Splitting a Delaunay Triangulation in Linear Time

Bernard Chazelle, Olivier Devillers, Ferran Hurtado, Mercè Mora, Vera Sacristan, Monique Teillaud

published in Algorithmica 34, 2002, p.39–46.

Splitting a Delaunay Triangulation in Linear Time*

Bernard Chazelle† Olivier Devillers‡ Ferran Hurtado§
Mercè Mora§ Vera Sacristán§ Monique Teillaud‡

1 Introduction

Computing the Delaunay triangulation of n points is well known to have an $\Omega(n \log n)$ lower bound. Researchers have attempted to break that bound in special cases where additional information is known.

The Delaunay triangulation of the vertices of a convex polygon is such a case where the lower bound of $\Omega(n \log n)$ does not hold. This problem has been solved in linear time with a deterministic algorithm of Agarwal, Guibas, Saxe and Shor [1]. Chew has also proposed a very simple randomized algorithm [8] for the same problem, which we sketch in Section 2.2. These two algorithms can also compute the skeleton of a convex polygon in linear time and support the deletion of a point from a Delaunay triangulation in time linear in its degree.

Another result is that if a spanning subgraph of maximal degree d of the Delaunay triangulation is known, then the remaining part of the Delaunay triangulation can be computed in $O(nd \log^\star n)$ expected randomized time

* *The French team was partially supported by Picasso French-Spanish collaboration program. The Spanish team was partially supported by CUR Gen. Cat. 1999SGR00356, Proyecto DGES-MEC PB98-0933 and Accin Integrada Francia-España HF99-112.*

†Princeton University, Computer Science Department, 35 Olden Street,Princeton, NJ 08544,USA. `http://ftp.cs.princeton.edu/~chazelle/`, chazelle@cs.princeton.edu

‡INRIA, BP93, 06902 Sophia-Antipolis, France. `www-sop.inria.fr/prisme/`, {Olivier.Devillers,Monique.Teillaud}@sophia.inria.fr

§Dept. Matemàtica Aplicada II, Univ. Politècnica de Catalunya, Pau Gargallo, 5, 08028 Barcelona, Spain. {hurtado,mora,vera}@ma2.upc.es, `www-ma2.upc.es/~geomc/`.

[14]. The Euclidean minimum spanning tree is an example of such a graph of bounded degree 6. This $O(n \log^\star n)$ result applies also if the points are the vertices of a chain monotone in both x and y directions but, in this special case, linear complexity has been achieved by Djidjev and Lingas [15] generalizing the result of Agarwal et al. for convex polygons.

Beside these results, where knowing some information on the points helps to construct the Delaunay triangulation, it has been proven that knowing the order of the points along any one given direction does not help [15].

Breaking a lower bound by using some additional information arises similarly in some other problems. One of the most famous is the triangulation of a simple polygon in linear time [6, 18, 2]. Other related problems are the constrained Delaunay triangulation of a simple polygon in $O(n)$ time [17]; the medial axis of a simple polygon in linear time [10]; the computation of one cell in the intersection of two polygons in $O\left(n(\log^\star n)^2\right)$ time [12]; the L^∞ Delaunay triangulation of points sorted along x and y axes in $O(n \log\log n)$ time [9]. Also, given the 3D convex hull of a set of blue points and the convex hull of the set of red points, the convex hull of all points can be computed in linear time [7].

The problem we address in this paper is the following: given the Delaunay triangulation $\mathcal{DT}(S)$ of a point set S in E^2, a partition of S into S_1, S_2, can we compute both $\mathcal{DT}(S_i)$ in $o(n \log n)$ time?

The reverse problem, given a partition of S into S_1, S_2, reconstruct $\mathcal{DT}(S)$ from $\mathcal{DT}(S_i)$, can be solved in linear time [7]. Indeed, the 3D convex hull of the vertices of two convex polyhedra can be computed in linear time [7] and, by standard transformation of the Delaunay triangulation to the convex hull, we get the result. This reverse operation can be used as the merging step of a divide and conquer algorithm.

In this paper, we propose an $O(n)$ randomized algorithm in the spirit of Chew's algorithm for the Delaunay triangulation of a convex polygon.

In some applications, we need to simplify a triangulation by removing several vertices at the same time (see for example [20]) this is usually done by choosing an independent set of small degree vertices to ensure good complexity. This paper allows us to relax that constraint and to have more flexibility to choose the vertices to remove according to the need of the application.

2 Preliminaries

We assume in the sequel that a triangulation allows constant time access from a triangle to its three neighbors and to its three vertices, and from a vertex to one incident triangle. This is provided by any reasonable representation of a triangulation, either based on triangles [4] or as in the *DCEL* or winged-edge structure [13, pp. 31-33].

2.1 Classical randomized incremental constructions

Randomized incremental constructions have been widely used for geometric problems [11, 3] and specifically for the Delaunay triangulation [5, 16, 14]. These algorithms insert the points one by one in a random order in some data structure to locate the new point and update the triangulation. The location step has an $O(\log n)$ expected complexity. The update step has constant expected complexity as can be easily proved by backwards analysis [19]. Indeed, the update cost of inserting the last point in the triangulation is its degree in the final triangulation. Since the last point is chosen randomly, its insertion cost is the average degree of a planar graph, which is less than 6.

2.2 Chew's algorithm for the Delaunay triangulation of a convex polygon

Chew's algorithm [8] for the Delaunay triangulation of a convex polygon uses the ideas above for the analysis of the insertion of the last point. The main idea is to avoid the location cost using the additional information of the convex polygon.

As noticed earlier, for any vertex v we know one of its incident triangles. In the case of Chew's algorithm, it is required that the triangle in question be incident to the convex hull edge following v in counterclockwise order.

The algorithm can be stated as follows:

1. Choose a random vertex p of the polygon $\mathcal{P}$.
2. Store the point q before p on the convex hull.
3. Compute the convex polygon $\mathcal{P} \setminus \{p\}$.

4. Compute recursively $\mathcal{DT}(S \setminus \{p\})$.

5. Let t be the triangle pointed to by q.

6. Create a triangle neighbor of t with p as vertex, flip diagonals if necessary using the standard Delaunay criterion and update links from vertices to incident triangles.

By standard backwards analysis, the flipping step has expected constant cost. Other operations, except the recursive call, require constant time. Thus we get a linear expected complexity.

The important thing is that we avoid the location step. Thus Chew's algorithm applies to other cases where the location step can be avoided, e.g. deletion of a point in a Delaunay triangulation.

3 Algorithm

3.1 General scheme

The main idea is similar to Chew's algorithm, that is to delete a random point $p \in S_i$ from $\mathcal{DT}(S)$, to split the triangulation and then to insert p in the triangulation $\mathcal{DT}(S_i \setminus \{p\})$ avoiding the usual location step. The location of p can be done by computing the nearest neighbor of p in S_i, which can be done in time $T(p) \log T(p)$ for some number $T(p)$ depending on p, whose expectation is $O(1)$. However, it is possible for example to have one point p, chosen with probability $\frac{1}{n}$ such that $T(p) = n$, which brings the expected cost to $E\left(T(p) \log T(p)\right) = \Omega(\log n)$. The idea is to choose two points p_α, p_β and to take for p the better of the two, in order to concentrate the distribution around its mean. Here is the algorithm:

Given $\mathcal{DT}(S)$,

1. Choose two random points $p_\alpha, p_\beta \in S$. Let $i, j \in \{1, 2\}$ such that $p_\alpha \in S_i$ and $p_\beta \in S_j$ (i and j do not need to be different).

2. Look simultaneously for the nearest neighbor of p_α in S_i and the nearest neighbor of p_β in S_j. As soon as one of the two is found, say the neighbor q of p_α in S_i, stop all searching and let p be p_α.

3. Remove p from $\mathcal{DT}(S)$ to get $\mathcal{DT}(S \setminus \{p\})$.

4. Recursively compute $\mathcal{DT}(S_1 \setminus \{p\})$ and $\mathcal{DT}(S_2 \setminus \{p\})$ from $\mathcal{DT}(S \setminus \{p\})$.

5. Determine the triangle of $\mathcal{DT}(S_i \setminus \{p\})$ incident to q that is traversed by the segment pq.

6. Apply the usual Delaunay flip procedure to obtain $\mathcal{DT}(S_i)$ from $\mathcal{DT}(S_i \setminus \{p\})$.

3.2 Combination lemmas

Note that in the algorithm, p is not a random point uniformly distributed among S, but one chosen among two random points. In this section, we investigate how this choice influences the mean value of some variable depending on p.

Let $X(p)$ be a positive random variable depending on a point p chosen uniformly at random among n points. $X(p)$ is bounded by n and $E(X)$ is the expected value of X. Y is an independent, identically distributed copy of X.

Lemma 1

$$E(\max\{X, Y\}) \leq 2 \cdot E(X).$$

Proof: This is a direct consequence of:

$$\begin{array}{rcll} E(\min\{X, Y\}) & \geq & 0 & \text{since } X \text{ and } Y \text{ are positive.} \\ E(\max\{X, Y\} + \min\{X, Y\}) & = & E(X + Y) = E(X) + E(Y) = 2 \cdot E(X) & \end{array}$$

□

Lemma 2 *If f is a non negative concave non decreasing function,*

$$E(\min\{X, Y\} \cdot f(\min\{X, Y\})) \leq E(X) \cdot f(E(X)).$$

Proof:

$$\begin{array}{rcll} & & 2 \cdot E(\min\{X, Y\} \cdot f(\min\{X, Y\})) & \\ & \leq & E(\min\{X, Y\} \cdot f(\max\{X, Y\}) + \max\{X, Y\} \cdot f(\min\{X, Y\})) & \\ & = & E(X \cdot f(Y) + Y \cdot f(X)) & \\ & = & E(X) \cdot E(f(Y)) + E(Y) \cdot E(f(X)) & \text{since } X \text{ and } Y \text{ are independent} \\ & = & 2 \cdot E(X) \cdot E(f(X)) & \\ & \leq & 2 \cdot E(X) \cdot f(E(X)) & \text{by concavity of } f. \end{array}$$

□

3.3 Algorithmic details and randomized analysis

Referring to the six different steps of the algorithm, here is a detailed cost analysis:

1. Done in time $O(1)$.

2. The nearest neighbor in S_i of a point $p \in S_i$ can be found in the following way. Start considering all the Delaunay edges incident to p in $\mathcal{DT}(S)$. Put them in a priority queue by increasing order of their distance to p. Explore the queue in the following way: each time that we consider a point q, there are two possibilities:

   - If $q \in S_i$, we are done: q is p's nearest neighbor in S_i.
   - If $q \notin S_i$, insert in the queue all its Delaunay neighbors, delete q and proceed to the following point in the queue.

   The correctness of this process is based on the fact that it simulates the way in which a circle centered in p would grow. In other words, if $q \in S_i$ is the point we are looking for, the algorithm computes and orders all the points that are closer to p than q (obviously, none of them belongs to S_i). The proof is based on the following observation.

   Fact. *Let S be a set of points. Let C be any disk in the plane that contains a point $s \in S$ on its boundary. Let $p_1, \ldots, p_k$ be all the points of S contained in C. Then s must have a Delaunay neighbor among $p_1, \ldots, p_k$.*

   Proof: Grow a circle C_s through s, tangent to C and interior to C, until it reaches the first point p_i (see Figure 1).

   The emptiness of C_s is obvious, and therefore sp_i is a Delaunay edge. □

   In this procedure, we have explored and ordered all the points that lie closer to p than q, together with all their neighbors. Can $T(p)$, the number of such points, be too big on average? As the randomly chosen point can belong either to S_1 or to S_2, we want to bound the following amount:

$$E(T) = \frac{1}{n}\left(\sum_{p\in S_1}\sum_{q\in D(p,NN_1(p))}\deg(q) + \sum_{p\in S_2}\sum_{q\in D(p,NN_2(p))}\deg(q)\right)$$

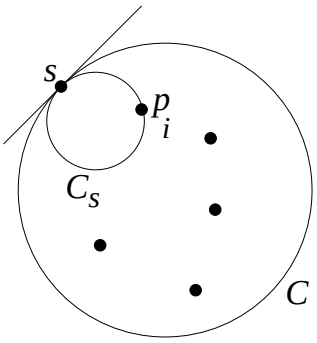

Figure 1: The points s and p_i are Delaunay neighbors.

where $NN_i(p)$ denotes the nearest neighbor of p in S_i, $D(p,s)$ is the disk of center p passing through s and $\deg(q)$ denotes the degree of q in $\mathcal{DT}(S)$.

We bound the summands in the following way:

$$\begin{aligned}
\sum_{p\in S_1} \sum_{q\in D(p,NN_1(p))} \deg(q) &= \sum_{q\in S_2} \sum_{p \text{ s.t. } q\in D(p,NN_1(p))} \deg(q) \\
&= \sum_{q\in S_2} \deg(q) \ \text{number}\{p \text{ s.t. } q \in D(p, NN_1(p))\} \\
&\leq 6 \sum_{q\in S_2} \deg(q).
\end{aligned}$$

The last inequality is due to the fact that the number of disks of the kind $D(p, NN_1(p))$ that can contain a point $q \in S_2$ is at most 6, because in the set $S_1 \cup \{q\}$ such a point p would have q as closest neighbor, and the maximum indegree of q in the nearest neighbor graph of $S_1 \cup \{q\}$ is 6.

Thus we get

$$E(T) \leq \frac{6}{n}\left(\sum_{q\in S_2} \deg(q) + \sum_{q\in S_1} \deg(q)\right) \leq 36.$$

Since the algorithm requires a priority queue, the cost of searching for q is $O(T \log T)$ if we use a balanced priority queue or even $O(T^2)$ if we

use a simple list to implement the queue and $E(T^2)$ cannot be bounded by a constant. But the time for deciding which of p_α and p_β will be p is the minimum of the times for finding the neighbors of p_α and p_β and thus expected to be constant by Lemma 2. This step has expected cost $O(1)$.

3. It is known that it can be done in time proportional to the degree of p in $\mathcal{DT}(S)$ with Chew's algorithm. Since for a random point, the expected degree is 6, the expected degree of p is smaller than 12 by Lemma 1. Hence, this step has expected cost $O(1)$.

4. If the cost of the algorithm is denoted $C(n)$, this step can be done in $C(n-1)$.

5. Exploring all the triangles incident to q takes time proportional to the degree of q in $\mathcal{DT}(S_i \setminus \{p\})$. But q is not a random point, but the nearest neighbor of p, itself chosen among two random points. We will prove below that the degree of the nearest neighbor in S_i of a random point $p \in S_i$ is at most 42, and thus by Lemma 1 the expected degree of q is less than 84 and this step can be done in time $O(1)$.

   Fact. *Given a random point p in a set of points R, the expected degree in $\mathcal{DT}(R \setminus \{p\})$ of the nearest neighbor of p in R is at most 42.*

   Proof: We have to consider the degree of a point in several graphs. Let $\deg_{NN}(q)$ be the indegree of q in the nearest neighbor graph of R, $\deg(q)$ be the degree of q in $\mathcal{DT}(R)$ and $\deg_p(q)$ be the degree of q in $\mathcal{DT}(R \setminus \{p\})$. It is known that $\deg_{NN}(q)$ is at most 6. When p is removed from $\mathcal{DT}(R)$ the new neighbors of q are former neighbors of p, thus $\deg_p(q) \leq \deg(p) + \deg(q)$. The expected value of $deg_p(NN(p))$ is:

$$\begin{aligned} E\left(\deg_p(NN(p))\right) &= \frac{1}{n}\sum_{p\in R} \deg_p(NN(p)) \\ &\leq \frac{1}{n}\sum_{p\in R} \left(\deg(p) + \deg(NN(p))\right) \\ &\leq 6 + \frac{1}{n}\sum_{p,q\in R\ q=NN(p)} \deg(q) \end{aligned}$$

$$\leq \quad 6 + \frac{1}{n} \sum_{q \in R} (\deg_{NN}(q) \deg(q))$$
$$\leq \quad 6 + \frac{1}{n} \sum_{q \in R} (6 \deg(q)) \leq 6 + 36 = 42$$

□

6. It is known that this step can be done in time proportional to the degree of p in $\mathcal{DT}(S_i)$, that is, in expected time $O(1)$ by Lemma 1.

As a conclusion, we have proven the following

Theorem 3 *Given a set of n points S and its Delaunay triangulation, for any partition of S into two disjoint subsets, S_1 and S_2, the Delaunay triangulations $\mathcal{DT}(S_1)$ and $\mathcal{DT}(S_2)$ can be computed in $O(n)$ expected time.*

4 Concluding remarks

4.1 Alternative ideas

We should mention several simpler ideas that do not work. A first idea consists in deleting all the points of S_2 from $\mathcal{DT}(S)$ in a random order, but the degree of a random point in S_2 cannot be controlled; in fact if we take points on the part of the unit parabola with positive abscissa, the Delaunay triangulation links the point of highest curvature to all others (see Figure 2). If we split the set into two parts along the parabola and we remove the highest-curvature half of the point set in a random order, then the probability of removing the highest curvature point increases as the set of point decreases and the expected time to remove half the points is $O(n \log n)$.

Another idea is to remove the points not at random, but by increasing degree, but in that case the set of points to remove must be kept sorted by degree, although the degrees change during the algorithm.

4.2 Convex hull in 3D

Through the projection of the plane on a paraboloid in 3D, Delaunay triangulations are closely related to convex hulls in three dimensions.

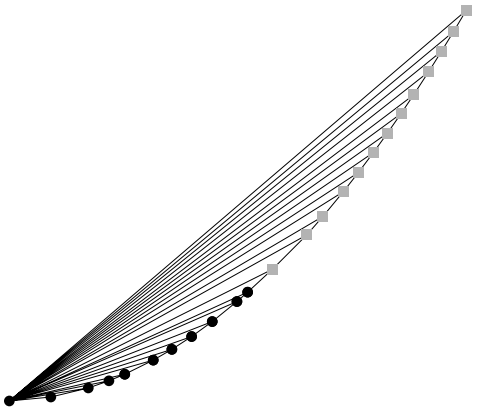

Figure 2: Points on a parabola

Unfortunately, our algorithm, or more precisely its complexity analysis, does not generalize to 3D convex hulls. In this paper we use the fact that the nearest neighbor graph is a subgraph of the Delaunay triangulation having bounded degree, and to generalize the algorithm we would need to define a neighboring relation which is a subgraph of the convex hull; several possibilities for such a subgraph exist but they do not provide bounded degree and thus the analysis does not generalize.

Acknowledgments

The authors thank Oswin Aichholzer for various discussions about this problem.

References

[1] A. Aggarwal, L. J. Guibas, J. Saxe, and P. W. Shor. A linear-time algorithm for computing the Voronoi diagram of a convex polygon. *Discrete Comput. Geom.*, 4(6):591–604, 1989.

[2] Nancy M. Amato, Michael T. Goodrich, and Edgar A. Ramos. Linear-time triangulation of a simple polygon made easier via randomization. In *Proc. 16th Annu. ACM Sympos. Comput. Geom.*, pages 201–212, 2000.

[3] Jean-Daniel Boissonnat, Olivier Devillers, René Schott, Monique Teillaud, and Mariette Yvinec. Applications of random sampling to on-line algorithms in computational geometry. *Discrete Comput. Geom.*, 8:51–71, 1992.

[4] Jean-Daniel Boissonnat, Olivier Devillers, Monique Teillaud, and Mariette Yvinec. Triangulations in CGAL. In *Proc. 16th Annu. ACM Sympos. Comput. Geom.*, pages 11–18, 2000.

[5] Jean-Daniel Boissonnat and Monique Teillaud. On the randomized construction of the Delaunay tree. *Theoret. Comput. Sci.*, 112:339–354, 1993.

[6] Bernard Chazelle. Triangulating a simple polygon in linear time. *Discrete Comput. Geom.*, 6(5):485–524, 1991.

[7] Bernard Chazelle. An optimal algorithm for intersecting three-dimensional convex polyhedra. *SIAM J. Comput.*, 21(4):671–696, 1992.

[8] L. P. Chew. Building Voronoi diagrams for convex polygons in linear expected time. Technical Report PCS-TR90-147, Dept. Math. Comput. Sci., Dartmouth College, Hanover, NH, 1986.

[9] L. P. Chew and S. Fortune. Sorting helps for Voronoi diagrams. *Algorithmica*, 18:217–228, 1997.

[10] F. Chin, J. Snoeyink, and C. A. Wang. Finding the medial axis of a simple polygon in linear time. *Discrete Comput. Geom.*, 21(3):405–420, 1999.

[11] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete Comput. Geom.*, 4:387–421, 1989.

[12] Mark de Berg, Olivier Devillers, Katrin Dobrindt, and Otfried Schwarzkopf. Computing a single cell in the overlay of two simple polygons. *Inform. Process. Lett.*, 63(4):215–219, August 1997.

[13] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, 1997.

[14] Olivier Devillers. Randomization yields simple $O(n \log^* n)$ algorithms for difficult $\Omega(n)$ problems. *Internat. J. Comput. Geom. Appl.*, 2(1):97–111, 1992.

[15] H. Djidjev and A. Lingas. On computing Voronoi diagrams for sorted point sets. *Internat. J. Comput. Geom. Appl.*, 5:327–337, 1995.

[16] Leonidas J. Guibas, D. E. Knuth, and Micha Sharir. Randomized incremental construction of Delaunay and Voronoi diagrams. *Algorithmica*, 7:381–413, 1992.

[17] Rolf Klein and Andrzej Lingas. A linear-time randomized algorithm for the bounded Voronoi diagram of a simple polygon. *Internat. J. Comput. Geom. Appl.*, 6:263–278, 1996.

[18] R. Seidel. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Comput. Geom. Theory Appl.*, 1(1):51–64, 1991.

[19] R. Seidel. Backwards analysis of randomized geometric algorithms. In J. Pach, editor, *New Trends in Discrete and Computational Geometry*, volume 10 of *Algorithms and Combinatorics*, pages 37–68. Springer-Verlag, 1993.

[20] J. Snoeyink and M. van Kreveld. Good orders for incremental (re)constructions. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 400–402, 1997.