Delaunay Triangulation

Sunghee Choi

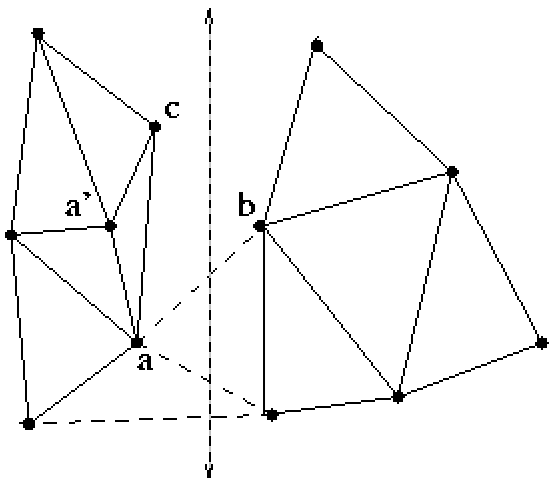
Algorithms

Voronoi and Delaunay : can compute one from the other in O(n) time. (Delaunay is simpler to compute.)

- edge flipping
- divide-and-conquer
  - O(n log n) worst case
  - split the points in half by a line
  - compute Delaunay triangulation of each piece
  - merge
- sweep
  - O(n log n) worst case
  - use sweep line paradigm in some way
- randomized incremental
  - O(n log n) expected

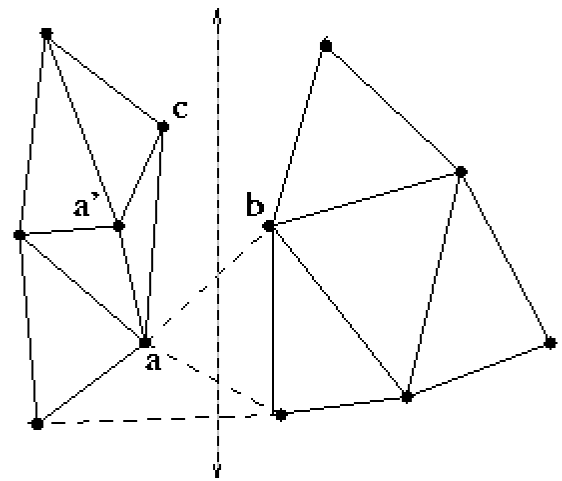
Divide and conquer

- The hard step is the merge: Given sets L and R, S= L∪R, separated by a vertical line, and their triangulations DT(L) and DT(R), compute DT(S).
- There are RR edges, LL edges, and cross edges.
- All new Delaunay edges in the merge are cross edges: if an RR edge is Delaunay now, it was before.
- Some triangles of DT(L) are deleted: those with circumcircles containing points of R.
  ...and symmetrically for DT(R).
- The cross edges are ordered along the split line, and consecutive edges share a vertex.



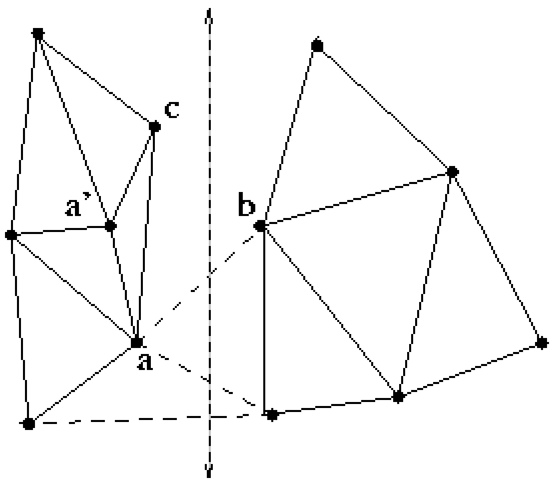
Divide and conquer

- The cross edges are built from bottom to top, starting from a cross edge that is a convex hull edge.
- Conceptually, maintain an empty disk on the current cross edge {a,b}, pushing it up, but keeping a and b on the bounding circle.
- The first site this "rising bubble" hits gives the next cross edge.
- That site is on an edge incident to a or b.



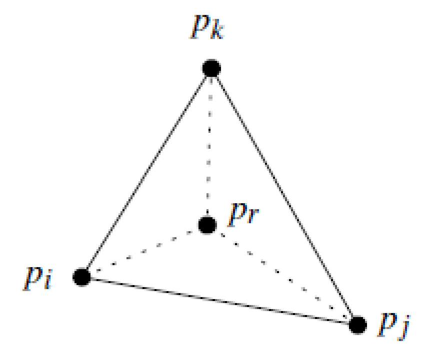
Divide and conquer

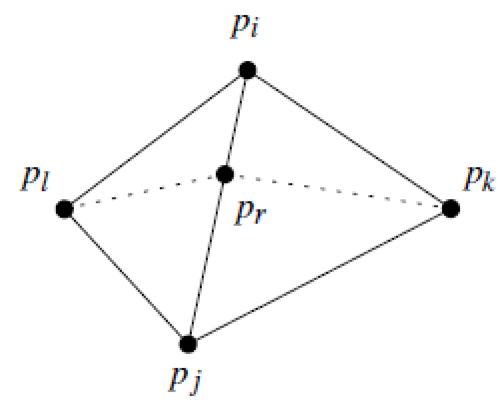
- To find it, walk around the edges incident to a, deleting those bounding triangles conflicting with b. Find the first edge {a,a'} not deleted.
- Similarly, walk around the edges incident to b, finding edge {b,b'}.
- Either b or b' is hit first by the "rising bubble", yielding the next cross edge {a,b'} or {a',b}.
- Since O(n) work is done in the merge, O(n log n) is needed overall.



Randomized incremental algorithm

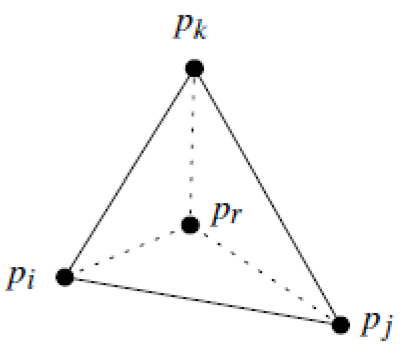
- Start with a large triangle that contains the set P. (far away so that they do not destroy any triangle in DT(P)).
- Later discard the initial triangle with their incident edges.
- Add points in random order and maintain DT.
- To insert a new point p_r,
  - Find the triangle containing p_r in current DT.
  - Add edges from p_r to the vertices of this triangle
  - If p_r lies on edge e of the triangle, add edges from p_r to the opposite vertices in triangles sharing e.

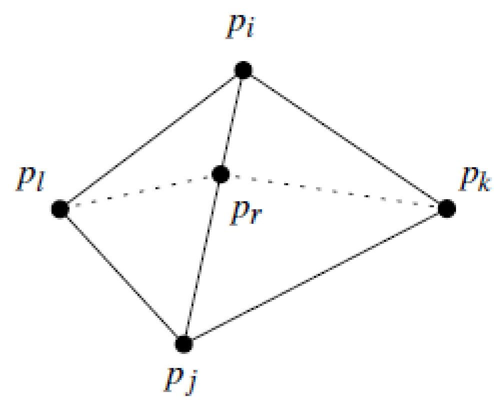




Randomized incremental algorithm

- Legalize edges that may need to be flipped.
  - LEGALIZE(p_r, p_i p_j) ,LEGALIZE(p_r, p_j p_k), LEGALIZE(p_r, p_k p_i)
  - LEGALIZE(p_r, p_i p_l) , LEGALIZE(p_r, p_l p_j) , LEGALIZE(p_r, p_j p_k) , LEGALIZE(p_r, p_k p_i)





Randomized incremental algorithm

- LEGALIZE(p_r, p_i p_j)

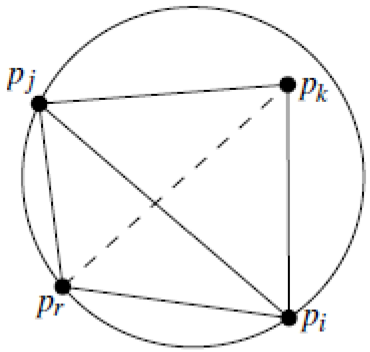
  If p_i p_j is illegal

  then let p_i p_j p_k be the triangle adjacent $p_r p_i$ p_j along p_i p_j

  Flip p_i p_j

  LEGALIZE(p_r, $p_i p_k$)

  LEGALIZE(p_r, $p_k p_j$)



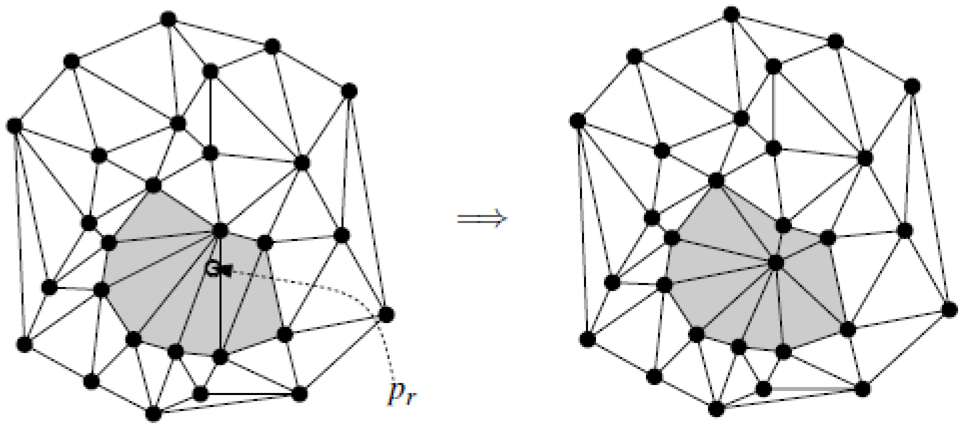
Incircle test

d lies inside triangle abc if and only if

$$\mathrm{inCircle}(a,b,c,d) \;=\; \det\begin{pmatrix} a_x & a_y & a_x^2 + a_y^2 & 1 \\ b_x & b_y & b_x^2 + b_y^2 & 1 \\ c_x & c_y & c_x^2 + c_y^2 & 1 \\ d_x & d_y & d_x^2 + d_y^2 & 1 \end{pmatrix} > 0.$$

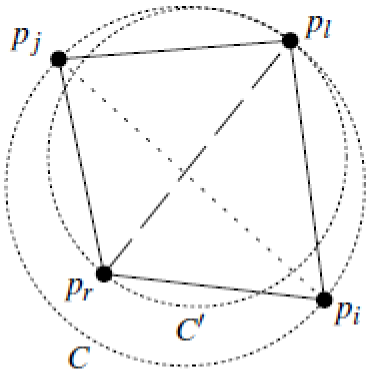
Correctness

- Need to prove that no illegal edges remain after all calls to LEGALIZE have been processed.
- Every new edge created due to the insertion of p_r is incident to p_r
- Every new edge is legal.
- An edge can only become illegal if one of its incident triangles changes.



Correctness

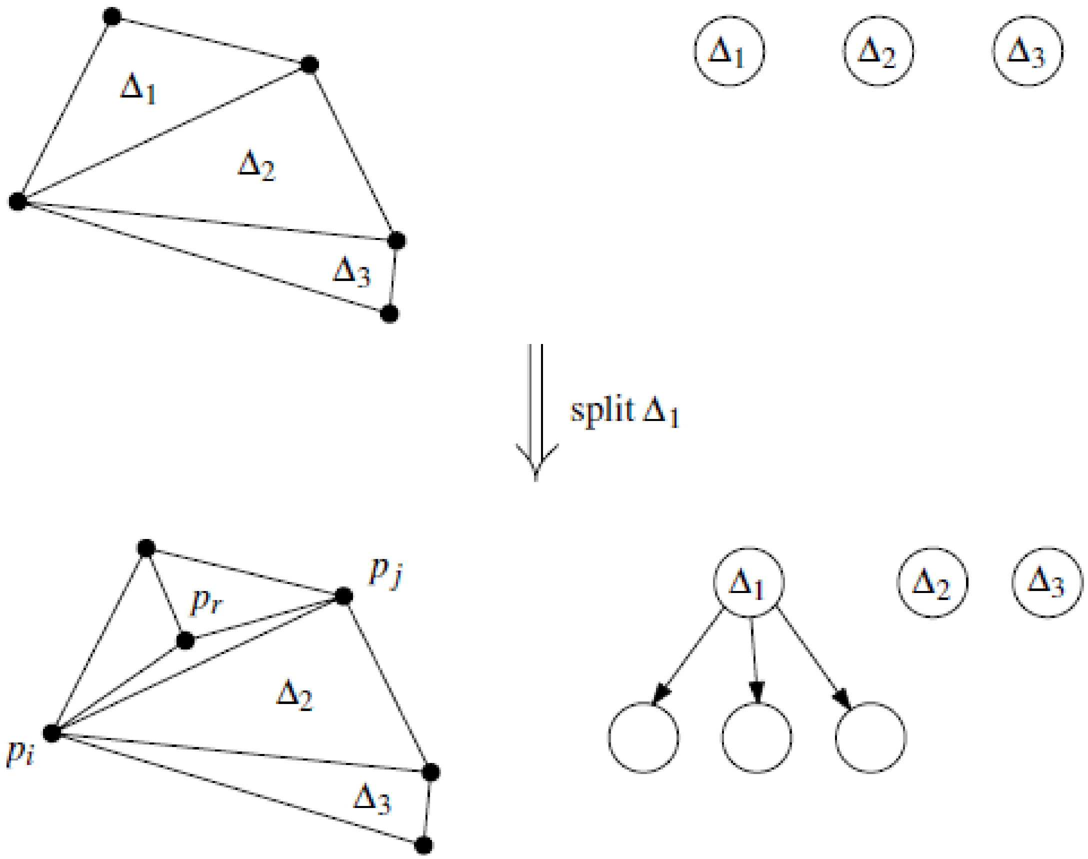
- Consider the first edges $p_r p_i$, $p_r p_j$, $p_r p_k$ (and perhaps $p_r p_l$) created.
- Shrink the circumcircle of p_i p_j p_k so that it passes through p_r and p_i. It is empty. Thus, $p_r p_i$ is Delaunay edge. (Similarly, for $p_r p_j$ and $p_r p_k$ (and for $p_r p_l$, if it exists.))
- Consider an edge flipped by LEGALIZE. Such an edge flip replaces an edge $p_i p_j$ of a triangle $p_i p_j p_l$ by an edge $p_r p_l$ incident to p_r. Since p_i p_j p_l was Delaunay triangle before the addition of p_r and its circumcircle C contains p_r, we can shrink C to obtain an empty circle C' with only $p_i p_j p_l$ on its boundary. Hence, $p_r p_l$ is Delaunay edge.



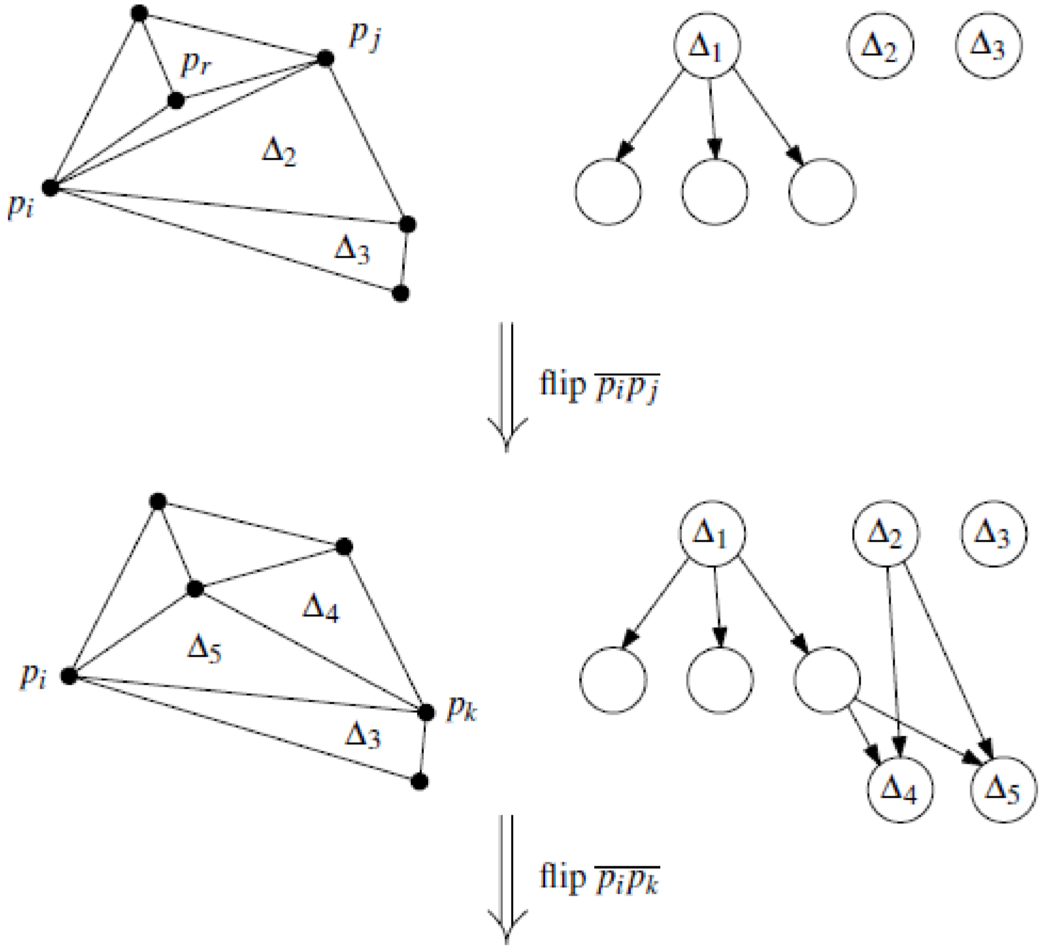
Point location

- How to find the triangle containing the point p_r?
- Use a similar idea to point location (trapezoidal map.)
- While we build DT, we also build a point location data structure D.
- D : directed acyclic graph
- Leaves of D : triangles of current DT.
- Internal nodes of D : triangles that were created but have been destroyed.
- Start with a single leaf node corresponding to the initial triangle.

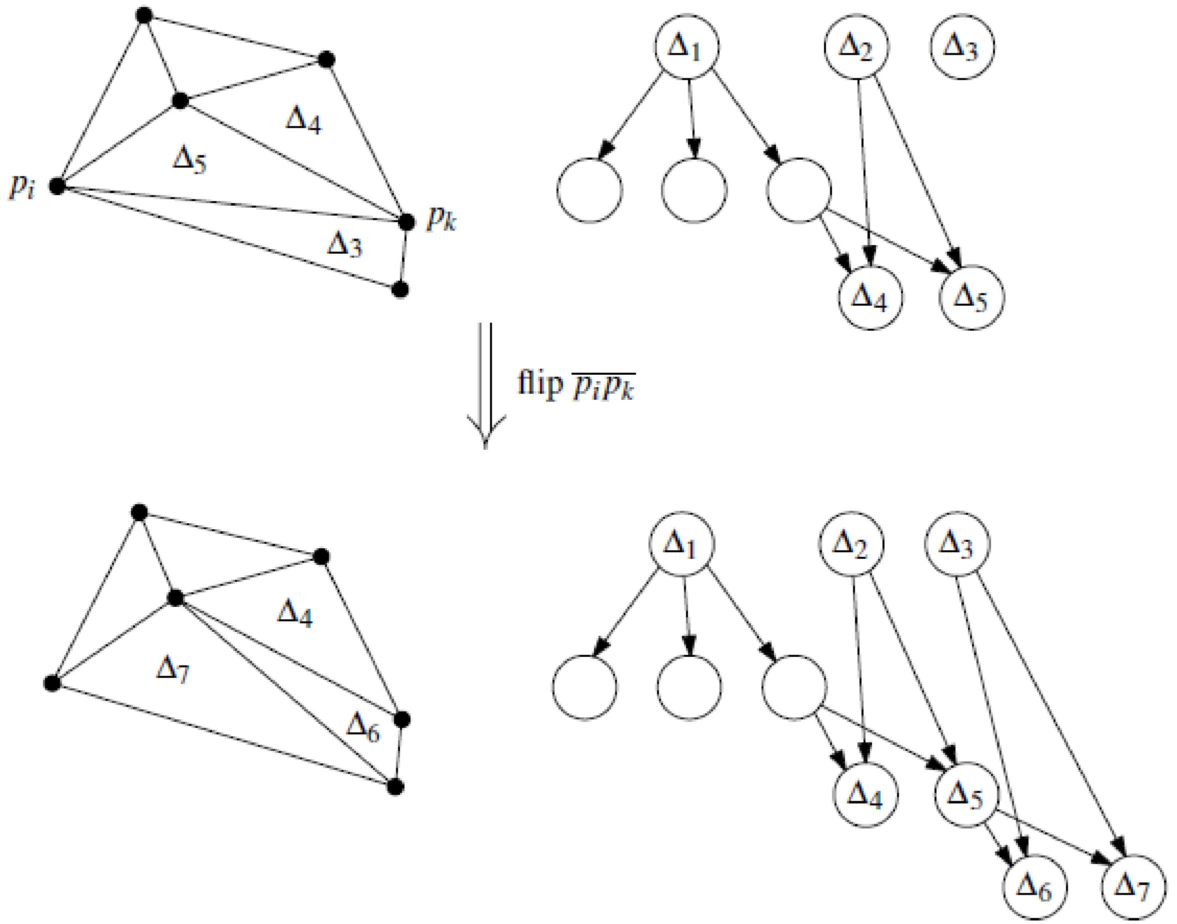
Point Location Data Structure



Point Location Data Structure



Point Location Data Structure



Point location

- Starting with the root node follow the links to the triangle containing p_r to find the leaf corresponding to the triangle in current triangulation that contains p_r.
- The out-degree of any node is at most 3.
- The point location takes linear time in the number of nodes on the search path (= number of triangles in D that contain p_r).

Analysis

- Structural changes generated by the algorithm (= number of triangles created during the course of the algorithm)?
- Lemma : The expected number of triangles created by the algorithm is at most 9n+1.
- Pf) When we insert p_r, we split 1 or 2 triangles, creating 3 or 4 new triangles, and 3 or 4 new edges. For every edge that we flip in LEGALIZE, we create two new triangles, creating edges incident to p_r.
- If after the insertion of p_r, there are k edges in DT incident to p_r, then we have created at most 2(k-3)+3 = 2k-3 new triangles.

Analysis

- If after the insertion of p_r, there are k edges in DT incident to p_r, then we have created at most 2(k-3)+3 = 2k-3 new triangles.
- What is the expected degree of p_r?
- Use backwards analysis!
- Consider the situation after insertion of p_r.
- DT has at most 3(r+3)-3-3 edges.
- 3 edges are edges of initial bounding triangle.
- Total degree of vertices is at most 2(3(r+3)-6-3)=6r.
- Expected degree of vertices is at most 6.
- E[number of triangles created by insertion of p_r] $\leq$ E[2k-3] =2E[k]-3 = 9
- Expected total number of created triangles is 1(initial triangle)+9n.

Analysis

- Theorem : DT of n points in plane can be computed in O(n log n) expected time and O(n) expected storage.
- Pf) Space : point location data structure D : every node corresponds to a triangle created by the algorithm. O(n) expected.
- Time : except for the time for point location, time spent is proportional to the number of created triangles = O(n) expected.
- Time for point location = O(number of triangles that contain p_r that were destroyed + 1(current Delaunay triangle containing p_r)).

Analysis

- A triangle $p_i\, p_j\, p_k$ can be destroyed
  - When a new point p_l has been inserted inside (or on the boundary of) $p_i\, p_j\, p_k$
  - An edge flip has replaced $p_i\, p_j\, p_k$ and adjacent triangle $p_i\, p_j\, p_l$. (Either $p_i\, p_j\, p_k$ was Delaunay triangle before p_l was inserted or $p_i\, p_j\, p_l$ was Delaunay triangle before p_k was inserted.)
- In all cases, we can charge the fact that the triangle $p_i\, p_j\, p_k$ was visited to a Delaunay triangle Δ that has been detroyed in the same stage as $p_i\, p_j\, p_k$ and such that the circumcircle of Δ contains p_r
- $K(\Delta)$: subset of points in P that lie in the circumcircle of Δ.
- The visit to a triangle during the location of p_r is charged to a triangle Δ with $p_r \in K(\Delta)$.
- A triangle Δ can be charged at most once for every one of the points in $K(\Delta)$.
- Therefore, total time for point location is $O(n + \Sigma\ card(K(\Delta)))$, where the summation is over all Delaunay triangles Δ created by the algorithm.
- Lemma 9.13 proves that $E[\Sigma\ card(K(\Delta))] = O(n \log n)$.