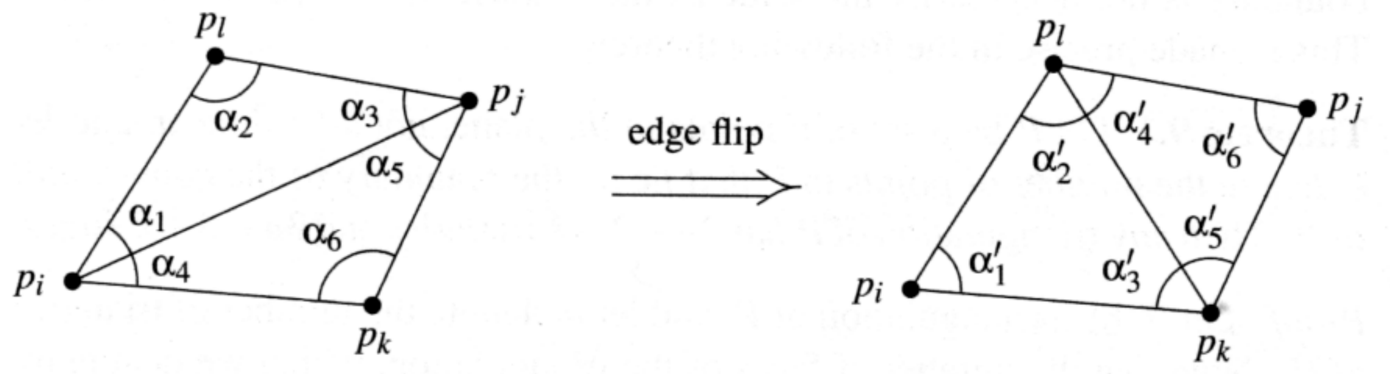
Angle Optimal Triangulations

Consider two adjacent triangles of T:

- If the two triangles form a convex quadrilateral, we could have an alternative triangulation by performing an *edge flip* on their shared edge.



Illegal Edges



- Edge *e* is illegal if:

$$\min_{1 \leqslant i \leqslant 6} \alpha_i < \min_{1 \leqslant i \leqslant 6} \alpha'_i.$$

- Only difference between *T* containing *e* and *T′* with *e* flipped are the six angles of the quadrilateral

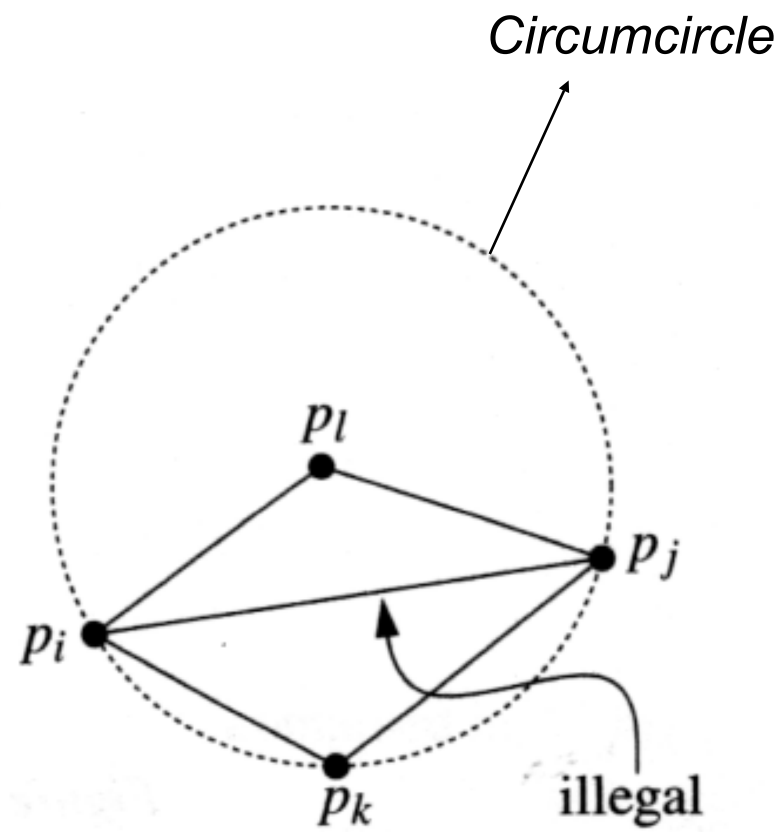
Illegal Triangulations

- If triangulation *T* contains an illegal edge e, we can make A(*T*) larger by flipping e

- In this case, *T* is an *illegal triangulation*

Testing for Illegal Edges

- If p_i, p_j, p_k, p_l form a convex quadrilateral and do not lie on a common circle, exactly one of p_ip_j and p_kp_l is an illegal edge.



- The edge p_ip_j is illegal iff p_l lies inside C.

Computing Legal Triangulations

1. Compute a triangulation of input points *P*

2. Flip illegal edges of this triangulation until all edges are legal

- Algorithm terminates because there is a finite number of triangulations

- *Too slow to be interesting…*

Delaunay Graphs

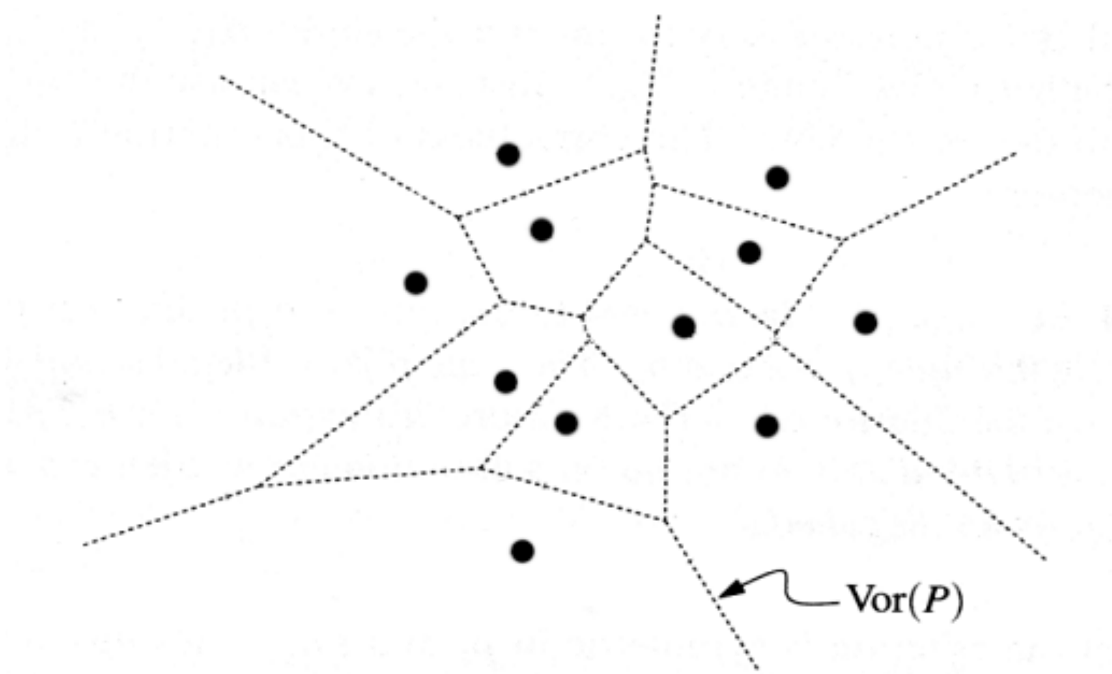
- Before we can understand an interesting solution to the terrain problem, we need to understand Delaunay Graphs

- Delaunay Graph of a set of points P is the dual graph of the Voronoi diagram of P

Delaunay Graphs

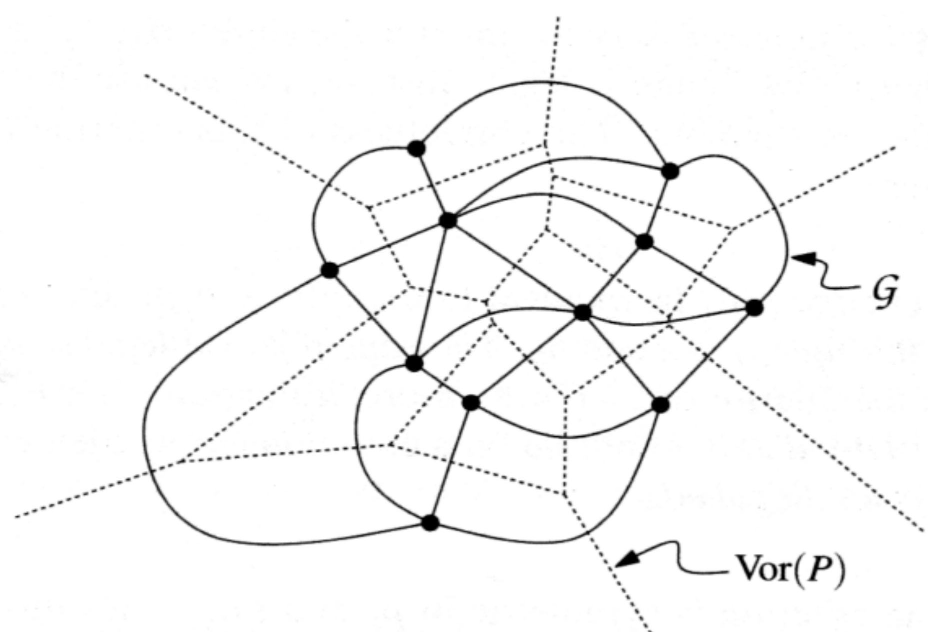
To obtain $\mathcal{DG}(P)$:

- Calculate $\mathcal{Vor}(P)$
- Place one vertex in each site of the $\mathcal{Vor}(P)$



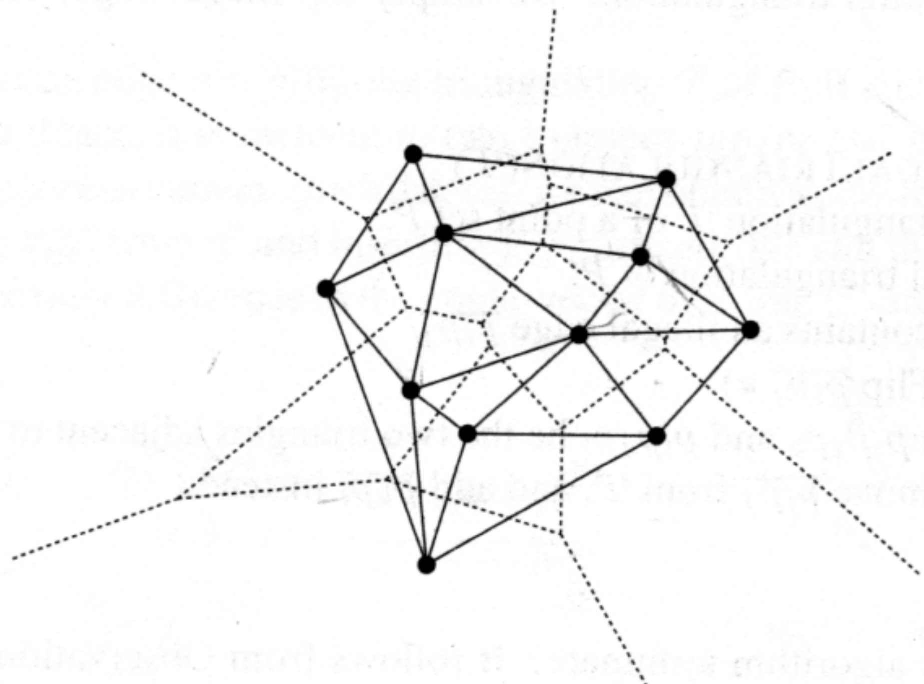
Constructing Delaunay Graphs

If two sites s_i and s_j share an edge (s_i and s_j are adjacent), create an arc between v_i and v_j, the vertices located in sites s_i and s_j



Constructing Delaunay Graphs

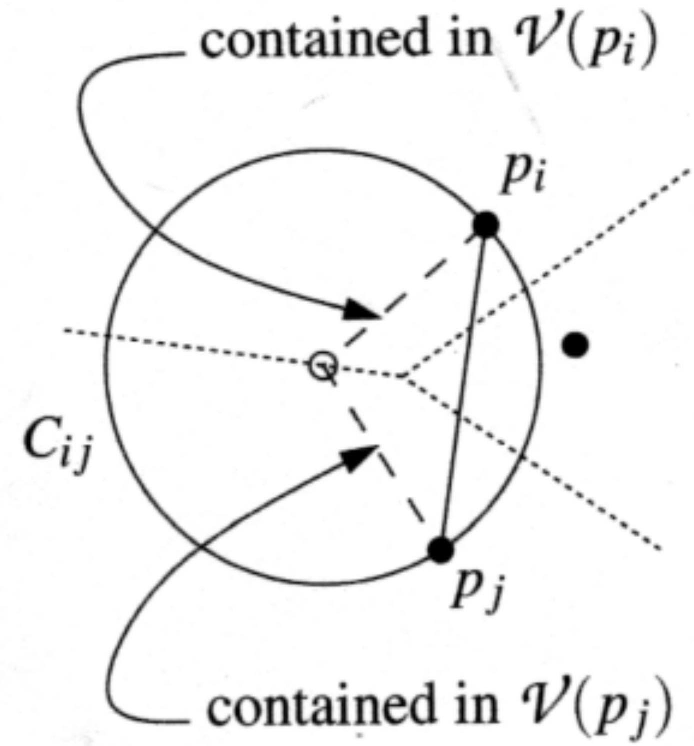
Finally, straighten the arcs into line segments. The resultant graph is $\mathcal{DG}$(P).



Properties of Delaunay Graphs

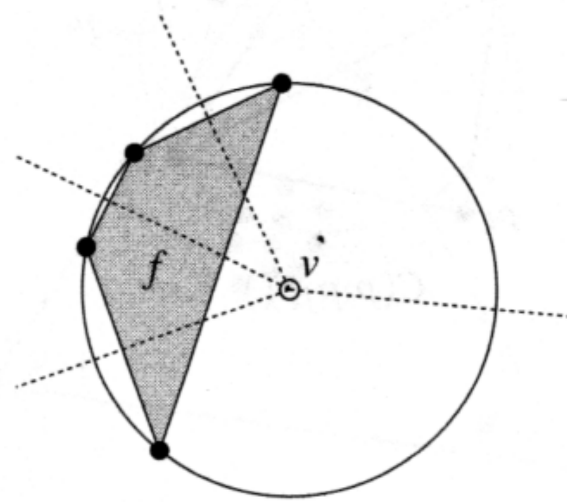
No two edges cross; $\mathcal{DG}(P)$ is a planar graph.

- Largest empty circle property



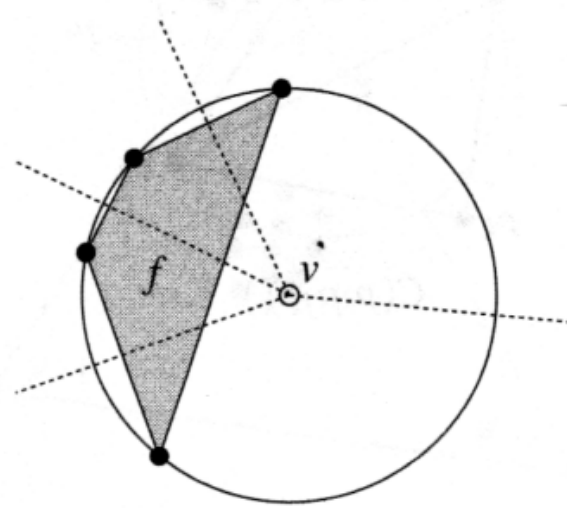
Delaunay Triangulations

- *Delaunay graph* is a triangulation if all points are in general position
  - No four or more points on a circle



Delaunay Triangulations

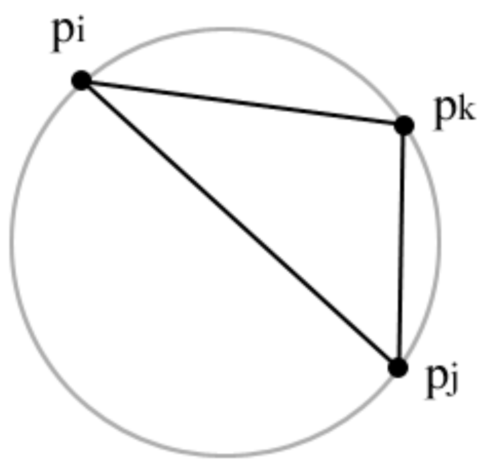
- These points form empty convex polygons, which can be triangulated.



Properties of Delaunay Triangles

From the properties of Voronoi Diagrams…

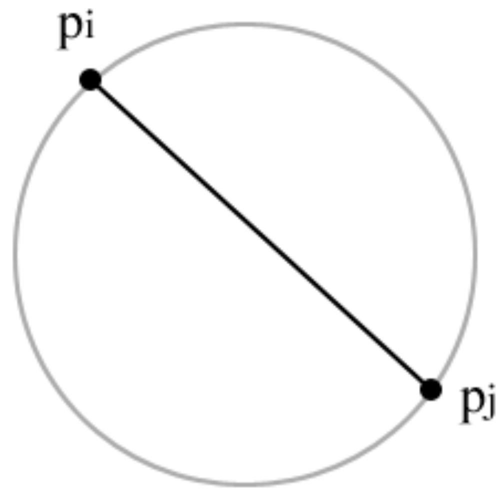
- Three points p_i, p_j, $p_k \in P$ are vertices of the same face of the $\mathcal{DG}(P)$ iff the circle through p_i, p_j, p_k contains no point of P on its interior.



Properties of Delaunay Triangles

From the properties of Voronoi Diagrams…

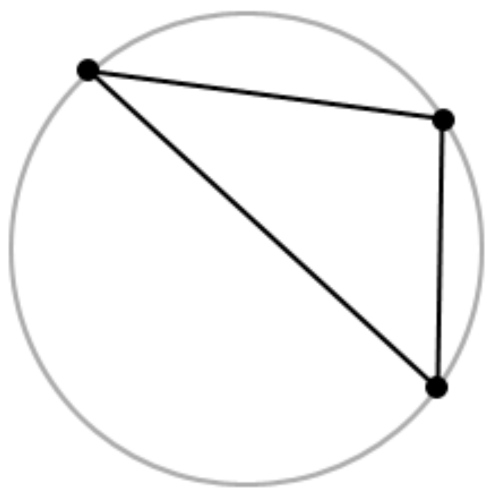
- Two points p_i, $p_j \in P$ form an edge of $\mathcal{DG}(P)$ iff there is a closed disc C that contains p_i and p_j on its boundary and does not contain any other point



Properties of Delaunay Triangles

From the previous two properties…

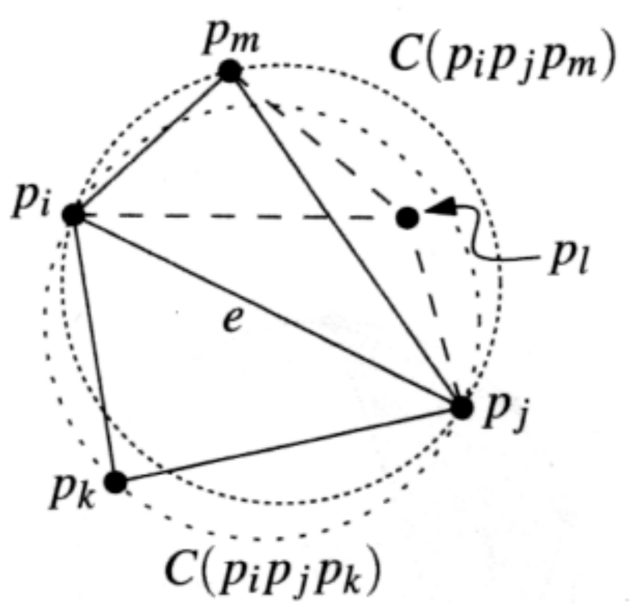
- A triangulation T of P is a $\mathcal{DT}(P)$ iff the circumcircle of any triangle of T does not contain a point of P in its interior



Legal Triangulations, revisited

A triangulation *T* of *P* is legal iff *T* is a $\mathcal{DT}$(P).

- DT → Legal: Empty circle property implies that all $\mathcal{DT}$ are legal
- Legal → DT: All legal triangles have empty circles



DT and Angle Optimal

The angle optimal triangulation is a $\mathcal{DT}$

- If P is in general position, $\mathcal{DT}(P)$ is unique and thus, is angle optimal

What if multiple $\mathcal{DT}$ exist for P?

- the minimum angle of each of the $\mathcal{DT}$ is the same
- Thus, all the $\mathcal{DT}$ are equally "good" for the terrain problem. All $\mathcal{DT}$ maximize the minimum angle

Terrain Problem, revisited

Therefore, the problem of finding a triangulation that maximizes the minimum angle is reduced to the problem of finding a Delaunay Triangulation.

So how do we find the Delaunay Triangulation?

How do we compute $\mathcal{DT}(P)$?

- We could compute $\mathcal{Vor}(P)$ then dualize into $\mathcal{DT}(P)$
  - Plane sweep algorithm O(n log n) time

- Instead, we will compute $\mathcal{DT}(P)$ using a randomized incremental method
  - O(n log n) expect time
  - Provide point location data structure
    - Good for height query

Algorithm Overview

1. Initialize triangulation *T* with a "big enough" helper bounding triangle that contains all points *P*.
2. Randomly choose a point p_r from *P*
3. Find the triangle Δ that p_r lies in
4. Subdivide Δ into smaller triangles that have p_r as a vertex
5. Flip edges until all edges are legal
6. Repeat steps 2-5 until all points have been added to *T*

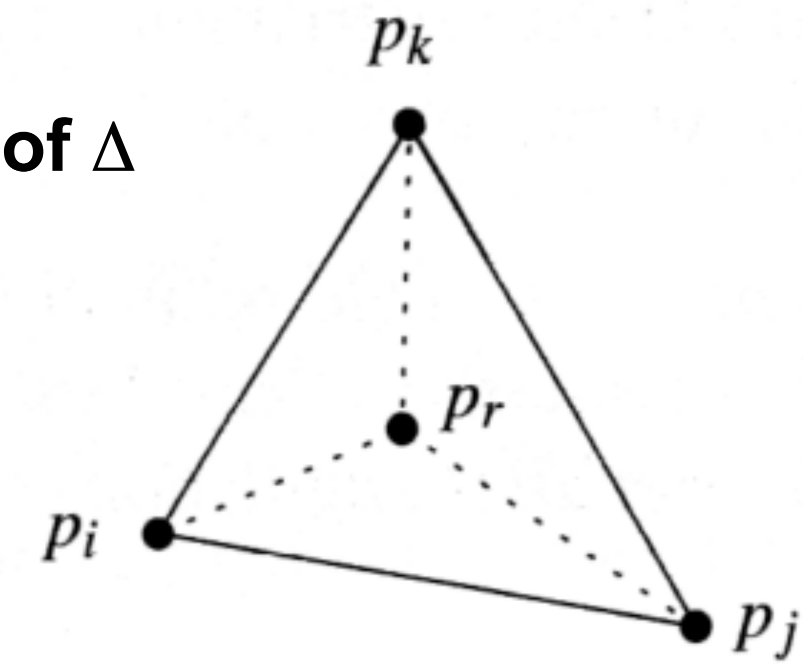
Let's skip steps 1, 2, and 3 for now…

Triangle Subdivision: Case 1 of 2

Assuming we have already found the triangle that p_r lives in, subdivide Δ into smaller triangles that have p_r as a vertex.

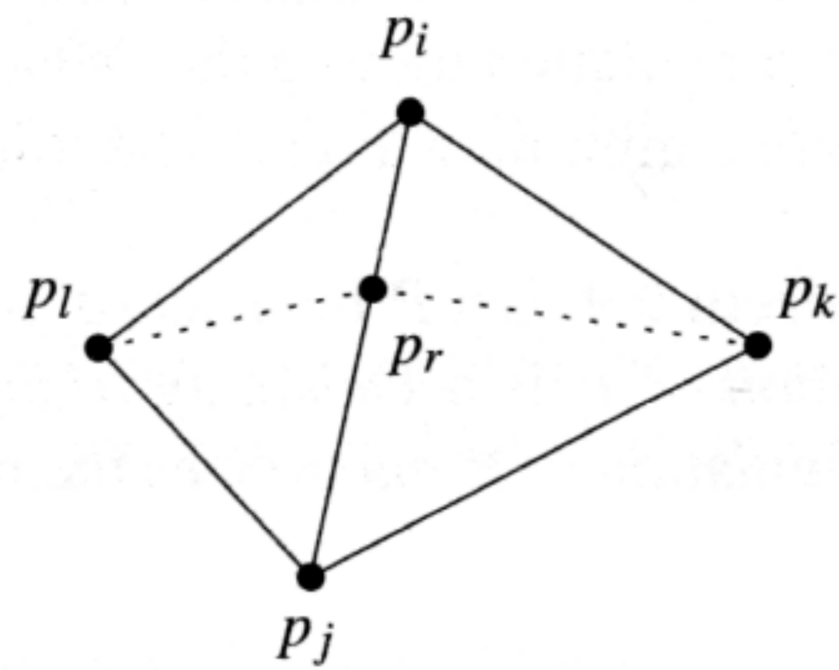
Two possible cases:

1) p_r lies in the interior of Δ



Triangle Subdivision: Case 2 of 2

2) p_r falls on an edge between two adjacent triangles



Which edges are illegal?

- Before we subdivided, all of our edges were legal
- After we add our new edges, some of the edges of T may now be illegal, but which ones?

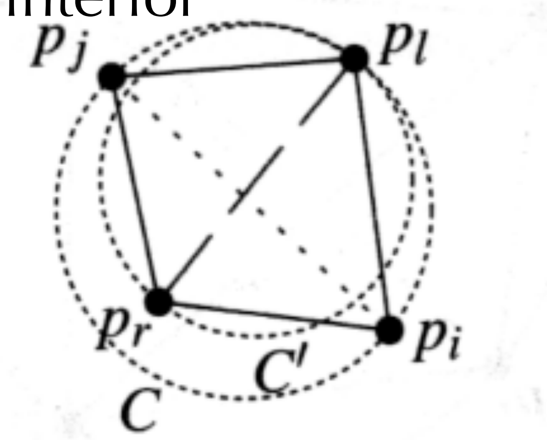
New Edges are Legal

Are the new edges (edges involving p_r) legal?

Consider **any** new edge $p_r p_l$.

Before adding $p_r p_l$,

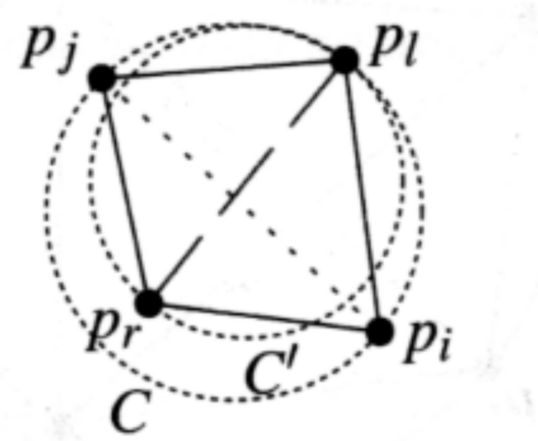
- p_l was part of some triangle $p_i p_j p_l$
- Circumcircle *C* of p_i, p_j, and p_l did not contain any other points of *P* in its interior



New edges incident to p_r are Legal

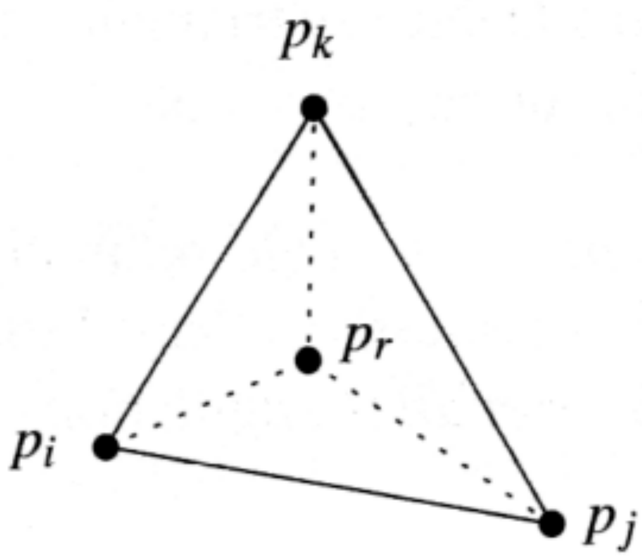
- If we shrink *C*, we can find a circle *C′* that passes through $p_r p_l$
- C′ contains no points in its interior.
- Therefore, $p_r p_l$ is legal.

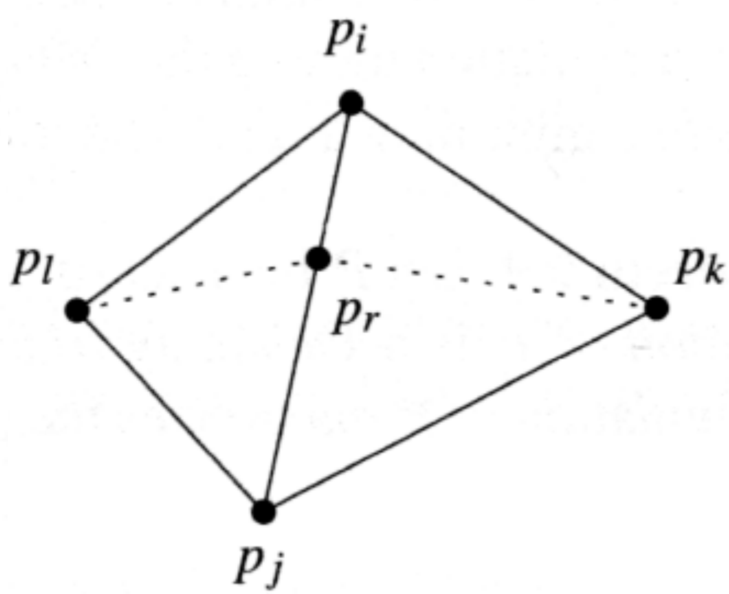
Any new edge incident p_r is legal



Outer Edges May Be Illegal

- An edge can become illegal only if one of its incident triangles changed
- Outer edges of the incident triangles $\{p_jp_k, p_ip_k, p_ip_j\}$ or $\{p_ip_l, p_lp_j, p_jp_k, p_kp_i\}$ may have become illegal.





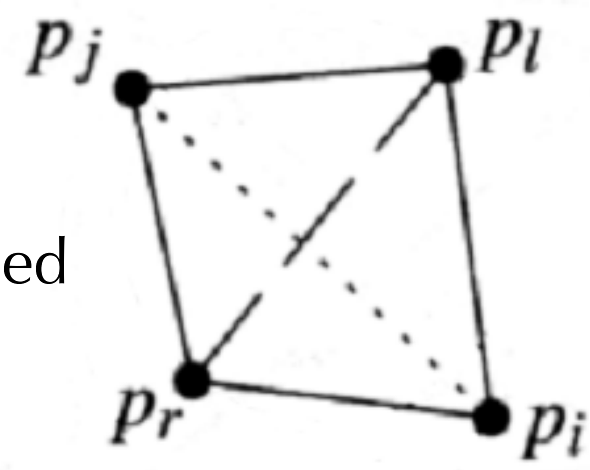
Flip Illegal Edges

- Now that we know which edges have become illegal, we flip them

- However, after the edges have been flipped, the edges incident to the new triangles may now be illegal.

- So we need to recursively flip edges…

LegalizeEdge



p_r = point being inserted

$p_i p_j$ = edge that may need to be flipped

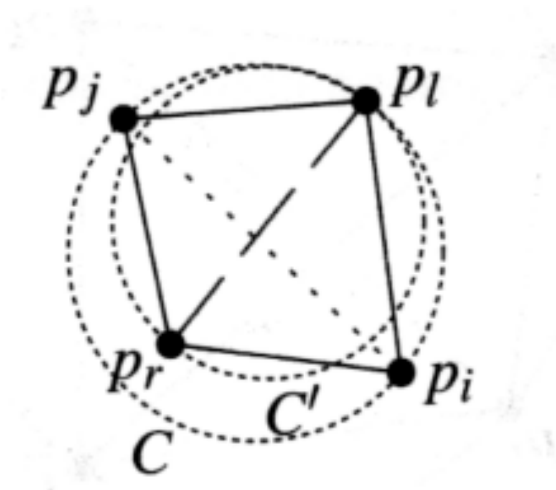
LEGALIZEEDGE(p_r, $p_i p_j$, T)

- **if** $p_i p_j$ is illegal
- **then** Let $p_i p_j p_l$ be the triangle adjacent to $p_r p_i p_j$ along $p_i p_j$
- Replace $p_i p_j$ with $p_r p_l$
- LEGALIZEEDGE(p_r, $p_i p_l$, T)

Flipped edges are incident to p_r

Notice that when LEGALIZEEDGE flips edges, these new edges are ALL incident to p_r

- By the same logic as earlier, we can shrink the circumcircle of $p_i p_j p_l$ to find a circle that passes through p_r and p_l
- Thus, the new edges are legal



Bounding Triangle

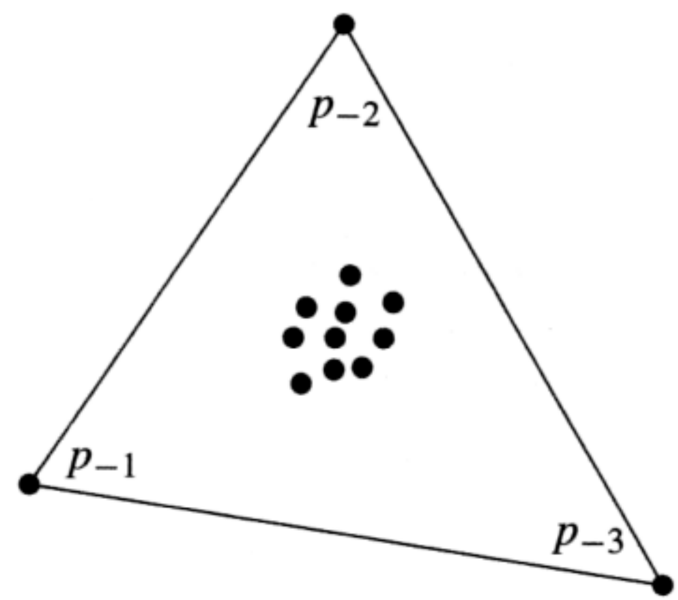
Remember, we skipped step 1 of our algorithm.

1. *Begin with a "big enough" helper bounding triangle that contains all points.*

Let $\{p_{-3}, p_{-2}, p_{-1}\}$ be the vertices of our bounding triangle.

"Big enough" means that the triangle:

- contains all points of P in its interior.
- will not destroy edges between points in P.



Triangle Location Step

Remember, we skipped step 3 of our algorithm.

3. Find the triangle T that p_r lies in

- Take an approach similar to Point Location approach

- Maintain a point location structure $\mathcal{D}$, a directed acyclic graph (DAG)

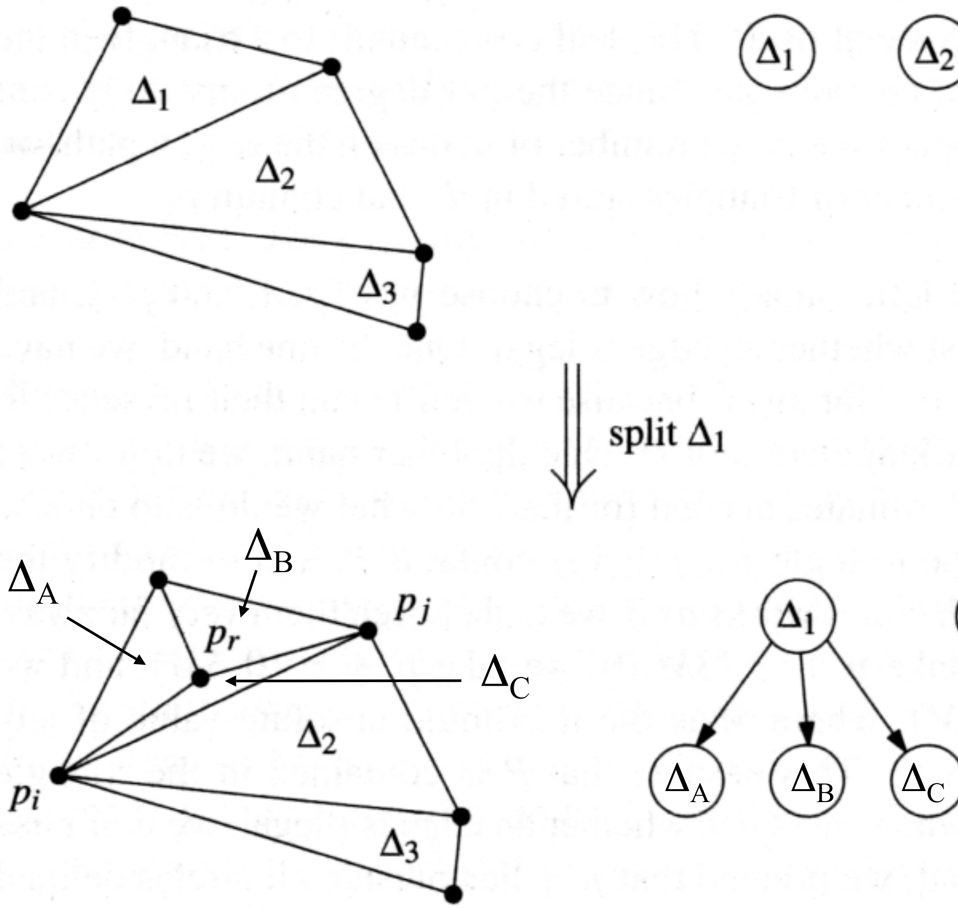
Structure of $\mathcal{D}$

- Leaves of $\mathcal{D}$ correspond to the triangles of the current triangulation.
- Maintain cross pointers between leaves of $\mathcal{D}$ and the triangulation.
- Begin with a single leaf, the bounding triangle $p_{-1}p_{-2}p_{-3}$

Subdivision and $\mathcal{D}$

- Whenever we split a triangle Δ_1 into smaller triangles Δ_a and Δ_b (and possibly Δ_c), add the smaller triangles to D as leaves of Δ_1

Subdivision and $\mathcal{D}$

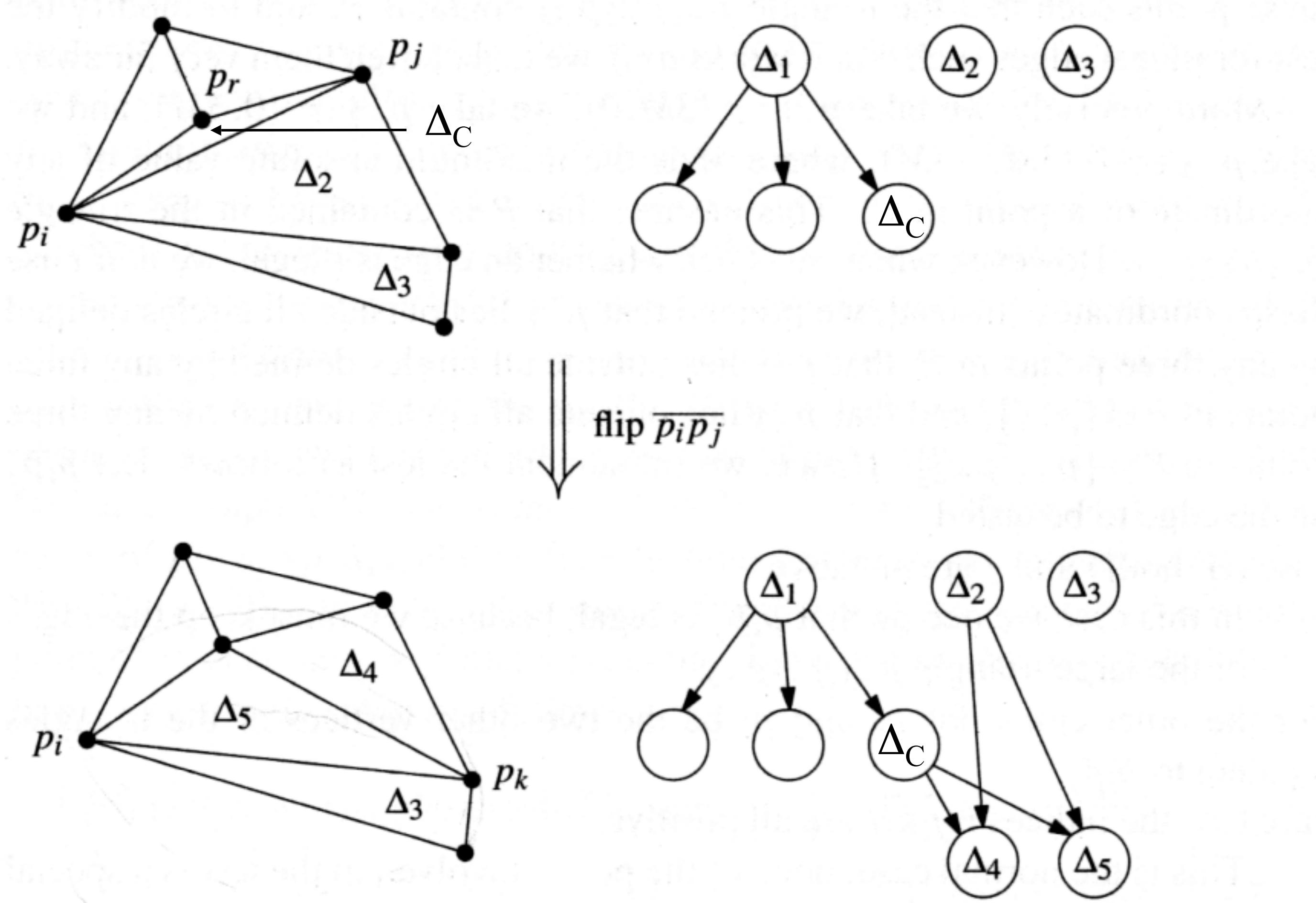


Edge Flips and $\mathcal{D}$

- Whenever we perform an edge flip, create leaves for the two new triangles.

- Attach the new triangles as leaves of the two triangles replaced during the edge flip.

Edge Flips and $\mathcal{D}$



Searching $\mathcal{D}$

p_r = point we are searching with

1. Let the current node be the root node of $\mathcal{D}$.
2. Look at child nodes of current node. Check which triangle p_r lies in.
3. Let current node = child node that contains p_r
4. Repeat steps 2 and 3 until we reach a leaf node.

Searching $\mathcal{D}$

- Each node has at most 3 children.
- Each node in path represents a triangle in $\mathcal{D}$ that contains p_r
- Therefore, takes O(number of triangles in $\mathcal{D}$ that contain p_r)

Properties of $\mathcal{D}$

Notice that the:

- Leaves of $\mathcal{D}$ correspond to the triangles of the current triangulation
- Internal nodes correspond to *destroyed triangles*, triangles that were in an earlier stage of the triangulation but are not present in the current triangulation

Algorithm Overview

1. Initialize triangulation *T* with helper bounding triangle. Initialize $\mathcal{D}$.
2. Randomly choose a point p_r from *P*.
3. Find the triangle Δ that p_r lies in using $\mathcal{D}$.
4. Subdivide Δ into smaller triangles that have p_r as a vertex. Update $\mathcal{D}$ accordingly.
5. Call LEGALIZEEDGE on all possibly illegal edges, using the modified test for illegal edges. Update $\mathcal{D}$ accordingly.
6. Repeat steps 2-5 until all points have been added to *T*.

Break

- 10 Min Break

Analysis Goals

- Expected storage required is:

  $O(n)$

- Expected running time of algorithm is:

  $O(n \log n)$

First, some notation…

- $P_r = \{p_1, p_2, \ldots, p_r\}$
  - Points added by iteration r

- $\Omega = \{p_{-3}, p_{-2}, p_{-1}\}$
  - Vertices of bounding triangle

- $\mathcal{DG}_r = \mathcal{DG}(\Omega \cup P_r)$
  - Delaunay graph as of iteration r

Sidetrack: Expected Number of Δs

Lemma 9.11 *Expected number of triangles created by* DELAUNAYTRIANGULATION *is* *9n+1*.

- In initialization, we create 1 triangle (bounding triangle).

Expected Number of Triangles

In iteration r where we add p_r,

- in the subdivision step, we create at most 4 new triangles. Each new triangle creates one new edge incident to p_r
- each edge flipped in LEGALIZEEDGE creates two new triangles and one new edge incident to p_r

<u>Expected Number of Triangles</u>

Let k = number of edges incident to p_r after insertion of p_r, the degree of p_r

- We have created at most 2(k-3)+3 triangles.
- -3 and +3 are to account for the triangles created in the subdivision step

The problem is now to find the expected degree of p_r

Expected Degree of p_r

Use backward analysis:

- Fix P_r, let p_r be a random element of P_r
- $\mathcal{DG}_r$ has 3(r+3)-6 edges
- Total degree of $P_r \leq 2[3(r+3)-9] = 6r$

E[degree of random element of P_r] ≤ 6

Triangles created at step r

Using the expected degree of p_r, we can find the expected number of triangles created in step r.

$\deg(p_r, \mathcal{DG}_r)$ = degree of p_r in $\mathcal{DG}_r$

$$
\begin{aligned}
\mathrm{E}[\text{number of triangles created in step } r] &\leqslant \mathrm{E}[2\deg(p_r, \mathcal{DG}_r) - 3] \\
&= 2\mathrm{E}[\deg(p_r, \mathcal{DG}_r)] - 3 \\
&\leqslant 2 \cdot 6 - 3 = 9
\end{aligned}
$$

Expected Number of Triangles

Now we can bound the number of triangles:

≤ 1 initial Δ + Δs created at step 1 + Δs created at step 2 + … + Δs created at step n

≤ 1 + 9n

Expected number of triangles created is 9n +1.

Storage Requirement

- $\mathcal{D}$ has one node per triangle created
- 9n+1 triangles created
- O(n) expected storage

Expected Running Time

Let's examine each step…

1. *Begin with a "big enough" helper bounding triangle that contains all points.*

   O(1) time, executed once = O(1)

2. *Randomly choose a point p_r from P.*

   O(1) time, executed n times = O(n)

3. *Find the triangle Δ that p_r lies in.*

   Skip step 3 for now…

Expected Running Time

4. *Subdivide Δ into smaller triangles that have p_r as a vertex.*

   O(1) time executed n times = O(n)

5. *Flip edges until all edges are legal.*

   In total, expected to execute a total number of times proportional to number of triangles created = O(n)

Thus, total running time without point location step is O(n).

Point Location Step

- Time to locate point p_r is
  O(number of nodes of $\mathcal{D}$ we visit)
  + O(1) for current triangle

- Number of nodes *of* $\mathcal{D}$ we visit
  = number of destroyed triangles that contain p_r

- A triangle is destroyed by p_r if its circumcircle contains p_r

We can charge each triangle visit to a Delaunay triangle whose circumcircle contains p_r

Point Location Step

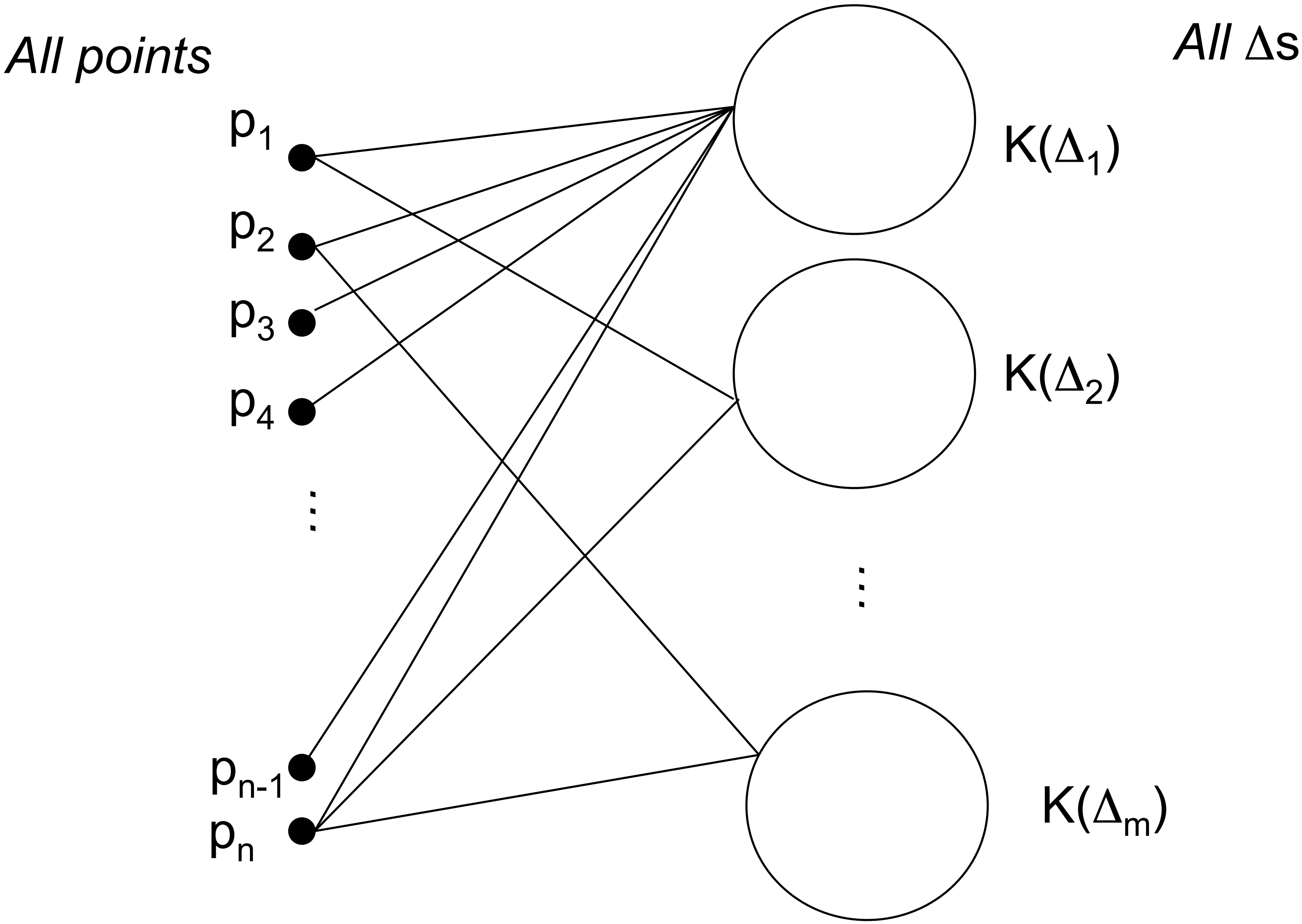
K(Δ) = subset of points in *P* that lie in the circumcircle of Δ

- When $p_r \in K(\Delta)$, charge to Δ.
- Since we are iterating through *P*, each point in K(Δ) can be charged at most once.

Total time:

$$O(n + \sum_{\Delta} \mathrm{card}(K(\Delta))),$$

Point Location Step



Point Location Step

We want to have O(*n* log *n*) time, therefore we want to show that:

$$\sum_{\Delta} \mathrm{card}(K(\Delta)) = O(n \log n),$$

Intuitions

When no points are added to the triangulation, for any Δ

$K(\Delta)=n$

When all points are added to the triangulation, for any Δ

$K(\Delta)=0$

When r points are added to the triangulation, for any Δ

Expected $K(\Delta)=n/r$

Point Location Step

Introduce some notation…

$\mathcal{T}_r$ = set of triangles of $\mathcal{DG}(\Omega \cup P_r)$

$\mathcal{T}_r \setminus \mathcal{T}_{r-1}$ triangles created in stage r

Rewrite our sum as:

$$\sum_{\Delta} \operatorname{card}(K(\Delta)) \Longrightarrow \sum_{r=1}^{n} \Bigg(\underbrace{\sum_{\Delta \in \mathcal{T}_r \setminus \mathcal{T}_{r-1}}}_{\textit{New } \Delta\text{s}} \operatorname{card}(K(\Delta)) \Bigg).$$

Point Location Step

More notation…

$k(P_r, q)$ = number of triangles $\Delta \in \mathcal{T}_r$ such that q is contained in Δ

$k(P_r, q, p_r)$ = number of triangles $\Delta \in \mathcal{T}_r$ such that q is contained in Δ and p_r is incident to Δ

$$\sum_{\Delta \in \mathcal{T}_r \setminus \mathcal{T}_{r-1}} \mathrm{card}(K(\Delta)) = \sum_{q \in P \setminus P_r} k(P_r, q, p_r).$$

Point Location Step

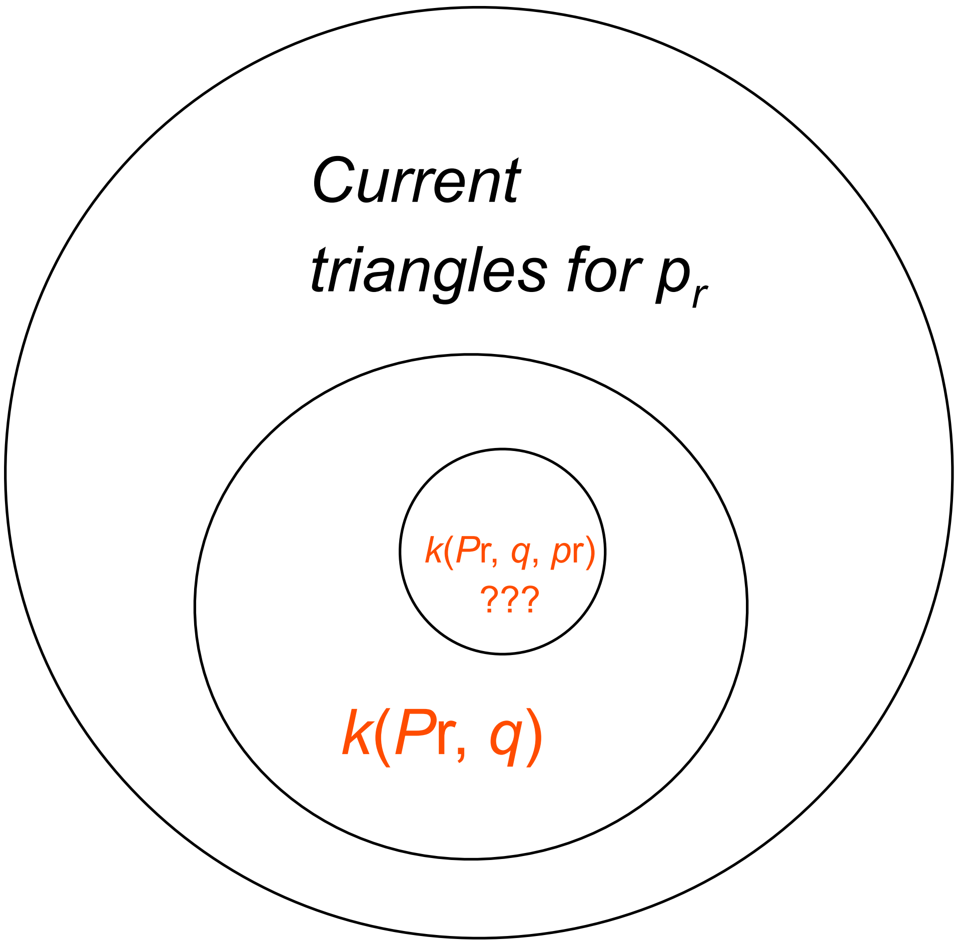
Find the $E[k(P_r, q, p_r)]$ then sum later…

- Fix P_r, so $k(P_r, q, p_r)$ depends only on p_r
- Probability that p_r is incident to a triangle is $3/r$
  (Backward analysis again!)

Thus:

$$E[k(P_r, q, p_r)] \leqslant \frac{3k(P_r, q)}{r}.$$

Point Location Step



Probability that p_r is incident to a triangle is $3/r$

Point Location Step

Using: $\mathrm{E}[k(P_r, q, p_r)] \leqslant \frac{3k(P_r, q)}{r}.$

We can rewrite our sum as:

$$\mathrm{E}\Big[\sum_{\Delta \in \mathcal{T}_r \setminus \mathcal{T}_{r-1}} \mathrm{card}(K(\Delta))\Big] \leqslant \frac{3}{r} \sum_{q \in P \setminus P_r} k(P_r, q).$$

Point Location Step

$$\mathrm{E}\Big[\sum_{\Delta \in \mathcal{T}_r \setminus \mathcal{T}_{r-1}} \mathrm{card}(K(\Delta))\Big] \leqslant \frac{3}{r} \sum_{q \in P \setminus P_r} k(P_r, q).$$

Conclusion

- Delaunay triangulation is optimal angle-maximize triangulation
- Delaunay triangulation can be done in O(nlogn) time using O(n) space
- Delaunay triangulation have many applications

Surface Reconstruction