Theoretical Computer Science 112 (1993) 339–354
Elsevier

On the randomized construction of the Delaunay tree*

Jean-Daniel Boissonnat and Monique Teillaud

Institut National de Recherche en Informatique et Automatique, 2004 Route des Lucioles, B.P. 109, 06561 Valbonne Cedex, France

Communicated by F.P. Preparata
Received January 1990
Revised February 1991

Abstract

Boissonnat, J.-D. and M. Teillaud, On the randomized construction of the Delaunay tree, Theoretical Computer Science 112 (1993) 339–354.

The Delaunay tree is a hierarchical data structure which is defined from the Delaunay triangulation and, roughly speaking, represents a triangulation as a hierarchy of balls. It allows a semidynamic construction of the Delaunay triangulation of a finite set of n points in any dimension. In this paper, we prove that a randomized construction of the Delaunay tree (and, thus, of the Delaunay triangulation) can be done in $O(n \log n)$ expected time in the plane and in $O(n^{\lceil \frac{d}{2} \rceil})$ expected time in d-dimensional space. These results are optimal for fixed d. The algorithm is extremely simple and experimental results are given.

1. Introduction

Much attention has been paid, in the computational geometry literature, to the Voronoï diagram of a finite set of points and to its geometric dual, the Delaunay triangulation. For general references, we refer the reader to the books by Preparata and Shamos [22] and by Edelsbrunner [10]. Aurenhammer recently wrote a very complete survey of this fundamental data structure [2].

In the plane, Shamos [26] established an $\Omega(n \log n)$ lower bound for the time for computing a planar triangulation and provided an algorithm reaching this bound, thus optimal in the worst-case sense. Other optimal algorithms have been proposed

Correspondence to: J.-D. Boissonnat, Institut National de Recherche en Informatique et Automatique, 2004 Route des Lucioles, B.P. 109, 06561 Valbonne Cedex, France. Email addresses of the authors: boissonn@alcor.inria.fr and teillaud@alcor.inria.fr.

* This work has been supported in part by the ESPRIT Basic Research Action Nr. 3075 (ALCOM).

by Lee and Schachter [17], Preparata and Hong [21], Guibas and Stolfi [14] and Fortune [11].

In higher-dimensional spaces, Klee [15] has shown that the Delaunay triangulation may have $\Omega(n^{\lceil \frac{d}{2} \rceil}/\lceil \frac{d}{2} \rceil !)$ simplices. See also [20, 25]. It is well known that the problem of computing the Delaunay triangulation of a set of n points in d-space reduces to that of computing the convex hull of a set of n points, obtained by lifting the initial ones to a paraboloid of revolution in $d+1$-space. The beneath–beyond algorithm of Seidel [24] (see also [10]) computes the convex hull of a set of n points in d-space in time $O(n^{\lceil \frac{d}{2} \rceil})$ for $d>2$. This algorithm is incremental, and the insertion of a new site reduces to an intersection problem in the dual space. It can be used to compute the Delaunay triangulation of n points in d-space in time $O(n^{\lfloor \frac{d}{2} \rfloor +1})$, which is optimal for odd dimensions.

The previously mentioned algorithms are rather involved, even in the plane, and a much simpler one, proposed by Green and Sibson [12] in two dimensions, is often preferred. This algorithm is incremental and can be generalized to higher-dimensional spaces as shown by Bowyer [6]. Each point is introduced one after another and the triangulation is updated after each insertion. The worst-case complexity of the algorithm is $O(n^2)$ in the 2-dimensional case and $O(n^{\lceil \frac{d}{2} \rceil +1})$ in the d-dimensional case. In order to improve the algorithm, the authors perform a walk from neighbor to neighbor which yields an estimated complexity of $O(n^{1+1/d})$ for homogeneous distributions of points. However, no precise analysis can be found in the papers and the performances tend to become rather poor with degenerate distributions of points, such as points belonging to a subset of lower dimension. The same remarks hold even when using sophisticated bucketing techniques such as the ones used by Asano et al. [1].

A clear advantage of this algorithm, however, is that it is quite simple and allows insertions of new points in a dynamic way. This has motivated further investigations. A first idea [5] to improve the time complexity of the incremental algorithm consisted of keeping all the incremental versions of the triangulation in a structure called the Delaunay tree. In order to do that efficiently, the points were introduced in a random way. Other randomized techniques appeared recently [7, 18, 13, 19]. A common point to all these randomized algorithms is that no distribution assumptions are made as it is the case, for example, in Dwyer [9]. Hence, the results remain valid for any set of points, provided that the points are inserted at random.

The algorithms in [7, 18, 19] are incremental in the sense that the points are introduced one at a time. But all the points need to be known in advance and maintained in an auxiliary data structure, the so-called conflict graph. The algorithms in [5, 13] do not impose such a restriction and, thus, are more "on-line".

The algorithm used in [13] has not been generalized to higher dimensions. The algorithms in [7, 19] work in any dimensions. However, as already noticed, they are static.

The algorithm based on the Delaunay tree can compute the Delaunay triangulation of a set of points in any dimension. In this paper, we prove that a randomized construction of the Delaunay tree (and, thus, of the Delaunay triangulation) can be done in $O(n \log n)$ expected time in the plane and in $O(n^{\lceil \frac{d}{2} \rceil})$ expected time in d-dimensional space, for fixed d. These bounds are best possible for fixed d. Indeed, in the plane, computing the Delaunay triangulation of n points lying on a convex curve can be reduced to sorting, which takes $\Omega(n \log n)$ time, even for a randomized algorithm. In d-space, for fixed d, the complexity is identical to the size of the triangulation in the worst case and, thus, is clearly optimal.

The algorithm is extremely simple and experimental results are given.

2. The Delaunay tree

2.1. *Definition and fundamental results about the Delaunay triangulation*

We just recall the definition of the Delaunay triangulation and some general results of importance for the sequel. More details can be found in [23, 22]. Let $\mathscr{E}^d$ be a d-dimensional euclidean space and $\mathscr{M}$ a set of n sites $M_1, \ldots, M_n$. We will assume that no subset of $d+2$ sites are cospherical or lie on a same hyperplane. The *Voronoï diagram* associated with $\mathscr{M}$ is a sequence $V_1, \ldots, V_n$ of convex polyhedra covering $\mathscr{E}^d$, where V_i consists of all the points of $\mathscr{E}^d$ that have M_i as a nearest site in the set $\mathscr{M}$:

$$V_i = \{P \in \mathscr{E}^d, \forall j, 1 \leqslant j \leqslant n, \delta(P, M_i) \leqslant \delta(P, M_j)\},$$

where δ denotes the euclidean distance.

The geometrical dual of the Voronoï diagram, obtained by linking the sites M_i whose Voronoï polyhedra are adjacent, is called the *Delaunay triangulation* of $\mathscr{M}$. The joins are taken to be straight line segments and we use them as a framework for a simplicial subdivision of space. The Delaunay triangulation is a collection of simplices covering the convex hull of $\mathscr{M}$. The half-spaces, not containing $\mathscr{M}$, whose boundary contains a facet of the convex hull of $\mathscr{M}$, will be considered as *infinite* simplices (they are defined by d sites of $\mathscr{M}$ and a site at infinity). Together with the infinite simplices, the triangulation is a tesselation of the whole space $\mathscr{E}^d$. A fundamental property of the triangulation is related to the balls circumscribing the simplices of the triangulation (the *Delaunay balls*). Note that the balls circumscribing infinite simplices are half-spaces.

Fundamental property:
No site of $\mathscr{M}$ belongs to the interior of a Delaunay ball.

If a site lies inside the circumscribing ball of a simplex, we say that the site is *in conflict* with the simplex. In these terms, the Delaunay triangulation is the set of simplices without conflict.

2.2. *Construction of the Delaunay tree*

The Delaunay tree is a hierarchical structure based on the incremental procedure of [12, 6]. During the incremental algorithm, each site is introduced one after another and the triangulation is updated after each insertion. Let M be a site to be introduced in the triangulation. All the simplices in conflict with M can no longer be simplices of the triangulation (and are removed in Green and Sibson's algorithm). The union of these simplices is a simply connected region $R(M)$. Let $F(M)$ denote the set of facets of the boundary of $R(M)$. The new simplices are obtained by linking M to the facets of $F(M)$ (Fig. 1).

The Delaunay tree is constructed in a similar way. But, instead of removing simplices during the different steps of the construction, we store all the simplices which have been constructed as nodes of the Delaunay tree and at each step we define relationships between simplices of the successive Delaunay triangulations. The aim of this structure is to find $R(M)$ efficiently.

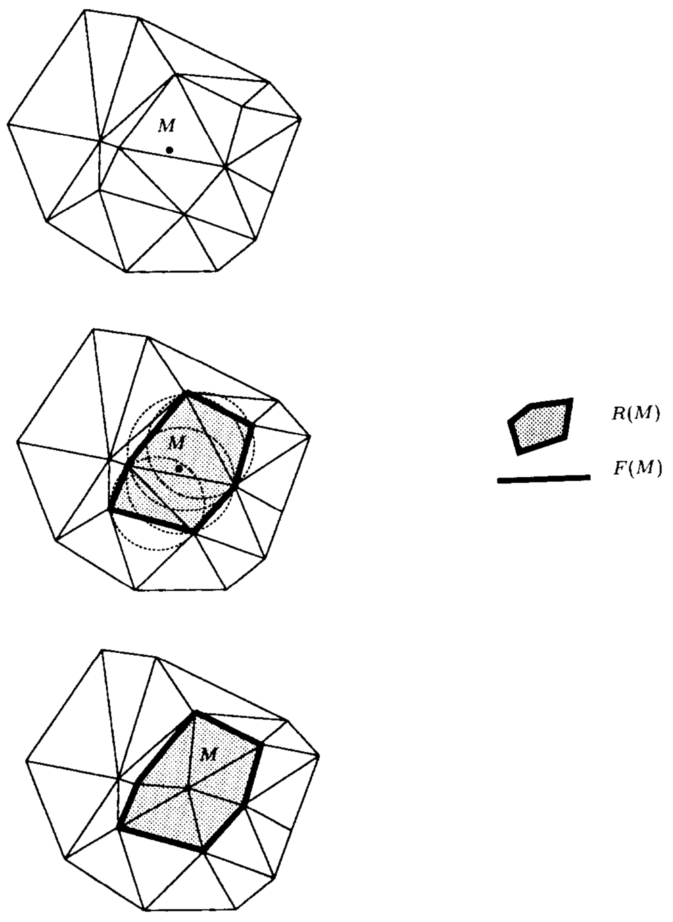


Fig. 1. Inserting a new site.

2.2.1. *Initialization step*

Let d denote the dimension of the space. For the initialization step we choose $d+1$ sites. They generate one finite simplex and $d+1$ infinite ones (see Fig. 2). These $d+2$ simplices will be the sons of the root of the tree.

2.2.2. *Inserting a new site M*

After the insertion of site M, the simplices in conflict with M are called *dead* and M is their *killer*. Observe that a killed simplex may not be incident to a facet of $F(M)$.

Let T be one of the simplices in conflict with M that has a facet F belonging to $F(M)$. We construct the new simplex S having vertex M and facet F. Let N be the simplex sharing facet F with T. Because the triangulation is a Delaunay one, we have the following property (see Fig. 3):

($\mathscr{P}$) The circumscribing ball of S is included in the union of the two balls circumscribing T and N.

This property is fundamental for the correctness of the algorithm, as will be seen in the sequel.

The newly created simplex S will be called *son of T* and *stepson of N* through facet F.

If we now insert a new site M' belonging to the circumscribing ball of S but not to that of N, S will be killed in turn, and its son S' having vertex M' and facet F will be another stepson of N. Thus, a node has at most one son and one list of stepsons through each facet, that is, 0–$d+1$ sons and 0–$d+1$ *lists* of stepsons.

Remark 2.1. Nevertheless, the total size of the stepson lists in the Delaunay tree is less than the number of nodes, since each newly created node (the $d+2$ first ones excepted) has exactly one stepfather. This is true in any dimension.

This hierarchical structure is called a *Delaunay tree* for short, but it is more exactly a rooted direct acyclic graph. This graph contains a tree: the tree whose links are the links between fathers and sons.

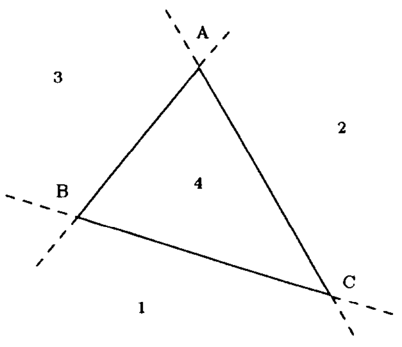


Fig. 2. Initialization step.

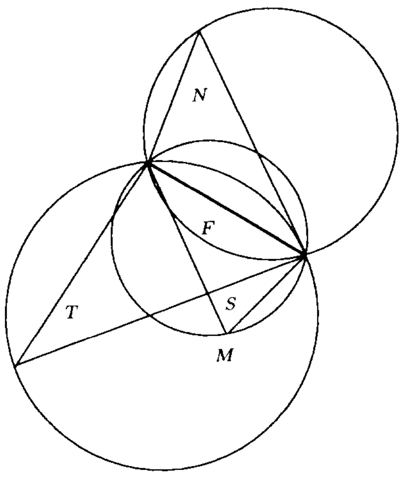


Fig. 3. Property ($\mathscr{P}$).

A simplex of the current triangulation is not dead and, so, corresponds to a node having no son, but possibly stepsons. We also maintain adjacency relationships between the simplices of the current triangulation. This will be developed in Section 2.3.2.

2.3. Using the Delaunay tree for constructing the Delaunay triangulation

Let M be a site to be introduced in the triangulation. Two steps are performed: first, we locate M in order to find the set $R(M)$ of all the simplices in conflict with M and then we create the new simplices.

2.3.1. Locating M

If M is in conflict with a simplex, we know, by Property ($\mathscr{P}$), that it is in conflict with its father or its stepfather. So, ($\mathscr{P}$) implies that we will be able to find all the simplices which are killed by M by exploring the Delaunay tree following Procedure location described in Fig. 4.

Remark 2.2. In fact, it is also possible to find only one simplex of $R(M)$ by searching the Delaunay tree, and to deduce the others using neighborhood relationships.

Remark 2.3. Let us show that all links from a node to its successive stepsons can be used to locate a new site. In Fig. 5, sites 1 7 are numbered in insertion order.

The subtree of the Delaunay tree corresponding to triangles (123), (234), (235), (256) and (237) is shown on the right-hand side of the figure.

If site 8 is now introduced, it is in conflict with (235) and (256), but not with (123). We can locate it using the link from (234) to (235) and the link from (235) to (256).

```
Procedure location($M$,$T$) :

    if $T$ has not been visited yet and $M$ is in conflict with $T$

        for each stepson $S$ of $T$ location($M$,$S$) ;

        for each son $S$ of $T$ location($M$,$S$) ;

        if $T$ is not dead then

            mark $T$ killed by $M$ ;

            add $T$ to the list $R(M)$ of the killed simplices

        endif

    endif
```

Fig. 4. Location of a site in the Delaunay tree.

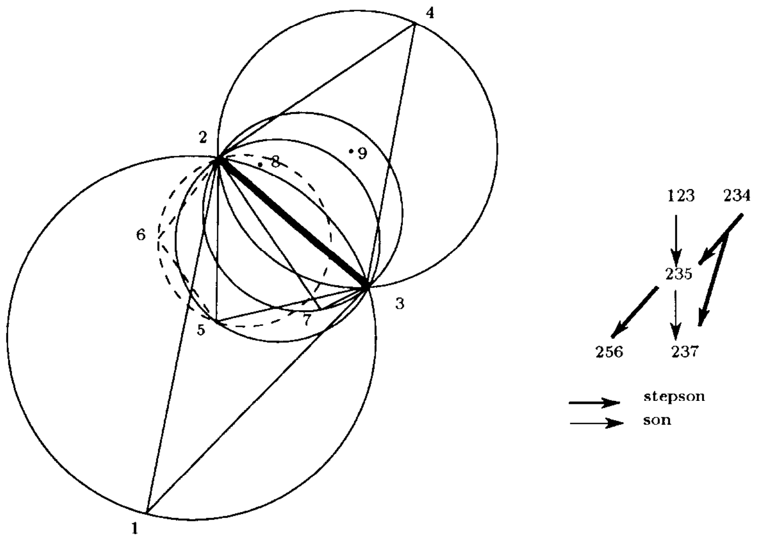


Fig. 5. All stepsons may be useful.

There exists no way to reach (256) from (234) through (237). If we want to locate 9, which is in conflict with (237) but not with (235), we must use the link from (234) to (237). Thus, it is necessary to store both stepsons of (234).

2.3.2. Creating the new simplices

We go through the list of the killed simplices. The facets of these simplices which will remain in the new simplices are those of $F(M)$. Let F be such a facet. We create

```
Procedure creation($M$):

    for each simplex $T$ killed by $M$
        for each neighbor $N$ of $T$ through a facet $F$
            if $M$ is not in conflict with $N$
                create the simplex $S$ having vertex $M$ and facet $F$ ;
                replace the adjacency relation between $N$ and $T$
                    by the adjacency relation between $N$ and $S$ ;
                create the relations: $S$ son of $T$ and $S$ stepson of $N$ through facet $F$
            endif
        endfor
    endfor ;
    create the adjacency relationships between the new simplices.
```

Fig. 6. Creating the new simplices.

the new simplex $S=(M, F)$ and the two relations between S, its father and its stepfather. The father of S is the killed simplex T containing facet F. The stepfather of S is the neighbor N of T through facet F (recall that we know the adjacency relationships in the current triangulation). Procedure creation is described in Fig. 6.

In order to create the adjacency relationships between the new simplices, we proceed as follows. Let F be a $(d-1)$-face of $F(M)$, and f be a $(d-2)$-face of F, we compute the sequence $T_1, T_2, \ldots, T_k$ of killed simplices, such that

- F is a facet of T_1,
- T_i is adjacent to T_{i-1} for $i=2, \ldots, k$,
- $f \subset T_i$ for $i=1, \ldots, k$,
- T_k has a facet F' belonging to $F(M)$, such that $F' \cap F = f$.

We declare that the new simplices MF and MF' are adjacent through facet Mf. By repeating this process for each $(d-1)$-face F of $F(M)$ and each $(d-2)$-face f of F, we obtain all the adjacency relationships between the new simplices.

Each killed simplex has at most $d(d+1)/2$ $(d-2)$-faces in $F(M)$ and, thus, is examined at most $d(d+1)/2$ times during this process.

3. Analysis of the randomized construction of the Delaunay triangulation

This section proves the main result of this paper, stated in the following theorem.

Theorem 3.1. *The randomized expected time for inserting the nth site in the Delaunay tree is* $O(\log n)$ *in two dimensions and* $O(n^{\lceil \frac{d}{2} \rceil - 1})$ *in dimension* $d > 2$. *The expected storage of the Delaunay tree is* $O(n^{\lceil \frac{d}{2} \rceil})$ *in dimension* $d \geqslant 2$.

(The constants depend on the dimension as a $(d+5)!/\lceil \frac{d}{2} \rceil!$ factor for the time complexity and a $1/(\lceil \frac{d}{2} \rceil - 1)!$ factor for the space complexity. We will omit the computations of these constants in this paper.)

A short proof for the complexity of the whole construction could be given by referring to Clarkson and Shor's [7] theorem for randomized incremental construction. Nevertheless, we sketch an alternate proof, which provides a nonamortized result.

We first introduce some additional notations and definitions. We define the width of a simplex to be the number of sites of $\mathcal{M}$ in conflict with that simplex $\mathcal{S}_k(\mathcal{M})$ will denote the set of simplices of width k and $\mathcal{S}_{\leqslant k}(\mathcal{M})$ the set of simplices of width at most k. $|\mathcal{A}|$ is the cardinality of a set $\mathcal{A}$. We define $s_k(n) = \max_{|\mathcal{M}|=n} |\mathcal{S}_k(\mathcal{M})|$, and $s_{\leqslant k}(n) = \max_{|\mathcal{M}|=n} |\mathcal{S}_{\leqslant k}(\mathcal{M})|$.

We also define a bicycle as a pair of simplices sharing a facet. A site is said to be in conflict with the bicycle, if it is in conflict with one of the two simplices but is not one of their vertices. $\mathcal{B}_k(\mathcal{M})$ will denote the set of bicycles of width k and $\mathcal{B}_{\leqslant k}(\mathcal{M})$ the set of bicycles of width at most k. We define $b_k(n)$ and $b_{\leqslant k}(n)$ in the same way as for the simplices.

Note that, when a site M is inserted, each new simplex S is the son of a simplex T and the stepson of a simplex N; ST and SN (Fig. 3) are bicycles without conflict (while NT has one conflict with M). More generally, an edge of the Delaunay tree corresponds to a bicycle.

The first lemma, due to Clarkson and Shor [7, Theorem 3.1], bounds the numbers of simplices and bicycles with width at most k defined by a set of n sites. The proof of this lemma uses the random sampling technique, and the fact that the number of simplices arising in the Delaunay triangulation of n sites is $O(n^{\lceil \frac{d}{2} \rceil})$ (see, for example, [15]) and, thus, that the number of bicycles of width zero has the same complexity (d is fixed).

Lemma 3.2. *The number of simplices having width at most k is, for* $k \geqslant 2$, *at most*

$$s_{\leqslant k}(n) = O(n^{\lceil \frac{d}{2} \rceil} k^{\lfloor \frac{d}{2} \rfloor + 1}).$$

The number of bicycles having width at most k is, for $k \geqslant 2$, *at most*

$$b_{\leqslant k}(n) = O(n^{\lceil \frac{d}{2} \rceil} k^{\lfloor \frac{d}{2} \rfloor + 2}).$$

Remark 3.3. For $k < 2$, we can trivially deduce the following bounds:

$$s_0(n) \leqslant s_{\leqslant 1}(n) \leqslant s_{\leqslant 2}(n) = O(2^{\lfloor \frac{d}{2} \rfloor + 1} n^{\lceil \frac{d}{2} \rceil})$$

and, similarly, for bicycles.

Remark 3.4. In the two-dimensional case, from [16], $s_k(n)$ and $b_k(n)$ are $O(k(n-k))$.

The sequel analyzes the insertion of the last site M of the set $\mathcal{M}$ of the n already inserted elements.

3.1. *Analysis of the randomized expected cost of Procedure* location

We compute the expected number of nodes $\mu(n)$ visited to locate the nth site M. A simplex S is visited during the location of M if S is the son or the stepson of a simplex T such that M is in conflict with T. So, we need here to count the number of edges of the Delaunay tree traversed during the location of the last site M, which is also the number of bicycles in conflict with M.

Let $B=ST$ be a bicycle of $\mathcal{B}_k(\mathcal{M})$. B is in conflict with M if one of the k sites in conflict with B is M, which happens with probability k/n. B appears before the insertion of M if the $k-1$ other sites in conflict with B have been inserted after the $d+2$ sites defining B; S is the son or the stepson of T if the site that created S has been inserted after the $d+1$ vertices of T. This occurs with probability $(d+1)!(k-1)!/(k+d+1)!$. The expected number of edges traversed by Procedure location is then obtained in the following way:

$$\mu(n)=\sum_{k=0}^{n-d-2}\sum_{B=ST\in\mathcal{B}_k(\mathcal{M})}\frac{k}{n}\,Prob\begin{pmatrix}S\text{ arises as a son}\\ \text{or a stepson of }T\\ \text{during the construction}\end{pmatrix}$$

$$=\sum_{k=0}^{n-d-2}\frac{(d+1)!\,|\mathcal{B}_k(\mathcal{M})|}{n(k+1)\cdots(k+d+1)}$$

Using Lemma 3.2, we deduce that $\mu(n)$ is $O(n^{\lceil \frac{d}{2}\rceil-1})$ in dimension d, and $O(\log n)$ in dimension 2.

3.2. *Analysis of the randomized expected cost of Procedure* creation

We first compute the number of nodes created by the nth site M. A given simplex is created by M if its width is zero after the insertion of M, and if M is one of its $d+1$ vertices. The expected number of created simplices is $\nu(n)=[(d+1)/n]\,|\mathcal{S}_0(\mathcal{M})|$, which is bounded by $[(d+1)/n]\,s_0(n)=O(n^{\lceil \frac{d}{2}\rceil-1})$.

The second part of the cost of inserting M is due to the killed simplices, which must be examined $O(d^2)$ times to create the adjacency relationships between the new simplices. The width of a simplex killed by M is one after the insertion of M; so, the expected number of simplices killed by M is $(1/n)|\mathcal{S}_1(\mathcal{M})|\leqslant(1/n)|\mathcal{S}_{\leqslant 1}(\mathcal{M})|$, which is less than $(1/n)s_{\leqslant 1}(n)=O(n^{\lceil \frac{d}{2}\rceil-1})$.

The cost of Procedure creation, for the insertion of the nth site is, thus, $O(n^{\lceil \frac{d}{2}\rceil-1})$ in dimension d.

3.3. *Analysis of the expected space used by the Delaunay tree*

The expected number $\eta(n)$ of nodes in the Delaunay tree is the number of all successive simplices arising in the Delaunay triangulation during the construction, and can be computed by summing the numbers $v(i)$ of simplices created by the ith site:

$$\begin{aligned}\eta(n) &= \sum_{i=1}^{n} v(i)\\ &= \mathrm{O}\left(\sum_{i=1}^{n} i^{\lceil d/2 \rceil - 1}\right)\\ &= \mathrm{O}(n^{\lceil d/2 \rceil}).\end{aligned}$$

We already know that the total number of edges in the Delaunay tree is linear in the number of nodes (Remark 2.1). The expected total storage is, thus, $\mathrm{O}(\eta(n))$.

This achieves the proof of Theorem 3.1.

Remark 3.5. The results stated in Theorem 3.1 are obtained by using worst-case bounds for the size of a Delaunay triangulation ($s_0(n)$ is used to bound $|\mathscr{S}_0(\mathscr{M})|$).

If we assume that the set of sites to be triangulated is uniformly distributed in the space, we know from the results of [9] that $|\mathscr{S}_0(\mathscr{M})|$ (and, thus, $|\mathscr{B}_0(\mathscr{M})|$) is $\mathrm{O}(n)$. Consequently, the random sampling technique used in Lemma 3.2 provides a linear complexity for the size of $\mathscr{S}_{\leqslant k}(\mathscr{M})$ and $\mathscr{B}_{\leqslant k}(\mathscr{M})$ and our algorithm runs in $\mathrm{O}(n \log n)$ expected time in any dimension.

4. Experimental results

The algorithm has been implemented in both dimensions 2 and 3. It is to be noted that the algorithm is extremely simple.

Moreover, the numerical computations involved are also very simple: When a simplex is created, we can compute the coordinates of the center of its circumscribing sphere, and its squared radius. This is achieved by first writing the equation of a sphere in d dimensions, then writing that the $d+1$ sites defining the simplex belong to the sphere, and solving the linear system which results from that, with $d+1$ equations and $d+1$ unknowns, in $\mathrm{O}(d^3)$ time.

The center and the squared radius of a simplex is computed once for all and stored in the corresponding node. For testing if a site is in conflict with a simplex, we only have to compute its squared distance to the center of the simplex, which costs $\mathrm{O}(d)$, and compare it with the squared radius. The constant announced in Section 3 has been computed according to this method.

The algorithm has run on many examples with different kinds of point distributions, in two and three dimensions.

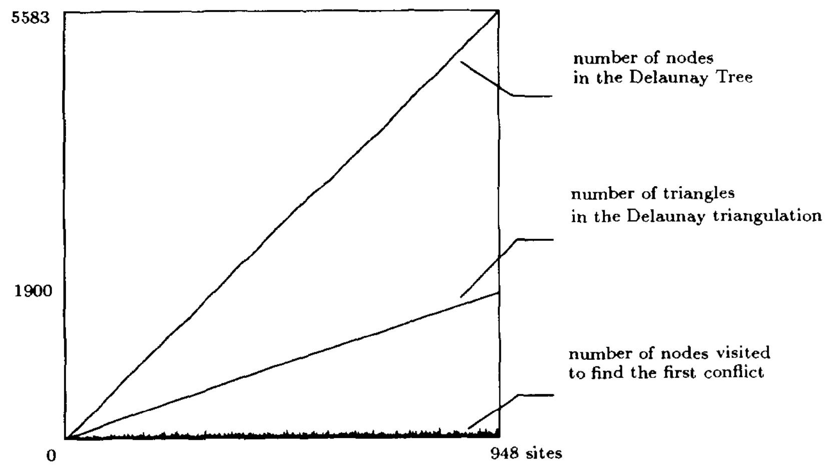


Fig. 7. Random sites in the plane.

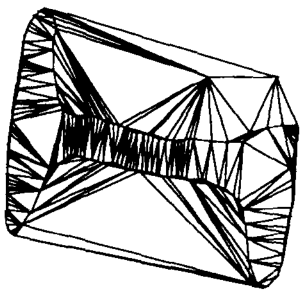
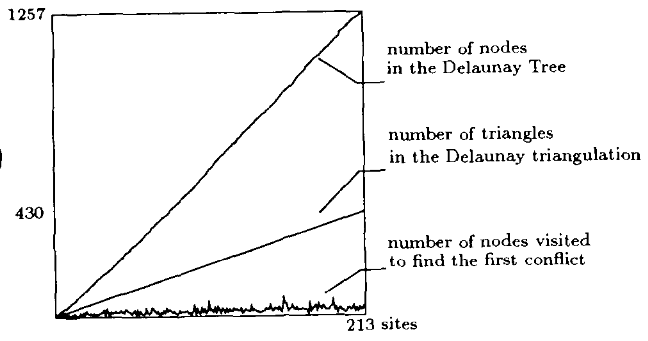


1257
430
213 sites
number of nodes in the Delaunay Tree
number of triangles in the Delaunay triangulation
number of nodes visited to find the first conflict

Fig. 8. A nonconvex curve.

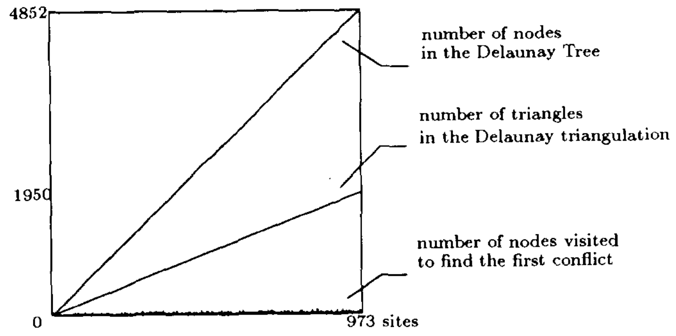


Fig. 9. An ellipsis.

In each case, several orders of insertion of the sites have been tried to analyze the running time down to the constants. All randomized orders yielded the same results. Some results are presented on Figs. 7–9 for the planar case.

Figures 10–12 present results in 3-space. Figures 10 and 11 show results in the case

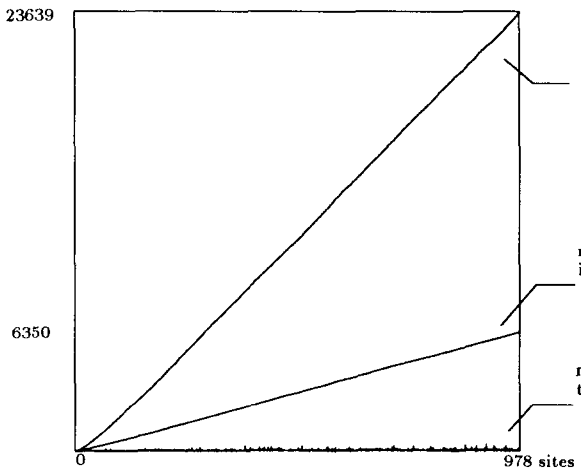
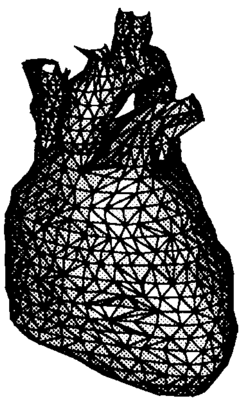
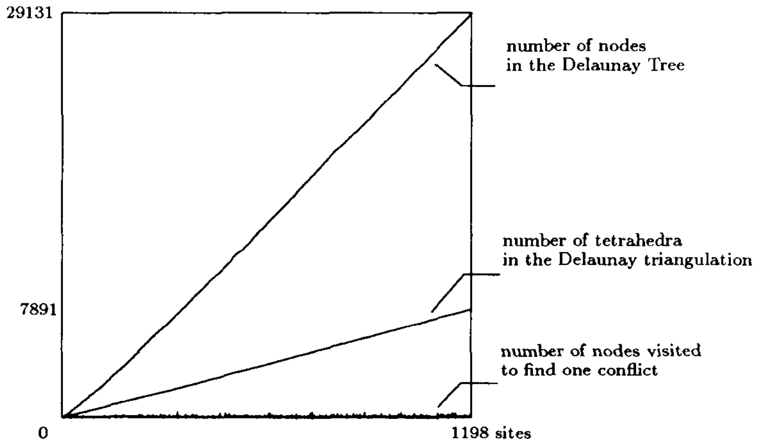


Fig. 10. Random sites in 3-space.





29131
7891
0
1198 sites
number of nodes in the Delaunay Tree
number of tetrahedra in the Delaunay triangulation
number of nodes visited to find one conflict

Fig. 11. A closed surface (a heart).

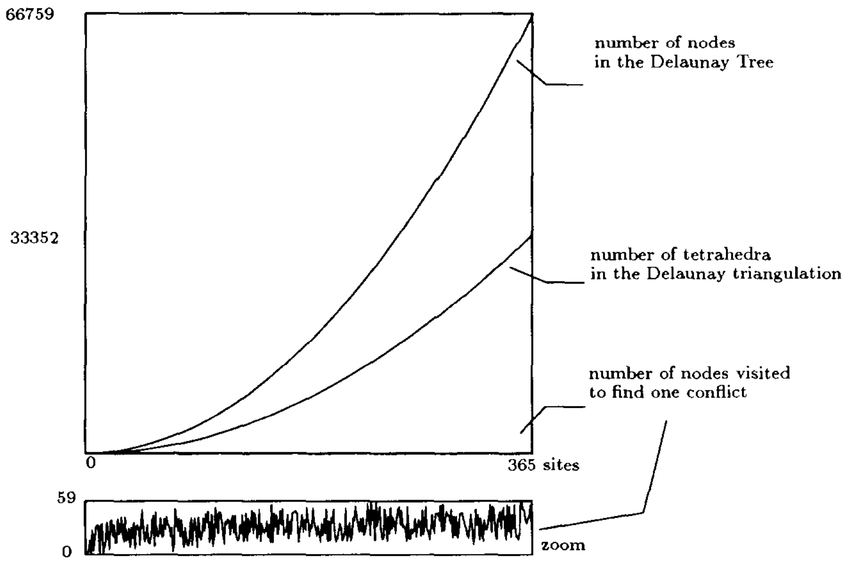


Fig. 12. A quadratic example.

where the size of the final Delaunay triangulation is linear in the number of sites, whereas Fig. 12 was obtained for a distribution where the number of tetrahedra was quadratic (the sites are lying on two noncoplanar line segments).

4.1. Storage of the Delaunay tree

The number of nodes in the Delaunay tree has been studied with respect to the number of inserted sites. This must be compared with the number of simplices in the final triangulation, which is the size of the output.

In the two-dimensional case, the ratio of the number of nodes in the Delaunay tree to the size of the output is less than 3, in any example. In the space, this ratio becomes 4 in the linear case, and only 2 in the quadratic case.

4.2. Inserting a site in the Delaunay tree

The chosen parameter to evaluate the cost of inserting a site is the number of nodes visited by Procedure location to find the first conflict. According to Remark 2.2, the remainder cost of this procedure consists of an output sensitive search in the Delaunay triangulation, and the cost of Procedure creation also depends on the number of modifications in the Delaunay triangulation.

The cost of locating a site in the Delaunay Tree appears to be very small compared to the size of the Delaunay triangulation.

Some empirical investigations have been done to evaluate the constants. In the plane, the variations of the number of nodes visited to find the first conflict can be roughly (after smoothing) assimilated to the variations of the function $x \mapsto 3\log_2 x$, in all examples. In the space, in the linear case, the function is $x \mapsto 7\log_2 x$, which has been explained in Remark 3.5.

We have computed in Section 3 the expected number of nodes visited by Procedure location to find all the simplices in conflict with a new site M, but no theoretical analysis has been done for the number of nodes visited to find the first conflict. We can see that, even in the quadratic case, this number remains logarithmic. Moreover, the constant appears to be better than in the linear case, since it is about 4.

5. Conclusion

We have shown that the Delaunay tree of n points can be constructed in randomized expected time $O(n \log n)$ in the plane and $O(n^{\lceil \frac{d}{2} \rceil})$ in d-space for $d>2$. Its randomized expected size is $O(n^{\lceil \frac{d}{2} \rceil})$ in dimension $d \geqslant 2$. The Delaunay tree allows to compute the Delaunay triangulation of n points within the same bounds, which is optimal for fixed d. A byproduct of this result is that the Delaunay tree can be used to locate a point inside a Delaunay triangulation in randomized expected time $O(\log n)$ in the plane and $O(n^{\lceil \frac{d}{2} \rceil - 1})$ in d-space.

An important point is that these results hold whatever the point distribution may be.

The algorithm is extremely simple and, moreover, the numerical computations involved are also very simple.

Experimental results in 2- and 3-dimensional spaces, for uniform as well as degenerate distributions of points, have provided strong evidence that this algorithm is very effective in practice.

In [8], the Delaunay tree has been made fully dynamic in the planar case, with a $O(\log \log n)$ expected complexity for removing a site.

Further investigations have been done to generalize the Delaunay tree to solve other applications efficiently. The k-Delaunay tree can be used to compute higher order Voronoï diagrams [4], and the IDAG is a more general data structure that can be used to design a large class of on-line randomized algorithms [3].

Acknowledgment

The authors thank Olivier Devillers for a careful reading of the paper, and Jean-Pierre Merlet for supplying us with his interactive drawing preparation system JP draw. We are also very grateful to the anonymous referee, who suggested many improvements in the earlier version of the paper.

References

[1] T. Asano, M. Edahiro, H. Imai and M. Iri, Practical use of bucketing techniques in computational geometry, in: G.T. Toussaint, ed., *Computational Geometry* (North-Holland, Amsterdam, 1985) 153–196.

[2] F. Aurenhammer, Voronoï diagrams – a survey of a fundamental geometric data structure, *ACM Comput. Surveys* **23** (1991) 345–405.

[3] J.D. Boissonnat, O. Devillers, R. Schott, M. Teillaud and M. Yvinec, Applications of random sampling to on-line algorithms in computational geometry, *Discrete Comput. Geom.* **8** (1992) 51–71.

[4] J.D. Boissonnat, O. Devillers and M. Teillaud, A semi-dynamic construction of higher order Voronoï diagrams and its randomized analysis, *Algorithmica*, to be published; available as Tech. Report INRIA 1207; abstract published in *2nd Canadian Conf. on Computational Geometry* (Ottawa 1990) 278–281.

[5] J.D. Boissonnat and M. Teillaud, A hierarchical representation of objects: the Delaunay tree, in: *2nd ACM Symp. on Computational Geometry* (Yorktown Heights, 1986) 260–268.

[6] A. Bowyer, Computing Dirichlet tesselations, *Comput. J.* **24** (2) (1981) 162–166.

[7] K.L. Clarkson and P.W. Shor, Applications of random sampling in computational geometry, II, *Discrete Comput. Geom.* **4** (5) (1989) 387–421.

[8] O. Devillers, S. Meiser and M. Teillaud, Fully dynamic Delaunay triangulation in logarithmic expected time per operation, in: *Computational Geometry Theory and Applications*, to be published; available as Tech. Report INRIA 1349; abstract published in Lecture Notes in Computer Science, Vol. 519 (*WADS '91*, 1991).

[9] R.A. Dwyer, Higher-dimensional Voronoï diagrams in linear expected time, *Discrete Comput. Geom.* **6** (1991) 343–367.

[10] H. Edelsbrunner, *Algorithms on Combinatorial Geometry* (Springer, Berlin, 1987).

[11] S. Fortune, A sweepline algorithm for Voronoï diagrams, *Algorithmica* **2** (1987) 153–174.

[12] P.J. Green and R. Sibson, Computing Dirichlet tesselations in the plane, *Comput. J.* **21** (1978) 168–173.

[13] L.J. Guibas, D.E. Knuth and M. Sharir, Randomized incremental construction of Delaunay and Voronoï diagrams, *Algorithmica* **7** (1992) 381–413.

[14] L.J. Guibas and J. Stolfi, Primitives for the manipulation of general subdivisions and the computation of Voronoï diagrams, *ACM Trans. Graphics*, **4** (2) (1985) 74–123.

[15] V. Klee, On the complexity of d-dimensional Voronoï diagrams, *Arch. Math.* **34** (1980) 75–80.

[16] D.T. Lee, On k-nearest neighbor Voronoï diagrams in the plane, *IEEE Trans. Comput.* **C-31** (1982) 478–487.

[17] D.T. Lee and B.J. Schachter, Two algorithms for constructing a Delaunay triangulation, *Internat. J. Comput. Inform. Sci.* **9** (3) (1980) 219–242.

[18] K. Mehlhorn, S. Meiser and C. Ó'Dúnlaing, On the construction of abstract Voronoï diagrams, *Discrete Comput. Geom.* **6** (1991) 211–224.

[19] K. Mulmuley, On obstruction in relation to a fixed viewpoint, in: *Proc. IEEE Symp. on Foundations of Computer Science* (1989) 592–597.

[20] I. Paschinger, Konvexe Polytope und Dirichletsche Zellenkomplexe, Ph.D. Thesis, Institut für Mathematik, Universität Salzburg, Austria, 1982.

[21] F.P. Preparata and S.J. Hong, Convex hulls of finite sets of points in two and three dimensions, *Comm. ACM* **2** (20) (1977) 87–93.

[22] F.P. Preparata and M.I. Shamos, *Computational Geometry: an Introduction* (Springer, Berlin, 1985).

[23] C.A. Rogers, *Packing and Covering* (Cambridge University Press, Cambridge, 1964).

[24] R. Seidel, A convex hull algorithm optimal for point sites in even dimensions, Tech. Report 14, Dept. of Computer Science, Univ. of British Columbia, Vancouver, BC, 1981.

[25] R. Seidel, The complexity of Voronoï diagrams in higher dimensions, in: *Proc. 20th Ann. Allerton Conf. on Communication, Control, Computing* (1982) 94–95.

[26] M.I. Shamos, Computational geometry, Ph.D. Thesis, Dept. of Computer Science, Yale University, 1978.