The Delaunay tetrahedralization from Delaunay triangulated surfaces

Sunghee Choi
Department of Computer Sciences
The University of Texas
Austin, TX 78712
sunghee@cs.utexas.edu

ABSTRACT

Given a surface mesh F in R^3 with vertex set S and consisting of Delaunay triangles, we want to construct the Delaunay tetrahedralization of S.

We present an algorithm which constructs the Delaunay tetrahedralization of S given a bounded degree spanning subgraph T of F. It accelerates the incremental Delaunay triangulation construction by exploiting the connectivity of the points on the surface. If the expected size of the Delaunay triangulation is linear, we prove that our algorithm runs in $O(n \log^* n)$ expected time, speeding up the standard randomized incremental Delaunay triangulation algorithm, which is $O(n \log n)$ expected time in this case.

We discuss how to find a bounded degree spanning subgraph T from surface mesh F and give a linear time algorithm which obtains a spanning subgraph from any triangulated surface with genus g with maximum degree at most $12g$ for $g > 0$ or three for $g = 0$.

Categories and Subject Descriptors

F.2 [**Theory of Computation**]: Analysis of Algorithms and Problem complexity; F.2.2 [**Analysis of Algorithms and Problem complexity**]: Nonnumerical algorithms and problems—*geometrical problems and computations*

General Terms

Algorithms

Keywords

Delaunay triangulation

1. INTRODUCTION

Delaunay triangulation is a fundamental and well-studied problem in computational geometry. Given a surface mesh F with a vertex set S and consisting of Delaunay triangles, we consider the problem of computing the Delaunay tetrahedralization of S. The problem has important applications including mesh generation and medial axis construction.

For arbitrary point sets in 3D, the randomized incremental algorithm has been the most popular solution for the 3D Delaunay triangulation. It incrementally constructs the Delaunay triangulation by adding points one by one in random order. The cost of insertion is dependent on the cost of locating a new point to add in the Delaunay triangulation of the already inserted points. The standard randomized incremental algorithm which uses an optimal point location scheme (e.g., the history DAG [6]) is theoretically optimal in expected time: it takes $O(n \log n)$ expected time if the expected size of the Delaunay triangulation of n points is $O(n)$, and $O(n^{1+k})$ if the expected size is $O(n^{1+k})$ for $0 < k \leq 1$.

Though the complexity of the 3D Delaunay triangulation can be quadratic for the worst case, it appears to be linear, in practice. Some theoretical results support this. Dwyer showed that if points are generated uniformly at random from the unit ball the expected size of the Delaunay triangulation is linear [9]. Recently, many researchers have studied the complexity of the 3D Delaunay triangulation for the special case of the points on a surface. Golin and Na [13] proved that the expected complexity of the 3D Delaunay triangulation of random points on convex polytopes is $\Theta(n)$. Attali and Boissonnat proved that the 3D Delaunay triangulation of a "light ϵ-sample" has complexity $O(n^{7/4})$. Erickson showed, in contrast, that there exist smooth surfaces that have a uniform sample whose Delaunay triangulation has quadratic size [10]. But, he also proved that for any *fixed* smooth surface, the Delaunay triangulation of any uniform ϵ-sample has complexity $O(n^{3/2})$ as n goes to infinity [11].

We assume for the theoretical analysis of our algorithm that the expected size of the Delaunay triangulation is linear. Our experiments [5] with samples from surfaces suggest that this assumption is usually correct.

We accelerate the point location by exploiting the connectivity of the boundary surface on which the input points lie. This idea is inspired by Seidel's randomized incremental trapezoidation algorithm of polygons [19]. By periodically tracing the boundary of the polygon and locating its segments in advance, he obtains an expected $O(n \log^* n)$ time algorithm for trapezoidation of simple polygons ($\log^* n$ is the largest integer l so that $\log^{(l)} n \geq 1$). Devillers applied the idea to the problem of 2D Delaunay triangulation given the Euclidean minimum spanning tree [8]. Our algorithm similarly traces the surface to speed up the location of unadded points later.

The improved running time of our algorithm is only proved in the case where the surface mesh is composed of Delaunay triangles. This case is important since many surface reconstruction algorithms produce surfaces that are subsets of the Delaunay triangulation. Though many such algorithms require the 3D Delaunay triangulation as a preprocessing step, there are some that avoid it. For example, the algorithms by Bernardini et al [3] or by Funke and Ramos [12] construct surfaces consisting of Delaunay triangles in near-linear time without computing the 3D Delaunay triangulation of all samples. Also there are other ways to obtain Delaunay triangulated surfaces without the 3D Delaunay triangulation. Cheng et al. [4] triangulate skin surfaces using Delaunay triangles. Attali and Lachaud [2] give a modified marching cubes algorithm [16] to construct an iso-surface all of whose triangles satisfy the Delaunay constraint in linear time. Constructing the 3D Delaunay triangulation given these surfaces can be useful, for instance, in generating a 3D mesh or medial axis.

We present and analyze an algorithm which constructs the Delaunay tetrahedralization given a bounded degree spanning subgraph T consisting of Delaunay edges in section 2. We show that the algorithm runs in $O(n \log^* n)$ expected time, if the expected size of the Delaunay triangulation is linear. In section 3, we discuss how to obtain a bounded degree spanning subgraph T from a surface F consisting of Delaunay triangles. We give a linear time algorithm to find a spanning subgraph with maximum degree at most $12g$ for $g > 0$ or three for $g = 0$ in any triangulated surface with genus g.

2. ALGORITHM

For a point set P, we denote the Delaunay triangulation of P as $D(P)$. For the analysis, we restrict the boundary surface F to be a subset of the Delaunay triangulation $D(S)$ of S and assume that we are given a spanning subgraph T of F with bounded degree d. In section 3, we discuss how to obtain such a spanning subgraph T given F. We assume that the points of S are in general position. Let $s_1, s_2, ..., s_n$ be a random ordering of the vertices of S and let $S_i = \{s_1, ..., s_i\}$ for $0 \leq i \leq n$.

To efficiently locate s_i in $D(S_{i-1})$ to construct $D(S_i)$, randomized incremental Delaunay triangulation algorithms maintain a location query structure $Q(S_{i-1})$. For our analysis, we use the theoretically optimal history DAG by Clarkson et al. [6] which exploits all the intermediate Delaunay tetrahedra created for our point location query structure. To locate s_i in $D(S_{i-1})$, using the history DAG, we start at the root and traverse all the intermediate Delaunay tetrahedra whose circumspheres contain s_i until we find the one in $D(S_{i-1})$ that contains s_i. After locating s_i in $D(S_{i-1})$ using $Q(S_{i-1})$, we update both the Delaunay triangulation $D(S_{i-1})$ and the query structure $Q(S_{i-1})$ to $D(S_i)$ and $Q(S_i)$.

In the worst case when the size of $D(S_i)$ is $O(i^2)$, locating

```
Tracing DT() {
    1. Let $h = 1$.
    2. For $1 \leq i \leq n$ do
        (a) If $i = N_h$, trace $T$ through $D(S_{N_h})$ to determine
            $d_{N_h}(s_k)$ for all $k > N_h$. Let $h = h + 1$.
        (b) Locate $s_i$ in $D(S_{i-1})$ using $Q(S_{i-1})$ starting from
            $d_{N_{h-1}}(s_i)$.
        (c) Insert $s_i$ and update $D(S_i)$ and $Q(S_i)$.
}
```

Figure 1: Tracing Delaunay Triangulation (TracingDT) algorithm

s_i in $D(S_{i-1})$ using $Q(S_{i-1})$ takes $O(i)$ expected time and updating $D(S_i)$ and $Q(S_i)$ takes $O(i)$ expected time. Thus the 3D Delaunay triangulation of n points takes $O(n^2)$ expected time. But, if the expected size of $D(S_i)$ is $O(i)$, the location takes $O(\log i)$ expected time and the update takes $O(1)$ expected time. For the remainder of the analysis, we assume that the expected size of $D(S_i)$ is $O(i)$, for all i.

Our algorithm in Figure 1 is inspired by Seidel's trapezoidation algorithm for simple polygons [19]. Let $N_h = \lceil n / \log^{(h)} n \rceil$. We denote $d_i(v)$ of a point v to be the Delaunay tetrahedron of $D(S_i)$ which contains v. We periodically traverse the spanning subgraph T and locate all the uninserted points in the intermediate Delaunay triangulation in advance, so that when we actually add a new point, we start from the Delaunay tetrahedron located in the last tracing step instead of at the root of the history DAG.

2.1 Analysis

We first analyze the cost of each tracing step 2a. Since $N_{\log^* n+1} > n$, step 2a is done only $\log^* n$ times. Each tracing step N_h for $1 \leq h \leq \log^* n$ is done by traversing the bounded degree spanning subgraph T and recording $d_{N_h}(s_k)$ for all $k > N_h$. Thus the cost is determined by the number of intersections between T and $D(S_{N_h})$. Here we need the restriction that T is a subset of $D(S)$. To count the expected number of intersections, we make some definitions and prove the following lemma about the intersection of a triangle of $D(R)$ for $R \subseteq S$ and a Delaunay edge of $D(S)$.

A sphere is *empty* of S if its interior contains no point $s \in S$. The *region* of Delaunay triangle abc is the union of the two empty circumspheres of the two Delaunay tetrahedra containing abc.

LEMMA 1. *Let abc be a triangle in $D(R)$ for any $R \subseteq S$. If edge vw in $D(S)$ intersects abc, at least one of $\{v, w\}$ lies in the region determined by abc in $D(R)$.*

PROOF. Suppose both v and w lie outside the region determined by abc in R. Let $abcd$, $abce$ be the two Delaunay tetrahedra adjacent to abc in $D(R)$. Consider $K = \{a, b, c, d, e, v, w\}$ Then, the tetrahedra $abcd$, $abce$ and the edge vw are in $D(K)$. Let B be the circumsphere of $abcd$,

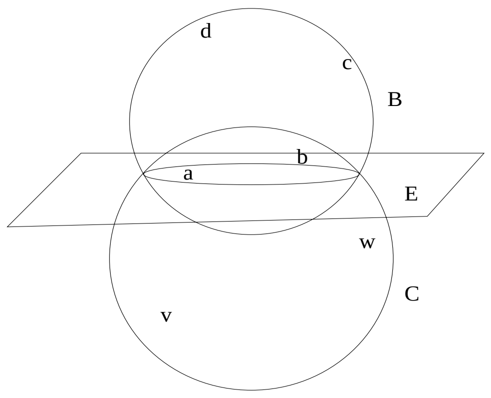

Figure 2: Proof of Lemma 1

and C be the circumsphere of a Delaunay tetrahedron $vwxy$ (x, y may be any pair of $\{a, b, c, d, e\}$) that contains vw in $D(K)$. Let E be the plane supporting the circle of the intersection of C and D. Both B and C are empty of K, because $abcd$ and $vwxy$ are in $D(K)$. So, $\{a, b, c, d\}$ and $\{v, w, x, y\}$ lie on the opposite sides of E as in Figure 2. Thus, vw cannot intersect abc. This is a contradiction. Therefore at least one of $\{v, w\}$ lies in the region determined by abc. $\square$

Lemma 1 may also be phrased and proved using the standard lifting which maps the Delaunay triangulation in R^3 to a lower convex hull in R^4.

LEMMA 2. *The expected number of intersections of $D(S_i)$ and T is $O(n)$.*

PROOF. By Lemma 1, if an edge vw in T intersects a triangle t in $D(S_i)$, at least one of {v,w} lies in the region determined by t in $D(S_i)$. So we count each edge of T adjacent to v as something that might intersect the triangle t, so each vertex v in the region of a triangle t can contribute to the number of intersections at most the degree $d(v)$ of v in T.

Let I denote the number of intersections of $D(S_i)$ and T. We bound I by summing over each triangle t of $D(S_i)$ the degrees $d(v)$ of vertices v of T that lie in the region of t. This is equivalent to summing over each of the vertices v of T the degree $d(v)$ of v times the number of regions containing v, which is bounded by the maximum degree d times the number of regions containing v. Since each Delaunay tetrahedron contains four Delaunay triangles, a vertex in a Delaunay circumsphere is counted as belonging to the four regions of the Delaunay triangles. Thus :

$$\begin{aligned} I &< \sum_{\substack{triangles\\ t\in D(S_i)}} \Big(\sum_{\substack{vertices\ v\in T\\ in\ region\ of\ t}} d(v)\Big) \\ &= \sum_{\substack{vertices\\ v\in T}} ((\#\text{ of regions containing } v) * d(v)) \\ &\le d \times \sum_{\substack{vertices\\ v\in T}} (\#\text{ of regions containing } v) \\ &= 4d \times \sum_{\substack{vertices\\ v\in T}} (\#\text{ of Del. circumspheres containing } v) \end{aligned}$$

We obtain the expected number of the Delaunay circumspheres of $D(S_i)$ that contain v using backwards analysis. If we insert vertex v in $D(S_i)$, the number of the Delaunay circumspheres of $D(S_i)$ which contains v is the number of Delaunay tetrahedra in $D(S_i)$ destroyed by the insertion of v. Since we are assuming that the expected size of $D(S_i)$ is $O(i)$, the expected number of the Delaunay tetrahedra destroyed by the insertion of v is $O(1)$. Hence, the expected number of intersections is $4nd \times O(1) = O(n)$. $\square$

THEOREM 3. *Tracing T in $D(S_i)$ takes $O(n)$ expected time.*

PROOF. Tracing time is determined by the number of intersections of $D(S_i)$ with T. By Lemma 2, the expected number of the intersections of $D(S_i)$ and T is $O(n)$. $\square$

So each tracing takes $O(n)$ expected time and since we trace $\log^* n$ times, all the tracing steps take $O(n \log^* n)$ expected time. Then the rest of the analysis is to determine the cost of locating and inserting a new point given the information from the tracing. This is basically the same as the proofs for the corresponding theorems of Seidel's trapezoidation algorithm [19] and Devillers' 2D Delaunay triangulation algorithm [8]. Nevertheless, we include the following lemma and theorem for completeness.

LEMMA 4. *If $d_k(s_i)$ is known, the expected cost of locating s_i in $D(S_{i-1})$ is $O(\log(i/k))$.*

PROOF. The cost of locating a point s_i in $D(S_{i-1})$ given $d_k(s_i)$ is proportional to the number of Delaunay tetrahedra in $D(S_j)$ for $k < j < i$ whose circumsphere contains s_i, which is $O(\log i) - O(\log k) = O(\log(i/k))$. $\square$

THEOREM 5. *The algorithm Tracing DT runs in $O(n \log^* n)$ expected time.*

PROOF. By Theorem 3, step 2a takes $O(n)$ expected time for fixed h and is executed $O(\log^* n)$ times, thus the total cost of for step 2a is $O(n \log^* n)$ expected time.

For $N_{h-1} < i \le N_h$, locating s_i takes $O(\log(i/N_{h-1}))$ expected time by Lemma 4. Since $i \le n$, $O(\log(i/N_{h-1}) =$

$O(\log(n/N_{h-1}) = O(\log^{(h)} n)$. For fixed h, step 2b is executed at most N_h times. Thus locating s_i for all $N_{h-1} < i \leq N_h$ takes $O(\log^{(h)} n) \times N_h = O(n)$ expected time. Since $h \leq \log^* n$, locating s_i for all $i \leq N_{\log^* n}$ takes $\log^* n \times O(n)$ expected time. For $N_{\log^* n} < i \leq n$, since $N_{\log^* n} \geq n/e$, locating s_i takes $O(\log(i/N_{\log^* n})) = O(1)$. Thus, the total cost of step 2b is $O(n \log^* n)$ expected time.

By the linear size assumption of the Delaunay triangulation, step 2c takes $O(1)$ expected time for each i, so the total cost is $O(n)$ expected time.

□

3. INPUT TO THE ALGORITHM

Even if the surface F is a subset of the Delaunay triangulation, it can have an arbitrarily large maximum degree (e.g., a point can have an arbitrary number of nearest neighbors). The analysis in the previous section is based on the assumption that we are given a spanning graph T of surface mesh F which is a subset of the Delaunay triangulation $D(S)$ and whose maximum degree d is bounded. Here we discuss how to obtain such input spanning graph T from F.

3.1 Special cases

One such spanning graph is the Euclidean minimum spanning tree of S. The Euclidean minimum spanning tree $EMST(S)$ of a point set S is the spanning tree of S that minimizes the sum of the lengths of the edges in the tree. Since it has bounded degree 6 in 2D and 12 in 3D [18], and is a subset of its Delaunay triangulation, the proofs of the previous section hold if we are given $EMST(S)$. Though $EMST(S)$ is usually constructed by computing $D(S)$ and running a minimum spanning tree algorithm on the edges of $D(S)$, it can also be computed without Delaunay triangulation in $O(n^{4/3} \log^{4/3} n)$ expected time. using the algorithm by Agarwal et al [1]. It is not of course guaranteed that a given surface triangulation F contains $EMST(S)$, but if F contains $EMST(S)$, we can find $EMST(S)$ from F in $O(n)$ expected time [14].

Some surface meshes already have bounded degree and we can just use any spanning tree of F as T. The iso-surfaces constructed from voxel data using the marching cubes algorithm [16] have a bounded degree. Though in general, the output of the marching cubes algorithm is not guaranteed to be Delaunay triangles, Attali and Lachaud's [2] modified marching cubes algorithm constructs iso-surfaces composed of Delaunay triangles.

LEMMA 6. *Let F be an iso-surface constructed from voxels. Then the degree of vertices in F is bounded.*

PROOF. By definition, the vertices of F are always on the voxel edges. A voxel edge is adjacent to at most 4 voxels. So for each vertex of F, the iso-surface edges adjacent to the vertex always lie in the 4 adjacent voxels. Each vertex lies in a loop in a voxel. Hence the maximum degree of a vertex is $2 \times 4 = 8$. □

```
bsg(G,C) {
    1. If $|V(G)| = 3$, output $H = G$, and we're done.
    2. If $C$ is a triangle, we choose a cycle vertex $v$ with degree
       greater than two as $z$.
       Otherwise, we choose $z$ from a list of cycle vertices
       with no chord.
    3. Let $z_1, z_2, ..., z_m$ be the neighbors of $z$ in the clockwise
       order such that $z_1, z_m \in V(C)$.
       Let $G' = G - z$ and $C'$ be the outer cycle of $G'$.
       Update chord information for new cycle $C'$.
       $H' = bsg(G', C')$
       If $m > 2$, output $H = H' - z_1z_2 - z_mz_{m-1} + z_1z +$
       $z_2z + z_mz$.
       If $m = 2$, $H = H' - z_1z_2 + z_1z + z_2z$.
}
```

Figure 3: Bounded degree spanning subgraph(bsg) algorithm: G is the triangulation homeomorphic to a disk and C is its boundary cycle.

Similarly, in case of the *locally uniform* samples defined by Funke and Ramos [12], it is shown that the surface Delaunay triangulation has a bounded degree. They also give a decimation algorithm which decimates dense samples into locally uniform samples in $O(n \log n)$ time.

3.2 Bounded degree spanning subgraph algorithm

If the surface F is not guaranteed to have bounded degree, we need to find a bounded degree spanning tree T from F. We give a linear time algorithm to construct a spanning subgraph with maximum degree at most three if $g = 0$ and $12g$ otherwise from any triangulation of a surface with genus g.

If the surface has genus greater than zero, Thomassen [20] showed that any triangulation of orientable surface S_g with a fixed genus g contains a spanning tree of maximum degree at most $5 \times 2^g - 2$. We obtain an improved bound $12g$ on maximum degree using the algorithm by Lazarus et al [15]. They find in $O(gn)$ time a set of $2g$ cycles on triangulated surfaces such that cutting the surface along them yields a topological disk called a polygonal schema. So we can convert triangulated surfaces with genus greater than zero into a topological disk in linear time. Since a vertex can appear in a cycle only once and there are $2g$ cycles, after cutting the surface along $2g$ cycles, a vertex may appear at most $4g$ times. Below we give a linear time algorithm to construct a spanning subgraph with maximum degree three given a triangulation homeomorphic to a disk. By identifying the vertices belonging to the cycles, we can obtain a spanning subgraph with maximum degree at most $4g \times 3 = 12g$.

Our algorithm for finding a spanning subgraph in a triangulation homeomorphic to a disk is described in Figure 3. It is inspired by the proof of Theorem 3.1 in [20] but gives a different constructive and somewhat simpler proof for the existence of a spanning subgraph with maximum degree three. We denote the set of the vertices of a graph G as $V(G)$. A

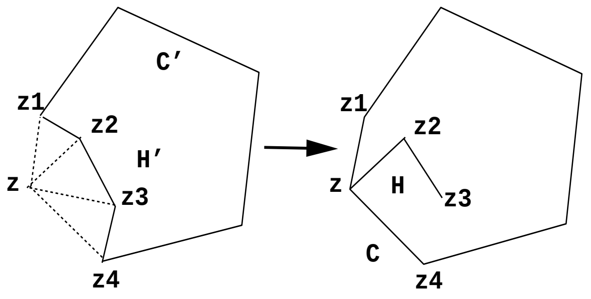

Figure 4: Constructing H from H'

chord of a vertex v in a cycle is a non-cycle edge between v and a vertex in the cycle. The algorithm is recursive. Each call $bsg(G, C)$ for the surface triangulation G and its outer cycle C outputs the spanning subgraph H with maximum degree at most three containing C. Initially, we start from the surface triangulation G. If $|V(G)| = 3$, G is the spanning subgraph H and we're done. Otherwise, we choose a cycle vertex z to remove from G to construct G' and its outer cycle C', and call $bsg(G', C')$ to obtain a spanning subgraph H' of G'. Then, we construct H from H' as in Figure 4; that is, we remove z_1z_2 and z_mz_{m-1} and add z_1z, z_2z and z_mz where $z_1, ..., z_m$ are the neighbors of z in a clockwise order.

LEMMA 7. *Given a triangulation G of S_0, we let any triangle of G be our initial cycle C. The algorithm $bsg(G, C)$ outputs a spanning connected subgraph of H with the maximum degree at most three.*

PROOF. First, to guarantee the termination of the algorithm, we need to show that during every recursive call to $bsg(G', C')$ we can find a vertex $z \in V(C')$ to remove. If C' is a triangle, there is a vertex in $V(C')$ that has degree greater than two, because otherwise $|V(C')| = 3$ and we're done. So assume that C' has n vertices (with $n > 3$). Since any triangulation of an n-gon has only $n-3$ diagonals, there is a vertex with no chord. Therefore, there is always a vertex $v \in V(C')$ to remove during every recursive call $bsg(G', C')$.

Second, we show that for each recursive call to $bsg(G', C')$, the outer cycle C' is homeomorphic to a circle. Whenever we remove a vertex z from C, its neighbors $z_1, .., z_m$ as defined in Figure 3 become the new vertices in the cycle. Since z has no chord, $z_2, ...z_{m-1}$ are not in C. C' is homeomorphic to a circle since it is obtained from C by removing zz_1, zz_m and adding $z_1z_2, z_2z_3, ..., z_{m-1}z_m$.

Finally, we prove that H is a spanning connected subgraph of F with maximum degree at most three. We first show that every recursive subcall to $bsg(G, C)$ returns a spanning subgraph H of G with maximum degree at most three such that $C' \subset H'$. We prove it by induction on $|V(G)|$. If $|V(G)| = 3$, $H = G$. So assume that $|V(G)| \geq 4$. Let z be any vertex with no chord and apply the induction hypothesis on $G - z$ to obtain H'. If the degree of vertex z is greater than two ($m > 2$, in Figure 3), $z_2, ..., z_{m-1} \in V(C)$. Then, H is obtained from H' by deleting the edges z_1z_2, z_mz_{m-1} and adding the edges z_1z, z_2z and z_mz. If the degree of vertex z is two ($m = 2$), then H is obtained from H' by deleting the edge z_1z_2 and adding the edges z_1z and z_2z. Either case, the degree of z in H is at most three, and the degrees of the rest of the vertices are the same as or less than in H'. Therefore, H is a spanning subgraph of G with maximum degree at most three. $\square$

LEMMA 8. *The algorithm $bsg(G, C)$ terminates in linear time.*

PROOF. For the analysis of the time complexity of the algorithm, we assume that for every vertex we are given a list of its neighbors in clockwise order. We maintain the cycle C as a doubly linked list and each vertex has markers for whether it is a cycle vertex and whether it has a chord. We also maintain a *no-chord list* - a list of cycle vertices that have no chord. At every step, we just choose the first vertex in no-chord list as z.

Removing a vertex z from G and modifying C to be the new outer cycle C' of $G' = G - z$ takes time proportional to the degree of z.

Also we need to update chord records for the new cycle C'. For each new cycle vertex $v \in V(C') \setminus V(C)$, we look at all its neighbors and if a neighbor w of v is a cycle vertex, we mark v as a chord vertex. If w was in no-chord list, we remove it from no-chord list. If v does not have a chord, we add v to the no-chord list. This takes $\sum_{v \in V(C') \setminus V(C)} d(v)$.

Since every vertex becomes a vertex in C just once and is removed just once, the total execution time is proportional to $\sum_{v \in V(G)} d(v) = O(2|E(G)|) = O(|V(G)|)$. $\square$

4. DISCUSSION

Our algorithm speeds up the location of a new point in randomized incremental Delaunay triangulation algorithms. But, the actual insertion cost of the new point and update of the Delaunay triangulation is not affected. Contrary to our expectation that the location time dominates the whole running time, our experiments [5] show that the location time and update time are pretty well balanced for several real Delaunay triangulation programs. Hence, this algorithm is not expected to improve the performance of the Delaunay triangulation programs in practice by more than a factor of two.

We conclude by asking an open question. When the points are generated by uniform distribution, we can compute $EMST$ in near-linear expected time [17; 7]. Can we compute $EMST$ of points from surface F more efficiently even if $EMST$ is not contained in F?

5. ACKNOWLEDGMENTS

The author would like to thank Nina Amenta for valuable insights, comments and advice.

6. REFERENCES

[1] P. Agarwal, H. Edelsbrunner, O. Schwarzkopf, and E. Welzl. Euclidean minimum spanning trees and

bichromatic closest pairs. *Discrete and Computational Geometry*, 6(5):407–422, 1991.

[2] D. Attali and J.-O. Lachaud. Constructing iso-surfaces satisfying the delaunay constraint; application to the skeleton computation. In *Proc. 10th International Conference on Image Analysis and Processing (ICIAP'99)*, pages 382–387. IEEE, 1999.

[3] F. Bernardini, J. Mittleman, H. Rushmeir, C. Silva, and G. Taubin. The ball-pivoting algorithm for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, 5(4):349–359, 1999.

[4] H.-L. Cheng, T. K. Dey, H. Edelsbrunner, and J. Sullivan. Dynamic skin triangulation. In *Proc. 12th Annu. ACM-SIAM Sympos. Discrete Algorithms*, 2001.

[5] S. Choi and N. Amenta. Delaunay triangulation programs on surface data. In *Proc. 13th Annu. ACM-SIAM Sympos. Discrete Algorithms*, January 2002.

[6] K. Clarkson, K. Mehlhorn, and R. Seidel. Four results on randomized incremental constructions. In *Proc. 9th Symposium on Theoretical Aspects of Computer Science*, 1992.

[7] K. L. Clarkson. An algorithm for geometric minimum spanning trees requiring nearly linear expected time. *Algorithmica*, 4(4):461–469, 1989.

[8] O. Devillers. Randomization yields simple $O(n \log^* n)$ algorithms for difficult $\Omega(n)$ problems. *International Journal of Computational Geometry and Applications*, 2(1):621–635, 1992.

[9] R. A. Dwyer. Higher-dimensional voronoi diagrams in linear expected time. *Discrete & Computational Geometry*, 6(4):343–367, 1991.

[10] J. Erickson. Nice point sets can have nasty delaunay triangulations. In *Proc. of 17th Annu. ACM Symposium on Computational Geometry*, June 2001.

[11] J. Erickson. Dense point sets have sparse delaunay triangulations. In *Proc. of the 13th. Annu. ACM-SIAM Symp. Discrete Algorithms*, January 2002.

[12] S. Funke and E. Ramos. Smooth-surface reconstruction in near-linear time. In *Proc. of the 13th Annu. ACM-SIAM Symp. Discrete Algorithms*, January 2002.

[13] M. Golin and H. Na. On the average complexity of 3d-voronoi diagrams of random points on convex polytopes. In *Proc. 12th Canadian Conf. Comput. Geom.*, pages 189–196, 1999.

[14] D. R. Karger, P. N. Klein, and R. E. Tarjan. A randomized linear-time algorithm to find minimum spanning trees. *Journal of the ACM*, 42(2):321–328, 1995.

[15] F. Lazarus, M. Pocciola, G. Vegter, and A. Verroust. Computing a canonical polygonal schema of an orientable triangulated surface. In *Proc. 17th Annu. ACM Symposium on Computational Geometry*, 2001.

[16] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *Computer Graphics (SIGGRAPH '87 Proceedings)*, 21(4):163–169, 1987.

[17] G. Narasimhan, M. Zachariasen, and J. Zhu. Experiments with computing geometric minimum spanning trees. In *Proceedings of ALENEX'00*, pages 183–196, 2000.

[18] G. Robins and J. S. Salowe. On the maximum degree of minimum spanning trees. In *Proc. 10th Annu. ACM Symposium on Computational Geometry*, pages 250–258, Stony Brook, New York, 1994.

[19] R. Seidel. A simple and fast randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Computational Geometry Theory and Applications*, 1(1):51–64, 1991.

[20] C. Thomassen. Trees in triangulations. *Journal of Combinatorial Theory*, Series B:56–62, 1994.