INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Learning Surfaces by Probing

Jean-Daniel Boissonnat — Leonidas J. Guibas — Steve Oudot

N° 5434

Décembre 2004

Thème SYM

Rapport de recherche

ISSN 0249-6399 ISRN INRIA/RR--5434--FR+ENG

Learning Surfaces by Probing

Jean-Daniel Boissonnat, Leonidas J. Guibas, Steve Oudot

Thème SYM — Systèmes symboliques
Projet Géométrica

Rapport de recherche n° 5434 — Décembre 2004 — 21 pages

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, nomal approximation, etc.

Key-words: Manifold learning, blind surface approximation, interactive surface reconstruction, surface meshing, Delaunay refinement

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles, BP 93, 06902 Sophia Antipolis Cedex (France)
Téléphone : +33 4 92 38 77 77 — Télécopie : +33 4 92 38 77 65

Reconstruction de surfaces par sondage

Résumé : Nous considérons le problème de la reconstruction d'une surface lisse inconnue, appelée S, qui borde un domaine $\mathcal{O}$ de $\mathbb{R}^3$. Le processus de découverte consiste à déplacer une sonde ponctuelle dans l'espace libre autour de $\mathcal{O}$, de manière à ce que cette sonde entre en contact avec S en divers points. Nous proposons une stratégie de sondage pour générer une série d'échantillons de S, à partir desquels peut être construite une surface triangulée approchant S avec n'importe quelle précision. Nous bornons le nombre d'opérations de sondage ainsi que le nombre de déplacements élémentaires de la sonde. Notre solution est une extension de précédents travaux sur les techniques de raffinement de Delaunay pour le maillage de surfaces. La surface approchante que nous générons vérifie toutes les propriétés des maillages obtenus par ces techniques, comme par exemple le bon type de topologie ou l'approximation des normales.

Mots-clés : Reconstruction de variétés, reconstruction aveugle de surfaces, reconstruction interactive de surfaces, maillage de surfaces, raffinement de Delaunay

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 blindly, 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 non-zero lower bound on its local feature size), the probing strategy proposed in this paper is inspired from Chew's algorithm [7] 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 [8]. 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 [17] 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 [13, 15] and some other probe models are analyzed by Rote [16]. 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 [14] 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.*

Theoretically speaking, we have at our disposal a *probing device*, that 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 mentionned 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 [9], minimum local feature size [3], etc. Having a positive reach is assured if S is $C^{1,1}$, i.e. S is C^1 and its normal vector field is Lipschitz.

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 [18, 19]. 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 W within any desired accuracy. Moreover, W recovers the exact topology of S and the error on the normal deviation of the facets of W 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, 7], 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 is analyzed in Section 6.

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 $W = \mathrm{Del}^{\omega}_{|S}(E)$.

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

H1 *W is a manifold without boundary;*

H2 *W has vertices on all the connected components of S;*

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

Specifically we proved that, under these conditions, W is ambient isotopic to S, at Hausdorff distance $O(\varepsilon^2)$ from S and the normal deviation is bounded by $O(\varepsilon)$. Moreover, S is included in the union of the balls of $\mathrm{Del}_{|S}(E)$ that circumscribe the facets of W. This implies that E is a 2ε-sample of S, that is: $\forall x \in S,\ |B(x, 2\varepsilon) \cap E| \geq 1$.

Hence, proving the correctness of Chew's algorithm, when run with some oracle ω, reduces to showing that assertions (H1), (H2) and (H3) are satisfied with $W = \mathrm{Del}^{\omega}_{|S}(E)$ upon termination.

In addition, we proved in [5] that the output point sample E of the algorithm is *sparse*, i.e. there exists a constant κ that does not depend on S nor on ε, such that: $\forall x \in S,\ |B(x, 2\varepsilon) \cap E| \leq \kappa$.

Since E is a sparse 2ε-sample of S, we have

Theorem 2.1 *[Theorem 5.5 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.*

$\mathrm{Area}(S)/\varepsilon^2$ and hence $|E|$ are proportional to the ε-*entropy* of S, i.e. the minimum number of spheres of radius ε that can cover S [12].

3 The probing algorithm

For the sake of clarity, we assume in sections 3, 4 and 5 that S is connected. We defer the treatment of several connected components to section 6. 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 is 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.

The main idea of the paper is to construct piecewise linear paths made of Voronoi edges.

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.

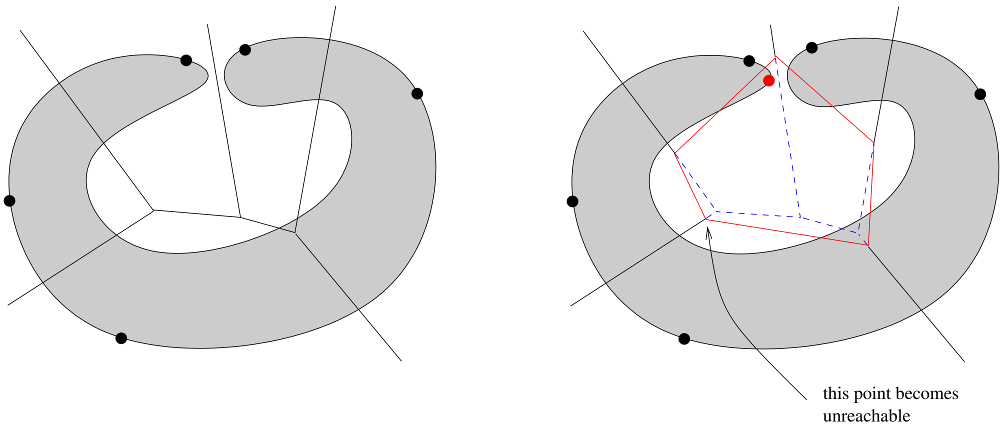

Figure 1: Connectivity changes in the Voronoi diagram. The red point is inserted in E and the red edges mark the boundary of its Voronoi cell.

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

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 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 the 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 $v.prev \leftarrow v$ by default 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, its *prev* field is never *NULL*. If v belongs to Ω, then we initialize $v.prev \leftarrow NULL$ by default.

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.

```
DETECT_ACCESS (v_start):
foreach neighbor v of v_start do
    PROBE edge [v_start, v];
    if ([v_start, v] ∩ S ≠ ∅) then
        add the dual of [v_start, v] to Del^v_|S(E);
    else if (v.prev = NULL) then
      // v becomes active
        if (v ∈ Ω) then
            v.prev ← v_start;
        else
            v.prev ← v;
        end if
        MOVE probing device from v_start to v;
        DETECT_ACCESS (v);
        MOVE probing device from v to v_start;
    end if
end foreach
```

Figure 2: Routine DETECT_ACCESS

Construction. Given an initial point set E of S, we compute $\mathrm{Del}^v_{|S}(E)$ by calling DETECT_ACCESS successively on all the vertices of $\mathrm{VG}(E)$ that lie outside Ω, including the infinite vertices[3].

[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 $\partial\Omega \cap \mathrm{VG}(E)$.

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 the *prev* pointer of every active vertex of $\mathrm{Vor}(E)$ 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:

- 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, then we move the probing device to v' and then 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 assumption is less than $0.091\ \mathrm{rch}(S)$. 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 $\varepsilon/2$. 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 Section 4.3 (lemma 4.2), (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 initialization step, the algorithm works like 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 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. In section 5, we will analyze the complexity of the algorithm.

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, as follows from a packing argument.

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.*

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

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: we first prove that (P1) and (P2) are verified after the initialization phase, and then we show that they still hold after each step of the algorithm.

• After the initialization phase, by construction, every active Voronoi vertex v has been given a free path $\pi(v)$ to some Voronoi vertex v' lying in $\mathbb{R}^3 \setminus \Omega$. Therefore, v can be reached from $\mathbb{R}^3 \setminus \Omega$ by moving the probing device along $\pi(v)$ from v' to v. Hence, (P1) holds.

Let w be a vertex of $\mathrm{VG}(E)$. If w belongs to a connected component of $\mathrm{VG}(E)\setminus\mathcal{O}$ that does not intersect $\mathbb{R}^3 \setminus \Omega$, then it is inactive after the initialization phase of the algorithm, because we call DETECT_ACCESS only on the vertices of $\mathbb{R}^3 \setminus \Omega$. Conversely, if w belongs to a connected component of $\mathrm{VG}(E) \setminus \mathcal{O}$ that intersects $\mathbb{R}^3 \setminus \Omega$, then there exists a free path in $\mathrm{VG}(E)$ from $\pi(w)$ to $\mathbb{R}^3 \setminus \Omega$. Let w' be the last vertex of $\pi(w)$. By definition, w' lies in $\mathbb{R}^3 \setminus \Omega$, while all the other vertices of $\pi(w)$ belong to Ω. Since we call DETECT_ACCESS on w', DETECT_ACCESS visits also w because all the vertices of $\pi(w)$ other than w' are inactive initially. Hence, w becomes active and (P2) holds 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) are verified by* 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, which means that $p \notin E$.

(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 since DETECT_ACCESS does not affect 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, is 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 v and w be two Voronoi vertices that lie in the same connected component of $\mathrm{VG}(E \cup \{p\}) \setminus \Omega$. There exists a free path π in $\mathrm{VG}(E \cup \{p\})$ that connects v to w. Let $\mathrm{V}(p)$ be the Voronoi cell of p in $\mathrm{Vor}(E \cup \{p\})$.
P2.1 If $\pi \cap \mathrm{V}(p) = \emptyset$, then π is made of edges of $\mathrm{Vor}(E)$, which means that v and w lie in the same connected component of $\mathrm{VG}(E) \setminus \mathcal{O}$. By (IH), v and w have the same status before the insertion of p. If this status is *active*, then they both remain such after the insertion of p because the update of $\mathrm{Del}^v_{|S}(E)$ does not affect the vertices that are already active. If the status is *inactive*, then, by (IH), all the vertices of path π are inactive before the insertion of p. Thus, if one of the vertices of π becomes active during the update of $\mathrm{Del}^v_{|S}(E)$, it is because this vertex is visited by DETECT_ACCESS, which implies that the whole path π is visited by DETECT_ACCESS since its vertices were all inactive before and since all its edges are free. In particular, v and w are both visited and hence both active afterwards. Therefore, in all cases v and w have the same status after p has been inserted.
P2.2 If $\pi \cap \mathrm{V}(p) \neq \emptyset$, *i.e.* if π contains some vertices and edges of the Voronoi cell of p, then we cut π into maximal pieces (w.r.t. inclusion) belonging either to $\mathrm{V}(p)$ or to $\mathrm{VG}(E \cup \{p\}) \setminus \mathrm{V}(p)$. Let $\pi_1, \cdots, \pi_k$ be these pieces.
– By the same argument as in P2.1, for each piece π_i that belongs to $\mathrm{VG}(E \cup \{p\}) \setminus \mathrm{V}(p)$, all the vertices of π_i have the same status, *active* or *inactive*, after p has been inserted.
– For each piece π_j that belongs to $\mathrm{V}(p)$, all the vertices of π_j are new Voronoi vertices. The ones that

lie outside Ω are active initially, whereas the others are inactive initially. If $\pi_j \setminus \Omega \neq \emptyset$, then, by running DETECT_ACCESS on the vertices of $\pi_j \setminus \Omega$, we make the other vertices of π_j active as well. Otherwise, all the vertices of π_j are inactive initially, hence visiting one of them with DETECT_ACCESS means visiting all of them. Therefore, they all have the same status, *active* or *inactive*, once $\mathrm{Del}^v_{|S}(E)$ has been updated.
– In addition, if π_i and π_{i+1} are two consecutive pieces of π, then one of them (say π_i) belongs to $\mathrm{V}(p)$ while the other one (π_{i+1}) belongs to $\mathrm{VG}(E \cup \{p\}) \setminus \mathrm{V}(p)$. Let v_i be their common vertex, e the edge of π_{i+1} incident to v_i and v_{i+1} the other vertex of e. v_{i+1} is a former Voronoi vertex while v_i is a new vertex. If v_i lies in $\mathbb{R}^3 \setminus \Omega$ (case a), or if v_{i+1} was active before the insertion of p (case b), then, according to section 3.1, the update of $\mathrm{Del}^v_{|S}(E)$ is performed by calling DETECT_ACCESS on v_i (case a) or on v_{i+1} (case b) right after the insertion of p, which implies that v_i and v_{i+1} become both active since edge e is free. Otherwise, v_i belongs to Ω and v_{i+1} was inactive before the insertion of p, hence v_i and v_{i+1} are both inactive right before the update of $\mathrm{Del}^v_{|S}(E)$, which implies that they have the same status afterwards since edge e is free.
– In conclusion, all the vertices of π have the same status after the insertion of p, which shows that (P2) is preserved. $\square$

Assertion (P1) states that the paths stored in our data structure can be effectively followed by the probing device. Assertion (P2) means that the status of the Voronoi vertices complies with the connected components of $\mathrm{VG}(E) \setminus \mathcal{O}$, which will be instrumental in guaranteeing the quality of the output of the algorithm.

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. Let E denote now 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 point sample with a *persistent facet* (a, b, c) cicumscribed by a Delaunay ball B centered on S of radius at most $\varepsilon/2$.

Lemma 4.2 *(a, b, c) remains a facet of $Del_{|S}(E)$ throughout the course of the algorithm.*

Proof Let us assume for a contradiction that, upon termination of the algorithm, (a, b, c) is no longer a facet of $\mathrm{Del}_{|S}(E)$. This means that B is no longer a Delaunay ball, *i.e.* that the algorithm has inserted a point x in the interior of B. We consider a vertex of (a, b, c), say a. Since a and x both lie in B, $\|a - x\|$ is at most twice the radius of B, hence $\|a - x\| \leq \varepsilon$. Now, since x is inserted by the algorithm, it is the center of some bad ball B' of $\mathrm{Del}^v_{|S}(E)$, whose radius is greater than ε. It follows that a belongs to the interior of B', which contradicts the fact that B' is a Delaunay ball. This ends the proof of the lemma. $\square$

It follows from lemma 4.2 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). $\square$

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 $\varepsilon < 0.091\,\varepsilon_S \leq 0.091\,\mathrm{rch}(S)$ upon termination. $\square$

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

L1 [Lemma 3.5 of [5]] *Two facets of $Del^v_{|S}(E)$ that share a vertex v do not overlap in projection onto $T(v)$, the plane tangent to S at v.*

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

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

L3 *Let ζ be a connected component of $VG(E) \setminus \mathcal{O}$. Either all the endpoints of ζ (which by definition belong to $VG(E) \cap S$) are centers of balls of $Del^v_{|S}(E)$ or none of them is.*
Proof Let p and q be two endpoints of ζ. By definition, p and q are centers of balls of $\mathrm{Del}_{|S}(E)$. If ζ contains no 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 is a 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 (L3). □

Using (L1), (L2) and (L3), 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 face dual to e. Every 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 $\mathrm{Del}_{|S}(E)$. By (L3), 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, e cannot be incident to more than two restricted Delaunay facets. Indeed, by (L1), the projections onto $T(v)$ (where v is a vertex of e) of the facets of $\mathrm{Del}^v_{|S}(E)$ incident to e pairwise do not overlap, thus they lie on different sides of the line supporting the projection of 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 2. Hence it is 2.

Since this is true for any edge of $\mathrm{Del}^v_{|S}(E)$, the facets of $\mathrm{Del}^v_{|S}(E)$ that are incident to any vertex v of $\mathrm{Del}^v_{|S}(E)$ form a set of umbrellas. Using (L1), we 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 and contradicting (L1). It follows that every facet of $\mathrm{Del}^v_{|S}(E)$ incident to v overlaps some facet of U in projection onto $T(v)$. Therefore, v can only have one umbrella. Since this is true for any vertex, $\mathrm{Del}^v_{|S}(E)$ is a 2-manifold without boundary. □

Since $\mathrm{Del}^v_{|S}(E)$ verifies H1-H3, it is a good approximation of S, according to section 2.

Theorem 4.3
- *$Del^v_{|S}(E)$ is ambient isotopic to S;*
- *the Hausdorff distance between $Del^v_{|S}(E)$ and S is $O(\varepsilon^2)$;*
- *$Del^v_{|S}(E)$ approximates S, in terms of normals and area, within an error of $O(\varepsilon)$;*
- *the balls of $Del^v_{|S}(E)$ cover S;*
- *E is a sparse 2ε-sample of S. It follows from theorem 2.1 that $|E| = O\left(\frac{Area(S)}{\varepsilon^2}\right)$.*

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]. It follows that $\mathrm{Del}^v_{|S}(E)$ 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 mentionned 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 should privilege 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 insert the points in $\mathrm{Del}(E)$ in an order that is close to the order of the points along $\mathcal{C}$ (see figure 3, first row), which makes the 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.

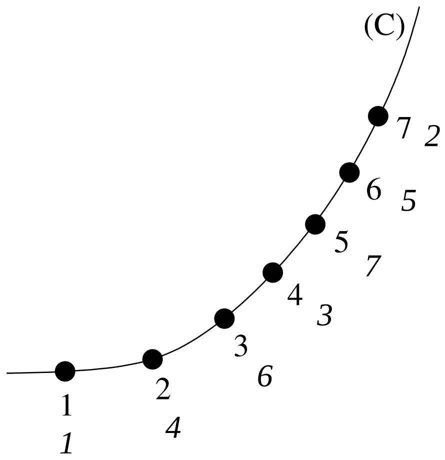

Figure 3: Two orders of insertion on a parabola

In the sequel, we analyse 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

In the data structure, we store 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.

A trivial bound on this number is $O(N^2)$, where N is the size of the output point sample. Indeed, every new point inserted in E creates at most $(2|E|-4)$ new Delaunay tetrahedra. But this bound is pessimistic since it does not take into account the fact that the points of E lie on a surface, a case in which the asymptotic size of $\mathrm{Del}(E)$ is $O(|E|\log|E|)$, as shown in [4].

Theorem 5.1 *The overall number of Voronoi vertices created during the course of the algorithm is $O\left(N^{5/4}\right)$, where N is the size of the output point sample.*

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

We will use the following result, stated as lemma 17 in [4]:

P *There exists an ε_0, depending on S only, such that for any $\varepsilon < \varepsilon_0$, every point of a sparse ε-sample E of S has $O\left(|E|^{1/4}\right)$ neighbors in Del(E).*

In the sequel, we take as ε_0 the minimum of the above (unknown) constant and of $0.091\,\mathrm{rch}(S)$.

- Let us first assume that $\varepsilon > \frac{\varepsilon_0}{4}$. The overall number of Voronoi vertices created by the algorithm is at most $|E|^2$, where E is the output point sample. By theorem 4.3, we have $|E|^2 = O\left(\mathrm{Area}^2(S)/\varepsilon^4\right) = O\left(\mathrm{Area}^2(S)/\varepsilon_0^4\right)$, a constant that depends only on the surface S.

- Let us now assume that $\varepsilon \leq \frac{\varepsilon_0}{4}$. Let i_0 be the first iteration of the algorithm at the end of which all the balls of $\mathrm{Del}^v_{|S}(E)$ have radii smaller than $\frac{\varepsilon_0}{4}$. In other words, i_0 is the first iteration such that $r(i_0) \leq \frac{\varepsilon_0}{4}$. Since $\frac{\varepsilon_0}{4} < 0.091\,\mathrm{rch}(S)$, $E(i_0)$ is an $\frac{\varepsilon_0}{2}$-sample of S, by theorem 4.3. Moreover,

Claim 5.1.1 *For any two iterations i and i' such that $i \geq i' \geq i_0$, we have $r(i) \leq 2r(i')$.*

Proof Let $B(i)$ be a ball of $\mathrm{Del}^v_{|S}(E(i))$ of largest radius. Its center $c(i)$ 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. Since $\varepsilon_0 < 0.091\,\mathrm{rch}(S)$, the balls of $\mathrm{Del}^v_{|S}(E(i_0))$ cover S, by theorem 4.3. Thus, $c(i)$ lies in a ball $B(c,r)$ of $\mathrm{Del}^v_{|S}(E(i'))$. We have $\mathrm{dist}(c(i), E(i')) \leq 2r \leq 2r(i')$. Moreover, since $i' \leq i$, we have $E(i') \subseteq E(i)$, and therefore $\mathrm{dist}(c(i), E(i)) \leq \mathrm{dist}(c(i), E(i'))$, which concludes the proof of the claim. □

Claim 5.1.2 *For any iteration $i \geq i_0$, $E(i)$ is a sparse $2r(i)$-sample of S, with $2r(i) \leq \varepsilon_0$.*

Proof Let i be an iteration of the algorithm, such that $i \geq i_0$. According to claim 5.1.1, we have $r(i) \leq 2r(i_0) \leq \frac{\varepsilon_0}{2} < 0.091\,\mathrm{rch}(S)$, thus $E(i)$ is a $2r(i)$-sample of S, with $2r(i) \leq \varepsilon_0$, according to theorem 4.3.

Let $x \in S$. To show that $E(i)$ is sparse, we count the points of $E(i)$ that lie in $B(x, 2r(i))$.
- Since $|E_{\mathrm{init}}| = 3$, the number of points of E_{init} that lie in $B(x, 2r(i))$ is at most 3.
- By definition of i_0, every point of $E(i_0) \setminus E_{\mathrm{init}}$ is the center of a Delaunay ball of radius greater than $\frac{\varepsilon_0}{4} \geq r(i_0)$, which is at least $\frac{1}{2}\,r(i)$, by claim 5.1.1. Moreover, at any iteration i' such that $i_0 \leq i' \leq i$, the point inserted in E is the center of a Delaunay ball of radius $r(i'-1)$, which is at least $\frac{1}{2}\,r(i)$, by claim 5.1.1. Therefore, the points of $E(i) \setminus E_{\mathrm{init}}$ are at least $\frac{1}{2}\,r(i)$ away from one another. Hence, they are centers of pairwise-disjoint balls of radius $\frac{1}{4}\,r(i)$. For every such ball B whose center lies in $B(x, 2r(i))$, B is included in $B(x, (2+\frac{1}{4})\,r(i))$. It follows that the number of points of $E(i) \setminus E_{\mathrm{init}}$ that lie in $B(x, 2r(i))$ is bounded by a constant, which concludes the proof of the claim. □

It follows from (P) and from claim 5.1.2 that, after iteration i_0 (which we do not know), the algorithm creates $O\left(|E|^{1/4}\right)$ new Voronoi vertices at each iteration. Thus, the total number of Voronoi vertices created by the algorithm is $O\left(|E(i_0)|^2 + N^{5/4}\right)$, where N is the size of the output point sample. Since $E(i_0)$ is a sparse $\frac{\varepsilon_0}{2}$-sample of S, by theorem 2.1 its size is $O\left(\mathrm{Area}(S)/\varepsilon_0^2\right)$, a constant that depends only on S. This ends the proof of theorem 5.1. □

In conclusion, the space complexity of the algorithm is $O\left(N^{5/4}\right)$.

Time complexity

Lemma 5.2 *The time complexity of our algorithm is $O\left(N^{5/4}\log N\right)$, where N is the size of the output point sample.*

Proof Let T be the overall number of Delaunay tetrahedra created by the algorithm. According to theorem 5.1, we have $T = O\left(N^{5/4}\right)$. We will show that the time complexity is $O(T \log T)$.

- The cost of maintaining 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 $\mathcal{Q}$ 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. $\square$

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 $O\left(N^{5/4}\right)$ by theorem 5.1, where N is the size of the output point sample.

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): *preliminary displacements* allow us to place the probing device to a vertex v before calling DETECT_ACCESS on v; *detection displacements* are performed inside the routine DETECT_ACCESS to locate the intersection points with the surface S. The overall cost of the detection displacements has been analyzed in the proof of lemma 5.2 and shown to be $O\left(N^{5/4}\right)$.

Lemma 5.3 *The overall cost of the preliminary displacements is $O\left(N^{9/4}\right)$.*

Proof The reasoning is similar in spirit to the one used in the proof of theorem 5.1, and we use the same notations.

- If $\varepsilon > \frac{\varepsilon_0}{4}$, then we give a simple bound. At each iteration i, the algorithm runs DETECT_ACCESS on several Voronoi vertices, which are either new vertices or former vertices that are neighbors of some new ones. For each such vertex v, the algorithm moves the probing device to v along two paths: one from its current position to $\mathbb{R}^3 \setminus \Omega$; the other from $\mathbb{R}^3 \setminus \Omega$ to v. Each path has a length bounded by the total number of Voronoi vertices, which is $O(|E(i)|^2)$. Since the number of new Voronoi vertices at iteration i is $O(|E(i)|)$, the overall number of edges travelled at iteration i during the preliminary displacements is $O(|E(i)|^3)$.

In conclusion, the overall cost of the preliminary displacements is $O\left(|E|^4\right)$, where E is the output point sample. Now, by theorem 4.3, we have $|E|^4 = O\left(\mathrm{Area}^4(S)/\varepsilon^8\right) = O\left(\mathrm{Area}^4(S)/\varepsilon_0^8\right)$, a constant that depends only on the surface S.

- If $\varepsilon \leq \frac{\varepsilon_0}{4}$, then according to claim 5.1.2, there exists an iteration i_0 of the algorithm such that, for every iteration $i \geq i_0$, $E(i)$ is a sparse $2r(i)$-sample of S, with $2r(i) \leq \varepsilon_0$.

It is proved in [3] that, since $E(i)$ is an ε_0-sample of S, with $\varepsilon_0 < 0.1\,\mathrm{rch}(S)$, every Voronoi cell 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* by the call to 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 preliminary displacements.

The lengths of these two paths are bounded by the overall number of Voronoi vertices created before iteration i. This number is $O\left(|E(i)|^{5/4}\right)$, by theorem 5.1. Hence, the overall cost of the preliminary displacements after iteration i_0 is $O\left(N.N^{5/4}\right) = O\left(N^{9/4}\right)$, where N is the size of the output point sample. $\square$

In conclusion, the displacement cost of the algorithm is $O\left(N^{9/4}\right)$.

6 Dealing with several connected components

Let $S_1, \cdots, S_n$ be the connected components of S. We assume that every component of $\mathcal{O}$ has a connected boundary. This is no real loss of generality since, otherwise, the probing device would not be able to probe some of the components. Under this assumption, $\mathcal{O}$ has n connected components exactly, $\mathcal{O}_1, \cdots, \mathcal{O}_n$, such that $\mathcal{O}_i$ is bounded by S_i, for all i. According to A1, for every component $\mathcal{O}_i$ we are given a point $o_i \in \mathcal{O}_i$. We assume that $\varepsilon < 0.05\, \varepsilon_S$.

Meshing one component of S

Basically, to handle several connected components, the algorithm behaves as if there were only one. For the initialization, it chooses any position p on $\partial\Omega$ and probes towards o_1, in order to build a persistent facet (a, b, c) – see Section 3.2. Notice that (a, b, c) may not lie on S_1, since the latter may be hidden from p by another connected component of S. Then, the algorithm acts as described in Section 3.2.

Upon termination, theorem 4.3 holds with S replaced by U, the union of the connected components of S that contain vertices of $\mathrm{Del}^v_{|S}(E)$. Let $\mathcal{O}^U$ be the union of the components of $\mathcal{O}$ whose boundaries belong to U, and let $\mathcal{O}^v$ be the bounded open set of $\mathbb{R}^3$ whose boundary is $\mathrm{Del}^v_{|S}(E)$.

Lemma 6.1 *$\mathcal{O}^U \triangle \mathcal{O}^v$, the symmetric difference between $\mathcal{O}^U$ and $\mathcal{O}^v$, is covered by the balls of $\mathrm{Del}^v_{|S}(E)$.*

Proof By definition, the balls of $\mathrm{Del}^v_{|S}(E)$ cover $\mathrm{Del}^v_{|S}(E)$. Moreover, by theorem 4.3, they also cover U. Let us build the power diagram $\mathcal{P}$ of the spheres of $\mathrm{Del}^v_{|S}(E)$.

It is well-known that every ball $B(c')$ of $\mathrm{Del}^v_{|S}(E)$ that intersects the cell $P(c)$ of c in $\mathcal{P}$, is included in $B(c)$ inside $P(c)$, that is, $B(c') \cap P(c) \subseteq B(c) \cap P(c)$. It follows that $B(c) \cap P(c)$ contains $\mathrm{Del}^v_{|S}(E) \cap P(c)$ and $U \cap P(c)$, since the balls of $\mathrm{Del}^v_{|S}(E)$ cover $\mathrm{Del}^v_{|S}(E)$ and U.

– In particular, for every edge e of $\partial P(c)$, $B(c) \cap e$ contains $\mathrm{Del}^v_{|S}(E) \cap e$ and $U \cap e$, which are the boundaries of $(\mathcal{O}^U \triangle \mathcal{O}^v) \cap e$. Since $B(c) \cap e$ is convex (because $B(c)$ and e are), it contains $(\mathcal{O}^U \triangle \mathcal{O}^v) \cap e$.

– For every face f of $\partial P(c)$, the boundary of $(\mathcal{O}^U \triangle \mathcal{O}^v) \cap f$ is made of $\mathrm{Del}^v_{|S}(E) \cap f$, $U \cap f$, and the intersections of $\mathcal{O}^U \triangle \mathcal{O}^v$ with several edges of ∂f. Since all these arcs are included in $B(c) \cap f$, which is convex because $B(c)$ and f are, $(\mathcal{O}^U \triangle \mathcal{O}^v) \cap f$ is also included in $B(c) \cap f$.

– The boundary of $(\mathcal{O}^U \triangle \mathcal{O}^v) \cap P(c)$ is made of $\mathrm{Del}^v_{|S}(E) \cap P(c)$, $U \cap P(c)$, and the intersections of $\mathcal{O}^U \triangle \mathcal{O}^v$ with several faces of $\partial P(c)$. Since all these surface patches are included in $B(c) \cap P(c)$, which is convex because $B(c)$ and $P(c)$ are, $(\mathcal{O}^U \triangle \mathcal{O}^v) \cap P(c)$ is also included in $B(c) \cap P(c)$. Since this is true for every center of ball of $\mathrm{Del}^v_{|S}(E)$, the proof of the lemma is complete. □

Let S_i be the component of S on which facet (a, b, c) lies. By lemma 4.2, (a, b, c) remains in $\mathrm{Del}^v_{|S}(E)$ throughout the course of the algorithm, thus $S_i \subseteq U$ and U is not empty. To see which components of S belong to U, we determine, for every $\mathcal{O}_j$, whether o_j verifies at least one of the following conditions:

C1 $\quad$ *$dist(o_j, E) < \varepsilon_S$*

C2 $\quad$ $o_j \in \mathcal{O}^v$

Lemma 6.2 *o_j verifies (C1) or (C2) iff $S_j \subseteq U$.*

Proof By lemma 6.1, the balls of $\mathrm{Del}^v_{|S}(E)$ cover $\mathcal{O}^U \triangle \mathcal{O}^v$. Therefore, every point p of $\mathcal{O}^U$ is included either in $\mathcal{O}^v$ or in the union of the balls of $\mathrm{Del}^v_{|S}(E)$. In the latter case, we have $\mathrm{dist}(p, E) \leq 2\,\varepsilon < \varepsilon_S$ since the radii of the balls of $\mathrm{Del}^v_{|S}(E)$ are at most ε.

Conversely, every point q of $\mathcal{O} \setminus \mathcal{O}^U$ is at distance at least $2\,\varepsilon_S$ from $\mathcal{O}^U$, because the components of S are farther than $2\,\mathrm{rch}(S) > 2\,\varepsilon_S$ from one another. It follows that q cannot lie in $\mathcal{O}^U \triangle \mathcal{O}^v$, since the latter is covered by the balls of $\mathrm{Del}^v_{|S}(E)$ which are centered on U and have radii at most $\varepsilon < \varepsilon_S$. Hence, $q \notin \mathcal{O}^v$ and $\mathrm{dist}(q, E) \geq 2\,\varepsilon_S$. □

Thanks to lemma 6.2, we know precisely which connected components of S have been meshed, by checking which of the o_j verifie (C1) or (C2).

– Checking (C1) with o_j reduces to finding the point of E that is closest to o_j, which can be performed by locating o_j in $\mathrm{Vor}(E)$.

– Regarding (C2), we notice that every Delaunay tetrahedron lies either completely inside $\mathcal{O}^v$ or completely outside $\mathcal{O}^v$, since the facets of $\mathrm{Del}^v_{|S}(E)$ belong to the Delaunay triangulation. Hence, it suffices to mark each tetrahedron as *interior* or *exterior*, and then to locate each o_j in $\mathrm{Del}(E)$.

In conclusion, (C1) and (C2) can be checked for all the o_j in $O(n \log |E| + T)$ time, where n is the number of connected components of S and T is the number of tetrahedra of $\mathrm{Del}(E)$.

Meshing the other components of S

To mesh other connected components of S, we create a persistent facet (a', b', c') on $S \setminus U$. This supposes that we can probe points from $S \setminus U$.

Lemma 6.3 *We can work out a point of $\mathbb{R}^3 \setminus \mathcal{O}$, reachable by the probing device, from which it is possible to probe points of $S \setminus U$.*

Proof As explained above, we know precisely which connected components of S belong to U. Hence, we know an i such that $S_i \cap U = \emptyset$. Let $p \in E$ be the point of E that is closest to o_i. By definition, the cell $\mathrm{V}(p)$ of p in $\mathrm{Vor}(E)$ contains o_i. It follows that the cells $V^+(p)$ and $V^+(o_i)$ in $\mathrm{Vor}(E \cup \{o_i\})$ have a non-empty intersection.

We compute the edges and vertices of the boundary of $V^+(o_i)$, and then we move the probing device along the free edges of the boundary of $\mathrm{V}(p)$, starting from an active vertex (which exists because p is incident to a facet of $\mathrm{Del}^v_{|S}(E)$), until we reach a vertex of $V^+(o_i)$.

It is proved in [3] that, since E is a 2ε-sample of U, with $2\varepsilon < 0.1\ \varepsilon_S \leq 0.1\ \mathrm{rch}(S)$, $\mathrm{V}(p)$ intersects U along a topological disk that divides $\mathrm{V}(p)$ into two components: one lies in $\mathcal{O}^U$, the other lies in $\mathbb{R}^3 \setminus \mathcal{O}^U$. Since the edges of $\partial\mathrm{V}(p)$ do not intersect $S \setminus U$, we will eventually find a vertex v of $V^+(o_i)$ by following the free edges of $\partial\mathrm{V}(p)$.

Once the probing device is at v, we probe from v towards o_i. Since o_i and v both lie in $V^+(o_i)$, which is convex, segment $[v, o_i]$ is included in $V^+(o_i)$. Hence, if $[v, o_i]$ intersects U, then every point of intersection lies in $V^+(o_i)$ and close to E (closer than $2\varepsilon < \mathrm{rch}(S)$) at the same time. This is impossible because every point of $V^+(o_i)$ is farther than $\mathrm{rch}(S)$ from E, since $\mathrm{dist}(o_i, E) \geq \mathrm{dist}(o_i, U) > 2\ \mathrm{rch}(S)$.

In conclusion, we have $V^+(o_i) \cap U = \emptyset$, hence segment $[v, o_i]$ intersects only $S \setminus U$. □

Once we have built a persistent facet (a', b', c') on $S \setminus U$, we run the algorithm with $E \cup \{a', b', c'\}$ as input point sample. By lemma 4.2, we are guaranteed that (a', b', c') will be in $\mathrm{Del}^v_{|S}(E)$ upon termination, which means that the number of connected components of U will have increased.

We go on like this, creating persistent facets on unmeshed components of S and running the algorithm again, until all the connected components of S are meshed. At this stage, all the $(o_j)_j$ verify (C1) or (C2), thus we know that we can stop. The algorithm has been run at most n times, where n is the number of connected components of S.

7 Implementation and results

We have implemented the algorithm using the CGAL library which provided us with several implementations of the Delaunay triangulation in 2D and in 3D. Results are reported in figures 4 and 5. The active part of the Voronoi graph is printed in blue, the inactive part in black.

In figure 4, we trace the algorithm on a $2D$ curve. Edges of $\mathrm{Del}^v_{|S}(E)$ are marked green or red, depending on whether they are circumscribed by a bad ball or a good ball of $\mathrm{Del}^v_{|S}(E)$. From top to bottom and from left to right:

- At the beginning, we probe three points of S that are far from one another, which implies that all the balls of $\mathrm{Del}^v_{|S}(E)$ are bad (all the edges of $\mathrm{Del}^v_{|S}(E)$ are green).
- During the course of the algorithm, bad balls disappear progressively. In the meantime, good balls appear.
- In the end, all the balls of $\mathrm{Del}^v_{|S}(E)$ are good (all the edges of $\mathrm{Del}^v_{|S}(E)$ are red).

In figure 5, we trace the algorithm on a surface embedded in $\mathbb{R}^3$. From top to bottom and from left to right:

- The algorithm starts with the creation of a persistent facet f (the small red facet). As stated in lemma 4.2, f deserves its name since it remains in $\mathrm{Del}_{|S}(E)$ throughout the process, even if theoretically it may disappear from $\mathrm{Del}^v_{|S}(E)$ (which is not the case in our example).

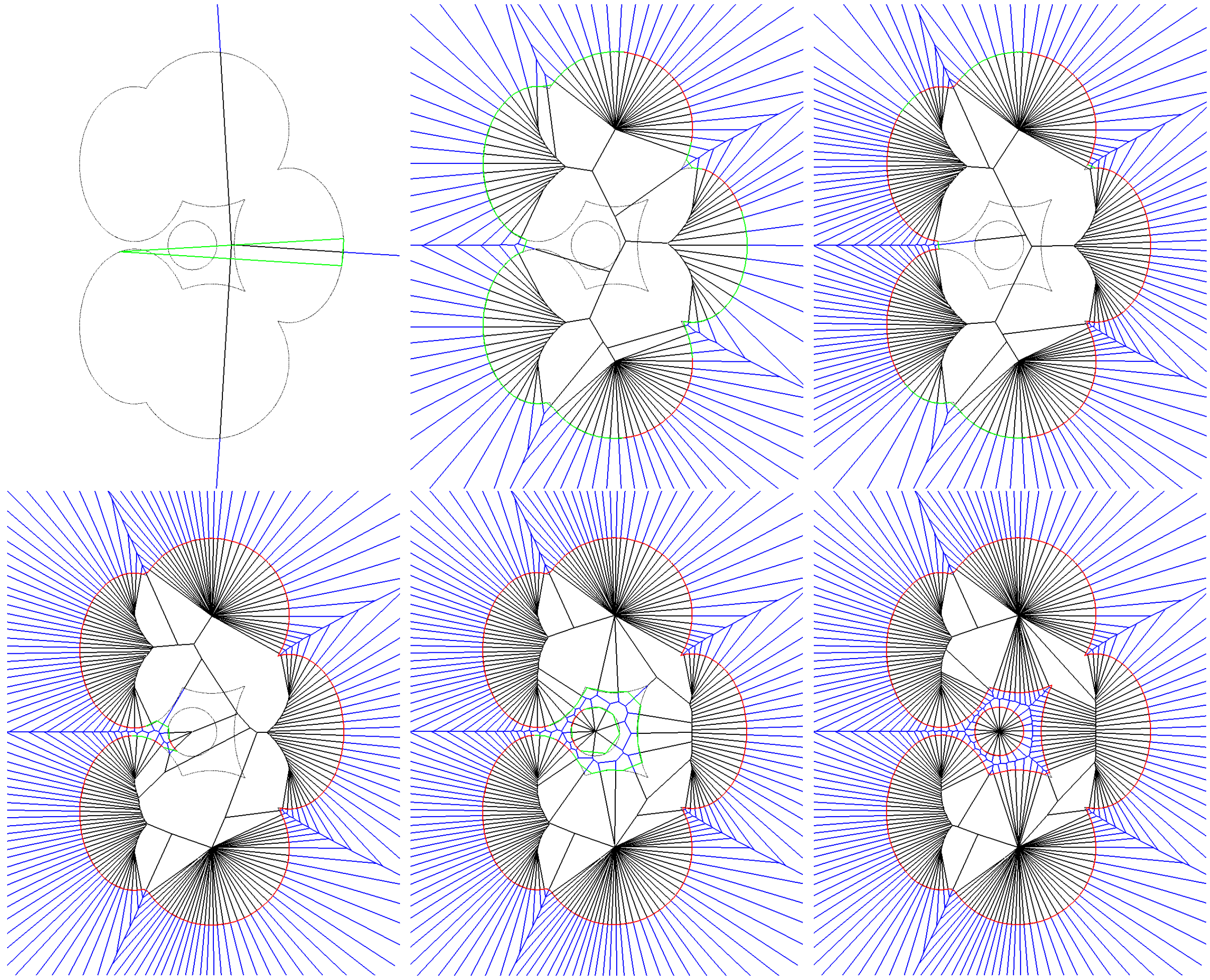

Figure 4: Course of the algorithm on a $2D$ shape made of disks and ellipses

- Then the algorithm refines the mesh. Since the algorithm considers the largest balls of $\mathrm{Del}^v_{|S}(E)$ first (thanks to $\mathcal{Q}$), it is only by the end of the process that the balls of $\mathrm{Del}^v_{|S}(E)$ become good (*i.e.* that the facets of $\mathrm{Del}^v_{|S}(E)$ become red).
- Before the balls of $\mathrm{Del}^v_{|S}(E)$ become good, the refinement process makes the mesh "propagate" through the object $\mathcal{O}$, until the topology of S is captured.

8 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?
– What are the exact trade-offs between optimizing combinatorial cost and displacement cost?
– We have assumed a perfect probing device. How can we mode 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?

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 (Solid Modeling 2004 special issue)*, 2005. To appear (preprint available at `ftp://ftp-sop.inria.fr/prisme/soudot/preprints/gmod.ps.gz`).

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

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

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

[9] M. de Carmo. *Differential Geometry of Curves and Surfaces.* Prentice Hall, Englewood Cliffs, NJ, 1976.

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

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

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

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

[17] 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.

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

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

Contents

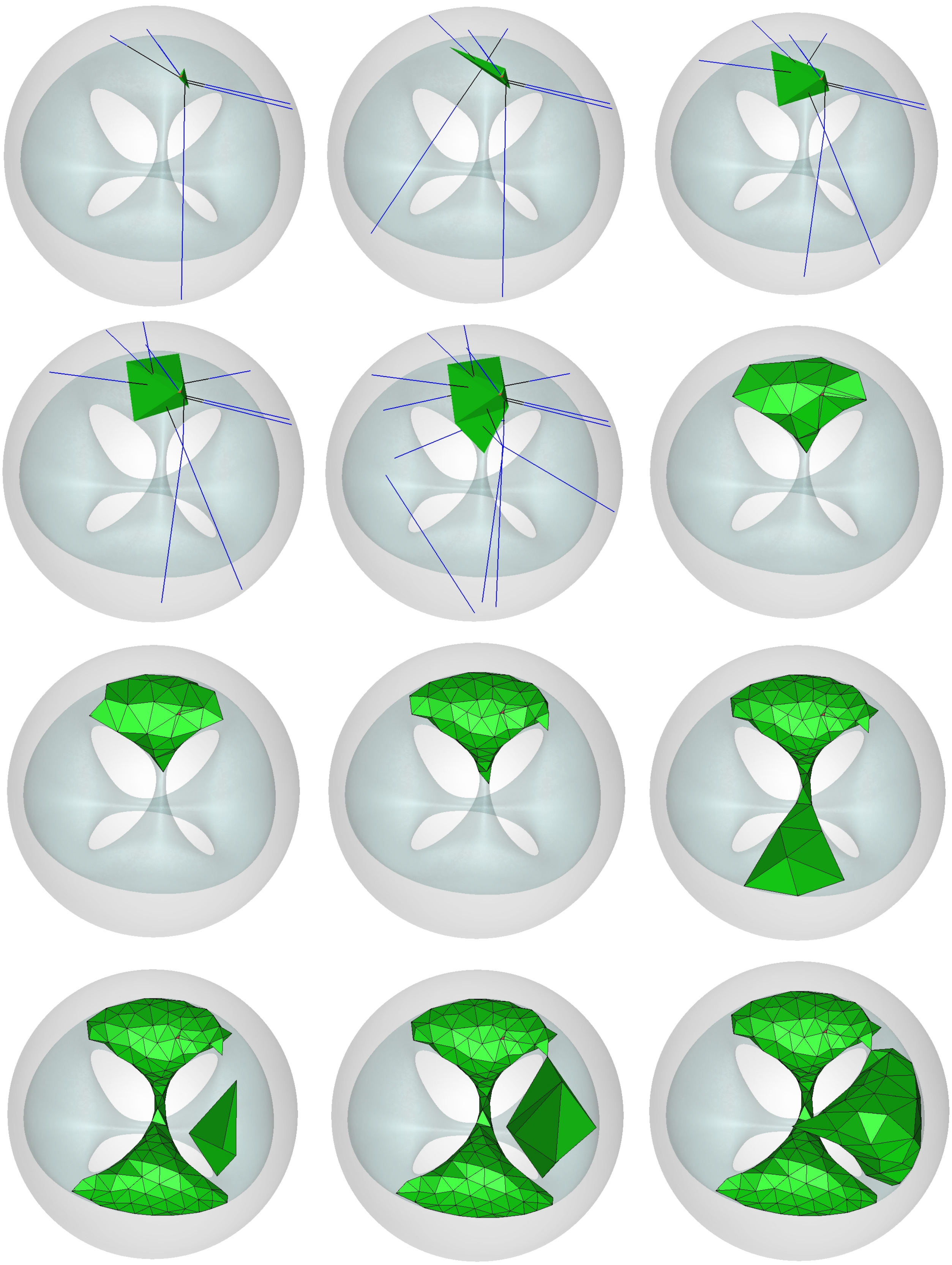

Figure 5: Course of the algorithm on a surface embedded in $\mathbb{R}^3$

Figure 5: Course of the algorithm on a surface embedded in $\mathbb{R}^3$ (cont'd)

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399