Updates on Voronoi Diagrams

João Dinis
Departamento de Física
Faculdade de Ciências, Universidade de Lisboa
1749-016 Lisboa, Portugal
jd@fc.ul.pt

Margarida Mamede
CITI, Departamento de Informática
Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa
2829-516 Caparica, Portugal
mm@di.fct.unl.pt

***Abstract*—The sweep line technique has been recently adapted to the sphere in order to build Voronoi diagrams of points on its surface. The resulting algorithm has proved to be simple and efficient, outperforming the freely available alternatives, which compute convex hulls of point sets in 3D. In this paper, we introduce two sweep algorithms for updating Voronoi diagrams, one for deleting and another for inserting a site, which are applicable to points on the sphere surface or on the plane.**

The algorithms operate directly on the doubly connected edge lists that implement the Voronoi diagram. This makes them preferable when the intended data is the Voronoi diagram, which happens, for instance, when natural neighbour interpolation is performed.

Both algorithms require linear space. Besides, insertion runs in linear time, which is worst-case optimal, whereas deletion runs in super-linear time. Although the deletion running time is not asymptotically optimal, both algorithms cope very well with degenerated cases, are efficient, and are practical to implement. Experimental results in both domains reveal that their performances are better than or similar to those of the CGAL library, which work on Delaunay triangulations.

***Keywords*-deletion and insertion in Voronoi diagrams; spherical Voronoi diagrams; doubly connected edge lists.**

I. Introduction

A Voronoi diagram is a useful tool to analyse geometrical data. The analyses often require the computation of a Voronoi diagram from an initial large data set, which must be updated afterwards by inserting or removing a small number of sites. The usual way to update a Voronoi diagram is by performing the equivalent operations on its dual, the Delaunay triangulation.

In an insertion, all triangles that are in *conflict* with the new site are removed, creating a hole, which is trivial to triangulate (in linear time). This is a well-known operation, which is part of the incremental algorithm for computing the Delaunay triangulation on the plane [10] or on the sphere [12].

Deletions are not so straightforward and the research has been focused mainly on the planar domain. Firstly, all triangles incident to the given site are deleted, creating a *hole*, which also needs to be triangulated. There are several algorithms for this purpose, whose running times depend on the number m of neighbours of the site to be removed.

The algorithm of Aggarwal *et al.* [1] is of great theoretical interest, because it runs in linear time in the worst-case. Nevertheless, it is too intricate to implement.

Devillers's algorithm [5] trades some efficiency for simplicity, running in $O(m \log m)$ time. The hole, which is delimited by a polygon, is triangulated incrementally, taking into account all *ears* of the polygon. Every ear has a priority, which is given by the power of the site that is being deleted with respect to the circle that circumscribes the ear. As an ear with the smallest priority is proved to belong in a final triangulation, it is selected and inserted into the Delaunay triangulation, shrinking the polygon. This process is iterated until a single triangle remains.

A similar but asymptotically slower alternative is the algorithm of Mostafavi *et al.* [11], whose running time is quadratic. It also considers the ears of the hole boundary, but does not rely on priorities. Instead, in each step, every ear is tested to determine if it belongs to the final triangulation. The test succeeds when the circle that circumscribes the ear does not contain any other vertex of the polygon.

Since the average vertex degree is six, quadratic but simpler algorithms may be competitive in practise. This is the case of boundary completion, the algorithm implemented in CGAL up to release 3.6.1 [2]. Here, an arbitrary edge of the polygon is selected and the corresponding triangle (which is not necessarily an ear) is found out and inserted, possibly dividing the polygon. The advantage of this strategy is that a polygon division may strongly reduce the number of comparisons at later stages.

It is important to mention that the algorithm of Chew [3], which builds the Voronoi diagram of a set of points located at the vertices of a convex polygon, has been adapted by the author to delete a site from a Voronoi diagram. It is a randomised algorithm with linear expected running time. The diagram of the region to update is computed incrementally, adding sites by a random order. A site is added as if its region was carved on the diagram, and the corresponding boundary is found through bisector intersections. The main drawback of this approach stems from the construction of the intermediate diagrams, because edges computed in a step may not belong to the final result. As we will see, our solution minimises changes: the edges of the original diagram are reused and simply linked to different endpoints.

In spite of not being explicitly mentioned in the literature, the above algorithms are all adaptable to the spherical domain. The large majority relies on the common in-circle tests, while Chew's algorithm is based on bisector intersections. Besides, the one of Devillers requires only a suitable redefinition of priority.

In this paper, we introduce algorithms for inserting and deleting a site directly in a Voronoi diagram, which avoids dealing with two distinct data structures. They operate on the doubly connected edge lists (DCELs) that are commonly used to implement Voronoi diagrams [4]. A DCEL provides direct access to the boundary of a site region, which is advantageous to some applications. Needless to say, with a Delaunay triangulation, region boundaries must be computed (by traversing the corresponding adjacent triangles). Notice that our algorithms do not depend on any particular DCEL feature, so they can work with other Voronoi diagram implementations, such as the Quad-Edge [9].

Natural neighbour interpolation [13] is an example of application that deals with Voronoi regions. In this interpolation method, the estimated value v_p at an arbitrary point p is easily defined by supposing that p is inserted in the Voronoi diagram of the data set, stealing areas from its neighbours to form its own region. Value v_p is the weighted average of the neighbour values, where the weights are the corresponding (relative) stolen areas. Thus, natural neighbour interpolation may be seen as a special case of site insertion.

Our algorithms are based on the sweep technique, which has already proved to produce simple and efficient procedures, not only on the plane [4], [8] but also on the sphere surface [7]. Both require $O(m)$ space. Concerning running times, they are $O(m \log m)$ for deletion and $O(m + t)$ for insertion, where t is the number of neighbours of the site nearest to the one that is inserted.

The rest of the paper is organised as follows. We start, in Section II, by delimiting the changes the update algorithms have to make to the Voronoi diagram. Then, Section III describes the basics of the sweep process, and Section IV presents the operations required on a DCEL. The deletion and the insertion algorithms are introduced and analysed in Sections V and VI. After that, Section VII reports on the experimental results, and Section VIII concludes with some comments on the research done in the paper.

II. Changes to Perform

Let V_k denote a Voronoi diagram with k sites, where s_j and R_j are the coordinates and the region of site j, respectively. Without loss of generality, we consider that the insertion algorithm computes V_k by adding site s_k to V_{k-1}, whereas the deletion algorithm computes V_{k-1} by removing site s_k from V_k.

Updates require only the recomputation of a small part of the Voronoi diagram, due to the properties of maximum empty circles. The algorithms scan the region to be updated with the same technique used by Fortune's algorithm for computing a Voronoi diagram of sites on the plane [4], [8]. The main difference is that, in our case, the sweep line is a circle centred at site s_k. The deletion algorithm sweeps R_k with the wave front starting on the region border and ending as a "line segment" in its interior. The insertion algorithm does the opposite: the wave front starts being a "line segment" with an endpoint at s_k and grows until it reaches the region border. Fig. 1b illustrates both processes.

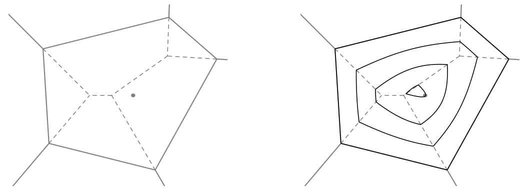

(a) (b)

Figure 1. (a) Dashed edges form a connected acyclic graph (with two inner edges). (b) Wave front sweeping the region to update.

The algorithms find a set of edges (and vertices), some of them to be added to V_k (in a deletion) or removed from V_{k-1} (in an insertion). Apart from polygon R_k, which is computed in an insertion, this set forms a connected acyclic graph G (see Fig. 1a). The *inner edges* of G are those edges whose both endpoints are in the interior of R_k.

The size and order of G may be deduced from the number m of neighbours of s_k. For n sites on the plane or on the sphere surface, Euler's formula states that $e \propto 3\,n$ and $v \propto 2\,n$, where e is the number of edges and v is the number of vertices [4]. Since the update of a single site implies a variation of three edges and two vertices in the diagram, graph G has $m - 3$ inner edges, whose endpoints define $m - 2$ vertices. So, to some extent, in a deletion, a polygon with m edges is replaced by a graph with $m-3$ inner edges, whereas, in an insertion, a graph with $m - 3$ inner edges is replaced by a polygon with m edges.

III. Hyperbolic Sweep

Both algorithms sweep R_k with a circular line, called the *sweep circle*, centred at s_k. The *wave front* is made of arcs of hyperbola, where each arc is defined by the sweep circle and a site. Before detailing the algorithms, we will show how a single arc of hyperbola sweeps half of the domain, how the intersections of two arcs scan half of the sites bisector, and how three arcs may define a vertex. Since these matters depend on the domain, Section III-A is dedicated to the sphere, while Section III-B deals with the plane.

A. Hyperbolic Sweep on the Sphere

The spherical domain is swept by a circular line, following the approach introduced by Dinis and Mamede [7]. In general, a point p on the sphere is defined by a pair (p_ρ, p_θ),

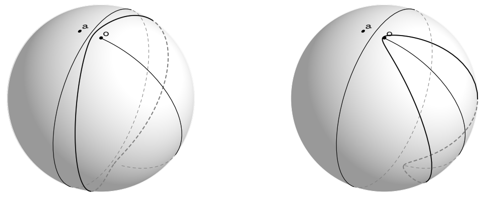


Figure 2. Scan of a half-sphere by a hyperbola (when r increases).

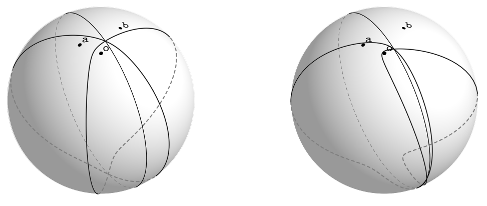


Figure 3. Scan of the half-bisector m_{ab} by the intersections of two hyperbolas (when r increases).

where p_ρ is the colatitude and p_θ is the longitude. But, in order to simplify some expressions, p may be given in 3D coordinates, which will be denoted by $\overrightarrow{p}$. We also assume that the sweep circle is always centred at the north pole o, that is, $s_k = o = (0, 0)$. As a consequence, the radius of the sweep circle, which is the length of an arc of great circle, is the colatitude of a parallel.

Let then $a = (a_\rho, a_\theta)$ be a site on the sphere and r be the radius of the sweep circle. Now, consider the loci of points $i = (i_\rho, i_\theta)$ such that

$$|i - a| = i_\rho + r\,. \tag{1}$$

So, i is the centre of a circle tangent to the sweep circle (with o in its interior) that contains a on its border. Equation 1 defines a branch of a spherical hyperbola whose foci are a and o (see Fig. 2). To simplify, these spherical hyperbola arms will be simply called *hyperbolas*.

To start with, recall that spherical hyperbolas are always closed curves (and congruent to spherical ellipses). The shape of the hyperbola is delimited by a spherical lune whose dihedral angle is $\lambda = 2\arccos(\csc(a_\rho/2)\sin(r/2))$. At $r = 0$, $\lambda = \pi$ and the hyperbola is degenerated into the foci bisector (which is a great circle). As r increases, λ decreases and the hyperbola closes in toward o (as in Fig. 2). Finally, when $r = a_\rho$, $\lambda = 0$ and the hyperbola is also degenerated, now into the arc of great circle that connects o to the antipode of a. Therefore, the hyperbola monotonically scans the half-sphere (that contains o) defined by the two foci bisector, as r varies from 0 to a_ρ.

The intersections of two hyperbolas, generated by sites a and b, scan half of the sites bisector. Let then $b = (b_\rho, b_\theta)$ be another site such that $b_\rho > a_\rho$ (c.f. Fig. 3), and m_{ab}, m_{ao}, and m_{bo} be the bisectors between sites (a, b), (a, o), and (b, o), respectively. First of all, remark that, due to (1), every intersection point i satisfies $|i - a| = |i - b| = i_\rho + r$, which means that i lies on m_{ab}.

The intersection points are at antipodal positions when $r = 0$, because the hyperbolas coincide with m_{ao} and m_{bo}, which are both great circles. As r increases, both intersections move towards each other, until they coincide. This happens for $r = a_\rho = \min(a_\rho, b_\rho)$, when one of the hyperbolas completes its sweep.

To understand how vertices are found, let $c = (c_\rho, c_\theta)$ be a third site. Without loss of generality, we assume that $a_\rho < c_\rho$, and that a, b and c are in strict clockwise order around o (as in Fig. 4). We are interested only in the inner envelope of the three hyperbolas with respect to o. So, let $\langle a\rangle$, $\langle b\rangle$, and $\langle c\rangle$ be the three arcs of hyperbola, whose common focus is o, $\langle a,b\rangle$, $\langle b,c\rangle$, and $\langle c,a\rangle$ be the corresponding arc intersections, and $v = (v_\rho, v_\theta)$ and $r_{\{a,b,c\}}$ be the centre and the radius of the circle that circumscribes the three sites. The issue is to determine the conditions under which arc intersections $\langle a,b\rangle$ and $\langle b,c\rangle$ meet at v, and arc $\langle b\rangle$ disappears. But $\langle a,b\rangle$ and $\langle b,c\rangle$ effectively scan v if $\langle c,a\rangle$ does not. To check if v is reached by $\langle c,a\rangle$, we may compare the relative position of v with respect to the great circle $\overline{o\,a}$ (because $a_\rho < c_\rho$). So, arc $\langle b\rangle$ disappears if:

$$(\overrightarrow{o} \times \overrightarrow{a}) \cdot \overrightarrow{v} < 0 \quad (v \text{ is scanned by } \langle b\rangle). \tag{2}$$

Notice that $\overrightarrow{o} \times \overrightarrow{a}$ is never the zero vector because sites a, c, and o are distinct, and $a_\rho < c_\rho$ implies that a and o are not at antipodal positions. The elimination of $\langle b\rangle$ occurs when the sweep circle radius $r = r_{\{a,b,c\}} - v_\rho$. Similar conditions apply if c is closer to o than a.

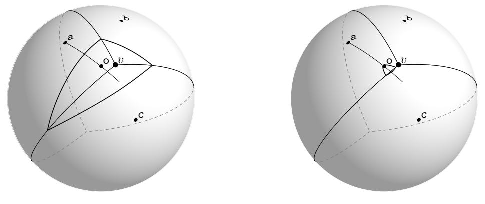


Figure 4. Finding a vertex by an arc elimination.

B. Hyperbolic Sweep on the Plane

Now, half-spheres are replaced by half-planes, great circles by lines, arcs of great circle by line segments, and the great circle distance by the Euclidean distance. In this case, we consider that points $p = (p_\rho, p_\theta)$ are given in polar coordinates, and that $\overrightarrow{p}$ stands for p in 2D coordinates. Furthermore, we still assume that s_k is placed at the origin o, i.e., $o = s_k = (0, 0)$.

In this context, given a site $a = (a_\rho, a_\theta)$ on the plane and the sweep circle radius r, the loci of points that satisfy (1) define a branch of hyperbola whose foci are a and o. The

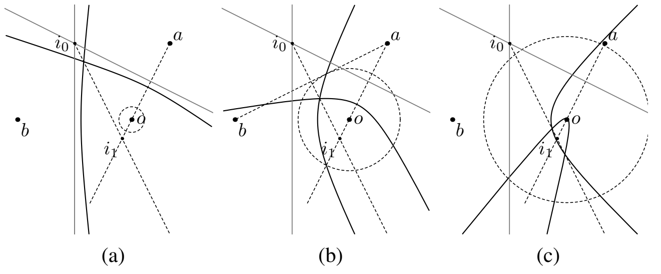


Figure 5. Scan of the half-bisector m_{ab} by the intersections of two hyperbolas, when $\overline{a \triangleright o}$ intersects m_{ab} (and r increases).

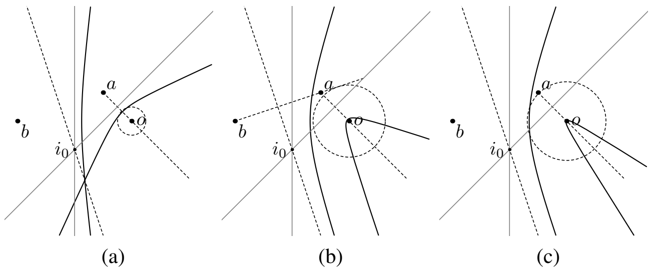


Figure 6. Scan of the half-bisector m_{ab} by the intersections of two hyperbolas, when $\overline{a \triangleright o}$ does not intersect m_{ab} (and r increases).

hyperbola parameters depend on a_ρ and r: $\epsilon = a_\rho / r$ is the eccentricity, and $\lambda = \arccos(r/a_\rho)$ is the asymptote slope.

This branch of hyperbola (also abbreviated to *hyperbola*) sweeps half of a plane (containing o), as r varies from 0 to a_ρ. When $r = 0$, it coincides with the bisector line between a and o; when r increases, the hyperbola asymptotes close in toward o; and when $r = a_\rho$, the hyperbola is the half-line with origin in o towards the opposite direction of a.

Although the intersections of two hyperbolas still scan half of the sites bisector, three distinct cases have to be analysed. In all of them, $b = (b_\rho, b_\theta)$ is a site such that $b_\rho > a_\rho$, and m_{ab}, m_{ao}, and m_{bo} are the bisector lines between sites (a, b), (a, o), and (b, o), respectively.

We start by assuming that a, b, and o are not co-linear and that the half-line $\overline{a \triangleright o}$ intersects m_{ab} (as depicted in Fig. 5). Initially, when $r = 0$, the hyperbolas coincide with m_{ao} and m_{bo}, and intersect at i_0. When $r = \text{distance}(o, \overline{a\,b})$, one asymptote of each hyperbola will be parallel to m_{ab} (see Fig. 5b) and a second intersection (which begins at infinity) starts to be traced. The scan of half of m_{ab} ends when $r = a_\rho$ because, since a is closer to o, its hyperbola is the first to complete the half-plane sweep. At that moment, both intersections meet (the one that started at i_0 and the one that started at infinity).

In a second configuration, the three sites are still not co-linear, but the half-line $\overline{a \triangleright o}$ does not intersect m_{ab} (c.f. Fig. 6). The half-bisector scan is performed similarly, except that it takes place when the sweep circle radius r varies from 0 to distance$(o, \overline{a\,b})$. At the beginning, the two arcs intersect at i_0 and, at the end (Fig. 6b), the asymptotes of the same

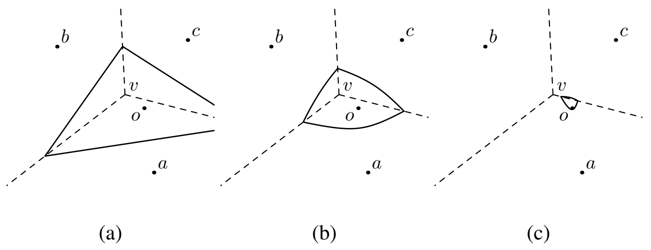


Figure 7. Finding a vertex in an originally closed envelope.

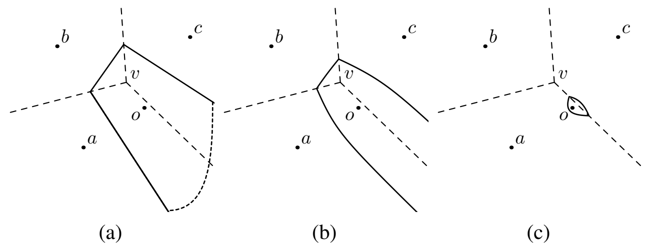


Figure 8. Finding a vertex in an artificially closed envelope.

arcs are parallel to each other and to m_{ab}. Remark that, in this case, only one side of each hyperbolic arc is brought into play.

The third case is when all sites are co-linear, and o is between a and b. Now the two hyperbolas face each other and both sides intersect at all times. The two intersection points begin at infinity and join half way between a and b (when $r = \min(a_\rho, b_\rho)$).

In the plane, the wave front is also the inner envelope of the hyperbolas (with respect to o) and its topological changes signal vertices locations. Let then $c = (c_\rho, c_\theta)$ be another site in the conditions shown in Fig. 7. That is, $a_\rho < c_\rho$ and a, b and c are in strict clockwise order around o. The three arcs of hyperbola are $\langle a \rangle$, $\langle b \rangle$, and $\langle c \rangle$, the corresponding arc intersections are $\langle a,b \rangle$, $\langle b,c \rangle$, and $\langle c,a \rangle$ (provided they exist), and the centre and the radius of the circle circumscribing the three sites are $v = (v_\rho, v_\theta)$ and $r = r_{\{a,b,c\}}$. In this case, for arc $\langle b \rangle$ to disappear, intersections $\langle a,b \rangle$ and $\langle b,c \rangle$ must converge and must reach v. Notice that the latter depends on the relative position of v with respect to line $\overline{o\,a}$ (because $a_\rho < c_\rho$). So, arc $\langle b \rangle$ disappears if:

$$\begin{cases} \left(\overrightarrow{c-a}^{\,i\frac{\pi}{2}}\right) \cdot \overrightarrow{b-a} < 0 & \text{(intersections converge)}, \\ \left(\overrightarrow{a}^{\,i\frac{\pi}{2}}\right) \cdot \overrightarrow{v} > 0 & (v \text{ is scanned by } \langle b \rangle) \end{cases} \quad (3)$$

where $\overrightarrow{x}^{i\theta}$ means $\overrightarrow{x}$ rotated θ clockwise. Once more, $\langle b \rangle$ is eliminated when the sweep circle radius $r = r_{\{a,b,c\}} - v_\rho$.

The arc elimination process may be extended to non-closed envelopes by considering an auxiliary arc (drawn with a dot line in Fig. 8a) that connects the two diverging half-

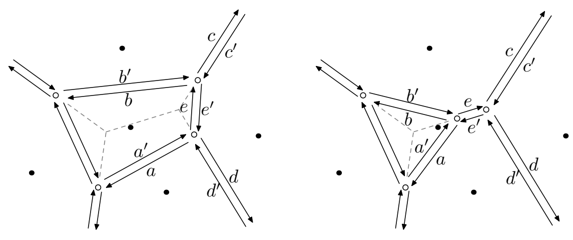


Figure 9. Edge flip by rotating e clockwise.

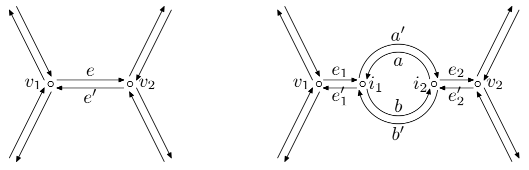


Figure 10. Inserting or removing a pair of edges.

lines. This auxiliary arc also disappears in the course of the sweep, like a regular one. In the case illustrated in Fig. 8, where the half-line $\overline{a \triangleright o}$ and m_{ac} intersect, the auxiliary arc disappears at $r = \text{distance}(o, \overline{a\,c})$ (see Fig. 8b), because the asymptotes of the two adjacent arcs become parallel and $\langle a,c \rangle$ begins to scan m_{ac}.

The trick of adding auxiliary arcs to open envelopes of hyperbolas allows to design the update algorithms as if all regions were bounded. For this purpose, we suppose that an auxiliary site, located at infinity, was inserted in the Voronoi diagram. Then, every unbounded region is artificially closed by adding an auxiliary edge, which represents an imaginary bisector between the corresponding site and the auxiliary site.

IV. Operations on a DCEL

The algorithms make use of three operations on a DCEL: edge flipping, and inserting or deleting a pair of edges.

The edge flip operation is illustrated in Fig. 9, where an edge e and its four adjacent edges are represented by their *twin* half-edges. When e is flipped, two adjacent edges (b and d) are reattached to the opposite endpoint of e, while the other two adjacent edges (a and c) remain unchanged.

Notice that, because edges are always shared by two regions, an edge flip removes an edge from the boundaries of two adjacent regions and adds it into the boundaries of two other adjacent regions. In particular, when an edge e is flipped, e is effectively deleted from the boundaries of the regions to which it belongs (c.f. the changes from Fig. 9a to Fig. 9b). Besides, the flip of an edge incident to a region has the effect of adding that edge into the region boundary (as it can be seen when e is flipped from Fig. 9b to Fig. 9a).

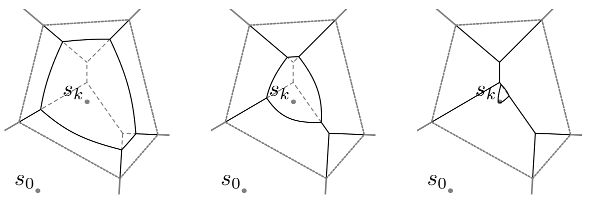


Figure 11. Deleting a site.

The deletion of a site is performed by contracting its region, releasing the edges into the neighbouring regions through a succession of edge flips, until the boundary is reduced to two edges. Then, this pair of edges is removed (in a single operation).

Basically, the insertion of a site performs that sequence of basic operations in the opposite direction. It begins with the insertion of a pair of edges, which is the new site initial boundary. Then, that boundary is expanded by a sequence of edge flips, absorbing edges from the neighbouring regions, until it reaches its final shape.

Given an edge e and a new site s_k, the insertion of a pair of edges associated with s_k in e consists in replacing e with four edges, adding two new endpoints, i_1 and i_2, as depicted in Fig. 10 (from left to right). The inner arcs are both made incident to i_1 and i_2. Actually, those arcs are self-loops, because initially i_1 and i_2 coincide. The outer edges e_1 and e_2, which result from the split of e, are incident to a new endpoint and to one of the original vertices. As we have already mentioned, this operation will be used to create and initialise the DCEL of s_k with the two inner arcs.

When s_k is removed, its DCEL ends with two arcs whose endpoints are all at the same location. The other two arcs that are incident to those endpoints (which are e_1 and e_2 in Fig. 10b) form up an edge of V_{k-1}. So, given the DCEL of s_k, the deletion of the corresponding pair of edges replaces e_1 and e_2 by a single edge (as it is shown in Fig. 10a).

V. Deletion Algorithm

The first algorithm deletes site s_k from V_k. Let then $s_i = (s^i_\rho, s^i_\theta)$, for every $i = 0, \dots, m-1$ and some $m < k$, be the neighbours of s_k (in V_k). For the sake of simplicity, we assume that $s_k = (0, 0)$ (that is, it is located at the north pole or at the plane origin) and s_0 is its nearest neighbour.

The sweep circle (centred at s_k) together with the m neighbours of s_k define m hyperbolas. The inner envelope of these hyperbolas is the *wave front*. It coincides with R_k when the sweep circle radius $r = 0$. As r increases, arcs are eliminated from the wave front, signalling the presence of new vertices and new edges. The process ends when the hyperbola generated by s_0 completes its sweep, that is, when $r = s^0_\rho$. So, the deletion algorithm can somehow be seen as a particular case of Fortune's algorithm [4], [8] where there are only *circle-events* (c.f. Fig. 11).

Algorithm 1 Delete s_k from V_k

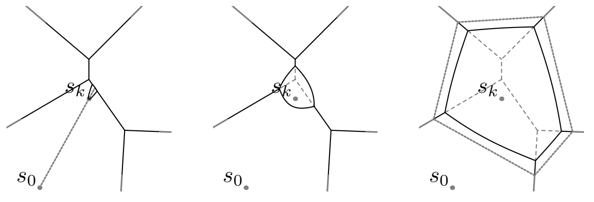
```
1: list: $w \leftarrow V_k$.DCEL($s_k$)              {wave front is $R_k$}
2: array: $a \leftarrow \emptyset$        {auxiliary array of length $w$.size()}
3: for every $e$ in $w$ do
4:     $k \leftarrow w$.compute_priority($e$)
5:     if $k < +\infty$ then
6:         $a$.add(($k, e$))
7:     end if
8: end for
9: queue: $q \leftarrow$ build_queue($a$)  {build the priority queue}
10: while $q \neq \emptyset$ do
11:     $(k, e) \leftarrow q$.delete_min()
12:     $p \leftarrow w$.previous($e$)
13:     $n \leftarrow w$.next($e$)
14:     $k_p \leftarrow w$.compute_priority($p$)
15:     $q$.insert_or_update_or_delete($p, k_p$)
16:     $k_n \leftarrow w$.compute_priority($n$)
17:     $q$.insert_or_update_or_delete($n, k_n$)
18:     $w$.flip_edge($e$)
19: end while
20: $V_k$.delete_pair_edges($w$)              {$w$ has 2 elements}
```

Algorithm 1 starts by initialising the wave front with the R_k DCEL (line 1) and, for each one of the initial arcs, it checks if the arc may disappear (that is, if (2) or (3) holds). If that is the case, the corresponding circle-event is added to a priority queue, which is ordered by ρ (lines 2–9). Then, while the priority queue is not empty, an event is removed and processed (lines 10–19). The event processing deletes an arc from the wave front (through an edge flip, which leaves a vertex behind), and updates the priorities associated with its two adjacent arcs. When the queue is exhausted, the wave front has two arcs which scan the same edge of the Voronoi diagram. Therefore, it remains to discard those arcs and to appropriately unite the split edge (line 20).

The proof of Proposition 1 (which is omitted due to the lack of space) assumes that the priority queue is implemented with a binary heap and an array (which associates every event in the priority queue with its position in the binary heap) and that every arc in the priority queue has a pointer to the corresponding DCEL cell of the Voronoi diagram. Recall that the number of processed events is $m-2$.

Proposition 1: Algorithm 1 deletes a site with m neighbours in $O(m \log m)$ time using $O(m)$ space.

As a matter of fact, our algorithm is quite similar to that of Devillers [5], exchanging edges for ears and vertices for triangles. The main difference relies on the priority definition, and the power of a site with respect to the circle circumscribing an ear involves the computation of the circle circumscribing three points (which is all we need to define a priority). Because of this high cost, simpler quadratic solutions are reported by the author to run faster in practise [6].



(a) (b) (c)

Figure 12. Inserting a site.

VI. Insertion Algorithm

Conceptually, the insertion algorithm (illustrated in Fig. 12) mimics the deletion algorithm in reverse order. As in the previous section, let s_k be located at the north pole or at the origin, and s_0 be the closest site to s_k. The sweep circle is also centred at s_k, but now its radius decreases from s_ρ^0 to zero. Like before, the wave front is the inner envelope of all hyperbolas. Recall that all regions are closed.

Let e_0 be the edge of R_0 that is intersected by the half-great circle (respectively, half-line) $\overline{s_0 \triangleright s_k}$, i be that intersection point, and s_1 be the neighbour of s_0 that shares e_0. The initial wave front (c.f. Fig. 12a) has two arcs, one generated by s_0 and the other by s_1. Moreover, it is the arc of great circle (respectively, the line segment) whose endpoints are s_k and i, and the two arc intersections coincide with i. As r decreases, the two arcs of hyperbola widen out and their intersections scan e_0 until a third arc becomes part of the wave front (see Fig. 12b). This occurs when one of the arc intersections reaches a vertex v of V_{k-1} (which is an endpoint of e_0). If the maximum empty circle γ of v (in V_{k-1}) contains s_k, the vertex is *in conflict* with s_k and must be removed. Notice that $v = (v_\rho, v_\theta)$ is in conflict with s_k if $r_\gamma - v_\rho \geq 0$, where r_γ is the radius of γ. In that case, an event is scheduled so that the wave front gains an arc and two new arc intersections, which will scan two new edges. The priority of that event is given by $r = r_\gamma - v_\rho$.

So, the task of the circular sweep is basically to find out and remove all vertices (and edges) that are in conflict with the site to be inserted, which characterise graph G referred to in Section II. But, since G is connected and acyclic, its extent is easily computed with a (partial) traversal of V_{k-1} that starts at some position known to belong to G. That is why edge e_0 will play a crucial role. It is important to notice that the diagram traversal may be performed in any order (that is, event priorities are irrelevant), as all scheduled events are independent and must be processed.

The first step of Algorithm 2 is to find edge e_0 (lines 1–6). The wave front begins with two arcs (line 7), whose intersections are on e_0, and the event queue is created (line 8). Then, two vertices are reached (the endpoints of e_0) and, if they are in conflict with s_k, the corresponding events are added into the queue (lines 9–15). When an event is processed, two new vertices are found and “enqueued”

Algorithm 2 Insert s_k in V_{k-1}; s_0 is the site nearest to s_k

```
1:  for every $e$ in $V_{k-1}$.DCEL($s_0$) do
2:    if $\overline{s_0 \triangleright s_k}$ intersects $e$ then
3:      $e_0 \leftarrow e$                          {initial edge found}
4:      break
5:    end if
6:  end for
7:  $V_{k-1}$.insert_pair_edges($e_0, s_k$)           {$R_k$ first 2 arcs}
8:  queue: $q \leftarrow \emptyset$                   {event queue is a FIFO}
9:  if $V_{k-1}$.exists_conflict($e_0$.origin(), $s_k$) then
10:   $q$.enqueue($e_0$)
11: end if
12: $e_1 \leftarrow V_{k-1}$.twin($e_0$)
13: if $V_{k-1}$.exists_conflict($e_1$.origin(), $s_k$) then
14:   $q$.enqueue($e_1$)
15: end if
16: while $q \neq \emptyset$ do
17:   $e \leftarrow q$.dequeue()
18:   $e' \leftarrow V_{k-1}$.previous($e$)
19:   if $V_{k-1}$.exists_conflict($e'$.origin(), $s_k$) then
20:     $q$.enqueue($e'$)
21:   end if
22:   $e'' \leftarrow V_{k-1}$.previous($V_{k-1}$.twin($e'$))
23:   if $V_{k-1}$.exists_conflict($e''$.origin(), $s_k$) then
24:     $q$.enqueue($e''$)
25:   end if
26:   $V_{k-1}$.flip_edge($e$)
27: end while
```

providing they are in conflict with s_k (lines 18–25). In addition, one arc and two arc intersections are always added into the wave front, which is done by an edge flip in the Voronoi diagram (line 26). This procedure is repeated until all scheduled events have been processed. At that moment, all edges of R_k are known as they correspond to the wave front arcs (when the sweep circle radius is zero).

The following result is a consequence of the queue policy.

Proposition 2: Algorithm 2 inserts a site s_k in $O(t+m)$ time using $O(m)$ space, where t is the number of neighbours of the site nearest to s_k in the initial Voronoi diagram and m is the number of neighbours of s_k in the resulting Voronoi diagram.

Even though the algorithm has been designed according to the sweep technique, its final version turns out to perform a graph traversal, like the insertion step of the incremental algorithm for computing a Delaunay triangulation. The main difference is that they update different data structures.

VII. Experimental Results

Now we present some experimental results that show how our update algorithms compare with those provided by the CGAL software library [2]. CGAL has been chosen because it is widely used, academically sound, and freely available in source code form. In the planar domain, we have compared our algorithms with the CGAL update algorithms that run on a Delaunay triangulation. In the spherical domain, since CGAL does not offer Delaunay triangulations of sites on a sphere surface, we have resorted to the Delaunay triangulation of points in 3D, which provides the convex-hull. Notice that any triangulation is suitable, but that is the only one where insertions and deletions can be made.

The CGAL (incremental) insertion algorithms are similar in both domains. First, there is a *walk* in the triangulation, which begins at an arbitrary location, to find a triangle (or a tetrahedron) in conflict with the new site. Then, the site is inserted. For our purposes, we have measured only the running times of the insertion phase.

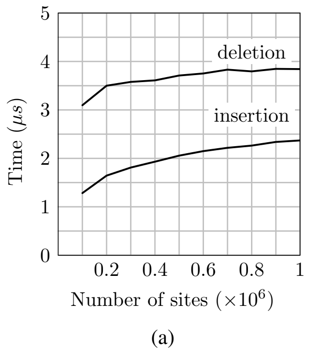
As we have already said, in the plane, up to CGAL 3.6.1 deletion is performed with the boundary completion algorithm. Since CGAL 3.7 (the last release, so far), the library uses a decision tree (by Devillers [6]) to choose the algorithm that is executed, which depends on the number m of neighbours of the site to be deleted. There is a specialised function for each $m = 3, \ldots, 7$, and the boundary completion routine is invoked only in the remaining cases. For this reason, in the plane our experiments have involved both deletion versions. Deletion in 3D starts by deleting all tetrahedrons incident to the site to be removed, which creates a hole. At the same time, all neighbouring sites are collected. Then, the algorithm computes the Delaunay triangulation of the neighbours, which is used to sew up the hole.

Due to the low average number of neighbours, we discarded the priority queue from our deletion algorithm. In fact, the algorithm performance improves when the delete_min operation is implemented with a sequential traversal of the wave front, which allows to eliminate all insertions, updates, and deletions of events.

Our algorithms have been coded in the C language and compiled with the GNU GCC C compiler (version 4.3.2) with optimisations. Programs have been executed on a computer running Linux, equipped with an Intel Xeon E5160 processor, running at 3.00 GHz, with 4 MB of cache.

Concerning data sets, sites are at random positions, distributed uniformly either on the $[0,1] \times [0,1]$ square or on the surface of the sphere. Locations on the sphere are in the 3D Cartesian coordinate system, so that running times can be comparable. For each domain, five independent data sets have been generated, with sizes $0.1 \times 10^6, 0.2 \times 10^6, \ldots, 1.0 \times 10^6$. Every presented result is the mean of the five measurements obtained from the executions with the sets of the same domain and size.

The execution of an insertion algorithm with a data set consists in the computation of the corresponding Voronoi diagram or Delaunay triangulation, by inserting sites successively in no particular order. After having built the Voronoi diagram or the Delaunay triangulation of a data set, the execution of a deletion algorithm removes all sites one by



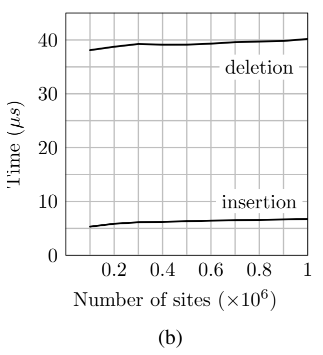
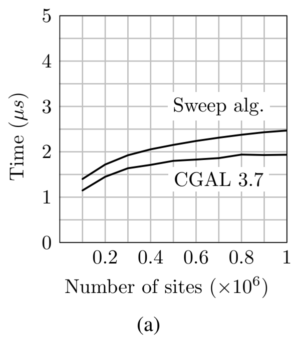


Figure 13. Average running times on the sphere. (a) Sweep algorithms. (b) CGAL algorithms.



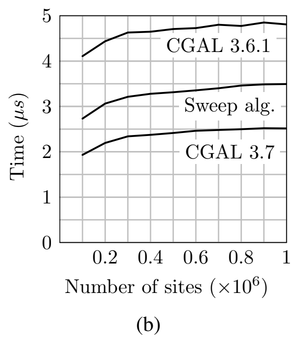


Figure 14. Average running times on the plane. (a) Insertion algorithms. (b) Deletion algorithms.

one, respecting the order of a random input permutation.

The average running times on the sphere are depicted in Fig. 13. It turns out that all algorithms get slower when the number of sites increases, because the cache capacity is limited and the hit rate decreases. Besides, there is a striking difference between the two approaches, given that the CGAL running times are larger than those of the sweep algorithms approximately by a factor of 2.8 for insertions, and 10 for deletions. The justification for these discrepancies is that 3D Delaunay triangulations are more expensive data structures. In particular, deletions are strongly penalised.

Fig. 14 plots the average running times on the plane, which also vary with the number of sites. Concerning insertions (see Fig. 14a), the CGAL algorithm outperforms ours (approximately by a factor of 1.3). Similarly, the three deletion algorithms have comparable performances (c.f. Fig. 14b). The latest version of CGAL is the fastest of all, with a twofold improvement from the previous release, which proves the effectiveness of the specialised functions. The sweep algorithm performs better than CGAL 3.6.1, but (approximately 1.4 times) worse than the current release.

VIII. Conclusions

We have presented two algorithms for updating Voronoi diagrams of points on a sphere surface or on a plane. In spite of the deletion running time being not optimal, both algorithms are efficient and practical to implement. Furthermore, they handle degenerated cases smoothly, which is a characteristic of the sweep based algorithms. It is worth mentioning that, even though many applications benefit from direct implementations of Voronoi diagrams, only a few algorithms cope straight with their updates.

The experimental results have shown that our algorithms are competitive on the plane. On the sphere, their performances were undoubtedly the best, which lead us to conclude that this domain requires specific algorithms. The sweep technique, which had already proved to suit the sphere particularities [7], allowed us to design simple and very efficient algorithms for updating spherical Voronoi diagrams.

References

[1] A. Aggarwal, L. Guibas, J. Saxe, and P. Shor. A linear time algorithm for computing the Voronoi diagram of a convex polygon. In *Proc. of STOC'87*, pages 39–45, 1987.

[2] CGAL: Computational Geometry Algorithms Library (release 3.6.1 and 3.7). (`http://www.cgal.org`).

[3] L. P. Chew. Building Voronoi Diagrams for Convex Polygons in Linear Expected Time. Technical Report PCS-TR90-147, Dartmouth College, Hanover, NH, January 1990.

[4] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, second edition, 2000.

[5] O. Devillers. On deletion in Delaunay triangulation. *Internat. J. Comput. Geom. Appl.*, 12:193–205, 2002.

[6] O. Devillers. Vertex removal in two-dimensional delaunay triangulation: Speed-up by low degrees optimization. *Comput. Geom. Theory Appl.*, 44:169–177, 2011.

[7] J. Dinis and M. Mamede. Sweeping the sphere. In *Proc. of ISVD'2010*, pages 151–160, 2010.

[8] S. Fortune. A sweepline algorithm for Voronoi diagrams. *Algorithmica*, 2:153–174, 1987.

[9] L. J. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. In *Proc. of STOC'83*, pages 221–234, 1983.

[10] C. L. Lawson. Software for C^1 surface interpolation. In J. Rice, editor, *Mathematical Software III*, pages 161–194. Academic Press, 1977.

[11] M. A. Mostafavi, C. Gold, and M. Dakowicz. Delete and insert operations in Voronoi/Delaunay methods and applications. *Comput. Geosci.*, 29(4):523–530, 2003.

[12] R. J. Renka. Algorithm 772: STRIPACK: Delaunay triangulation and Voronoi diagram on the surface of a sphere. *ACM Trans. on Mathematical Software*, 23(3):416–434, 1997.

[13] R. Sibson. A brief description of natural neighbor interpolation. In *Chapter 2 in Interpolating multivariate data*, pages 21–36. John Wiley & Sons, New York, 1981.