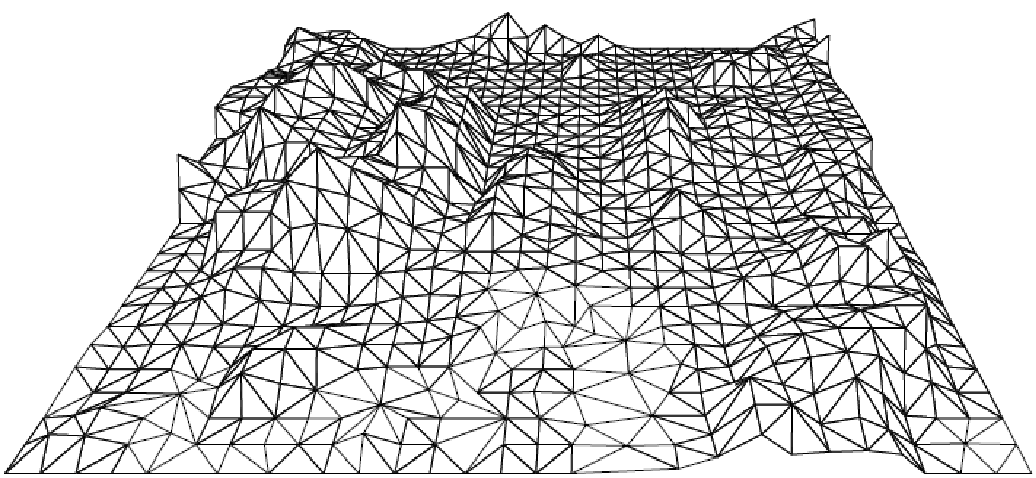
Delaunay Triangulations

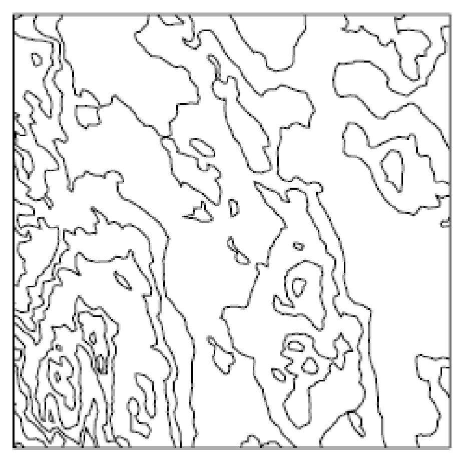
Prepared By Zohreh Mohseni

Motivation: Terrains

- Set of data points $A \subset R^2$
- Height $f(p)$ defined at each sample point p in A

  A terrain can be visualized with a perspective drawing like the one in below figure, or with contour lines

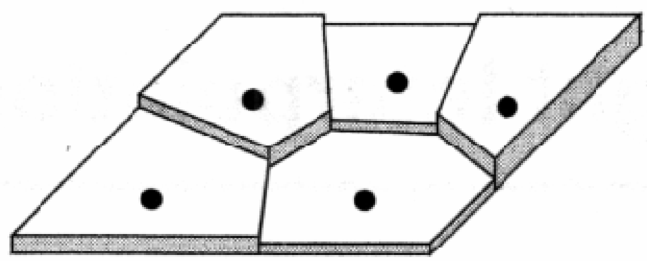




How can we most naturally approximate height of points not in p?

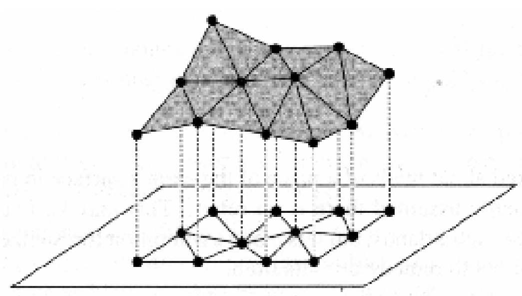
Option: Discretize

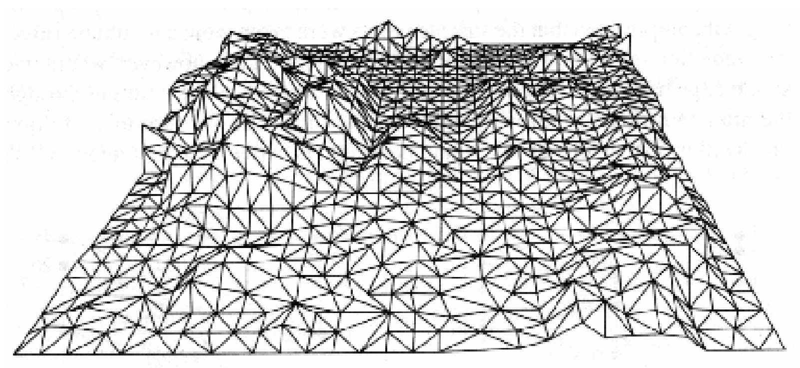
- Let f(p) = height of nearest sample point for points not in p



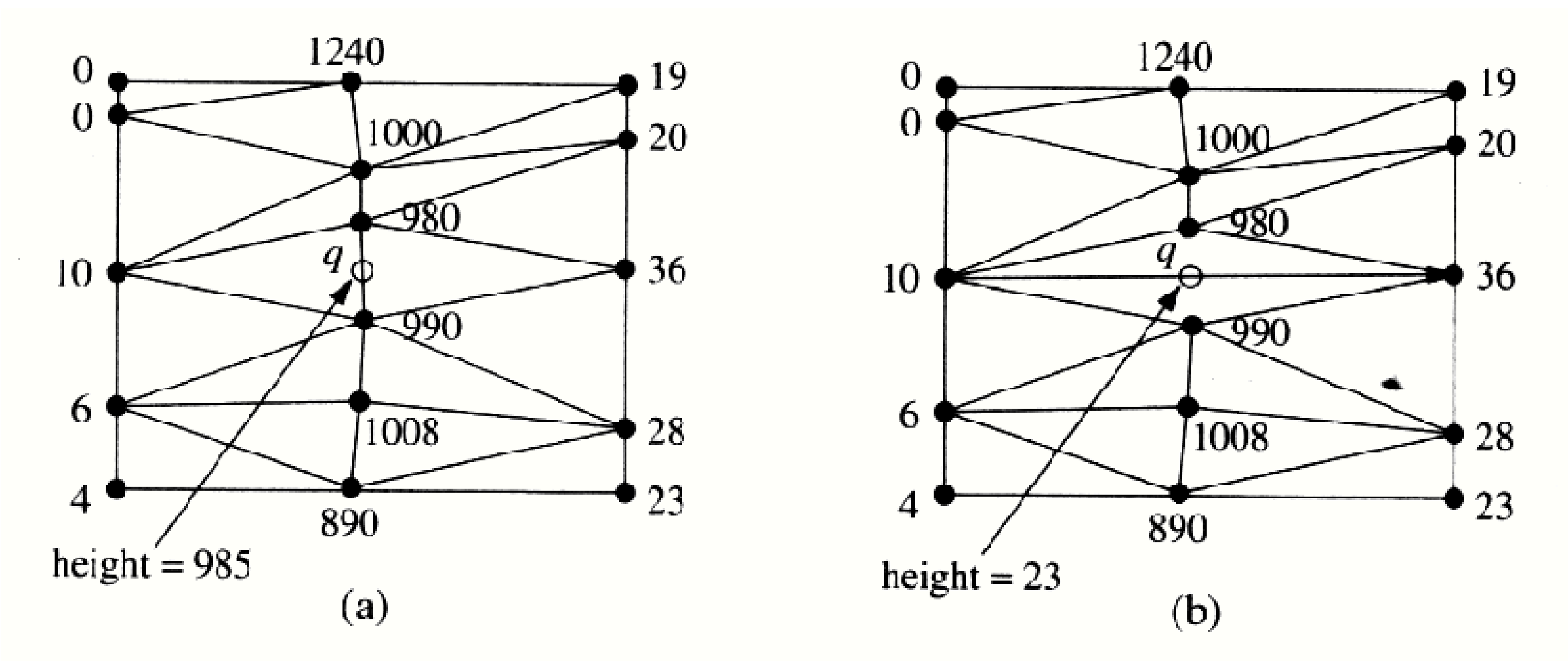
Better Option: Triangulation

- Polyhedral terrain: Determine a triangulation of p in R^2, then raise points to desired height
- triangulation: planar subdivision whose bounded faces are triangles with vertices from p





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

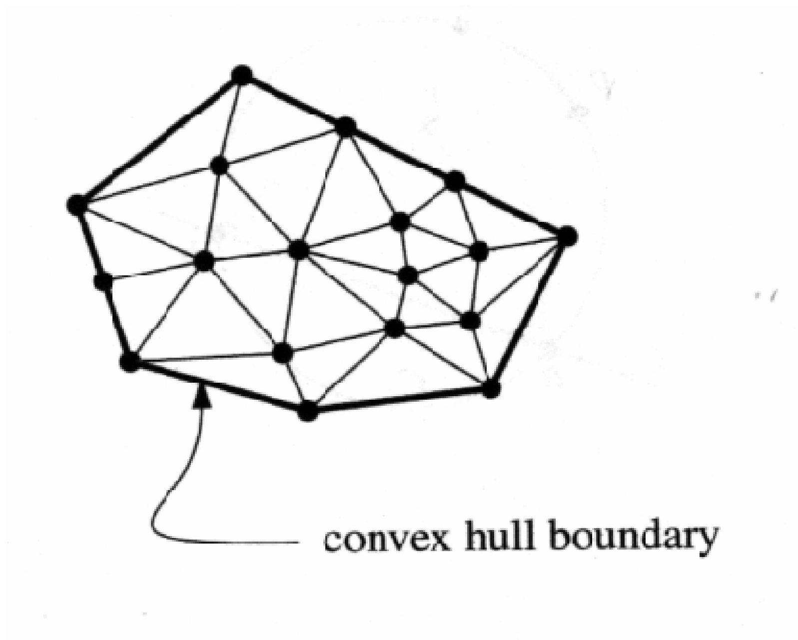


Triangulation: Formal Definition

- maximal planar subdivision: a subdivision S such that no edge connecting two vertices can be added to S without destroying its planarity
- triangulation of set of points P: a maximal planar subdivision whose vertices are elements of P

Triangulation is made of triangles

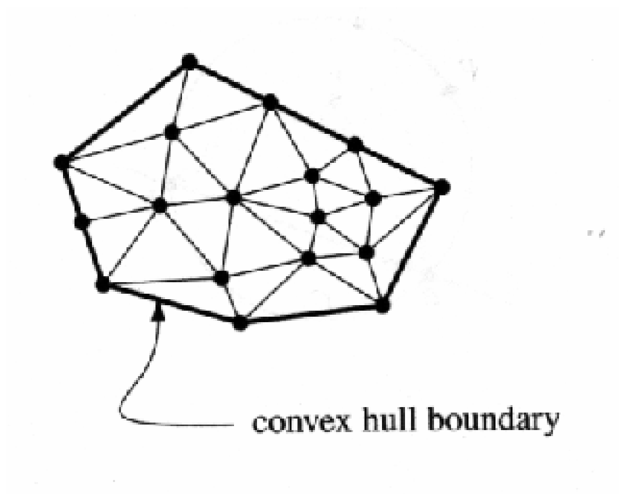
- Outer polygon must be convex hull
- Internal faces must be triangles, otherwise they could be triangulated further



Theorem 9.1

For P consisting of n points,not all collinear. all triangulations contain 2n-2-k triangles, 3n-3-k edges

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



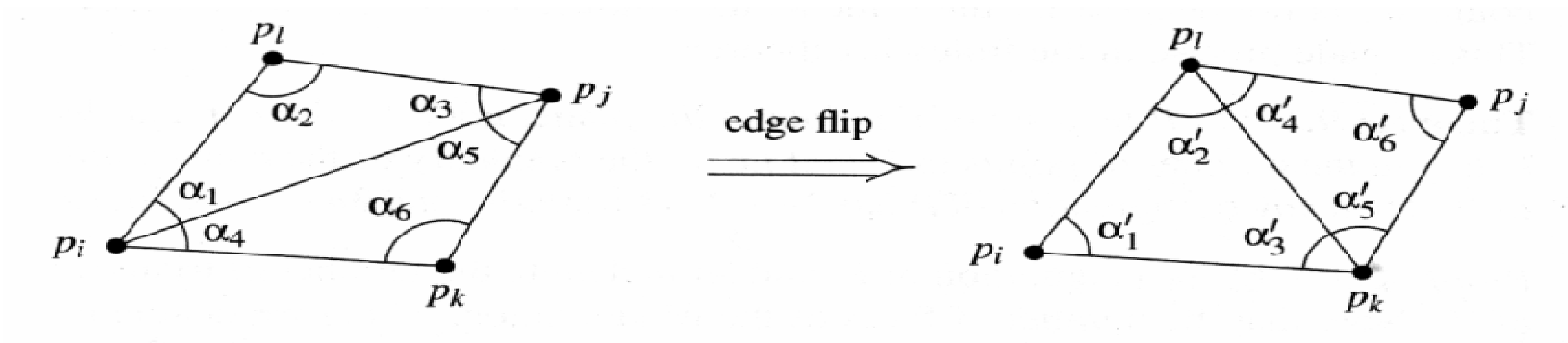
Angle Optimal Triangulations

- Create angle vector of the sorted angles of triangulation T, $(\alpha_1, \alpha_2, \alpha_3, \ldots, \alpha_{3m}) = A(T)$ 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 $\alpha_i > \alpha'_i$
- Best triangulation is triangulation that is *angle optimal, i.e. has the largest angle vector.*

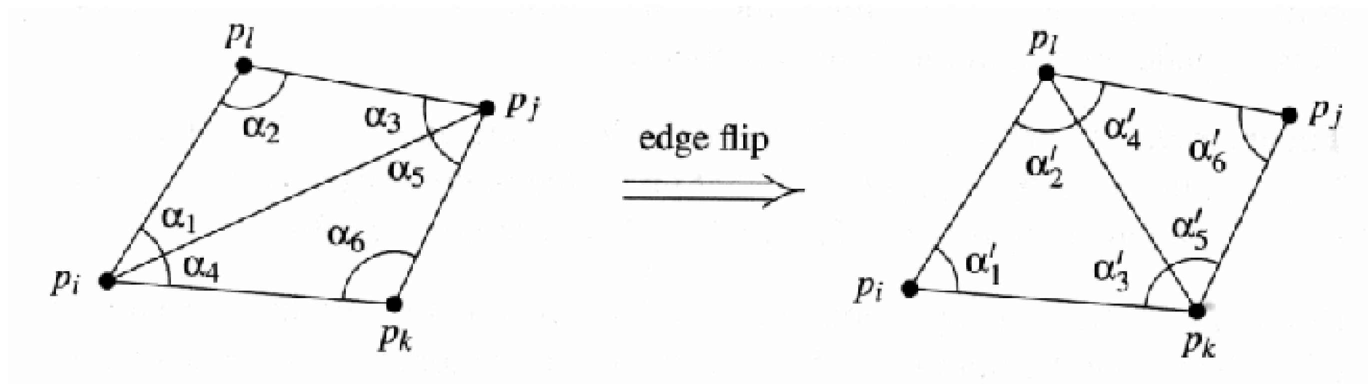
Angle Optimal Triangulations

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.

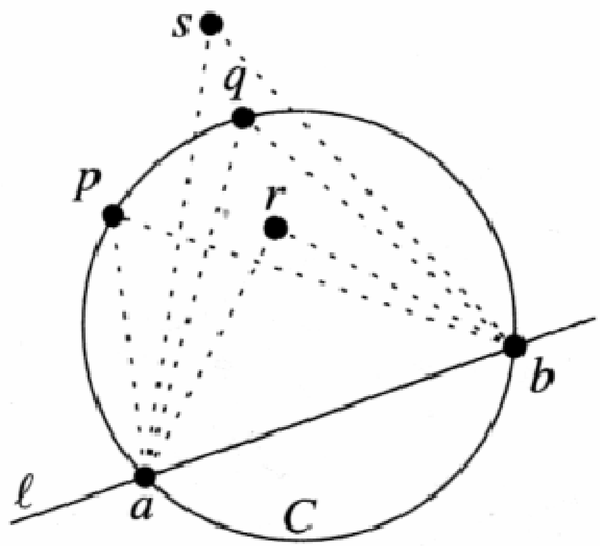
Illegal Triangulations

- 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.

Thales’s Theorem (Theorem 9.2)

- We can use Thales’ Theorem to test if an edge is legal without calculating angles

Let C be a circle, l a line intersecting C in points a and b and p, q, r, and s points lying o the same side of l. Suppose tha *p and q lie on C, that r lies inside C, and that s lies outside C. Then:*

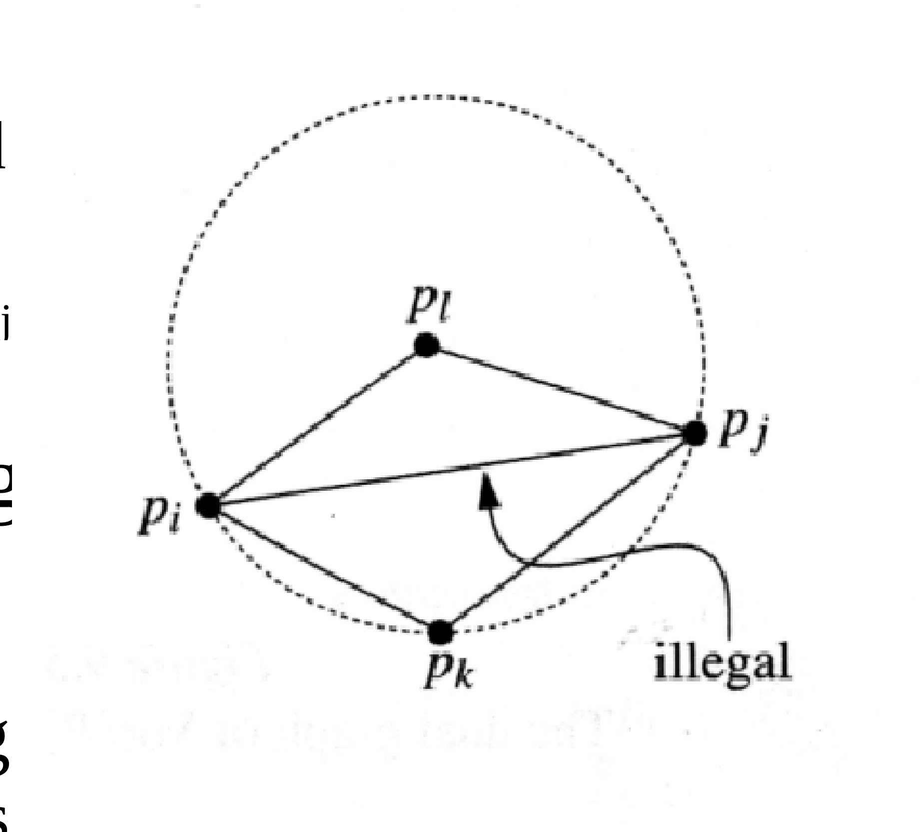


$$\measuredangle arb > \measuredangle apb = \measuredangle aqb > \measuredangle asb.$$

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 illeg edge.



- The edge p_ip_j is illeg
  – Proved using Thales's

p_i-p_j-p_k is smaller than the angle p_i-p_l-p_k

Algorithm *LEGALTRIANGULATION(T)*

Input. Some triangulation T of a point set P.

Output. A legal triangulation of P.

1. while T contains an illegal edge pipj
2. do (Flip $p_i p_j$)
3. Let $p_i p_j p_k$ and $p_i p_j p_l$ be the two triangles adjacent to $p_i p_j$.
4. Remove $p_i p_j$ from T, and add $p_k p_l$ instead.
5. Return T

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…

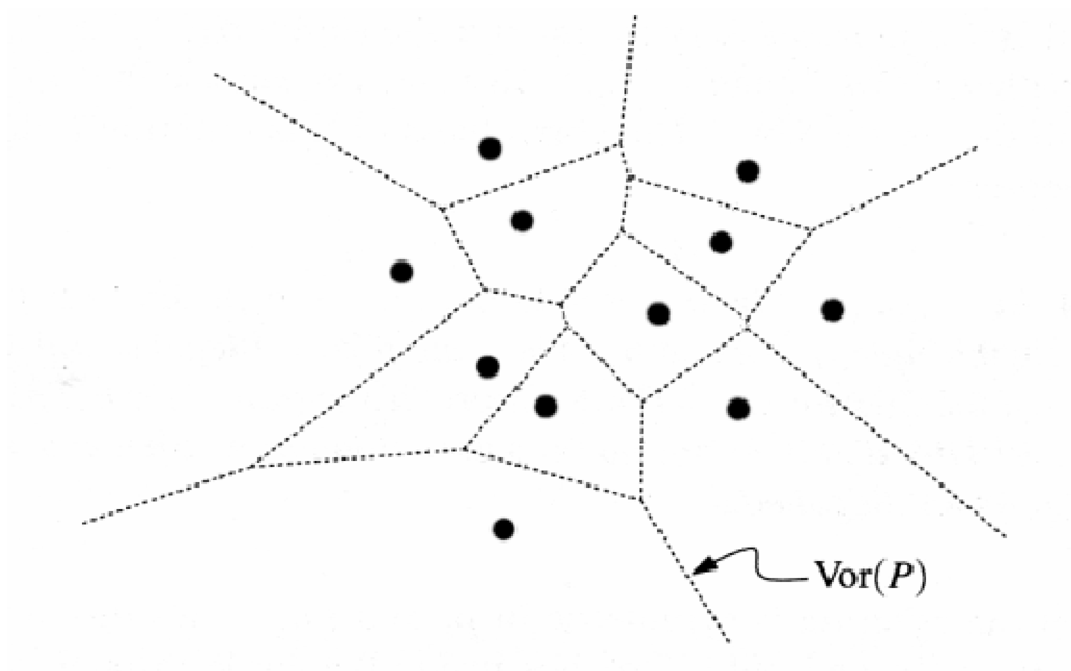
Sidetrack: Delaunay Graphs

- 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

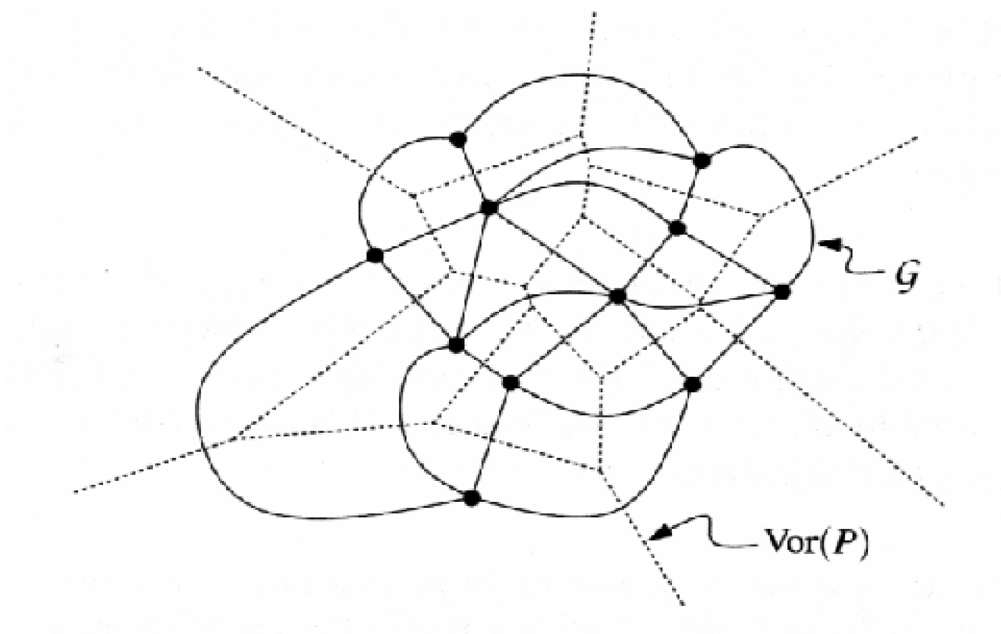
To obtain DG(P):

- Calculate Vor(P)
- Place one vertex in each site of the Vor(P)



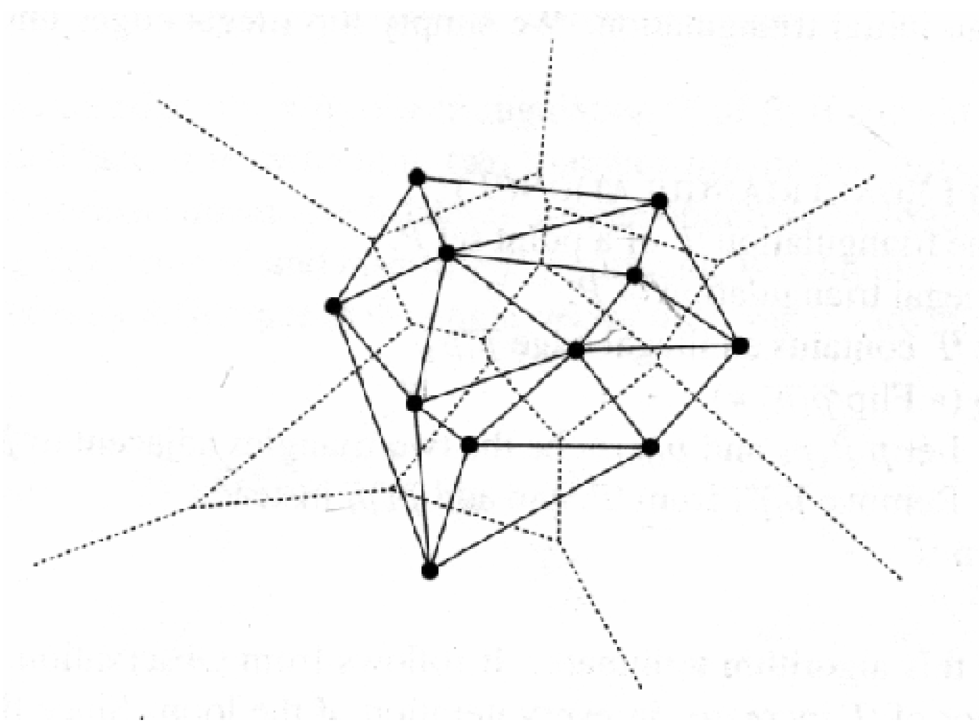
Constructing Delaunay Graphs

If two sites s_i and s_j share an edge (i.e., are adjacent), create an arc between vi and vj, the vertices located in sites si and sj



Constructing Delaunay Graphs

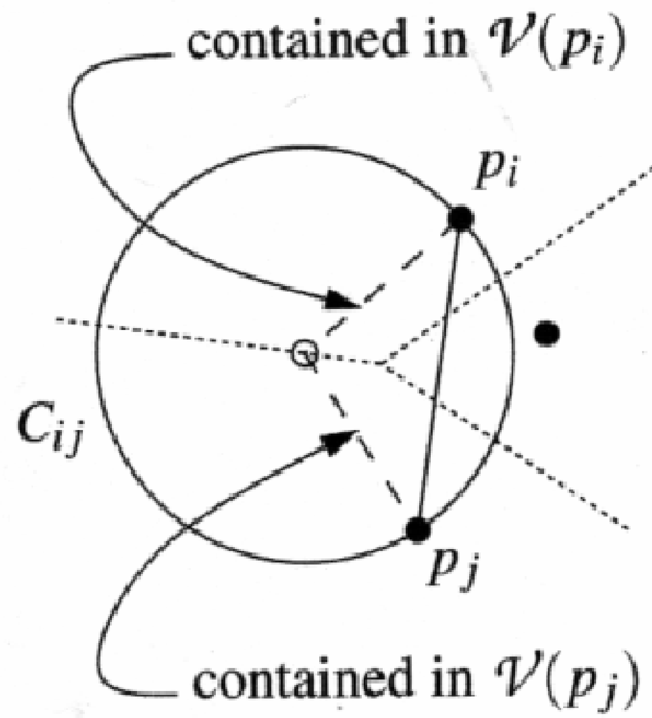
Finally, straighten the arcs into line segments. The resultant graph is DG(P).



Properties of Delaunay Graphs

Theorm 9.5: DG(P) is a plane graph.

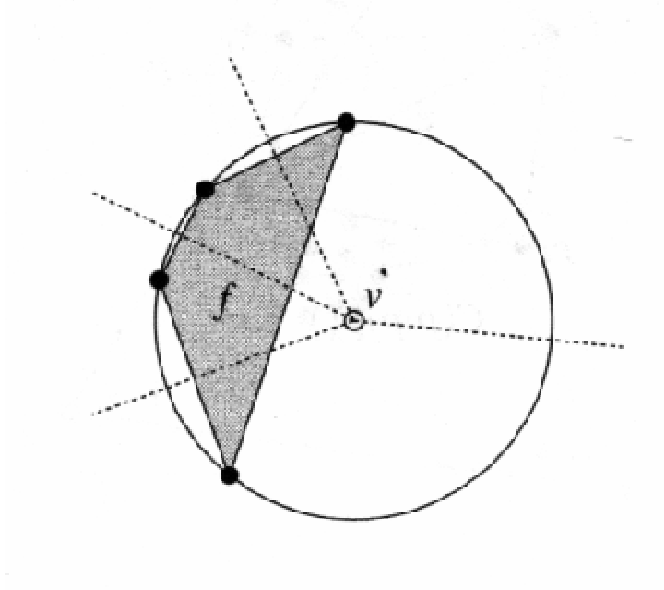
- Proved using the empty circle property of Voronoi diagrams



Delaunay Triangulations

• The edges around a face correspond to the Voronoi edges incident to the corresponding Voronoi vertex.

- Some sets of more than 3 points of Delaunay graph may lie on the same circle.
- These points form empty convex polygons, which can be triangulated.
- Delaunay Triangulation is a triangulation obtained by adding 0 or [illegible]nay Graph.



From the properties of Voronoi Diagrams…

Theorem 9.6: let p be a set of points in the plane:

- Three points $p_i, p_j, p_k \in P$ are vertices of the same face of the $DG(P)$ iff the circle through p_i, p_j, p_k contains no point of P on its interior.
- two points $p_i, p_j \in P$ from an edge of the $DG(P)$ iff there is a closet disc C that contains p_i and p_j on its boundary and does not contain any other poin of P.

Theorem 9.7

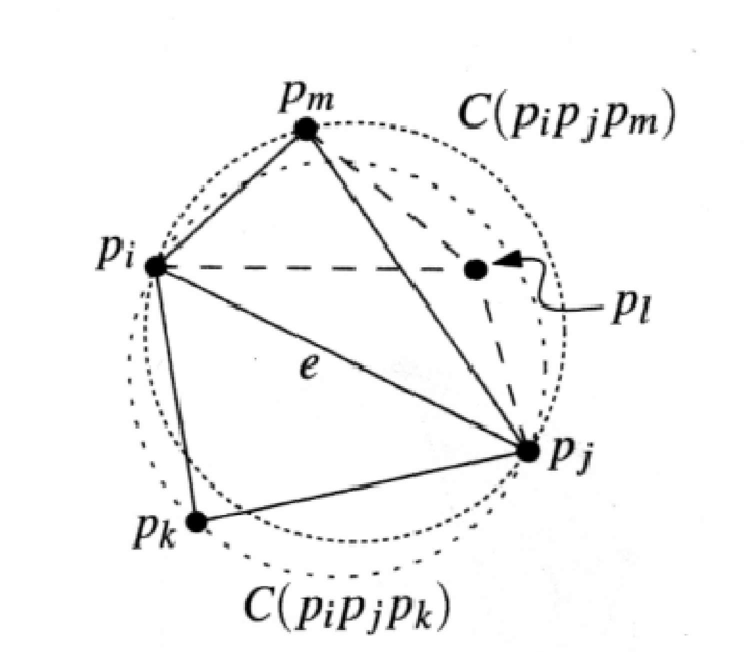
Let P be a set of points in the plane, and let T be a triangulation of p then T is a Delaunay triangulation of p iff the circumcircle of any triangle of T dos not contain a point of p in its interior.

Theorem9.8

A triangulation T of P is legal iff T is a DT(P).

•DT→Legal: Empty circle property

• Legal→DT: assume legal and not empty circle property



Theorem9.9

let p a set of points in the plane. Any angle-optimal triangulation of P is a DT of P. furthermore, any DT of P maximize the minimum angle over all triangulations of P.

What if multiple DT exist for P?

- Not all DT are angle optimal.
- By Thales Theorem, the minimum angle of each of the DT is the same.

- Thus, all the DT are equally “good” for the terrain problem. All 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 DT(P)?

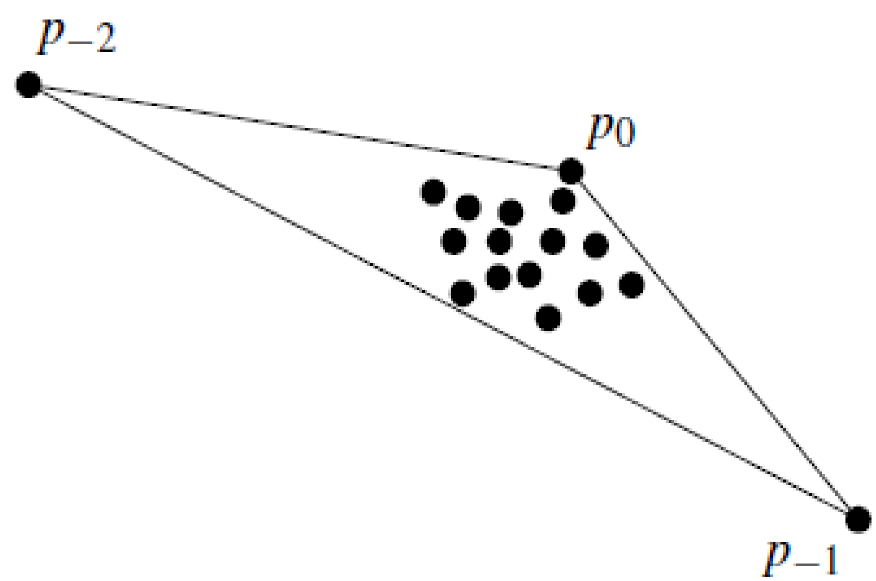
- Compute Vor(P) then dualize into DT(P).
- We could also compute DT(P) using a randomized incremental method.

randomized incremental method:

• start with a large triangle that contains the set P ($p_0p_{-1}p_{-2}$).

• p_0 is highest point of P.

• choose p_{-1} and p_{-2} far enough away(two extra point).



randomized incremental method:

•computing a DT of $P \cup \{p_0 p_{-1} p_{-2}\}$ by randomized incremental method.

•adds the points in random order and it maintains a DT of the current point set (added point is p_r).

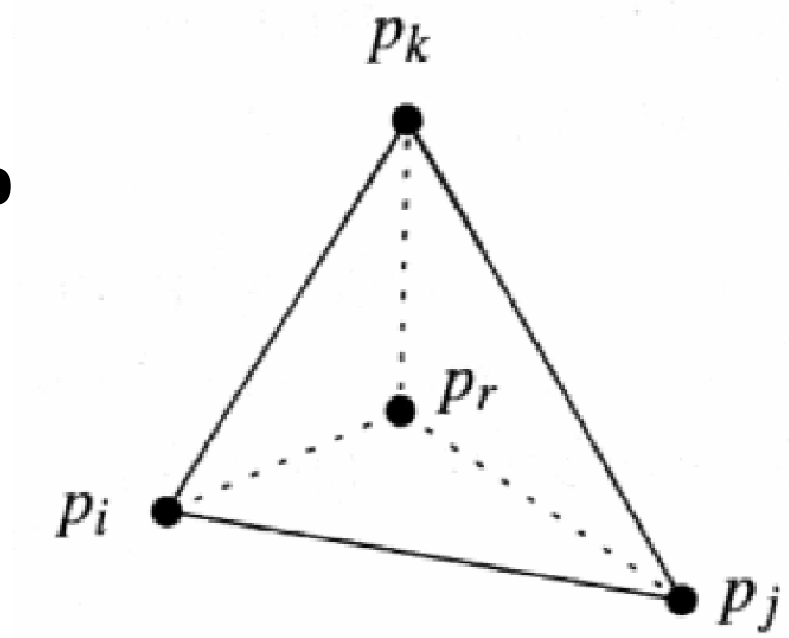
•First find the triangle of the current triangulation that contains p_r.

Triangle Subdivision: Case 1 of 2

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

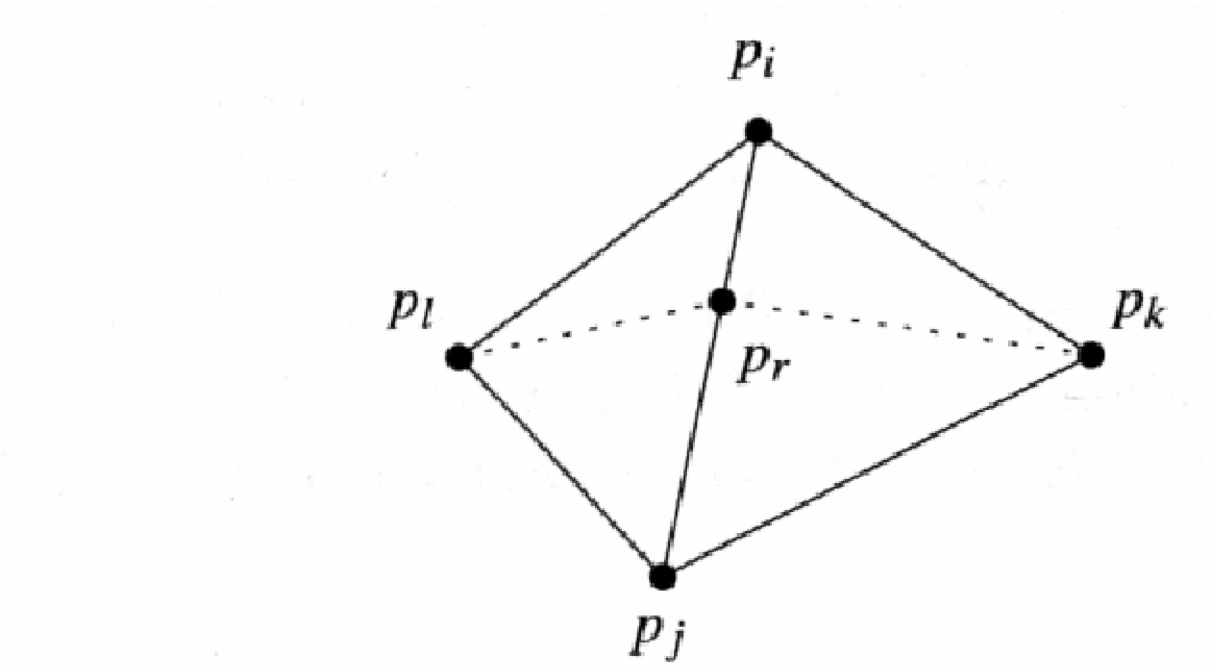
Two possible cases:

1) p_r lies in the interior o



Triangle Subdivision: Case 2 of 2

2) p_r falls on an edge between two adjacent triangles

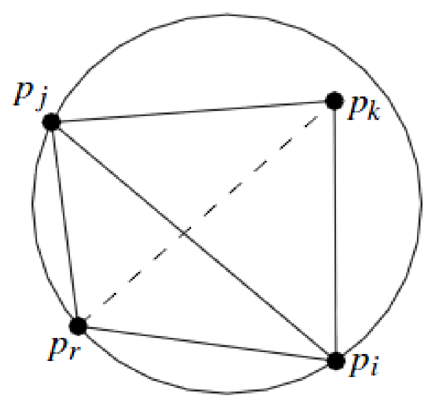


Algorithm DELAUNAYTRIANGULATION(P)
Input. A set P of $n+1$ points in the plane.
Output. A Delaunay triangulation of P.

Let p_0 be the lexicographically highest point of P, that is, the rightmost among the points with largest y-coordinate.
Let p_{-1} and p_{-2} be two points in $\mathbb{R}^2$ sufficiently far away and such that P is contained in the triangle $p_0 p_{-1} p_{-2}$.
Initialize $\mathcal{T}$ as the triangulation consisting of the single triangle $p_0 p_{-1} p_{-2}$.
Compute a random permutation $p_1, p_2, \ldots, p_n$ of $P \setminus \{p_0\}$.
for $r \leftarrow 1$ **to** n
  do ($*$ Insert p_r into $\mathcal{T}$: $*$)
    Find a triangle $p_i p_j p_k \in \mathcal{T}$ containing p_r.
    if p_r lies in the interior of the triangle $p_i p_j p_k$
      then Add edges from p_r to the three vertices of $p_i p_j p_k$, thereby splitting $p_i p_j p_k$ into three triangles.
        LEGALIZEEDGE($p_r, \overline{p_i p_j}, \mathcal{T}$)
        LEGALIZEEDGE($p_r, \overline{p_j p_k}, \mathcal{T}$)
        LEGALIZEEDGE($p_r, \overline{p_k p_i}, \mathcal{T}$)
      else ($*$ p_r lies on an edge of $p_i p_j p_k$, say the edge $\overline{p_i p_j}$ $*$)
        Add edges from p_r to p_k and to the third vertex p_l of the other triangle that is incident to $\overline{p_i p_j}$, thereby splitting the two triangles incident to $\overline{p_i p_j}$ into four triangles.
        LEGALIZEEDGE($p_r, \overline{p_i p_l}, \mathcal{T}$)
        LEGALIZEEDGE($p_r, \overline{p_l p_j}, \mathcal{T}$)
        LEGALIZEEDGE($p_r, \overline{p_j p_k}, \mathcal{T}$)
        LEGALIZEEDGE($p_r, \overline{p_k p_i}, \mathcal{T}$)
Discard p_{-1} and p_{-2} with all their incident edges from $\mathcal{T}$.
return $\mathcal{T}$

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

1. (* The point being inserted is p_r, and $\overline{p_i p_j}$ is the edge of $\mathcal{T}$ that may need to be flipped. *)
2. **if** $\overline{p_i p_j}$ is illegal
3. **then** Let $p_i p_j p_k$ be the triangle adjacent to $p_r p_i p_j$ along $\overline{p_i p_j}$.
4. (* Flip $\overline{p_i p_j}$: *) Replace $\overline{p_i p_j}$ with $\overline{p_r p_k}$.
5. LegalizeEdge($p_r, \overline{p_i p_k}, \mathcal{T}$)
6. LegalizeEdge($p_r, \overline{p_k p_j}, \mathcal{T}$)

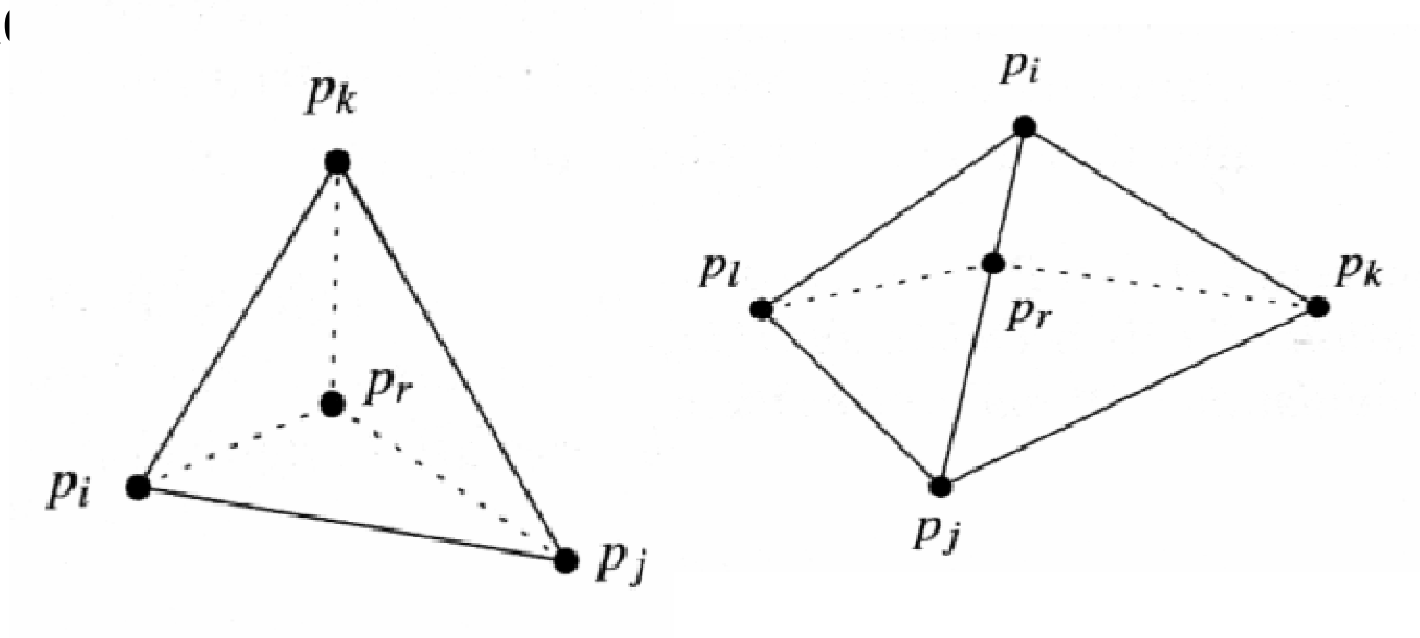


Which edges are illegal?

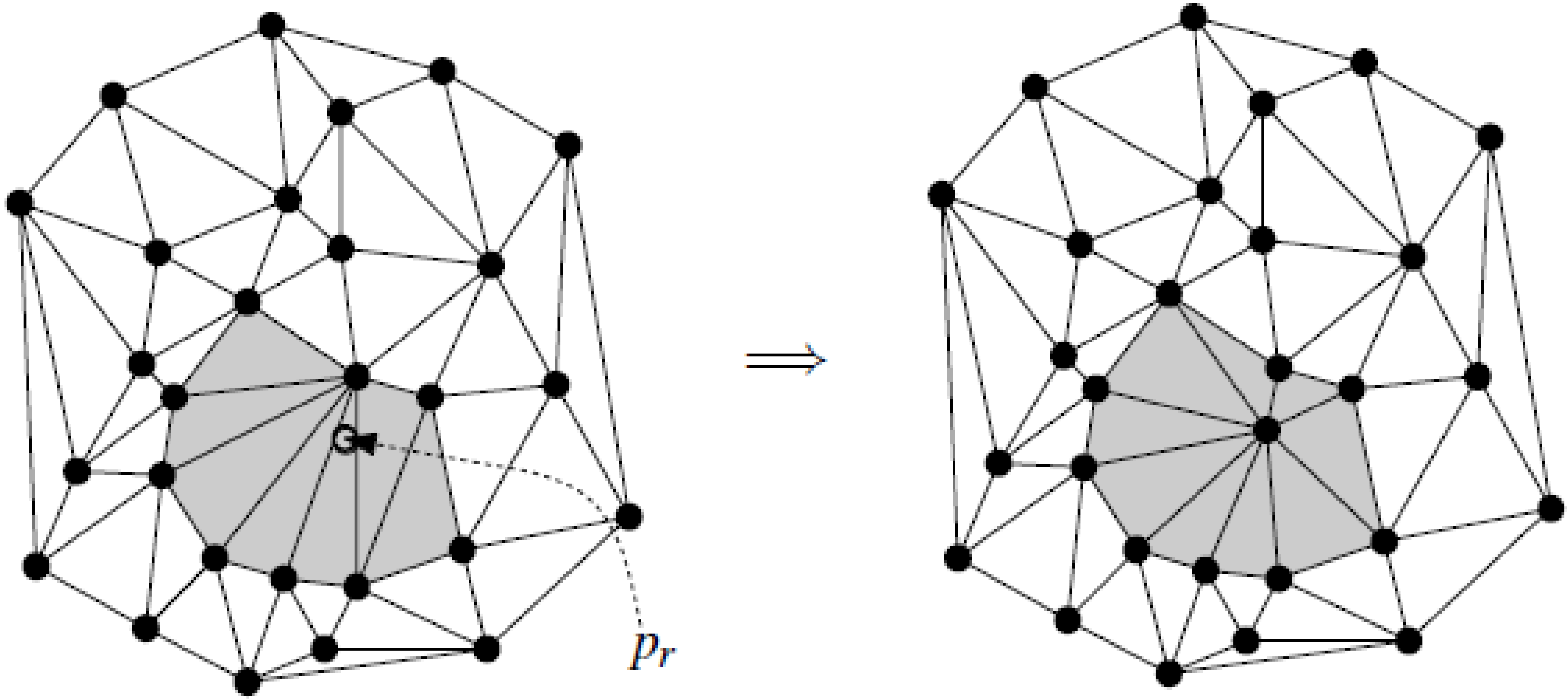
- Before it subdivided, all of edges were legal.
- After add new edges, some of the edges of T may now be illegal, but which ones?

Outer Edges May Be Illegal

- An edge can only become illegal if one of its incident triangles changed.
- Outer edges of the incident triangles $\{p_i p_j p_k\}$ or $\{p_i p_j p_k, p_i p_j p_l\}$ may have become ill



Example:



Correctness of the algorithm:

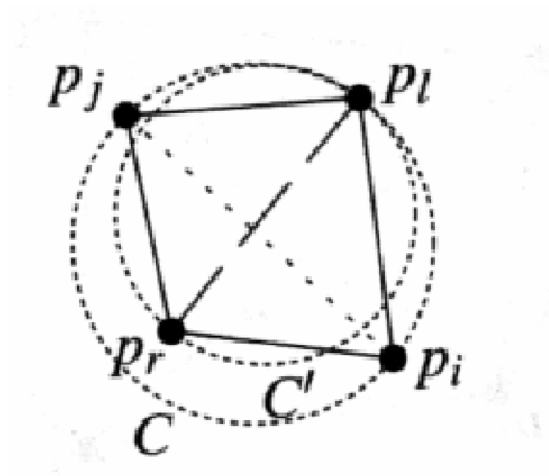
- Prove that no illegal edges remaine after all calls to LEGALIZEEDGE have been processed.

- An edge can only become illegal if one of its incident triangles changed,this proves that the algorithm tests any edge that may become illegal.

- Note that the algorithm cannot get into an infinite loop.

Lemma 9.10

Every new edge created in DelaunayTriangulation or in LegalizeEdge during the insertion of p_r is an edge of DG of $\{p\text{-}2,\ldots,p_r\}$



1)New edges incident to p_r are legal

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

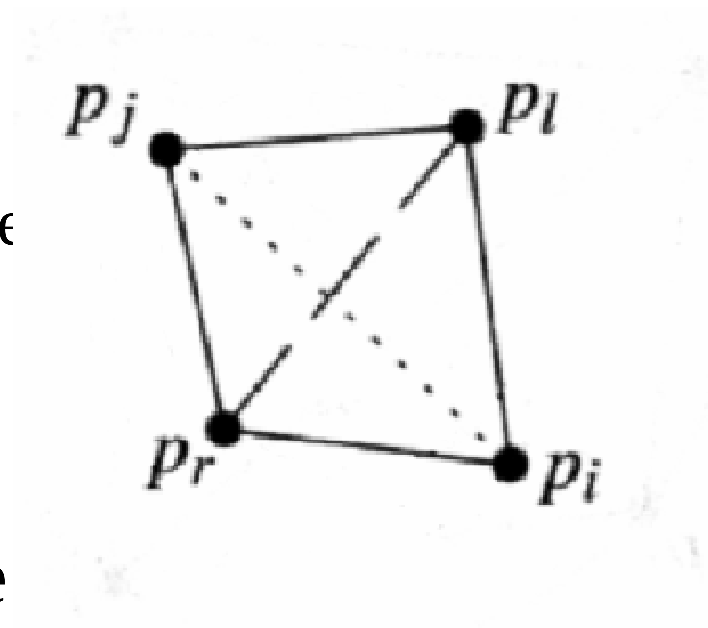
Any new edge incident p_r is legal

2)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 flippe



LEGALIZEEDGE(p_r, $p_i p_j$, T)
1. if $p_i p_j$ is illegal
2. then Let $p_i p_j p_l$ be the triangle $p_r p_i p_j$ along $p_i p_j$
3. Replace pipj with prpl
4. LEGALIZEEDGE(p_r, $p_i p_l$, T)
5. LEGALIZEEDGE(p_r, $p_l p_j$, T)

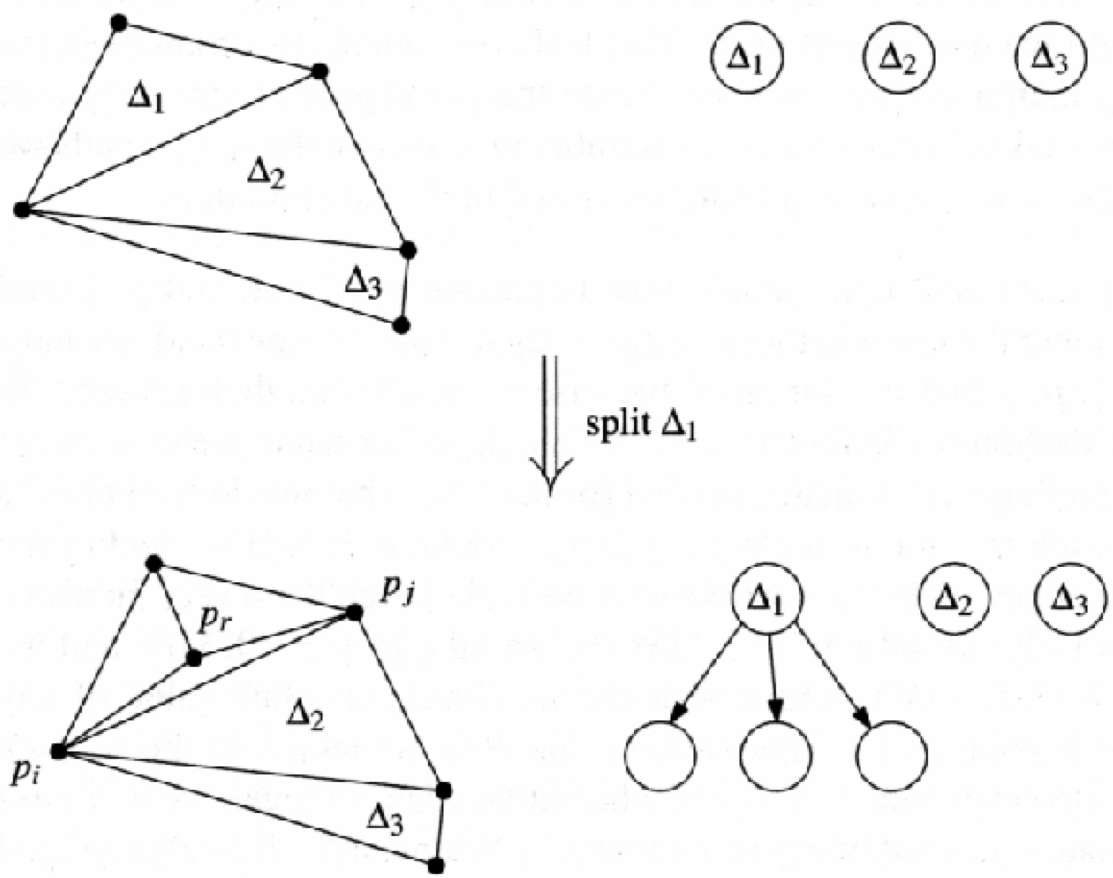
Structure of D

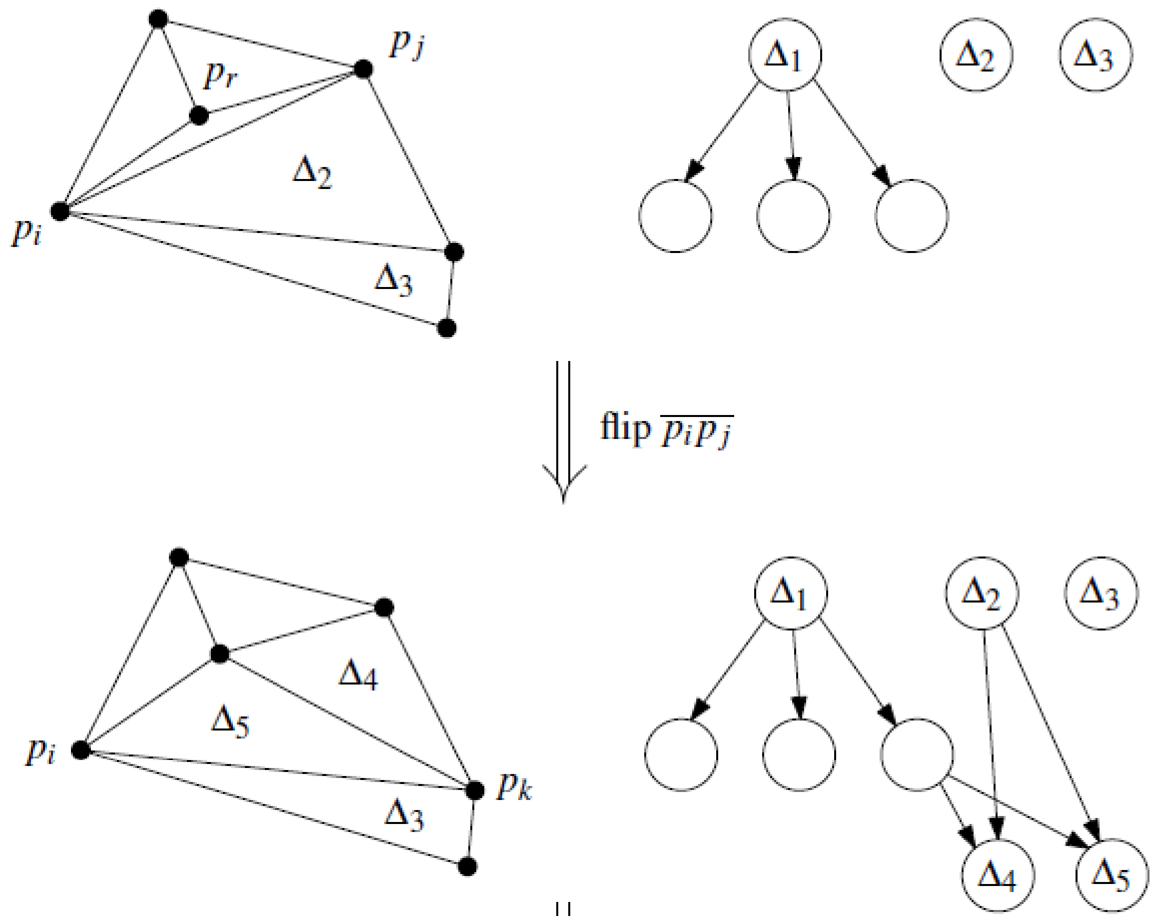
- Leaves of D correspond to the triangles of the current triangulation.
- Maintain cross pointers between leaves of D and the triangulation.
- Begin with a single leaf, the bounding triangle $p_{-1}p_{-2}p_0$

Subdivision and D

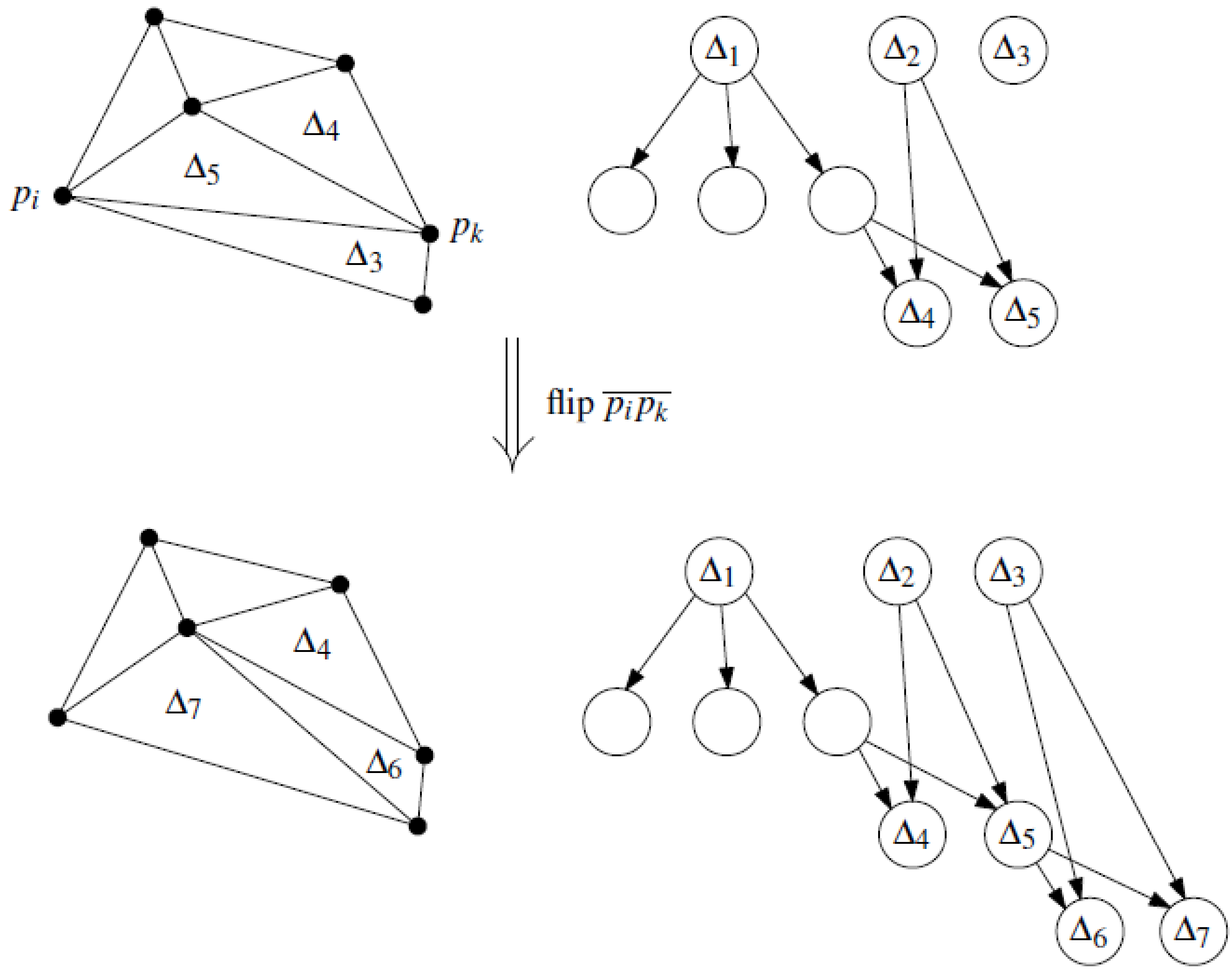
- spilt a triangle $p_i p_j p_k$ of the current triangulation into three(or tow) new triangles
- in D :
  - add three(or tow) new leaves to D
  - make the leaf for $P_i P_j P_k$ into an internal node without going pointers to those three(or tow) leaves.
- So an internal node at most gets three outgoing pointers.

example:


Δ_1
Δ_2
Δ_3
Δ_1
Δ_2
Δ_3
split Δ_1
p_j
p_r
Δ_2
p_i
Δ_3
Δ_1
Δ_2
Δ_3



p_j
p_r
Δ_2
p_i
Δ_3
Δ_1
Δ_2
Δ_3
flip $\overline{p_i p_j}$
Δ_4
Δ_5
p_i
p_k
Δ_3
Δ_1
Δ_2
Δ_3
Δ_4
Δ_5


p_i
p_k
Δ_1
Δ_2
Δ_3
Δ_4
Δ_5
flip $\overline{p_i p_k}$
Δ_6
Δ_7

Searching D

p_r = point we are searching with

1. Let the current node be the root node of 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 D

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

- p_{-1}:lies outside every circle defined by three non-collinear points of P and such that the clockwise of the points of P around p_{-1} is identical to their ordering.
- p_{-2}:lies outside every circle defined by three non-collinear points of $P \cup \{P_{-1}\}$ and such that the counterclockwise of the points of $P \cup \{P_{-1}\}$ around p_{-2} is identical to their ordering.

Bounding Triangle

Remember, we skipped step 1 of our algorithm.

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

Let $\{p_0, p_{-1}, p_{-2}\}$ be the vertices of our bounding triangle.

"B*ig enough" means that the triangle:*

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

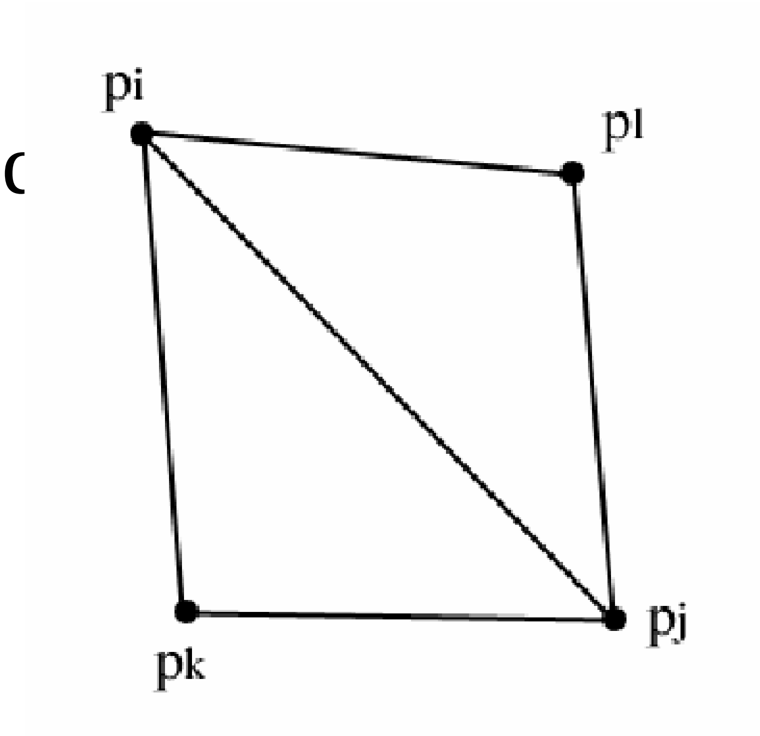
Considerations for Bounding Triangle

- We could choose large values for p_0, p_{-1}, p_{-2} ,but that would require potentially huge coordinates.

- Instead, we'll modify our test for illegal edges, to act as if we chose large values for bounding triangle.

Modified Illegal Edge Test

$p_i p_j$ is the edge being tested
p_k and p_l are the other two vertices of the triangles incident to $p_i p_j$



Our illegal edge test falls into one of 3 cases.

Illegal Edge Test, Case 1

Case 1) Indices i and j are both negative or zero

•$p_i p_j$ is an edge of the bounding triangle.

•$p_i p_j$ is legal, want to preserve edges of bounding Triangle.

Illegal Edge Test, Case 2:

Case 2) Indices i, j, k, and l are all non negative.

- This is the normal case.
- pipj is illegal iff p_l lies inside the circumcircle of pi,pj,pk.

Illegal Edge Test, Case 3:

In this case $p_i p_j$ is legal iff min(k,l)<min(i,j):

1)Exactly one of i, j, k, l is negative:this points outside the circle defined by the other three poins and method is correct.

2)min(i,j) and min(k,l) are negative:p_{-2} lies outside any circle defined by tree points in $P \cup \{P_{-1}\}$ implies that the method is correct.

Analysis Goals

- Expected running time of algorithm is: O(n log n)
- Expected storage required is: O(n)

First, some notation…

- Pr = {p1, p2, …, pr}
  - Points added by iteration r
- $\{p_0, p_{-1}, p_{-2}\}$
  - Vertices of bounding triangle
- D*Gr = DG(*$P_r \cup \{P_{-2}.P_{-1}.P_0\}$*)*
  - Delaunay graph as of iteration r

Lemma9.11:

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

Expected Number of Triangles

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

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.

Expected Number of Triangles

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

Expected Degree of pr

Use backward analysis:

•Fix Pr, let pr be a random element of Pr

•D*Gr has 3(r+3)-6 edges*

• *Total degree of Pr   2[3(r+3)-9] = 6r*

E[degree of random element of Pr]= 6

Triangles created at step r

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

deg(pr, DGr) = degree of pr in DGr

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

Expected Number of Triangles

Now we can bound the number of triangles:
1 initial Triangle + 9n created Triangle

Expected number of triangles created is 9n+1.

Theorem 9.12:

The DT of a set p of n points in the plane can be computed in o(n logn) expected time,using o(n) expectd storage.

Storage Requirement:

- 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 pr from P.

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

3. Find the triangle that pr 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 D we visit)
  + O(1) for current triangle
- Number of nodes of 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(Δ), charge to Δ.
- Since we are iterating through P, each point in K(Δ) can be charged at most once.

Total time for point location:

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

Point Location Step

If p is a point set in general position, then:

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

Point Location Step

Introduce some notation…

T_r = set of triangles of DG(P_r)

$T_r \setminus T_{r-1}$ triangles created in stage r

Rewrite our sum as:

$$\sum_{r=1}^{n} \left(\sum_{\Delta \in \mathcal{T}_r \setminus \mathcal{T}_{r-1}} \mathrm{card}(K(\Delta)) \right).$$

Point Location Step

More notation…

$k(P_r, q)$ = *number of triangles* $\Delta \in T_r$ *such that* q is contained in Δ

$k(P_r, q, p_r)$ = *number of triangles* $\Delta \in T_r$ *such that* q *is contained in* Δ *and* p_r *is incident to* Δ

Rewrite our sum as:

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

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 $3/r$

Thus:

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

Point Location Step

Using:

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

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

Now find $E[k(P_r, p_{r+1})]$…

- Any of the remaining n-r points is equally likely to appear as p_{r+1}

So:

$$\mathbf{E}\left[k(P_r, p_{r+1})\right] = \frac{1}{n-r} \sum_{q \in P \setminus P_r} k(P_r, q).$$

Point Location Step

Using:

$$\mathrm{E}\big[k(P_r, p_{r+1})\big] = \frac{1}{n-r} \sum_{q \in P \setminus P_r} k(P_r, q).$$

We can rewrite our sum as:

$$\mathrm{E}\Big[\sum_{\Delta \in \mathcal{T}_r \setminus \mathcal{T}_{r-1}} \mathrm{card}(K(\Delta))\Big] \leqslant 3\left(\frac{n-r}{r}\right) \mathrm{E}\big[k(P_r, p_{r+1})\big].$$

Point Location Step

Find $k(P_r, p_{r+1})$

- number of triangles of T_r that contain p_{r+1}
- these are the triangles that will be destroyed when p_{r+1} is inserted; $T_r \setminus T_{r+1}$
- Rewrite our sum as:

$$E\Big[\sum_{\Delta \in \mathcal{T}_r \setminus \mathcal{T}_{r-1}} \operatorname{card}(K(\Delta))\Big] \leqslant 3\left(\frac{n-r}{r}\right) E\big[\operatorname{card}(\mathcal{T}_r \setminus \mathcal{T}_{r+1})\big].$$

Point Location Step

Remember, number of triangles in triangulation of n points with k points on convex hull is 2n-2-k

- T_m has 2(m+3)-2-3=2m+1
- T_{m+1} has two more triangles than T_m

Thus, card($T_r \setminus T_{r+1}$)

= card(triangles destroyed by p_r)

= card(triangles created by pr) – 2

= card($T_{r+1} \setminus T_r$) - 2

We can rewrite our sum as:

$$\mathrm{E}\Big[\sum_{\Delta \in \mathcal{T}_r \setminus \mathcal{T}_{r-1}} \mathrm{card}(K(\Delta))\Big] \leqslant 3\left(\frac{n-r}{r}\right)\Big(\mathrm{E}\left[\mathrm{card}(\mathcal{T}_{r+1} \setminus \mathcal{T}_r)\right] - 2\Big).$$

Point Location Step

Remember we fixed P_r earlier…

- Consider all Pr by averaging over both sides of the inequality, but the inequality comes out identical.

E[*number of triangles created by* p_r]
  = E[*number of edges incident to* p_{r+1} *in* T_{r+1}]
  = 6

Therefore:

$$\mathbf{E}\Big[\sum_{\Delta \in \mathcal{T}_r \setminus \mathcal{T}_{r-1}} \operatorname{card}(K(\Delta))\Big] \leqslant 12\left(\frac{n-r}{r}\right).$$

Analysis Complete

$$\mathrm{E}\Big[\sum_{\Delta\in\mathcal{T}_r\setminus\mathcal{T}_{r-1}} \mathrm{card}(K(\Delta))\Big] \leqslant 12\left(\frac{n-r}{r}\right).$$

If we sum this over all r, we have shown that:

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

And thus, the algorithm runs in O(n log n) time.