The delaunay hierarchy — **Source link** ⧉

Olivier Devillers

Institutions: French Institute for Research in Computer Science and Automation

Published on: 01 Apr 2002 - International Journal of Foundations of Computer Science (World Scientific Publishing Company)

Topics: Pitteway triangulation, Constrained Delaunay triangulation, Delaunay triangulation, Minimum-weight triangulation and Bowyer–Watson algorithm

Related papers:

- Computational Geometry: Algorithms and Applications

- Incremental constructions con BRIO

- Fast randomized point location without preprocessing in two- and three-dimensional Delaunay triangulations

- Walking in a Triangulation

- Optimal Search in Planar Subdivisions

The Delaunay Hierarchy

Olivier Devillers

The Delaunay Hierarchy

Olivier Devillers

Abstract

We propose a new data structure to compute the Delaunay triangulation of a set of points in the plane. It combines good worst case complexity, fast behavior on real data, small memory occupation and the possibility of fully dynamic insertions and deletions.

The location structure is organized into several levels. The lowest level just consists of the triangulation, then each level contains the triangulation of a small sample of the level below. Point location is done by walking in a triangulation to determine the nearest neighbor of the query at that level, then the walk restarts from that neighbor at the level below. Using a small subset (3%) to sample a level allows a small memory occupation; the walk and the use of the nearest neighbor to change levels quickly locate the query.

keywords: computational geometry, geometric computing, randomized algorithms, Delaunay triangulation, dynamic algorithms.

1 Introduction

The computation of the Delaunay triangulation of a set of n points in the plane is one of the classical problems in computational geometry and plenty of algorithms have been proposed to solve it.

These Delaunay algorithms can have different characteristics:

- Optimal on worst case data, i.e. $O(n \log n)$ time.

- Good complexity on random data only

- Randomized

- On-line vs off-line

In the current trade-off between algorithmic simplicity, practical efficiency and theoretical optimality, practitioners often opt for simplicity and practical

efficiency taking the risk of having bad performance on some special kind of data.

Our aim is to reconcile many of the above aspects, namely to obtain an incremental algorithm using simple data structures, having a good practical performance on realistic input and still having a provable $O(n \log n)$ expected randomized computation time in the worst case.

Previous related work

Our work is strongly related to some previous algorithms for Delaunay triangulation. All these algorithms are incremental and their complexity is randomized, they use some location structure to find where the new point is inserted, and then update the triangulation.

The first idea for a randomized incremental construction for the Delaunay triangulation [5] uses a location structure based on the history of the Delaunay triangulation: the Delaunay tree. Point p_i is inserted at time i; to find the position of p_n in the triangulation, p_n is located in all the triangulations at times 1 to $n-1$; the location at time $i+1$ is deduced from the location at time i. This idea yields an expected $O(n \log n)$ complexity [6, 20] if the points are inserted in a random order. The drawbacks of this approach are the following: the location structure consists of the history of the construction and thus strongly depends on the insertion order, and the additional memory needed cannot be controlled. (The expected memory is proved to be $O(n)$ and is experimentally about twice the size of the final triangulation.)

Mulmuley [25] proposed a location structure independent of the insertion order. The structure has $O(\log n)$ levels, each level being a random sample of the level below. At each level, the Delaunay triangulation of the points is computed, and the overlapping triangles at different levels are linked to enable location of new points. This structure has the advantage of being independent of the order of insertion, of ensuring an $O(\log^2 n)$ location time for any point, and of allowing deletions in an easier way than the Delaunay tree [13]. However, the additional memory is still important and the location structure is not especially simple.

Mücke, Saias and Zhu [24] proposed a very simple structure and algorithm called *jump and walk* to handle triangulation of random points. The structure reduces to a random subset of $\sqrt[3]{n}$ points and pointers, from each of these points, to an incident triangle in the Delaunay triangulation. A new point is located by finding the nearest neighbor in the sample by brute force, and walking in the triangulation starting at that point. For evenly distributed points, the expected number of points in the zone of a point of the sample is $O(n^{\frac{2}{3}})$ and since the complexity of the walk has the behavior of a square root [17], we get an $O(\sqrt[3]{n})$ time for locating a point and an $O(n^{\frac{4}{3}})$ complexity for constructing the whole triangulation. The simplicity of that algorithm makes it competitive with many $O(n \log n)$ algorithms, but for some data (for example points on a curve) the complexity may increase to $O(n^{\frac{5}{3}})$.

Overview

Our approach uses a structure with levels similar to Mulmuley, but with simpler relations between levels. This allows a better control of the memory overhead. The transition between two levels is not direct as in Mulmuley but, similarly to Mücke, Saias and Zhu's algorithm, uses a walk to locate a point in a triangulation.

In Section 2 we present the algorithm, in Section 3 we prove that the expected complexity of constructing the Delaunay triangulation is $O(n \log n)$. The generalization to higher dimensions is explained in Section 4. In Section 5, the parameters of the data structure are then tuned to minimize the constant in the case of random points and are shown to yield an excellent behavior, we pay special attention to the comparison with the method of Mücke, Saias and Zhu. Finally we give some implementation remarks and practical results in Section 6.

2 Algorithm

Let $\mathcal{S}$ be a set of n sites in the plane. The aim is to compute the Delaunay triangulation $\mathcal{DT}_\mathcal{S}$ of $\mathcal{S}$ and to maintain it efficiently under insertions and deletions.

2.1 The location structure

The algorithm uses a data structure composed of different levels. Level i contains the Delaunay triangulation $\mathcal{DT}_i$ of a set of sites $\mathcal{S}_i$.

The sets $\mathcal{S}_i$ forms a decreasing sequence of random subsets of $\mathcal{S}$ based on a Bernoulli sampling technique [23, 26]:

$$\mathcal{S} = \mathcal{S}_0 \supseteq \mathcal{S}_1 \supseteq \mathcal{S}_2 \supseteq \ldots \supseteq \mathcal{S}_{k-1} \supseteq \mathcal{S}_k$$

$$Prob(p \in \mathcal{S}_{i+1} \mid p \in \mathcal{S}_i) = \frac{1}{\alpha} \in]0, 1[.$$

The data structure is fairly simple: it contains the points of $\mathcal{S}$ and the triangles of all the triangulations $\mathcal{DT}_i$. A point $p \in \mathcal{S}$ such that $p \in \mathcal{S}_i \subseteq \ldots \subseteq \mathcal{S}_0$ and $p \notin \mathcal{S}_{i+1}$ is said to be a *vertex of level i* and has a link to a Delaunay triangle of $\mathcal{DT}_j$ incident to p for all j between 0 and i. A triangle of $\mathcal{DT}_i$ has links to its three neighbors in $\mathcal{DT}_i$ and to its three vertices. The number k of levels is not fixed; for each point, random trials decide its level, and the point with the highest level determines k.

2.2 Location of a query

To locate a query q, we start at a known vertex v_{k+1} of the highest level k. Then we search for v_k, the vertex of $\mathcal{DT}_k$ nearest to q. Since v_k is also a vertex

of $\mathcal{DT}_{k-1}$, we search for v_{k-1}, the nearest neighbor of q in $\mathcal{DT}_{k-1}$, starting at v_k. The search is continued descending the different levels. At each level i, the nearest vertex v_i of q in $\mathcal{DT}_i$ is determined.

At level i the search for v_i is carried out in three phases:

- First phase: from v_{i+1}, we have a link to a triangle of $\mathcal{DT}_i$ having v_{i+1} as a vertex. All triangles incident to v_{i+1} are explored to find the triangle containing the segment $v_{i+1}q$.

- Second phase: all the triangles of $\mathcal{DT}_i$ intersected by $v_{i+1}q$ are visited, walking along the segment $v_{i+1}q$ up to the triangle t_i that contains q.

- Third phase: from t_i, we have to find the nearest neighbor v_i of q. To this aim, we visit the triangles that may be intersected by line segment qv_i: consider a visited triangle $ww'w''$ of $\mathcal{DT}_i$ such that q lies within the circle C' through w, w' and w''. Assume without loss of generality than $|qw| \leq |qw'|, |qw''|$ and let C be the circle of center q through w, the relevant neighbors of $ww'w''$ are visited. Irrelevant neighbors are determined by the following rules:

    - Do not examine an already visited triangle.
    - C does not cut C' between w' and w'', thus the line segment qv_i can not intersect $w'w''$ and the neighbor of $ww'w''$ through $w'w''$ does not need to be examined (Figure 1a).
    - if the absolute value of the angle qww' (resp. qww'') is greater than $\frac{\pi}{2}$ then qv_i can not intersect ww' (resp. ww'') since C (which contains qv_i) does not intersect ww' (resp. ww''). The neighbor of $ww'w''$ through ww' (resp. ww'') does not need to be examined (Figure 1b).

    This phase starts at triangle t_i and selects among the vertices of the visited triangles the closest to q.

Figure 1c show the triangles visited by the different phases of the search.

2.3   Updates

Because of its simplicity, the data structure is fairly easy to update. Maintaining it dynamically provides a fully dynamic triangulation algorithm. The links between the different levels do not use any complicated data structure. The vertices just know a triangle at all levels in which they appear.

To delete a point from $\mathcal{S}$, just delete the corresponding vertex at all the levels where it appears, which can be done in time sensitive to d the degree of that vertex. On average $d = 6$ and thus some of the following algorithms can be used: a complicated algorithm [1] of deterministic complexity $O(d)$, a simple randomized $O(d)$ algorithm [8], solutions of complexity $O(d \log d)$ [12] or even
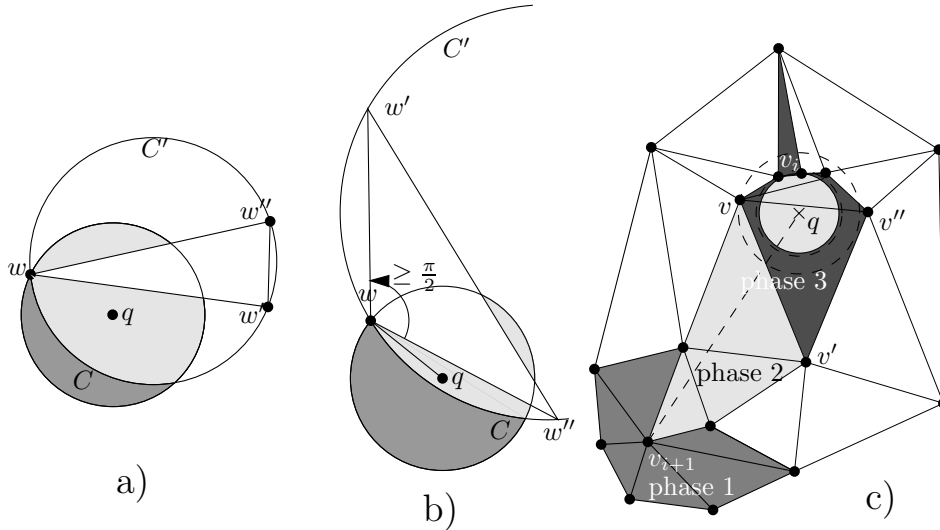
Figure 1: Search for v_i.

simpler $O(d^2)$ algorithms. The simple sub-optimal solutions may be good in practice since d is small on average.

Inserting a point in $\mathcal{S}$ reduces to locating it at all levels, computing its level i and inserting the new vertex at all levels $j, 0 \leq j \leq i$ (which is sensitive to the degree of the new vertex since it is already located). The insertion at each level is done using a standard algorithm [21].

3 Worst-case randomized analysis

The analysis will rely on the randomization in the construction of the random subsets $\mathcal{S}_i$ and on the random order to insert the points of $\mathcal{S}$. In this section, no assumption applies to the data distribution, which can be the worst case distribution. As usual in theoretical computational geometry, we make only an asymptotic analysis and give rough upper bounds for the constants. In the next section, parameter α will be tuned to get a tight constant in the special case of evenly distributed points.

Let $\mathcal{S}$ be a set of n points organized in the structure described in Section 2 and q a point to be inserted in $\mathcal{S}$. Since we have assumed a random insertion order, q is a random point of $\mathcal{S} \cup \{q\}$.

We denote $n_i = |\mathcal{S}_i|$ and $\mathcal{R}_i = \mathcal{S}_i \cup \{q\}$.

Notice that, thanks to the random insertion order, q is a random element of $\mathcal{R}_i$: an element v of $\mathcal{R}_i$ is equal to q with probability $\frac{1}{n_i+1}$. $\mathcal{R}_i$ can be considered as a random subset of $\mathcal{R}_{i-1}$, indeed, given an element v of $\mathcal{R}_{i-1}$, it belongs to $\mathcal{R}_i$ with probability $\frac{1}{n_{i-1}+1} + \frac{n_{i-1}|}{n_{i-1}+1}\frac{1}{\alpha}$ (the first term is the probability that $v = q$ and the second term is the probability that $v \in \mathcal{S}_i$). In the sequel, we denote by $E(X)$ the expected value of some quantity X, the expectation is on the whole randomization process, that is, q is random in $\mathcal{R}_i$ and $\mathcal{S}_i$ is a random subset of $\mathcal{S}_{i-1}$.

The cost of exploring all the triangles incident to v_{i+1} at the first phase of the walk of level i is the degree of v_{i+1} in $\mathcal{DT}_i$. The cost of the second phase is the number of triangles intersected by segment $v_{i+1}q$. The cost of the third phase is the number of visited triangles during the search of v_i from t_i.

Lemma 1 *The expected degree of v_i in $\mathcal{DT}_{i-1}$ is $O(1)$.*

Proof: The difficulty is that v_i is not a random point in $\mathcal{R}_i$ but the nearest neighbor $NN(q)$ of a random point q in $\mathcal{R}_i$. We will use the fact that qv_i is an edge of $\mathcal{NN}$, the nearest neighbor graph of $\mathcal{R}_i$, which has maximum degree 6 [27]. We denote by $d_{\mathcal{G}}^{\circ}(v)$ the degree of v in some graph $\mathcal{G}$.

$$
\begin{aligned}
E\left(d_{\mathcal{DT}_{i-1}}^{\circ}(NN(q))\right) &= E\left(\frac{1}{|\mathcal{R}_i|}\sum_{q\in\mathcal{R}_i} d_{\mathcal{DT}_{i-1}}^{\circ}(NN(q))\right) \\
&= E\left(\frac{1}{|\mathcal{R}_i|}\sum_{q\in\mathcal{R}_i} d_{\mathcal{DT}_{\mathcal{R}_{i-1}}}^{\circ}(q) + d_{\mathcal{DT}_{\mathcal{R}_{i-1}}}^{\circ}(NN(q))\right) \\
&\leq 6 + E\left(\frac{1}{|\mathcal{R}_i|}\sum_{v,q\in\mathcal{R}_i,v=NN(q)} d_{\mathcal{DT}_{\mathcal{R}_{i-1}}}^{\circ}(v)\right) \\
&\leq 6 + E\left(\frac{1}{|\mathcal{R}_i|}\sum_{v\in\mathcal{R}_i} d_{\mathcal{NN}}^{\circ}(v) d_{\mathcal{DT}_{\mathcal{R}_{i-1}}}^{\circ}(v)\right) \\
&\leq 6 + E\left(\frac{1}{|\mathcal{R}_i|}\sum_{v\in\mathcal{R}_i} 6 d_{\mathcal{DT}_{\mathcal{R}_{i-1}}}^{\circ}(v)\right) \\
&\leq 6 + 36 \leq 42
\end{aligned}
$$

$\diamond$

Lemma 2 *Given $w \in \mathcal{R}_i$, the expected number of vertices q of $\mathcal{R}_i$ such that w belongs to the disk of center q and passing through the nearest neighbor of q in $\mathcal{R}_{i+1}$ is less than 6α.*
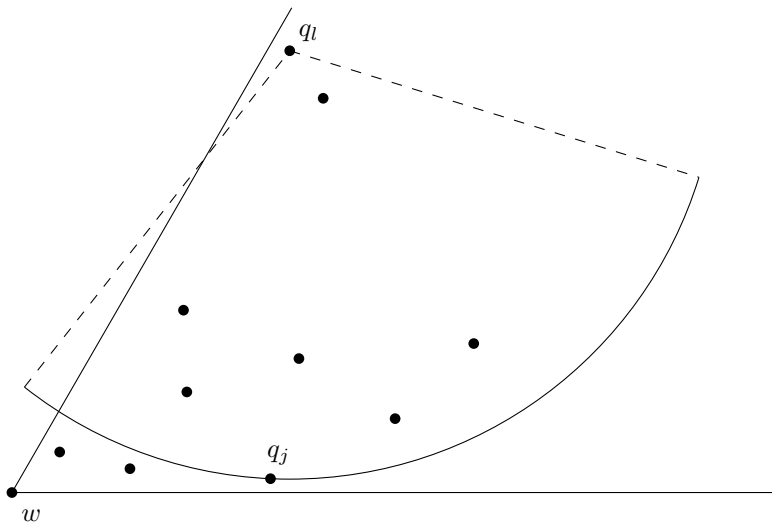
Figure 2: For the proof of Lemma 2

Proof: Let $w \in \mathcal{R}_i$ and let $w = q_0, q_1, q_2 \ldots q_k$ be the points of $\mathcal{R}_i$ lying in a wedge of angle $\frac{\pi}{3}$ having apex w sorted by increasing distance to w. Clearly, a disk of center q_l passing through q_j $(j < l)$ cannot contain w (see Figure 2) and thus, if $q = q_l$, a necessary condition for w to be in the disk having as diameter the segment defined by q and the nearest neighbor of q in $\mathcal{R}_{i+1}$ is that no point of $\{q_0, \ldots q_{l-1}\}$ is in the sample $\mathcal{R}_{i+1}$. It happens with probability $(1 - \frac{1}{\alpha})^l$.

Using six wedges around w to cover the whole plane, and summing over the choice of $q \in \mathcal{R}_i$ we get the claimed result. Notice that the disk of center q and passing through $NN(q)$ contains the disk of diameter $qNN(q)$, thus the expected number of vertices q of $\mathcal{R}_i$ belonging to the disk of diameter $qNN(q)$ is less than 6α. $\diamond$

Lemma 3 *The expected number of edges of $\mathcal{DT}_i$ intersecting segment qv_{i+1} is $O(\alpha)$.*

Proof: Let e be an edge of $\mathcal{DT}_i$ intersecting segment qv_{i+1}. If e belongs to $\mathcal{DT}_{\mathcal{R}_i}$, it means that e is an internal edge of the region retriangulated when q is inserted in $\mathcal{DT}_i$.

The expected number of such edges is 3 since q is a random point in $\mathcal{R}_i$, and that number equals the average degree of q in $\mathcal{R}_i$ minus 3.

If e belongs to $\mathcal{DT}_{\mathcal{R}_i}$, one end-point w of e must belong to the disk of diameter qv_{i+1}, denoted disk$[qv_{i+1}]$, (otherwise any disk through the end-points of e must contain q or v_{i+1} and e cannot be a Delaunay edge).

7

The expected number of edges of $\mathcal{DT}_{\mathcal{R}_i}$ intersecting disk$[qv_{i+1}]$ is bounded by the sum of the degrees of the vertices in disk$[qv_{i+1}]$

$$E(\#\{e \in \mathcal{DT}_{\mathcal{R}_i} \text{ having an end-point} \in \mathcal{R}_i \cap \text{disk } [qv_{i+1}]\})$$

$$= \frac{1}{|\mathcal{R}_i|} \sum_{q \in \mathcal{R}_i} \sum_{w \in \mathcal{R}_i \cap \text{disk } [qv_{i+1}]} d^{\circ}_{\mathcal{DT}_{\mathcal{R}_i}}(w)$$

$$= \frac{1}{|\mathcal{R}_i|} \sum_{w \in \mathcal{R}_i} d^{\circ}_{\mathcal{DT}_{\mathcal{R}_i}}(w) \cdot \#\{q \in \mathcal{R}_i | w \in \text{disk } [qv_{i+1}]\}$$

$$\leq \frac{1}{|\mathcal{R}_i|} \sum_{w \in \mathcal{R}_i} d^{\circ}_{\mathcal{DT}_{\mathcal{R}_i}}(w) 6\alpha \quad \text{using Lemma 2}$$

$$\leq 36\alpha \quad \text{using the bound of 6 on the average degree of } w$$

Notice that Lemma 2 was established for a fixed w and a random q which allows it to be used inside the sum over w. Thus we get a total expected cost for the walk bounded by $36\alpha + 3$. $\diamond$

Lemma 4 *The expected number of triangles of $\mathcal{DT}_i$ visited during the search for v_i from t_i is $O(\alpha)$.*

Proof: The triangles examined in Phase 3, either are triangles of $\mathcal{DT}_i \setminus \mathcal{DT}_{\mathcal{R}_i}$ and the expected number of such triangles is 4, or are triangles of $\mathcal{DT}_i \cap \mathcal{DT}_{\mathcal{R}_i}$ and have a vertex in the disk of center q passing through v_{i+1}. Thus we can argue as in Lemma 3, denoting disk $|_c qv_{i+1}]$ the disk of center q through v_{i+1}:

$$E(\#\{t \in \mathcal{DT}_{\mathcal{R}_i} \text{having a vertex} \in \text{disk } |_c qv_{i+1}]\})$$

$$\leq \frac{1}{|\mathcal{R}_i|} \sum_{q \in \mathcal{R}_i} \sum_{w \in \mathcal{R}_i \cap \text{disk } |_c qv_{i+1}]} d^{\circ}_{\mathcal{DT}_{\mathcal{R}_i}}(w)$$

$$\leq \frac{1}{|\mathcal{R}_i|} \sum_{w \in \mathcal{R}_i} d^{\circ}_{\mathcal{DT}_{\mathcal{R}_i}}(w) |\{q \in \mathcal{R}_i | w \in \mathcal{R}_i \cap \text{disk } |_c qv_{i+1}]\}|$$

$$\leq \frac{1}{|\mathcal{R}_i|} \sum_{w \in \mathcal{R}_i} d^{\circ}_{\mathcal{DT}_{\mathcal{R}_i}}(w) 6\alpha \quad \text{using Lemma 2}$$

$$\leq 36\alpha \quad \text{using the bound on the average degree of } w$$

The expected number of visited triangles is less than $4 + 36\alpha$. $\diamond$ An alternative solution for Phase 3 consists of simulating the insertion of q in $\mathcal{DT}_i$ in order to select all the points which are neighbors of q in $\mathcal{DT}_{\mathcal{R}_i}$. Since we know that q and v_i are neighbors in $\mathcal{DT}_{\mathcal{R}_i}$, v_i can be selected from these vertices. In that case the visited triangles are those of $\mathcal{DT}_i \setminus \mathcal{DT}_{\mathcal{R}_i}$ plus their neighbors, that is, an expected number of less than 10. This approach involves more complicated tests than the approach analyzed above.

Theorem 5 *The expected cost of inserting the nth point in the Delaunay hierarchy is $O(\alpha \log_\alpha n)$*

Proof: By linearity of expectation, Lemmas 1, 3 and 4 prove that the expected cost at one level is $O(\alpha)$. Since the expected height of the structure is $\log_\alpha n$, we get the claimed result. (The analysis is similar to the analysis for skip lists [23].) $\diamondsuit$

Theorem 6 *The construction of the Delaunay hierarchy of a set of n points is done in expected time $O(\alpha n \log_\alpha n)$ and $O(\frac{\alpha}{\alpha-1}n)$ space. The expectation is on the randomized sampling and the order of insertion, with no assumption on the point distribution.*

Proof: Easy corollary of Theorem 5. $\diamondsuit$

4 Higher dimensional case

The previous algorithm and its analysis generalize easily to higher dimensions. The algorithm uses several levels of samples. At level i it uses a walking strategy to locate the simplex containing a query, then its nearest neighbor v_i can be found by simulating the insertion of q in $\mathcal{DT}_i$.

The analysis of the previous section uses essentially the fact that the nearest neighbor graph is of bounded degree 6 and the fact that the expected degree of a point in a triangulation is also 6. The cost of walking at level i is the product of these two numbers. In higher dimensions, the degree of the nearest neighbor graph remains bounded by some constant depending on the dimension. Unfortunately the expected degree of a random point is not bounded in higher dimensions. As usual in analysis of incremental randomized constructions [4], we can write aexpress the complexity as sensitive to the expected size of the Delaunay triangulation of a sample. If $f(r)$ is the expected size of the Delaunay triangulation of a random sample of size r of $\mathcal{S}$, then the expected degree of a random point in the triangulation of that sample is $\frac{f(r)}{r}$.

Lemmas 1, 2, 3 and 4 generalize easily: Lemma 1 gives that the expected degree of v_i is $O\left(\frac{f(r)}{r}\right)$. Lemma 2 remains unchanged (up to a constant) and Lemmas 3 and 4 give bounds of $O\left(\alpha \frac{f(r)}{r}\right)$ for the Phases 2 and 3 of the walk. The expected complexity of walking at level i is $O(\alpha \frac{f(\alpha^i n)}{\alpha^i n})$. Summing on the different levels yields an expected complexity of $\sum_{i=0}^{\infty} \alpha \frac{f(\alpha^i n)}{\alpha^i n}$ to locate a point in the Delaunay hierarchy.

There exist several bounds for $f(r)$; in the worst case $f(r) = r^{\lceil \frac{d}{2} \rceil}$ and we get Theorems 7 and 8 below. If the points are evenly distributed, $f(r) = O(r)$ [18] and the complexity of inserting a point is logarithmic. Other results on $f(r)$ exist, for example if there is a minimal distance between points [19] or if the points belong to a surface [2].

Theorem 7 *The expected cost of inserting the nth point in a d-dimensional Delaunay hierarchy for any constant ratio α is $O\left(n^{\lceil \frac{d-2}{2} \rceil}\right)$.*

Proof: The expected cost at level i is bounded by the maximum degree of the nearest neighbor graph multiplied by $\alpha(\alpha^i n)^{\lceil \frac{d-2}{2} \rceil}$, summing for i going from 0 to ∞ gives $O\left(\frac{\alpha}{1-\alpha^{\lceil \frac{d-2}{2} \rceil}} n^{\lceil \frac{d-2}{2} \rceil}\right)$. $\diamond$

Theorem 8 *The construction of the d-dimensional Delaunay hierarchy of a set of n points can be done in expected time and space $O\left(n^{\lceil \frac{d}{2} \rceil}\right)$. The expectation is on the randomized sampling and the order of insertion, with no assumption on the point distribution.*

Proof: Easy corollary of Theorem 7. $\diamond$

5 Tuning parameters

We have proved that our structure has an asymptotically worst case optimal cost $O(n \log n)$ for any set of points in the plane. The constant hidden in the O actually varies with α, the ratio between two levels of the hierarchy. In this section, we propose a more precise analysis, not in the worst case, but for points evenly distributed in a square[1] to tune the parameter α in that case.

This section needs a careful analysis of the constant involved in the different phases of the algorithm which implies a precise counting of the arithmetic operations involved.

5.1 Phase 1

We can assume that $d^{\circ}_{\mathcal{DT}_i}(v_{i+1}) = 6$ (and not only ≤ 36 as proved in Lemma 1). And thus if the turn around v_{i+1} is clockwise or counterclockwise depending on the position of segment $v_{i+1}q$ with respect to the starting triangle, and assuming that this position is random around v_{i+1}, the expected number of orientation tests[2] is 3. Figure 3 shows the different cases to average; the edges $v_{i+1}w$ such that an orientation test $v_{i+1}wq$ is performed are indicated for a typical degree 6 vertex in the triangulation.

---

[1] We denote by γ the point density, that is, the expected number of points in any unit square.

[2] An orientation test takes three points p, q and r and reports if the triangle is clockwise, counterclockwise or degenerate.
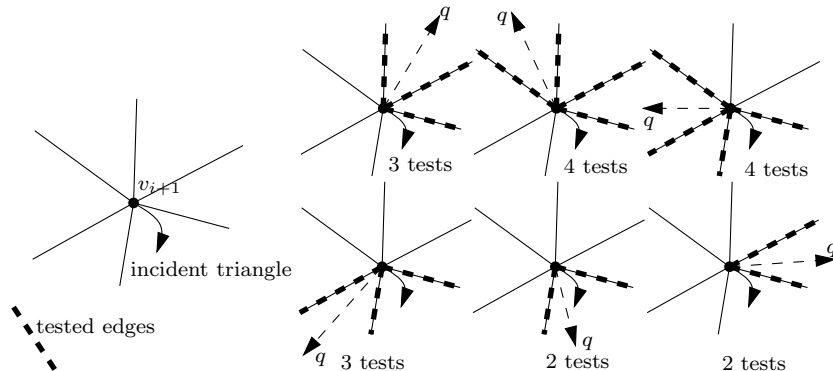
Figure 3: Number of orientation tests in Phase 1

5.2 Phase 2

Devroye et al. [17, 7] proved that the expected number of edges of a Delaunay triangulation of random points crossed by a line segment of length l is $O(l\sqrt{\gamma})$ where γ is the point density. Our experiments show that the constant is 2.

The expected number of points in the disk of center q passing through v_{i+1} is $\alpha - 1$. Indeed, if the points of $\mathcal{R}_i$ are sorted by increasing distance from q, v_{i+1} is the first point in $\mathcal{R}_{i+1}$, thus the number of points in the disk is k with probability $(1 - \frac{1}{\alpha})^k \frac{1}{\alpha}$ (probability that the k first points are not in the sample and the $k + 1^{st}$ point is), and the expected number is $\frac{1}{\alpha} \sum k(1 - \frac{1}{\alpha})^k = \alpha - 1$. Thus if l is the length of qv_{i+1} the density of points in $\mathcal{DT}_i$ is $\gamma = \frac{\alpha}{\pi l^2}$.

Thus we conclude that the expected number of edges of $\mathcal{DT}_i$ intersecting segment qv_{i+1} is $i2l\sqrt{\gamma} = 2l\sqrt{\frac{\alpha}{\pi l^2}} = \frac{2\sqrt{\alpha}}{\sqrt{\pi}}$.

For each edge ww' crossed, two orientation tests are performed [14]: if w is the newly examined vertex, orientations of triangles wqv_{i+1} and qww' are computed.

5.3 Phase 3

Phase 3 is more difficult to analyze precisely, but a rough bound is that the number of candidate vertices examined (with shortest distance) is less than two and that we examine less than 8 triangles in total.

In fact, we modified Phase 3 so that instead of really searching for v_i, the nearest neighbor of q in $\mathcal{S}_i$, we just define v_i as the nearest among the three vertices of t_i. This modified Phase 3 is reduced to three distance computations and two comparisons.
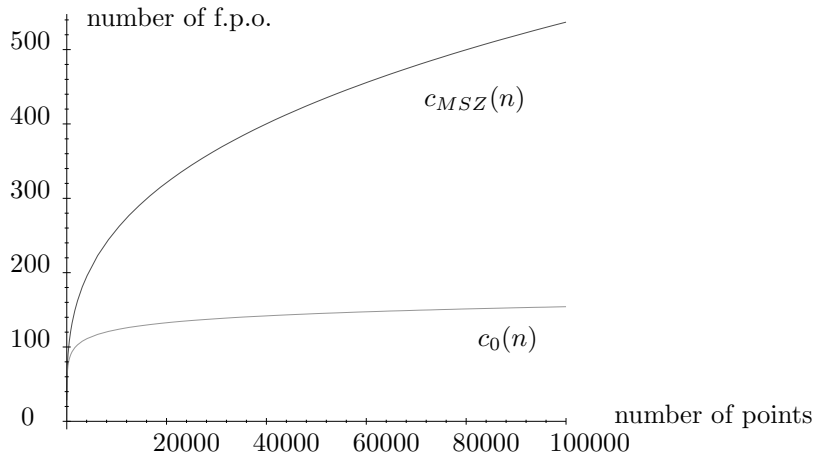
11

Figure 4: Comparison of the number of floating point operations between $c_0(n)$ and $c_{MSZ}(n)$ for $\alpha = 40$ and $\beta = 1$.

5.4 Tuning α

A precise counting [10] of floating point operations (f.p.o.) yields a complexity of $36 + 6.2\sqrt{\alpha}$ at each level. Since the number of levels is $\log_\alpha n = \frac{\log_2 n}{\log_2 \alpha}$ we get a cost of $c_0(n) = (32 + 6.2\sqrt{\alpha}) \left\lceil \frac{\log_2 n}{\log_2 \alpha} \right\rceil$ which is close to its minimum ($c_0(n) \in [13.3 \log_2 n, 14 \log_2 n]$) for $\alpha \in [18, 90]$, with the minimum occuring when $\alpha \simeq 40$.

5.5 Comparison with [24]

A similar counting [10] of f.p.o. in the jump and walk algorithm, using a random sample of $\beta \sqrt[3]{n}$ points, produces a cost of $c_{MSZ}(n) = 17 + \sqrt[3]{n} \left(\frac{6.2}{\sqrt{\beta}} + 5\beta \right)$ which is close to its minimal value for $0.5 < \beta < 1$.

As shown by the comparison of the two curves in Figure 4, our method is potentially much better than jump and walk [24], even for a small number of points. However, this method of analyzing our approach hides the discontinuity of the cost, since the effective number of levels is necessarily an integer. To have a better understanding of what happens for a small number of points, we plot the cost of inserting a point in a structure having a fixed number of levels.

The classical walk from a random point in the structure costs $c_{walk}(n) = 17 + 6.2\sqrt{n}$ which is also the cost of inserting in our structure up to the time a second level is created.

12

When k levels have been created, the cost is

$$c_k(n) = c_{walk}\left(\frac{n}{\alpha^k}\right) + 15k + k \cdot c_{walk}(\alpha).$$

We can alternatively mix this multilevel approach with Mücke et al's. sampling at the first level of the structure. In that case, the cost is

$$c_k^{\star}(n) = c_{MSZ}\left(\frac{n}{\alpha^k}\right) + 15k + k \cdot c_{walk}(\alpha).$$

This comparison (see Figure 5) shows that jump and walk [24] ($c_1^{\star}(n)$) becomes better than the simple walk ($c_1(n)$) for $n > 40$. The two level delaunay hierarchy ($c_2(n)$) becomes better than the simple walk ($c_1(n)$) for $n > 180$ and better than jump and walk [24] ($c_1^{\star}(n)$) for $n > 600$. The main information is that our structure should be significantly better than [24] for $10000 < n$.

6 Implementation

6.1 Deletion

The above structure supports insertions and queries as explained above, but also deletions. Since there is no complicated data structure to maintain, deletions can be handled by just deleting the removed point at each level where it appears.

This can be done in output-sensitive time [9, 1], and thus the deletion of a random point is done in expected constant time since a point appears at an expected constant number of levels and its expected degree k is also constant.

From a practical point of view, and to keep the simplicity of the algorithm, a simpler suboptimal algorithm is better. Point deletion can be done in $O(k \log k)$ time in any dimension [12]. The hole created by the removal of all faces incident to the removed points is filled by adding the new simplices in some good order.

6.2 Arithmetic degree

A basic geometric test (also called predicate) such as the orientation test usually reduces to the evaluation of the sign of some polynomial; the degree of such a polynomial is a useful measure of the complexity of the predicates used by the algorithm [22]. The algorithm above is designed to make a parsimonious use of high degree tests. More precisely, the location phase uses only orientation tests on three points in Phases 1 and 2, and distance computations and angle comparisons with $\frac{\pi}{2}$ in Phase 3. All these tests have degree 2. Updating the Delaunay triangulation clearly needs to use in-circle tests which are of degree 4.

In higher dimensions, Phases 1 and 2 use degree d tests; Phase 3 and point insertion use degree $d + 2$ tests.
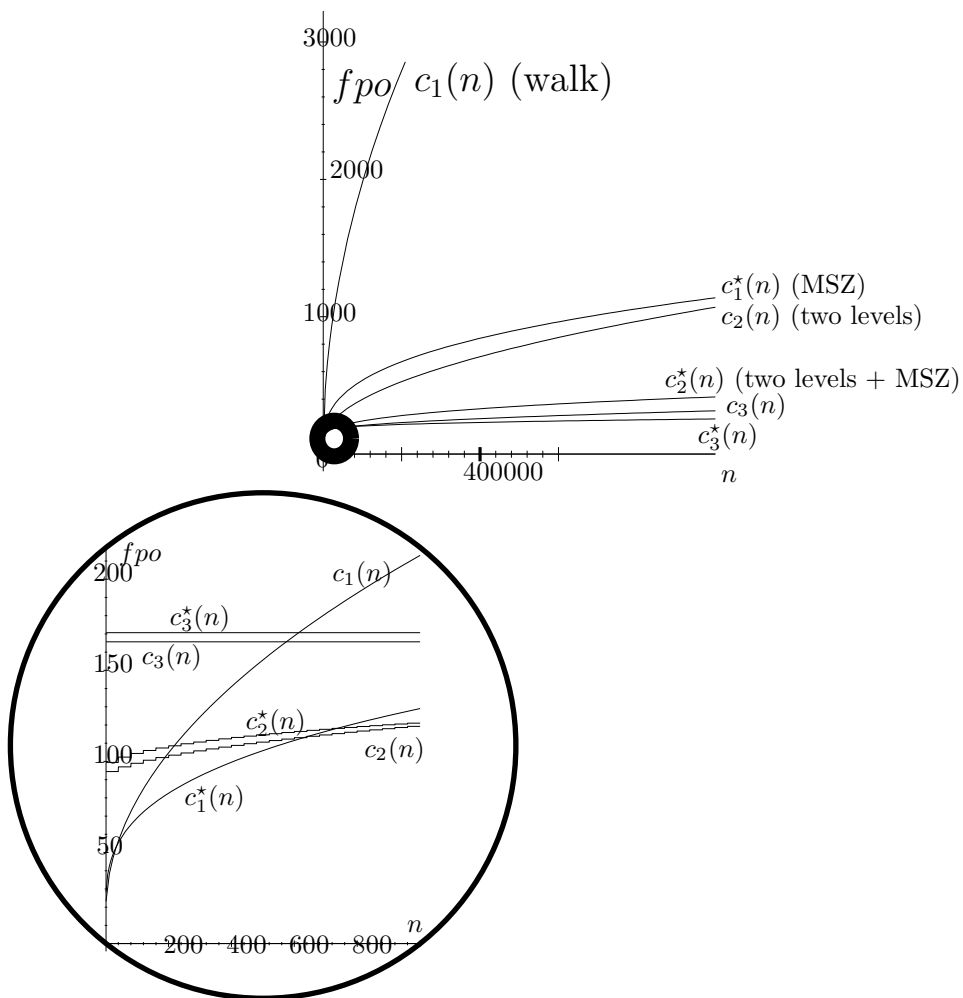
Figure 5: Comparison of the number of floating point operations between $c_k(n)$ and $c_k^\star(n)$ for $\alpha = 40$.

14

6.3 Robustness issues and degeneracies

Degeneracies are solved by handling special cases: if two points have the same coordinates, then the insertion is not done, if four points are cocircular, then the last point inserted is considered as outside the disk defined by the others. This corresponds to the standard perturbation scheme [28] on the convex hull of the points projected on the paraboloid in 3 dimensions. This generalizes in any dimension.

We use exact arithmetic for 24 bits integers, and thus the coordinates of our points are integers in range $[-16.777.216, 16.777.216]$ (up to a multiplication by a power of 2). Using this restricted kind of number, double precision computation is exact on degree 2 tests and almost never leads to precision problems on degree 4 predicates. Nevertheless, the exactness of all computations are verified by an arithmetic filter and exact computation is performed if needed.

In dimension three, we use the same kind of numbers. The orientation test has degree 3 and the in sphere test have degree 5. All computations are done using filtered floating point computations and exact computation if needed [15, 16].

6.4 Experimental results

We have investigated the performance of our algorithm in different situations.

All point sets have 1.000.000 points (except *moment*), times are the time[3] for reading the points and computing the Delaunay triangulation in seconds.

6.4.1 Sensibility to the input distribution

We have used the different kinds of input distributions described below:

2D	*square*	points evenly distributed in a square
	ellipse	points evenly distributed on an ellipse
	mixed	95% points evenly distributed on an ellipse plus 5% points evenly distributed in a square
	parabola	points evenly distributed on the parabola $y = x^2$ and rounded to 24 bits
	circle	points evenly distributed on a circle (rounded)
3D	*cube*	points evenly distributed in a cube
	ellipsoid	points evenly distributed on an ellipsoid
	object	points measured on an object with a 3D sensor
	sphere	points evenly distributed on a sphere (rounded)
	moment	points evenly distributed on the moment curve $y = x^2, z = x^3$ (rounded), 5.000 points only

---

[3]Running times have been obtained on a PentiumIII 730 MHz 640 Mo. Code is in C++ compiled with g++ under Linux.

The following table gives the execution times. We can observe that the running times are of the same order for the different kinds of distribution (except for the quadratic example in 3D). By playing with the parameters, we can use the same software to use the Hierarchy with or without the cubic root sampling of Mcke, Saias and Zhu, the cubic root sampling without the hierarchy, or a simple walk in the triangulation. The Delaunay hierarchy runs much faster than the cubic root sampling in two dimensions. In three dimensions, the advantage is less important but still significant, especially on some non uniform distributions.

data	Hierarchy + sampling	Hierarchy only	cubic root sampling	walk
square	19	20	56	488
ellipse	28	35	106	3380
mixed	31	31	708	392
parabola	33	35	170	
circle	22	22	23	22
cube	90	94	102	245
ellipsoid	95	98	197	774
object	88	88	108	174
sphere	71	72	72	85
moment, 5000	62	62	63	71

6.4.2 Sensibility to the ratio between levels

The table below shows the running time for different point distributions and depending of the ratio used to sample the levels. The line *square* in this table agrees with the theoretical results of Section 5.4 that the behavior of the algorithm should be good for a ratio between levels between 18 and 90; this validates our counting of floating point operations as a relevant measure to establish this optimal value of the ration α. . For non uniform distributions, the table shows that a small ratio in the interval $[18, 90]$ should be prefered. In all other tables, we choose a ratio of 30. Using this ratio of 30 and for one million points, the hierarchy has usually 3 levels of samples.

ratio between levels	3	5	10	20	30	80	200	500
square	27	23	20	19	19	21	24	27
mixed	29	25	24	28	31	47	75	113
ellipse	27	25	24	26	27	37	53	46
cube	147	123	106	100	97	96	97	98
object	130	110	95	89	89	87	89	91

6.4.3 Sensibility to randomness

The complexity is randomized, that is if you are unlucky with the random sample or the random order of insertion the running time may increase. As shown by the following table which gives the running time of different execution

with different choices for the random sampling, the variance of the expected running time of the algorithm is in fact very low and running times have a variation of less than 1%. Since the times vary within this interval of 1% for different runs of the algorithm, times in other tables have been rounded to the precision of the second.

mixed	30.8	30.8	31.4	31.3	31.0
object	88.3	87.2	87.9	88.6	86.9

6.4.4 Comparison with other software

We have compared the two dimensional version with some Delaunay software available on the WWW:

- `qhull` by Bradford Barber and Hannu Huhdanpaa, duality with 3D convex hull [3] (available at http://www.geom.umn.edu/locate/qhull). This algorithm is static.

- `div-conquer` by Jonathan Shewchuk, divide and conquer [29] algorithm using a quadtree scheme. This algorithm is static.

- `sweep` by Jonathan Shewchuk, implements Fortune's plane sweep algorithm, which is static.

- `incremental` by Jonathan Shewchuk, incremental with Mücke et al. localization, this algorithm is semi-dynamic and can be easilly modified to be fully dynamic.
  These three codes support exact arithmetic on `double` (available at http://www.cs.cmu.edu/~quake/triangle.research.html).

- `Dtree` Delaunay tree structure[6] provides a semi-deynamic algorithm (available at http://www.inria.fr/prisme/logiciel/del-tree.html).

- `hierarchy` this paper.

The execution times[4] in seconds are in the table below. Our method is significantly faster than the other incremental method, specially in the ellipse cases. Our method is about 50% slower than the divide and conquer algorithm and is clearly the best fully dynamic algorithm.

---

[4]This set of runs have been done on a Sun-Ultra1 200 MHz 128Mo.

distribution	size	qhull	sweep	div-conq	incr	Dtree	hierarchy
random	5000	0.65	0.21	0.11	0.29	1.4	0.14
random	50000	8.0	3.6	1.6	6.6	17	2.3
random	500000	101	53	22	150	swap	31
mixed	5000	0.54	0.21	0.13	0.75	1.3	0.20
mixed	50000	7.8	3.2	2.16	42	16	3.5
mixed	500000	420	46	29	2100	swap	49
ellipse	5000	0.83	0.18	0.14	2.1	1.3	0.21
ellipse	50000	57	2.8	2.4	110	14	3.7
ellipse	500000	swap	39	33	1400	swap	55
parabola	5000	3.9	0.16	0.11	2.0	1.2	0.16
parabola	50000	790	2.7	2.0	110	14	3.0
parabola	500000	swap	39	28	1800	swap	45
circle	5000	93	0.17	0.17	0.52	1.4	0.14
circle	50000	220	3.1	1.8	11	15	2.4
circle	500000	swap	22	43	240	swap	36

7 Conclusion

We have proposed a new hierarchical data structure to compute the Delaunay triangulation of a set of points in the plane. It combines good worst case randomized complexity, fast behavior on real data, small memory occupation and dynamic updates (insertion and deletion of points).

Referring to Su and Drysdale's study of several techniques [30] and our comparisons with Shewchuk implementations [29] of some of these techniques, we have shown that our implementation is competitive with other approaches on random data; the only faster approach for large inputs is the divide and conquer algorithm which is not dynamic. Furthermore, we can prove that the performance remains good on pathological inputs.

The main idea of our structure is to perform point location using several levels. The lowest level just consists of the triangulation, then each level contains the triangulation of a small sample of the levels below. Point location is done by walking along a line segment in a triangulation to determine the nearest neighbor of the query at that level, then the walk restarts from that neighbor at the level below. Other walking strategies have been proposed [14] and may be more efficient in practice but there is no theoretical result on the complexity of these alternative walks. Location at the highest level is done using jump and walk [24] which is efficient for small sets of points.

One characteristic of the Delaunay hierarchy is that the best time performance is obtained with a ratio of about three per cent between two levels and thus a very small number of levels (three or four typically) and thus good performances in terms of memory occupation. The structure is simple and does

not need additional features such as buckets.

Such structures can be generalized to other problems. The two main ingredients of the proofs are bounds on the maximal degree of the nearest neighbor graph and the expected degree of a random vertex in the Delaunay triangulation. The first generalizes well in higher dimensions, while the second becomes a data sensitive parameter (constant for random points, $n^{\lceil (d-1)/2 \rceil}$ in the worst case). A generalization for computing the trapezoidal map can also be done.

Code

A demo version compiled for Sun Solaris is available at
http://www.inria.fr/prisme/logiciels/del-hierarchy/.

Acknowledgements

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] Dominique Attali and Jean-Danial Boissonnat. Complexity of the delaunay triangulation of points on a smooth surface. Research Report, to appear, INRIA, 2001.

[3] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa. The Quickhull algorithm for convex hull. Technical Report GCG53, Geometry Center, Univ. of Minnesota, July 1993.

[4] 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.

[5] Jean-Daniel Boissonnat and Monique Teillaud. A hierarchical representation of objects: The Delaunay tree. In *Proc. 2nd Annu. ACM Sympos. Comput. Geom.*, pages 260–268, 1986.

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

[7] P. Bose and L. Devroye. Intersections with random geometric objects. Technical report, School of Computer Science, McGill University, 1995. Manuscript.

[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. Constrained Delaunay triangulations. In *Proc. 3rd Annu. ACM Sympos. Comput. Geom.*, pages 215–222, 1987.

[10] O. Devillers. Improved incremental randomized Delaunay triangulation. Research Report 3298, INRIA, 1997.

[11] Olivier Devillers. Improved incremental randomized Delaunay triangulation. In *Proc. 14th Annu. ACM Sympos. Comput. Geom.*, pages 106–115, 1998.

[12] Olivier Devillers. On deletion in Delaunay triangulation. In *Proc. 15th Annu. ACM Sympos. Comput. Geom.*, pages 181–188, 1999.

[13] Olivier Devillers, Stefan Meiser, and Monique Teillaud. Fully dynamic Delaunay triangulation in logarithmic expectedtime per operation. *Comput. Geom. Theory Appl.*, 2(2):55–80, 1992.

[14] Olivier Devillers, Sylvain Pion, and Monique Teillaud. Walking in a triangulation. This volume.

[15] Olivier Devillers and Franco P. Preparata. A probabilistic analysis of the power of arithmetic filters. *Discrete Comput. Geom.*, 20:523–547, 1998.

[16] Olivier Devillers and Franco P. Preparata. Further results on arithmetic filters for geometric predicates. *Comput. Geom. Theory Appl.*, 13:141–148, 1999.

[17] Luc Devroye, Ernst Peter Mücke, and Binhai Zhu. A note on point location in Delaunay triangulations of random points. *Algorithmica*, 22:477–482, 1998.

[18] R. A. Dwyer. Higher-dimensional Voronoi diagrams in linear expected time. *Discrete Comput. Geom.*, 6:343–367, 1991.

[19] Jeff Erickson. Nice point sets can have nasty Delaunay triangulations. In *Proc. 17th Annu. ACM Sympos. Comput. Geom.*, pages 96–105, 2001.

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

[21] C. L. Lawson. Software for C^1 surface interpolation. In J. R. Rice, editor, *Math. Software III*, pages 161–194. Academic Press, New York, NY, 1977.

[22] Giuseppe Liotta, Franco P. Preparata, and Roberto Tamassia. Robust proximity queries: An illustration of degree-driven algorithm design. *SIAM J. Comput.*, 28(3):864–889, 1998.

[23] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, New York, NY, 1995.

[24] Ernst P. Mücke, Isaac Saias, and Binhai Zhu. Fast randomized point location without preprocessing in two- and three-dimensional Delaunay triangulations. In *Proc. 12th Annu. ACM Sympos. Comput. Geom.*, pages 274–283, 1996.

[25] K. Mulmuley. Randomized multidimensional search trees: Dynamic sampling. In *Proc. 7th Annu. ACM Sympos. Comput. Geom.*, pages 121–131, 1991.

[26] K. Mulmuley. *Computational Geometry: An Introduction Through Randomized Algorithms*. Prentice Hall, Englewood Cliffs, NJ, 1993.

[27] M. S. Paterson and F. F. Yao. On nearest-neighbor graphs. In *Proc. 19th Internat. Colloq. Automata Lang. Program.*, volume 623 of *Lecture Notes Comput. Sci.*, pages 416–426. Springer-Verlag, 1992.

[28] R. Seidel. The nature and meaning of perturbations in geometric computing. In *Proc. 11th Sympos. Theoret. Aspects Comput. Sci.*, volume 775 of *Lecture Notes Comput. Sci.*, pages 3–17. Springer-Verlag, 1994.

[29] J. R. Shewchuk. Triangle: Engineering a 2d quality mesh generator and Delaunay triangulator. In *First Workshop on Applied Computational Geometry*. Association for Computing Machinery, May 1996.

[30] P. Su and R. Drysdale. A comparison of sequential Delaunay triangulation algorithms. *Comput. Geom. Theory Appl.*, 7:361–386, 1997.