Learning Smooth Objects by Probing

Jean-Daniel Boissonnat
INRIA
2004, route des lucioles
06902 Sophia-Antipolis
boissonn@sophia.inria.fr

Leonidas J. Guibas
Dept. Computer Science
Stanford University
Stanford, CA 94305
guibas@cs.stanford.edu

Steve Oudot
INRIA
2004, route des lucioles
06902 Sophia-Antipolis
soudot@sophia.inria.fr

ABSTRACT

We consider the problem of discovering a smooth unknown surface S bounding an object $\mathcal{O}$ in $\mathbb{R}^3$. The discovery process consists of moving a point probing device in the free space around $\mathcal{O}$ so that it repeatedly comes in contact with S. We propose a probing strategy for generating a sequence of surface samples on S from which a triangulated surface can be generated which approximates S within any desired accuracy. We bound the number of probes and the number of elementary moves of the probing device. Our solution is an extension of previous work on Delaunay refinement techniques for surface meshing. The approximating surface we generate enjoys the many nice properties of the meshes obtained by those techniques, e.g. exact topological type, normal approximation, etc.

Categories and Subject Descriptors: I.3.5 [Computer Graphics]: Curve, surface, solid, and object representations

General Terms: Algorithms, Theory

Keywords: Manifold learning, blind surface approximation, interactive surface reconstruction, surface meshing, Delaunay refinement

1. INTRODUCTION

A great deal of work in computational geometry and related communities has focussed on the problem of surface reconstruction from scattered data points sampled on the surface. The computational geometry community was the first to describe local sampling conditions under which the geometry of the underlying surface can provably be approximated well and its topology fully recovered [2, 3]. These sampling conditions, however, may require prior information about the surface that is not readily available or may be verified and tested only after the fact (that is, after all the samples have been taken), if at all. As a result undesirable oversampling or undersampling may occur — in the former case sampling effort is wasted; in the latter provable reconstruction is impossible. In practice, the difficulty of testing these conditions means that the reconstruction algorithm is applied without concern for theoretical guarantees.

A different and much less explored approach is to use the sampling conditions to guide the sampling process as the samples are being generated. Certain physical acquisition processes can allow this type of fine control over the sampling process. In this paper we consider the problem of discovering the shape of an unknown object $\mathcal{O}$ of $\mathbb{R}^3$ through an adaptive process of probing its surface from the exterior. A probe is issued along a ray whose origin lies outside $\mathcal{O}$ and returns the first point of $\mathcal{O}$ hit by the ray. Successive probes may require the probing device to be moved through the free space outside $\mathcal{O}$. The goal is to find a strategy for the sequence of probes that guarantees a precise approximation of $\mathcal{O}$ after a minimal number of probes. Note that this problem involves an interesting bootstrapping issue, as the underlying surface is only known to the probing algorithm through the samples already taken. Thus, differently from most existing work in surface reconstruction, the data are not given all at once prior to the reconstruction phase but must instead be computed iteratively, each new probe depending on the outcomes of the previous probes. Furthermore, collision avoidance between the probing device and $\mathcal{O}$ must be observed at all times.

Given a surface of known positive reach (with a positive lower bound on its local feature size), the probing strategy proposed in this paper is inspired from Chew's algorithm [8] for Delaunay-based mesh refinement. Delaunay balls bounding surface facets are refined if they are too big. This refinement process is accomplished by moving our point probing device among current or prior edges of the dual Voronoi diagram known to lie in free space, before issuing a probe along the Voronoi edge dual to the facet to be refined. Our main contribution in this paper is the new probing algorithm proposed, the data structures used to find collision-free paths for the probing device, and the analysis of the total cost of this sampling procedure, including the number of probes made, the displacement cost for moving the device, and the combinatorial complexity of the construction. Our approach suggests numerous open problems that deserve further investigation.

1.1 Previous work

The above problem belongs to the class of geometric probing problems, pioneered by Cole and Yap [9]. Geometric probing, also known as blind approximation or interactive reconstruction, is motivated by applications in robotics. In this context, our probe model described above is called a tactile or finger probe. Geometric probing finds applications in other areas and gave rise to several variants. In particular, other probe models have been studied in the literature, e.g. line probes (a line moving perpendicular to a direction), X-ray probes (measuring the length of intersection between a line and the object), as well as their counterparts in higher dimensions.

We classify the probing algorithms into two main categories, exact or approximate, depending on whether they return the exact shape of the probed object or an approximation. An exact probing algorithm can only be applied to shapes that can be described by a finite number of parameters like polygons and polyhedra. In fact, most of the work on exact geometric probing is for convex polygons and polyhedra. See [18] for a survey of the computational literature on the subject. Although it has been shown that, using enhanced finger probes, a large class of non convex polyhedra can be exactly determined [1, 6], exact probing is too restrictive for most practical applications.

Approximate probing algorithms overcome this deficiency by considering the accuracy of the desired reconstruction as a parameter. The goal is to find a strategy that can discover a guaranteed approximation of the object using a minimal number of probes. The general problem is ill-posed, since we cannot conclude anything about the shape of the object if we have only local information about the shape. Some global information or prior knowledge is required to restrict the class of shapes being approximated. An important class is the class of convex shapes. Probing strategies have been proposed for planar convex objects using line probes [14, 16] and some other probe models are analyzed by Rote [17]. Observe that approximating a convex object using hyperplane probes is nothing else than approximating its supporting function.

As far as we know, probing non convex (non polyhedral) objects has not been studied. The problem has some similarity with surface approximation. In particular, the marching cube algorithm [15] and our recent Delaunay refinement surface mesh generator [5] provide blind approximations of a surface since the surface needs to be known only through an oracle that typically decides whether a line segment intersects the surface or not. However, the probing problem differs from surface approximation in an essential way: we cannot place the probing device at will anywhere but need to plan the motion of the probing device to its next probing location. Differently from the convex case, we cannot simply probe from infinity and need to determine finite positions outside the object where to place the probing device. Moreover, in order to reach such positions, we need to determine paths along which the probing device can be safely moved without colliding with the object.

1.2 Statement of the problem

Let $\mathcal{O}$ be a bounded open set of $\mathbb{R}^3$ and S its boundary. The goal is to approximate S by a probing tool that can locate points on S. The following assumption allows us to localize $\mathcal{O}$ within $\mathbb{R}^3$, preventing indefinite searches.

A1 *For every connected component $\mathcal{O}_i$ of $\mathcal{O}$, we know a point o_i that belongs to $\mathcal{O}_i$.*

Assumption A2 bounds the area of interest and allows us to obtain initial locations and paths for the probing device without bumping into $\mathcal{O}$.

A2 *We know a convex and compact subset Ω of $\mathbb{R}^3$ that contains S (and hence also $\mathcal{O}$). We denote by $\partial\Omega$ the boundary of Ω.*

We have at our disposal a *probing device*, which is an oracle that, once placed at some point p of $\mathbb{R}^3 \setminus \mathcal{O}$, can be oriented towards any direction d and then tasked to return the first point of transverse intersection between S and the ray defined by (p, d). The probing device can move freely in $\mathbb{R}^3 \setminus \mathcal{O}$ but cannot penetrate $\mathcal{O}$. Such a device can be constructed in practice, using for instance a laser with three DOFs of displacement and two DOFs of rotation, that can cast a ray in any direction and measure its distance to the point where the ray hits the object.

We assume that the probing device provides *exact information.* The outcome of a probe is a point on the boundary of the object.

We need also to define the *accuracy measure* for our reconstruction. The accuracy will be measured by the Hausdorff distance. Since the measured points are on the boundary S of the object, the accuracy of the reconstruction will be ε iff any point of S is at distance at most ε from a measured point. In other words, the set E of measured points is a ε-sample of S.

As mentioned above, to be able to make any reconstruction claims, we need to restrict the class of shapes we probe. We consider here those with *positive reach.* The reach of a surface S, denoted by $\mathrm{rch}(S)$, is the infimum over S of the distance of a point of S to the medial axis of S. The reach has been previously used in many contexts and has received various names: reach [12], normal injectivity radius [7], minimum local feature size [3], etc. Having a positive reach is ensured if S is $C^{1,1}$, i.e. S is C^1 and its normal vector field is Lipschitz [12].

A3 *We know a positive constant $\varepsilon_S \leq rch(S)$.*

Finally, we need a *model of computation* to analyze the complexity of our algorithm. Following the perception-action-cognition paradigm, we distinguish between the information or probing cost, the displacement cost, and the combinatorial cost. This distinction is also reminiscent of the difference made between combinatorial and informational complexity in the work on information-based computation [19, 20]. The probing cost measures the number of probes and indicates the amount of information that becomes available to our algorithm. The displacement cost accounts for the motion of the probing device. The combinatorial cost measures the arithmetic operations and comparisons required, as well as the maintenance cost of the data structures. It is not possible in general to optimize all costs simultaneously.

1.3 Overview of the paper

Under assumptions A1-A3, we show in this paper that S can be approximated by a triangulated surface $\hat{S}$ within any desired accuracy. Moreover, $\hat{S}$ recovers the exact topology of S and the error on the normal deviation of the facets of $\hat{S}$ is also bounded.

The paper is organized as follows. Since our solution is an extension of previous work on Delaunay refinement for surface meshing [5, 8], we recall Chew's algorithm and its main properties in Section 2. In Section 3 we describe the probing algorithm, present its main properties in Section 4, and analyze its complexity in Section 5. In these sections, the surface S is assumed to be connected, for simplicity. The case of a surface with more than one connected component will be analyzed in the journal version of the paper.

2. CHEW'S ALGORITHM

Chew's surface mesh generator is a greedy incremental algorithm that inserts sample points on S and maintains the *Delaunay triangulation* of the sample E *restricted* to S, defined below.

Data structure. Given a point set $E \subset S$, the Delaunay triangulation of E restricted to S, $\mathrm{Del}_{|S}(E)$, is the subset of the 3-dimensional Delaunay triangulation $\mathrm{Del}(E)$ of E made of the facets whose dual Voronoi edges intersect S. Every point of intersection of a Voronoi edge with S is the center of a *ball of* $Del_{|S}(E)$, *i.e.* a Delaunay ball centered on S. In practice, only a subset of $\mathrm{Del}_{|S}(E)$ can be computed, since S is known through an oracle ω that is not assumed to detect all the intersection points of S with the edges of the Voronoi diagram of E. The subset of $\mathrm{Del}_{|S}(E)$ that ω detects is noted $\mathrm{Del}^{\omega}_{|S}(E)$ and stored as a subcomplex of $\mathrm{Del}(E)$. Each time a point is added to E, only the part of the Voronoi diagram that has changed after the insertion of the point has to be queried by the oracle ω.

Algorithm. Chew's algorithm takes as input the surface S, a positive value ε, as well as an optional initial point sample E. If no initial point sample is given, then the algorithm constructs one in the same way as our probing algorithm – see Section 3.1. $\mathrm{Del}^{\omega}_{|S}(E)$ is then computed querying every edge of the Voronoi diagram of E using oracle ω.

At each iteration, the algorithm inserts a new point into E and updates $\mathrm{Del}^{\omega}_{|S}(E)$. Each point inserted into E is the center of a *bad ball of* $Del^{\omega}_{|S}(E)$, that is, a ball of $\mathrm{Del}_{|S}(E)$ whose center c has been detected by ω and whose radius is greater than ε. The algorithm stops when there are no more bad balls of $\mathrm{Del}^{\omega}_{|S}(E)$, which will eventually happen if ε is positive since S is compact. Upon termination, the algorithm returns E as well as $\hat{S} = \mathrm{Del}^{\omega}_{|S}(E)$.

Guarantees on the output. In [5], we proved that Chew's algorithm returns a triangulated complex $\hat{S}$ that is a good approximation of S provided that the following conditions are satisfied:

H1 *$\hat{S}$ is a manifold without boundary;*

H2 *$\hat{S}$ has vertices on all the connected components of S;*

H3 *Every facet f of $\hat{S}$ is circumscribed by a ball of $Del_{|S}(E)$, of center $c \in S$ and of radius at most ε for $\varepsilon < 0.091\ rch(S)$.*

THEOREM 2.1. *Under H1-H3, we have:*

- *$\hat{S}$ is ambient isotopic to S;*
- *the Hausdorff distance between $\hat{S}$ and S is at most $4.5\dfrac{diam(S)}{rch(S)^2}\ \varepsilon^2 = O\left(\varepsilon^2\right)$;*
- *$\hat{S}$ approximates S, in terms of normals and area, within an error of $O(\varepsilon)$;*
- *S is covered by the balls of $Del_{|S}(E)$ that circumscribe facets of $\hat{S}$;*
- *E is a 2ε-*sample *of S: $\forall x \in S$, $|E \cap B(x, 2\varepsilon)| \geq 1$.*

Moreover, it is proved in [5] that E is *sparse*: an r-sample E' is *sparse* if there is a constant κ that does not depend on S nor on r, such that $\forall x \in S$, $|E' \cap B(x, r)| \leq \kappa$. Thus,

THEOREM 2.2. *[Theorem 5.4 of [5]]*
$|E| = O\left(\iint_S \frac{dx}{\varepsilon^2}\right) = O\left(\frac{Area(S)}{\varepsilon^2}\right)$, *where the constant in the O does not depend on S nor on ε.*

In order to construct a PL approximation of S within a Hausdorff error of $\delta > 0$, it suffices to take:

$$\varepsilon = \mathrm{rch}(S)\sqrt{\frac{\delta}{4.5\ \mathrm{diam}(S)}}$$

The size of the output point set is then $O\left(\varepsilon^{-2}\right) = O\left(\delta^{-1}\right)$, which is optimal in the convex case [13]. Notice that the constraint on ε given by H3 yields a constraint on δ: $\delta < 0.04\ \mathrm{diam}(S)$. In practice, $\mathrm{rch}(S)$ can be approximated by ε_S, and $\mathrm{diam}(S)$ by the diameter of Ω.

3. THE PROBING ALGORITHM

For the sake of clarity, we assume that S is connected. We defer the treatment of several connected components to the full version of the paper. According to A1, we know a point $o \in \mathcal{O}$.

If we except the moves of the probing device, our algorithm is very similar to Chew's algorithm. The main difference concerns the oracle that is used to discover the surface S. In our case, to check whether a Voronoi edge e intersects S or not, we must first move our probing device to one of its endpoints. This requires two things: first, that at least one endpoint v of e be located in $\mathbb{R}^3 \setminus \mathcal{O}$; second, that we know a *free path* from $\mathbb{R}^3 \setminus \Omega$ (where the probing device can move freely) to v, *i.e.* a continuous curve included in $\mathbb{R}^3 \setminus \mathcal{O}$ that goes from $\mathbb{R}^3 \setminus \Omega$ to v.

DEFINITION 3.1. *Given a point set E, the* Voronoi graph *of E, $VG(E)$, is the graph made of the vertices and edges of the Voronoi diagram of E.*

Our basic intuition is to constrain the probing device to move along the edges of $\mathrm{VG}(E) \setminus \mathcal{O}$, which are called the *free edges*[1]. A difficulty arises from the fact that, when a new point p is inserted in E, some of the current Voronoi vertices and edges may disappear. It follows that portions of $\mathrm{VG}(E) \setminus \mathcal{O}$ that could be reached by the probing device from $\mathbb{R}^3 \setminus \Omega$ before the insertion of p may no longer be reachable afterwards — see Figure 1 for an illustration.

To overcome this difficulty, once a free path $\pi(v)$ from $\mathbb{R}^3 \setminus \Omega$ to some Voronoi vertex v has been found, we store $\pi(v)$ in memory so that v will remain reachable by the probing device permanently. Hence our paths are made of two types of edges: edges that belong to the current Voronoi graph, and edges that do not but were edges in some former Voronoi diagram.

By moving the probing device along our free paths, and by probing from each visited Voronoi vertex towards its neighbor vertices in $\mathrm{Vor}(E)$, we can detect a subset $\mathcal{I}$ of the points of $\mathrm{VG}(E) \cap S$ and construct a subcomplex of $\mathrm{Del}_{|S}(E)$ called the *visible restricted Delaunay triangulation of* E, or simply $\mathrm{Del}^{v}_{|S}(E)$. Every point of $\mathcal{I}$ is the center of a Delaunay

[1]More generally, any object (point, segment, curve etc.) that lies outside $\mathcal{O}$ is said to be *free*.

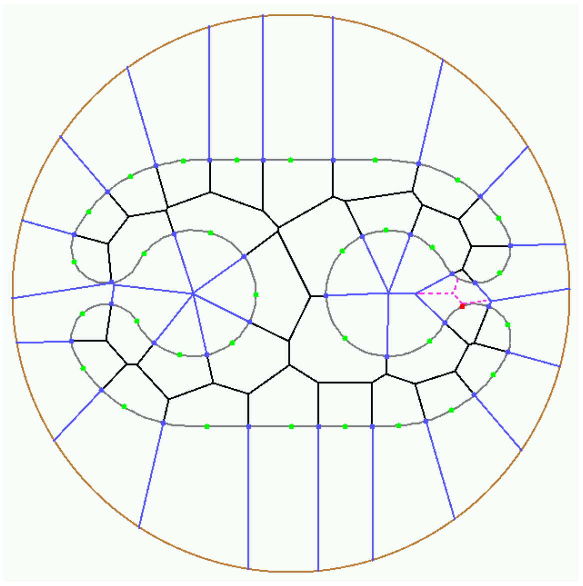

Figure 1: The insertion of the red point splits $\mathbf{VG}(E) \setminus \mathcal{O}$ into two connected components, one of which is then no longer reachable from $\mathbb{R}^3 \setminus \Omega$. Old Voronoi edges are dashed.

ball, called *ball of* $Del^v_{|S}(E)$, that circumscribes a facet of $\mathrm{Del}^v_{|S}(E)$.

3.1 Data structure

We proceed as in Chew's algorithm, by storing $\mathrm{Del}^v_{|S}(E)$ as a subcomplex of $\mathrm{Del}(E)$. Inside every Delaunay tetrahedron, we mark each of the four facets as being or not being part of $\mathrm{Del}^v_{|S}(E)$. This way, every Delaunay facet is marked twice since it belongs to two Delaunay tetrahedra.

In order to store the paths for the probing device, every Voronoi vertex[2] v is given a pointer *prev* to the previous vertex on a path from $\mathbb{R}^3 \setminus \Omega$ to v. By convention, $v.prev = NULL$ means that we know no free path from $\mathbb{R}^3 \setminus \Omega$ to v. In such a case, v is said to be *inactive*. Otherwise, v is called *active*.

If a newly created Voronoi vertex v belongs to $\mathbb{R}^3 \setminus \Omega$, then we set $v.prev \leftarrow v$ since v can be reached by the probing device. In particular, an infinite Voronoi vertex (*i.e.* the endpoint at infinity of an unbounded Voronoi edge) always lies outside Ω, which is compact. Thus, the *prev* field of an infinite vertex is never *NULL*. If v belongs to Ω, then we initialize $v.prev \leftarrow NULL$.

To construct and then update $\mathrm{Del}^v_{|S}(E)$, we use a routine named DETECT_ACCESS, introduced in Figure 2. Starting from an active vertex v_{start}, DETECT_ACCESS performs a depth-first traversal of $\mathrm{VG}(E) \setminus \mathcal{O}$ to see which previously inactive vertices can be reached by the probing device from v_{start} through free edges of the Voronoi graph.

Initial construction. Given an initial point set E of S, we compute $\mathrm{Del}^v_{|S}(E)$ by moving the probing device successively to all the vertices of $\mathrm{VG}(E)$ that lie outside Ω (including the infinite vertices[3]). For every such vertex v, we set $v.prev \leftarrow v$ and then we call DETECT_ACCESS on v.

```
DETECT_ACCESS ($v_{\text{start}}$):
 // Precondition: $v_{\text{start}}$ is active
foreach neighbor $v$ of $v_{\text{start}}$ do
    PROBE edge $[v_{\text{start}}, v]$;
    if ($[v_{\text{start}}, v] \cap S \neq \emptyset$) then
        add the dual of $[v_{\text{start}}, v]$ to $\mathrm{Del}^v_{|S}(E)$;
    else if ($v.prev = NULL$) then
      // $v$ becomes active
        if ($v \in \Omega$) then
            $v.prev \leftarrow v_{\text{start}}$;
        end if
        MOVE the probing device from $v_{\text{start}}$ to $v$;
        DETECT_ACCESS ($v$);
        MOVE the probing device from $v$ to $v_{\text{start}}$;
    end if
end foreach
```

Figure 2: Routine DETECT_ACCESS

After the initialization phase, every Voronoi vertex that can be reached from $\mathbb{R}^3 \setminus \Omega$ by walking along edges of $\mathrm{VG}(E) \setminus \mathcal{O}$ is active. Moreover, every active vertex is given a free path to $\mathbb{R}^3 \setminus \Omega$.

Update. Each time a new point p is to be inserted in E, we update $\mathrm{Del}^v_{|S}(E)$ as follows:

- before the insertion, we look at the active vertices of $\mathrm{Vor}(E)$ that no longer exist in $\mathrm{Vor}(E \cup \{p\})$. By definition, they lie in $\mathrm{V}(p)$, the cell of p in $\mathrm{Vor}(E \cup \{p\})$. We keep these vertices in memory and we leave their *prev* pointers unchanged. This way, every active vertex will remain active in the sequel and will keep its path to $\mathbb{R}^3 \setminus \Omega$.
- after the insertion, we look at the new vertices of the Voronoi diagram (including the infinite ones), which by definition are the vertices of $\mathrm{V}(p)$. For any such vertex v, we need to determine whether v can be reached from $\mathbb{R}^3 \setminus \Omega$ through edges of $\mathrm{VG}(E) \setminus \mathcal{O}$:
  - either $v \in \mathbb{R}^3 \setminus \Omega$, in which case we set $v.prev \leftarrow v$, we move the probing device to v and we call DETECT_ACCESS on v.
  - or $v \in \Omega$, in which case we look at the only neighbor v' of v that is not a vertex of $\mathrm{V}(p)$. If v' is active and if edge $[v, v']$ is free (which we can easily determine since $[v, v']$ is included in a former Voronoi edge that has been probed from v'), then we move the probing device to v' and we call DETECT_ACCESS on v'.

3.2 The algorithm

The algorithm takes as input a user-defined value ε such that $0 < \varepsilon < 0.091\, \varepsilon_S$, which by A3 is less than $0.091\,\mathrm{rch}(S)$. As explained in Section 2, ε can be chosen with respect to a certain threshold $\delta < 0.04\,\mathrm{diam}(S)$, such that the output of the algorithm will be a PL approximation of S within a Hausdorff error of δ.

[2]In practice, it is its dual Delaunay tetrahedron that we consider. However, for simplicity, we will identify Delaunay tetrahedra with Voronoi vertices in the sequel.

[3]Processing the infinite vertices in the same manner as the other ones simplifies the presentation but is not quite satisfactory since it involves moving the probing device to infinity. However, this can be avoided easily by clipping $\mathrm{VG}(E)$ by $\partial\Omega$ and calling DETECT_ACCESS on all the intersection points of $\partial\Omega \cap \mathrm{VG}(E)$.

The algorithm starts by computing an initial point set E made of three points of S. To do so, it places the probing device at a point p of $\partial\Omega$ and probes from p towards point o. Since $o \in \mathcal{O}$ and $p \in \mathbb{R}^3 \setminus \mathcal{O}$, the probing device finds a point $a \in S$, such that the segment $[p, a]$ is free. The algorithm then chooses two other directions, very close to direction $[p, a)$, so that the probing device will find two other points of S, namely b and c, such that triangle (a, b, c) is circumscribed by a sphere centered on S of radius at most $\varepsilon/3$. The algorithm sets $E = \{a, b, c\}$ and builds $\mathrm{Del}^v_{|S}(E)$ as described in Section 3.1. By construction, (a, b, c) is a facet of $\mathrm{Del}^v_{|S}(E)$. Moreover, as shown in [5] (Lemma 7.1), (a, b, c) will remain in $\mathrm{Del}_{|S}(E)$ throughout the process[4]. For this reason, we call it a *persistent facet.* The *bad* balls of $\mathrm{Del}^v_{|S}(E)$, *i.e.* the balls of $\mathrm{Del}^v_{|S}(E)$ whose radii are greater than ε, are stored in a priority queue $\mathcal{Q}$ where they are sorted by decreasing radius.

After the construction of the initial point set, the algorithm works as Chew's algorithm, using the probing device to answer the oracle. Specifically, the data structure is $\mathrm{Del}^v_{|S}(E)$, and the bad balls of $\mathrm{Del}^v_{|S}(E)$ are stored in $\mathcal{Q}$. While $\mathcal{Q}$ is not empty, the algorithm retrieves from $\mathcal{Q}$ the bad ball $B(c, r)$ of largest radius and inserts its center c in E. The algorithm then updates $\mathrm{Del}^v_{|S}(E)$ as described in Section 3.1, and updates $\mathcal{Q}$ as follows:

– the former bad balls that disappear because of the insertion of c are removed from $\mathcal{Q}$;
– the new bad balls that are created by the insertion of c are inserted in $\mathcal{Q}$.

The algorithm stops when $\mathcal{Q}$ is empty, that is, when no ball of $\mathrm{Del}^v_{|S}(E)$ is bad. The algorithm then returns E and $\mathrm{Del}^v_{|S}(E)$.

4. CORRECTNESS OF THE ALGORITHM AND QUALITY OF THE APPROXIMATION

In this section, we analyze the probing algorithm. We prove that it terminates in Section 4.1. In Section 4.2, we exhibit two invariants that are instrumental in proving the geometric properties of the output surface in Section 4.3. The analysis of the complexity of the algorithm is deferred to Section 5.

4.1 Termination

After the initialization phase, every point that is inserted in E belongs to S and is the center of a Delaunay ball of radius greater than ε. It follows that the points inserted in E are at distance at least ε from one another. Since ε is positive and S is compact, only finitely many points are inserted in E.

4.2 Invariants of the algorithm

Proposition 4.1. *The following assertions hold throughout the course of the algorithm:*

P1 *All active Voronoi vertices can be reached from $\mathbb{R}^3 \setminus \Omega$ by moving the probing device along current or former Voronoi edges.*

P2 *Any two Voronoi vertices that lie in the same connected component of $VG(E) \setminus \mathcal{O}$ have the same status,* active *or* inactive*.*

Proof. We proceed by induction. Clearly, (P1) and (P2) are verified after the initialization phase. Let us now consider a step of the algorithm during which a new point (say p) is inserted in E and $\mathrm{Del}^v_{|S}(E)$ is updated. Our induction hypothesis is the following:

IH *Assertions (P1) and (P2) hold in set E before the insertion of p.*

We will prove successively that (P1) and (P2) are still verified after the insertion of p. In the sequel, E denotes the point sample before the insertion of p.

(P1) Let v be a vertex that is active after the insertion of p.
P1.1 If v existed and was already active before the insertion of p, then its path $\pi(v)$ to $\mathbb{R}^3 \setminus \Omega$ remains unchanged since all the vertices of $\pi(v)$ are kept in memory and detect_access does not change the status of active vertices. It follows that v is reachable by the probing device from $\mathbb{R}^3 \setminus \Omega$ after the insertion of p, since it was so before by (IH).
P1.2 If v did not exist or was not active before the insertion of p, then v is visited by detect_access during the update of $\mathrm{Del}^v_{|S}(E)$. Since we run detect_access only on new vertices lying in $\mathbb{R}^3 \setminus \Omega$ and on former active vertices, v is given a free path either to a new vertex lying in $\mathbb{R}^3 \setminus \Omega$, or to a former active vertex which, as explained in P1.1, remains reachable by the probing device after the insertion of p. In both cases, v is reachable by the probing device from $\mathbb{R}^3 \setminus \Omega$.

(P2) Let us prove that the vertices v and w of any free edge e of $\mathrm{VG}(E \cup \{p\})$ have the same status after the update of $\mathrm{Del}^v_{|S}(E)$. It will then follow, by transitivity, that (P2) still holds after the insertion of p.
P2.1 If a vertex of e (say v) is visited by detect_access during the update of $\mathrm{Del}^v_{|S}(E)$, then it becomes active if not so before, and detect_access visits also w if the latter is not active. Thus, v and w are both active afterwards.
P2.2 If neither v nor w is visited by detect_access, then they keep their status during the update of $\mathrm{Del}^v_{|S}(E)$. Hence, it suffices to prove that they have the same status right before the update of $\mathrm{Del}^v_{|S}(E)$. If neither v nor w is a vertex of $\mathrm{V}(p)$, then they are both old Voronoi vertices, and e is an old edge, which implies that v and w have the same status, by (IH). If one of them belongs to $\mathrm{V}(p)$, then none can be active, since otherwise the algorithm would run detect_access on the one(s) that is(are) active, hereby contradicting the hypothesis of P2.2. □

4.3 Geometric properties of the output

As explained in Section 2, in order to guarantee that the algorithm constructs a good approximation of S, it suffices to prove that $\mathrm{Del}^v_{|S}(E)$ verifies assertions (H1), (H2) and (H3) upon termination of the algorithm. Now let E denote the output point sample.

Proof of H2. Since we assumed that S is connected, it suffices to check that $\mathrm{Del}^v_{|S}(E)$ is not empty when the algorithm halts. Recall that the algorithm constructs an initial

[4]Notice however that (a, b, c) is not guaranteed to remain in $\mathrm{Del}^v_{|S}(E)$.

point sample with a *persistent facet* (a, b, c) cicumscribed by a Delaunay ball B centered on S of radius at most $\varepsilon/3$. As shown in [5] (Lemma 7.1), (a, b, c) remains a facet of $\mathrm{Del}_{|S}(E)$ throughout the course of the algorithm. It follows that $\mathrm{VG}(E) \cap S$ is not empty upon termination of the algorithm. Since $\mathrm{VG}(E)$ is connected, at least one point p of $\mathrm{VG}(E) \cap S$ belongs to the same connected component of $\mathrm{VG}(E) \setminus \mathcal{O}$ as some infinite Voronoi vertex. By (P2), p can be "seen" from an active Voronoi vertex. Hence, $\mathrm{Del}^v_{|S}(E)$ is not empty, which proves (H2). □

Proof of H3. By definition, every facet of $\mathrm{Del}^v_{|S}(E)$ is circumscribed by a ball of $\mathrm{Del}^v_{|S}(E)$. Since the algorithm eliminates the balls of $\mathrm{Del}^v_{|S}(E)$ that have radii greater than ε, all the balls of $\mathrm{Del}^v_{|S}(E)$ have radii at most $\varepsilon < 0.091\ \varepsilon_S$ upon termination. By A3, ε is less than $0.091\ \mathrm{rch}(S)$. □

As established in Section 3 of [5], assertion (H3) alone induces a few local properties, such as:

L1 [Lemma 3.4 of [5]] *Two facets of $\mathrm{Del}^v_{|S}(E)$ that share an edge form a dihedral angle greater than $\frac{\pi}{2}$.*

L2 [Lemma 3.6 of [5]] *An edge of $\mathrm{Vor}(E)$ cannot intersect S in more than one point x such that $\mathrm{dist}(x, E) < 0.091\ \mathrm{rch}(S)$. Hence, every edge of $\mathrm{Vor}(E)$ contains at most one center of ball of $\mathrm{Del}^v_{|S}(E)$.*

L3 [Proposition 3.10 of [5]] *The balls of $\mathrm{Del}^v_{|S}(E)$ intersect S along* pseudo-disks*, i.e. topological disks that pairwise intersect along topological disks and whose boundaries pairwise intersect in at most two points.*

To prove (H1), we need yet another result, which is a direct consequence of assertion (P2):

L4 *Let ζ be a connected component of $\mathrm{VG}(E) \setminus \mathcal{O}$. Either all the points of $\partial\zeta \cap S$ are centers of balls of $\mathrm{Del}^v_{|S}(E)$, or none of them is.*

Proof. Let p and q be two points of $\partial\zeta \cap S$. By definition, p and q are centers of balls of $\mathrm{Del}_{|S}(E)$. If ζ contains no (finite or infinite) Voronoi vertex, then it is made of one piece of a Voronoi edge only. Therefore, p and q cannot be detected by the probing device, and none of them can be the center of a ball of $\mathrm{Del}^v_{|S}(E)$. If ζ contains some Voronoi vertices, then, by (P2), all the Voronoi vertices in ζ have the same status, *active* or *inactive*. In the first case, p and q are both centers of balls of $\mathrm{Del}^v_{|S}(E)$. In the second case, none of them is, which ends the proof of (L4). □

Using L1-L4, we can now prove Assertion (H1).

Proof of H1. We first show that every edge of $\mathrm{Del}^v_{|S}(E)$ is incident to exactly two facets of $\mathrm{Del}^v_{|S}(E)$. We then prove that every vertex of $\mathrm{Del}^v_{|S}(E)$ has only one *umbrella*. An umbrella of a vertex v is a set of facets of $\mathrm{Del}^v_{|S}(E)$ incident to v whose adjacency graph is a cycle.

Let e be an edge of $\mathrm{Del}^v_{|S}(E)$. We denote by $\mathrm{V}(e)$ the Voronoi facet dual to e. Notice that $\partial\mathrm{V}(e) \cap S \neq \emptyset$, since e belongs to $\mathrm{Del}_{|S}(E)$. It follows that any connected component ξ of $\partial\mathrm{V}(e) \setminus \mathcal{O}$ is a simple polygonal arc, whose endpoints lie on S and are centers of balls of $\mathrm{Del}_{|S}(E)$. Moreover, ξ is included in a connected component of $\mathrm{VG}(E) \setminus \mathcal{O}$. Thus, by (L4), either both endpoints of ξ are centers of balls of $\mathrm{Del}^v_{|S}(E)$, or none of them is. It follows that the total number of centers of balls of $\mathrm{Del}^v_{|S}(E)$ that lie on $\partial\mathrm{V}(e)$ is even. Then, by (L2), the number of edges of $\partial\mathrm{V}(e)$ that contain centers of balls of $\mathrm{Del}^v_{|S}(E)$ is even. Equivalently, the number of facets of $\mathrm{Del}^v_{|S}(E)$ that are incident to e is even.

In addition, two facets of $\mathrm{Del}^v_{|S}(E)$ incident to e form a dihedral angle greater than $\frac{\pi}{2}$, by (L1). It follows that e cannot be incident to more than three facets of $\mathrm{Del}^v_{|S}(E)$.

In conclusion, the number of facets of $\mathrm{Del}^v_{|S}(E)$ incident to e is even, at least 1 (because e is an edge of $\mathrm{Del}^v_{|S}(E)$), and at most 3. Hence it is 2.

Since this is true for any edge of $\mathrm{Del}^v_{|S}(E)$, the facets of $\mathrm{Del}^v_{|S}(E)$ incident to a given vertex v of $\mathrm{Del}^v_{|S}(E)$ form a set of umbrellas. Using (L3), one can prove that they form only one umbrella – see Proposition 4.2 of [5]. Basically, if U is an umbrella, then v lies in the interior of the projection of U onto $T(v)$, since otherwise there would be two consecutive facets of U overlapping each other, which would imply by (L3) that one of their vertices lies inside a *surface patch*[5] (which is impossible since surface patches are empty of points of E). It follows that v belongs to the interior of the union R of the surface patches of the facets of U, since these patches are pseudo-disks (by L3). Then, using (L3) again, it is not difficult to prove that any facet of $\mathrm{Del}^v_{|S}(E)$ incident to v that does not belong to U has one of its vertices in the interior of R, which contradicts the fact that surface patches are empty of points of E.

Therefore, a vertex of $\mathrm{Del}^v_{|S}(E)$ can only have one umbrella. It follows that $\mathrm{Del}^v_{|S}(E)$ is a 2-manifold without boundary. □

Since $\mathrm{Del}^v_{|S}(E)$ verifies H1-H3, it is a good approximation of S, according to Theorem 2.1. In particular, E is a 2ε-sample of S, and it is sparse, as mentioned in Section 2. This implies that $|E| = O\left(\frac{\mathrm{Area}(S)}{\varepsilon^2}\right)$, by Theorem 2.2. Moreover, if $\varepsilon < 0.05\ \varepsilon_S$, then E is a 0.1-sample of S, and hence $\mathrm{Del}_{|S}(E)$ is homeomorphic to S, by Theorem 2 of [3]. As a consequence, $\mathrm{Del}^v_{|S}(E)$ and $\mathrm{Del}_{|S}(E)$ are equal, since they are homeomorphic and since $\mathrm{Del}^v_{|S}(E)$ is a subcomplex of $\mathrm{Del}_{|S}(E)$.

5. COMPLEXITY OF THE ALGORITHM

As mentioned in the introduction, the complexity of the algorithm has three components: the *combinatorial cost* that measures the memory space and time needed to store, construct and update the data structures; the *probing cost* that counts the number of probes performed by the probing device; the *displacement cost* that measures the effort spent in moving the probing device. Depending on the context, one can give emphasis to one type of cost or the other.

Notice that it is not possible in general to optimize all costs simultaneously. Take for instance a parabola $\mathcal{C}$ embedded in $\mathbb{R}^2$, as shown in Figure 3. Any Delaunay-based algorithm that optimizes the displacements of the probing device will somehow follow the curve $\mathcal{C}$, inserting the points of E more or less in their order along $\mathcal{C}$ (see Figure 3, first

[5]The *surface patch* of a facet f of $\mathrm{Del}^v_{|S}(E)$ is the intersection of S with the ball of $\mathrm{Del}^v_{|S}(E)$ circumscribing f.

row). This makes the overall complexity of the incremental Delaunay triangulation quadratic. Differently, our algorithm will insert the points in an order defined by the *largest empty ball* criterion (see Figure 3, second row), which does not optimize the displacement cost but makes the combinatorial cost linear (in 2D).

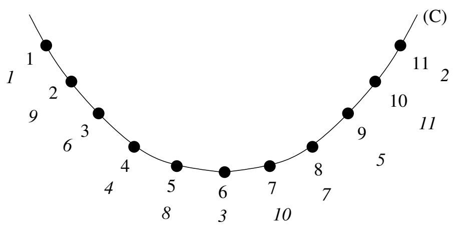

Figure 3: Two orders of insertion on a parabola

In the sequel, we analyze the combinatorial cost, probing cost and displacement cost separately. Since our algorithm enforces the probing device to move along the Voronoi edges, the size of the Voronoi diagram has a direct impact on all three costs.

5.1 Combinatorial cost

Space complexity

The data structure stores the current Delaunay triangulation as well as some of the former Voronoi vertices. Since every vertex is stored at most once, the size of the data structure is at most the total number of Voronoi vertices created during the course of the algorithm. We will bound this number w.r.t. the Hausdorff distance δ between $\hat{S}$ and S.

Let E_{init} be the initial point sample constructed by the algorithm. We have $|E_{\text{init}}| = 3$. For every iteration i of the algorithm, we call $E(i)$ the point set E at the end of iteration i. $E(i) \setminus E(i-1)$ contains precisely the point $p(i)$ inserted in E at iteration i, and $E(i-1) \setminus E_{\text{init}}$ is the set of all points inserted before iteration i. We call $r(i)$ the radius of the largest ball of $\text{Del}^v_{|S}(E(i))$. Since the algorithm always inserts the center of the ball of $\text{Del}^v_{|S}(E)$ of largest radius, $p(i)$ is at a distance $r(i-1)$ from $E(i-1)$.

Let Z be the set of all points of S with an osculating sphere bounding an open ball that does not intersect S. To bound the number of Voronoi vertices created, we will use the following result, stated as Lemma 17 in [?]:

Lemma 5.1. *There exist four constants ε_0, C_0, K_1 and K_2, depending only on S, such that, for any sparse ε-sample E of S, with $\varepsilon \leq \varepsilon_0$, the number of Delaunay edges incident to a vertex p of Del(E) is at most $K_1\ \varepsilon^{-1/2}$ if dist(p, Z) $\leq C_0 \sqrt{\varepsilon}$ and at most K_2 / dist(p, Z) otherwise.*

In the sequel, we take as ε_0 the minimum of the above (unknown) constant ε_0 and of 0.091 rch(S). Let i_0 be the first iteration of the algorithm at the end of which all the balls of $\text{Del}^v_{|S}(E)$ have radii at most $\frac{\varepsilon_0}{4}$. In other words, i_0 is the first iteration such that $r(i_0) \leq \frac{\varepsilon_0}{4}$. Since $\frac{\varepsilon_0}{4} <$ 0.091 rch(S), $E(i_0)$ is an $\frac{\varepsilon_0}{2}$-sample of S, by Theorem 2.1.

Lemma 5.2. *For any two iterations i and j such that $j \geq i \geq i_0$, we have $r(j) \leq 2r(i)$.*

Proof. Let $B(j)$ be a ball of $\text{Del}^v_{|S}(E(j))$ of largest radius. Its center $c(j)$ lies on S. Since $i \geq i_0$, we have $E(i_0) \subseteq E(i)$. Hence, $E(i)$ is an ε_0-sample of S, and the balls of $\text{Del}^v_{|S}(E(i))$ cover S, by Theorem 2.1. Thus, $c(j)$ lies in a ball $B(c,r)$ of $\text{Del}^v_{|S}(E(i))$. We have $\text{dist}(c(j), E(i)) \leq 2r \leq 2r(i)$. Moreover, since $i \leq j$, $E(i)$ is included in $E(j)$. It follows that $r(j) = \text{dist}(c(j), E(j)) \leq \text{dist}(c(j), E(i)) \leq 2r(i)$, which concludes the proof of the lemma. $\square$

Lemma 5.3. *For any iteration $i > i_0$, $E(i)$ is a $2r(i)$-sample of S, with $2r(i) \leq \varepsilon_0$, and the points of $E(i) \setminus E_{init}$ are farther than $\frac{r(i-1)}{2}$ from one another and from E_{init}.*

Proof. Let i be any iteration of the algorithm such that $i > i_0$. According to Lemma 5.2, we have $r(i) \leq 2r(i_0) \leq \frac{\varepsilon_0}{2}$, thus $E(i)$ is a $2r(i)$-sample of S, with $2r(i) \leq \varepsilon_0$, according to Theorem 2.1. In addition, by definition of i_0, every point of $E(i_0) \setminus E_{\text{init}}$, when inserted in E, is the center of a Delaunay ball of radius greater than $\frac{\varepsilon_0}{4} \geq r(i_0)$, which is at least $\frac{1}{2}\ r(i-1)$, by Lemma 5.2. Moreover, at any iteration k such that $i_0 < k \leq i$, the point inserted in E is the center of a Delaunay ball of radius $r(k-1)$, which is at least $\frac{1}{2}\ r(i-1)$, by Lemma 5.2. Therefore, the points of $E(i) \setminus E_{\text{init}}$ are at least $\frac{1}{2}\ r(i-1)$ away from one another and from E_{init}. $\square$

Let i be an iteration of the algorithm, such that $i \geq i_0$. Let $j > i$ be the first iteration such that $r(j) \leq \frac{r(i)}{8}$. By Lemma 5.3, $E(j)$ is a $2r(j)$-sample of S, with $2r(j) \leq \frac{r(i)}{4}$. We call $E(i,j)$ the set of the points inserted by the algorithm between iterations i (excluded) and j (included). We have $E(i,j) = E(j) \setminus E(i)$.

Lemma 5.4. *For any k such that $i < k \leq j$, $E(k)$ is a sparse $6r(i)$-sample of S.*

Proof. By Lemma 5.3, $E(k)$ is a $2r(k)$-sample of S. Since $2r(k) \leq 4r(i) \leq 6r(i)$ (Lemma 5.2), $E(k)$ is a $6r(i)$-sample. To prove that $E(k)$ is sparse, we count the points of $E(k)$ that lie in $B(x, 6r(i))$, for any $x \in S$.
- Since $|E_{\text{init}}| = 3$, the number of points of E_{init} that lie in $B(x, 6r(i))$ is at most 3.
- By Lemma 5.3, the points of $E(k) \setminus E_{\text{init}}$ are farther than $\frac{r(k-1)}{2}$ from one another. Now, $\frac{r(k-1)}{2}$ is at least $\frac{r(i)}{16}$, since $i < k \leq j$. It follows that the points of $E(k) \setminus E_{\text{init}}$ are centers of pairwise-disjoint balls of radius $\frac{1}{32}\ r(i)$. For every such ball B whose center lies in $B(x, 6r(i))$, B is included in $B(x, (6 + \frac{1}{32})\ r(i))$. It follows that the number of points of $E(k) \setminus E_{\text{init}}$ that lie in $B(x, 6r(i))$ is bounded by a constant, which shows that $E(k)$ is sparse and hereby concludes the proof of Lemma 5.4. $\square$

Lemma 5.5. *$E(i,j)$ is a sparse $6r(i)$-sample of S.*

Proof. Let u be a point of $E(i+1)$. By Corollary 4.13 of [5], u is a vertex of $\text{Del}_{|S}(E(i+1))$. $\text{Del}_{|S}(E(i+1))$ is a 2-manifold without boundary, thus u has at least three neighbors v_1, v_2, v_3 in $\text{Del}_{|S}(E(i+1))$. Since $|E_{\text{init}}| = 3$, at least one point among $\{u, v_1, v_2, v_3\}$ belongs to $E(i+1) \setminus E_{\text{init}}$. Let us call it w. By Lemma 5.3, w is farther than $\frac{r(i)}{2}$ from the other points of $\{u, v_1, v_2, v_3\}$. Thus, u is farther than $\frac{r(i)}{2}$ from one of its neighbors, say v_1. Any point $x \in S$ belonging to the Voronoi face $\text{V}(u) \cap \text{V}(v_1)$ is farther than $\frac{r(i)}{4}$ from u. Hence, x is closer to some point u' of $E(i+1,j)$ than to u, since otherwise $E(j)$ could not be a $\frac{r(i)}{4}$-sample of S. As a consequence, the distance from any point $y \in S \cap \text{V}(u)$ to $E(i+1,j)$ is at most:

$$\begin{aligned} \text{dist}(y, u') &\leq \text{dist}(y,u) + \text{dist}(u,x) + \text{dist}(x,u') \\ &\leq \text{dist}(y,u) + 2\ \text{dist}(u,x) \end{aligned}$$

Since $E(i+1)$ contains $E(i)$, $E(i+1)$ is a $2r(i)$-sample of S. Thus, $\mathrm{dist}(y,u) \le 2r(i)$ and $\mathrm{dist}(u,x) \le 2r(i)$, which implies that $\mathrm{dist}(y, E(i+1,j)) \le 6r(i)$. Since this is true for any $u \in E(i+1)$, $E(i+1,j)$ is a $6r(i)$-sample of S. So is $E(i,j)$, for $E(i,j) \supset E(i+1,j)$.

In addition, by definition of j, any point of $E(i,j)$, right before its insertion, is the center of a Delaunay ball of radius greater than $\frac{r(i)}{8}$. It follows that the points of $E(i,j)$ are farther than $\frac{r(i)}{8}$ from one another. Hence, by the same argument as in the proof of Lemma 5.4, $E(i,j)$ is sparse. □

Lemma 5.6. *During the insertion of the points of $E(i,j)$, the algorithm creates $O\left(|E(i,j)|\log|E(i,j)|\right)$ Delaunay edges.*

Proof. Let $\varepsilon_i = 6r(i)$. The reasoning is similar in spirit to that of Lemma 18 of [?], although with an additional subtelty. We assume that Z is a set of curves of finite length, and we decompose S into strips parallel to Z, of width $C_0\,\sqrt{\varepsilon_i}$, where C_0 is a constant defined in [?] that depends on S but not on ε_i. Let Z_k denote the k^{th} strip ($k \ge 0$). The points of Z_k lie at a distance of Z ranging from $kC_0\sqrt{\varepsilon_i}$ to $(k+1)C_0\sqrt{\varepsilon_i}$.

Since $E(i,j)$ is a sparse ε_i-sample of S (Lemma 5.5), its size is $C\varepsilon_i^{-2}$, for some constant C depending on S, and the number of points of $E(i,j)$ that lie in a given strip Z_k is $C\varepsilon_i^{-3/2}$. Moreover, for any i' such that $i < i' \le j$, $E(i')$ is a sparse ε_i-sample of S (Lemma 5.4), thus Lemma 5.1 applies to the point inserted at iteration i'. Summing the contributions of all the points of $E(i,j)$, we find that the number n of Delaunay edges created by the insertion of the points of $E(i,j)$ is at most:

$$C\varepsilon_i^{-3/2}\frac{K_1}{\sqrt{\varepsilon_i}}+\sum_{k\ge 1} C\varepsilon_i^{-3/2}\frac{K_2}{kC_0\sqrt{\varepsilon_i}} = K_1C\varepsilon_i^{-2}+\frac{K_2C\varepsilon_i^{-2}}{C_0}\sum_{k\ge 1}\frac{1}{k}$$

Recall that the size of $E(i,j)$ is $C\varepsilon_i^{-2}$. Moreover, the number of strips Z_k is $C' \,/\, C_0\sqrt{\varepsilon_i}$, where C' depends only on S. It follows that n is at most:

$$K_1\,|E(i,j)| + \frac{K_2}{C_0}\,|E(i,j)|\left(1+\sum_{2\le k\le \frac{C'}{C_0C^{1/4}}|E(i,j)|^{1/4}}\frac{1}{k}\right)$$

i.e. $O\left(|E(i,j)|\right) + O\left(|E(i,j)|\right)\left(1+O\left(\log|E(i,j)|\right)\right)$. □

Theorem 5.7. *The total number of Voronoi vertices created during the course of the algorithm is $O\left(N\log N\right)$, where $N = O\left(\varepsilon^{-2}\right) = O\left(\delta^{-1}\right)$ is the size of the output point set. This bound holds for the space complexity of the algorithm.*

Proof. We divide the output point set E into clusters. More precisely, i_0 is defined as above, and for any $k \ge 1$, we define i_k as the first iteration such that $r(i_k) \le \frac{r(i_{k-1})}{8}$. Let l be the last iteration of the algorithm. We assume without loss of generality that $l = i_K$, for some K. We have $E = E(i_0) \cup \bigcup_{0\le k<K} E(i_k, i_{k+1})$. By Lemma 5.6, every cluster $E(i_k,i_{k+1})$ generates $|E(i_k,i_{k+1})|\log|E(i_k,i_{k+1})|$ Delaunay edges. It follows that the overall number of Delaunay edges created is at most:

$$\begin{aligned} & |E(i_0)|^2 + \textstyle\sum_{0\le k<K}|E(i_k,i_{k+1})|\log|E(i_k,i_{k+1})| \\ \le\; & |E(i_0)|^2 + \textstyle\sum_{0\le k<K}|E(i_k,i_{k+1})|\log|E| \\ \le\; & |E(i_0)|^2 + |E|\log|E| \end{aligned}$$

where $|E(i_0)| = O\left(\mathrm{Area}(S)\,/\,\varepsilon_0^2\right)$, which depends only on S. The theorem follows, since the number of Voronoi vertices is linear w.r.t. the number of Voronoi edges. □

Time complexity

Lemma 5.8. *The time complexity of the algorithm is*

$$O\left(N\log^2 N\right) = O\left(\varepsilon^{-2}\log^2\frac{1}{\varepsilon}\right) = O\left(\frac{1}{\delta}\log^2\frac{1}{\delta}\right)$$

Proof. Let T be the overall number of Delaunay tetrahedra created by the algorithm. According to Theorem 5.7, we have $T = O\left(N\log N\right)$. We will show that the time complexity is $O(T\log T)$.

- The cost of maintaining $\mathrm{Del}(E)$ is $O(T)$ since no point location is performed in our case.

- The cost of updating $\mathrm{Del}^v_{|S}(E)$ is also $O(T)$ since detect_access stops each time it reaches an active vertex and any vertex that becomes active remains so. Hence, the number of times a vertex is visited is at most the total number of incident Voronoi edges created by the algorithm.

- Since a Voronoi edge is probed from its vertices, it contains at most two centers of balls of $\mathrm{Del}^v_{|S}(E)$. Hence, the cost of maintaining the priority queue $\mathcal{Q}$ of bad balls of $\mathrm{Del}^v_{|S}(E)$ is $O(T\log T)$ since the total number of centers of balls of $\mathrm{Del}^v_{|S}(E)$ inserted in $\mathcal{Q}$ (and then retrieved from it) is at most twice the total number of Voronoi edges created during the process. □

5.2 Probing cost

The algorithm probes only along the Voronoi edges and from their vertices. Since every Voronoi edge has two vertices, it is probed at most twice. Hence, the total number of probes is at most twice the total number of Voronoi edges created during the process, which is $O\left(N\log N\right)$= $O\left(\varepsilon^{-2}\log\frac{1}{\varepsilon}\right)$= $O\left(\frac{1}{\delta}\log\frac{1}{\delta}\right)$, by Theorem 5.7.

5.3 Displacement cost

We bound the total number of Voronoi edges travelled by the probing device. During the update of $\mathrm{Del}^v_{|S}(E)$, two types of displacements are performed (see Section 3.1): *detection displacements* are performed inside the routine detect_access to locate the intersection points with the surface S; *positioning displacements* are performed during the update of $\mathrm{Del}^v_{|S}(E)$, when the probing device is moved from one place of $\mathrm{VG}(E)$ to another, before issuing a new sequence of probes.

Lemma 5.9. *The displacement cost of the algorithm is $O\left(N^2\log N\right)$= $O\left(\varepsilon^{-4}\log\frac{1}{\varepsilon}\right)$ = $O\left(\delta^{-2}\log\frac{1}{\delta}\right)$.*

Proof. The overall cost of the detection displacements has been analyzed in the proof of Lemma 5.8 and shown to be $O\left(N\log N\right)$.

In addition, according to Lemma 5.3, for every iteration $i > i_0$, $E(i)$ is a $2r(i)$-sample of S, with $2r(i) \le \varepsilon_0$. It is proved in [3] that, since $2r(i) < 0.1\ \mathrm{rch}(S)$, every Voronoi cell of $\mathrm{Vor}(E(i))$ intersects S along a topological disk that divides the cell into two components: one lies in $\mathcal{O}$, the other lies in $\mathbb{R}^3 \setminus \mathcal{O}$. Therefore, if $p(i)$ is the point inserted in E at iteration i, then, right after its insertion, all the vertices of its Voronoi cell $\mathrm{V}(p(i))$ that can be reached by the probing device will be marked *active* when calling detect_access on a neighbor of the vertex of $\mathrm{V}(p(i))$ that is considered first. As a consequence, the algorithm has to call detect_access only once before $\mathrm{Del}^v_{|S}(E(i))$ is fully updated. Hence, at iteration i, two paths only are followed by the probing device during the positioning displacements.

The lengths of these two paths are bounded by the overall number of Voronoi vertices created before iteration i. This number is $O(N \log N)$, by Theorem 5.7. Hence, the overall cost of the positioning displacements after iteration i_0 is $O(N.N \log N)$. $\Box$

The bound of Lemma 5.9 is almost tight, since on some input surfaces the displacement cost of the algorithm has been proved to be $\Omega(N^2)$.

6. IMPLEMENTATION AND RESULTS

We have implemented the algorithm using the CGAL library which provided us with implementations of the Delaunay triangulation in 2D and in 3D. Results on a planar curve and on a surface are reported in Figures 4 and 5. The active part of the Voronoi graph is printed in blue, the inactive part in black. The faces of $\mathrm{Del}^v_{|S}(E)$ are shown in red or in green, depending on whether they are circumscribed by a good or a bad ball of $\mathrm{Del}^v_{|S}(E)$. In the 2D example, the inactive part of the Voronoi graph is shown only in the first image, for clarity.

7. CONCLUSION

Many important questions are left open by this work, including:
– Can the number of probes be reduced to $O(N)$, where N is the size of the output point sample? In the case where $\mathcal{O}$ is a set of pairwise disjoint convex sets, the number of Voronoi vertices created outside $\mathcal{O}$ is linear w.r.t. N, hence the number of probes is $O(N)$.
– What are the exact trade-offs between optimizing combinatorial cost and displacement cost?
– We have assumed a perfect probing device. How can we model uncertainty in the probes? (See [10, 11] for some related results).
– Can the approach be extended to piecewise smooth surfaces?
– In practice a physical scaffold has to be present around the object being sampled to support the probing device. Can we show similar results for a probing device whose motions obey realistic constraints?
– Can we extend the approach to more general manifolds in higher dimensions? In particular, can we avoid computing full Delaunay triangulations whose cost becomes prohibitive?

8. ACKNOWLEDGEMENTS

We thank the anonymous referees for their insightful comments. The first and third authors were partially supported by the European Union through the Network of Excellence AIM@SHAPE Contract IST 506766. The second author was supported in part by NSF CARGO grant 0138456, NSF ITR grant 0205671 and NSF FRG grant 0354543.

9. REFERENCES

[1] Panagiotis D. Alevizos, Jean-Daniel Boissonnat, and Mariette Yvinec. Non-convex contour reconstruction. *J. Symbolic Comput.*, 10:225–252, 1990.

[2] N. Amenta, S. Choi, T. K. Dey, and N. Leekha. A simple algorithm for homeomorphic surface reconstruction. In *Proc. 16th Annu. ACM Sympos. Comput. Geom.*, pages 213–222, 2000.

[3] Nina Amenta and Marshall Bern. Surface reconstruction by Voronoi filtering. *Discrete Comput. Geom.*, 22(4):481–504, 1999.

[4] Dominique Attali, Jean-Daniel Boissonnat, and André Lieutier. Complexity of the Delaunay triangulation of points on surfaces: the smooth case. In *Proc. 19th Annu. ACM Sympos. Comput. Geom.*, pages 201–210, 2003.

[5] Jean-Daniel Boissonnat and Steve Oudot. Provably good sampling and meshing of surfaces. *Graphical Models (special issue on Solid Modeling)*, 2005. To appear.

[6] Jean-Daniel Boissonnat and Mariette Yvinec. Probing a scene of non-convex polyhedra. *Algorithmica*, 8:321–342, 1992.

[7] M. Do Carmo. *Riemannian Geometry.* Birkhäuser, Boston, Basel, Berlin, 1992.

[8] L. P. Chew. Guaranteed-quality mesh generation for curved surfaces. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pages 274–280, 1993.

[9] R. Cole and C. K. Yap. Shape from probing. *J. Algorithms*, 8(1):19–38, March 1987.

[10] T. K. Dey and S. Goswami. Provable surface reconstruction from noisy samples. In *Proc. 20th Annu. ACM Sympos. Comput. Geom.*, pages 330–339, 2004.

[11] D. P. Dobkin, H. Edelsbrunner, and C. K. Yap. Probing convex polytopes. In *Proc. 18th Annu. ACM Sympos. Theory Comput.*, pages 424–432, 1986.

[12] H. Federer. *Geometric Measure Theory.* Springer-Verlag, 1970.

[13] P. M. Gruber. Approximation of convex bodies. In Peter M. Gruber and J. M. Wills, editors, *Convexity and its Applications*, pages 131–162. Birkhäuser, Basel, Switzerland, 1983.

[14] M. Lindenbaum and A. M. Bruckstein. Blind approximation of planar convex sets. *IEEE Trans. Robot. Autom.*, 10(4):517–529, August 1994.

[15] W. Lorensen and H. Cline. Marching cubes: a high resolution 3d surface construction algorithm. *Comput. Graph.*, 21(4):163–170, 1987.

[16] T. J. Richardson. Approximation of planar convex sets from hyperplanes probes. *Discrete and Computational Geometry*, 18:151–177, 1997.

[17] Günter Rote. The convergence rate of the Sandwich algorithm for approximating convex functions. *Computing*, 48:337–361, 1992.

[18] Steven S. Skiena. Geometric reconstruction problems. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 26, pages 481–490. CRC Press LLC, Boca Raton, FL, 1997.

[19] J. F. Traub, G. W. Wasilkowski, and H. Wozniakowski. *Information-based Complexity.* Academic Press, 1988.

[20] J. F. Traub and A. G. Werschulz. *Complexity and Information.* Cambridge University Press, 1998.

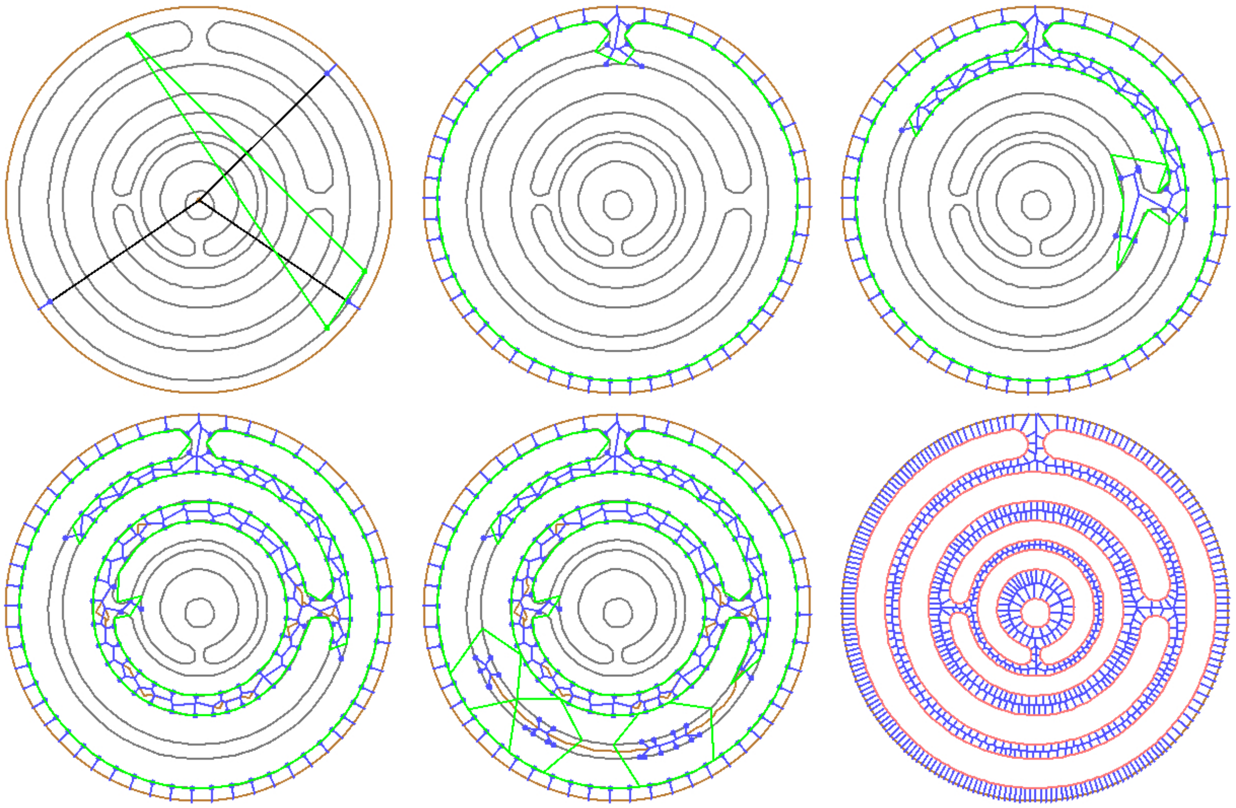

Figure 4: Course of the algorithm on a curve in $\mathbb{R}^2$

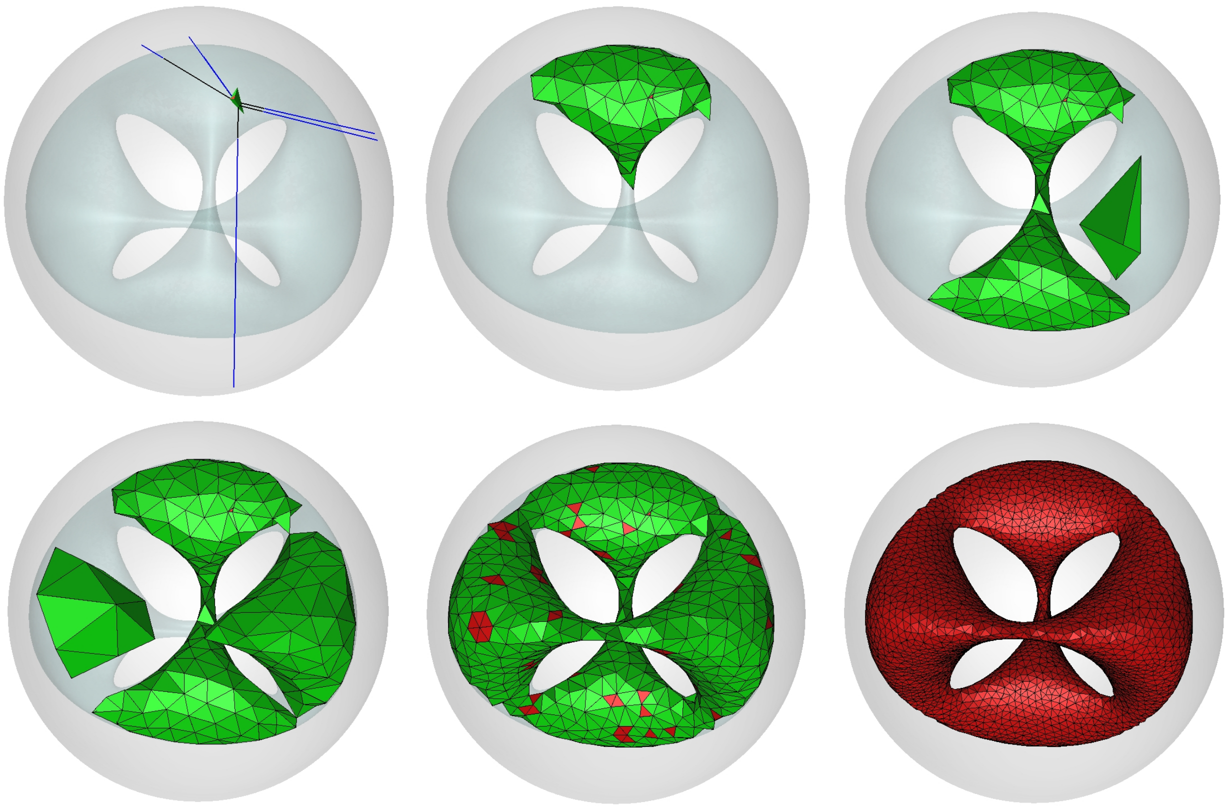

Figure 5: Course of the algorithm on a surface in $\mathbb{R}^3$