Additively Weighted Voronoi Diagram on the Oriented Projective Plane

Guilherme A. Pinto and Pedro J. de Rezende
{guialbu,rezende}@dcc.unicamp.br

Relatório Técnico IC–00-03

Fevereiro de 2000

Additively Weighted Voronoi Diagram on the Oriented Projective Plane

Guilherme A. Pinto and Pedro J. de Rezende
{guialbu,rezende}@dcc.unicamp.br

Abstract

We consider Voronoi diagrams defined on the oriented projective plane $\mathbb{T}^2$. In this geometry, the closest and furthest site diagrams are antipodal. We give a simple on-line incremental algorithm for constructing the additively weighted diagram. This diagram, which may be disconnected in Euclidean plane, is always connected in $\mathbb{T}^2$ and has exactly $3n-6$ edges and $2n-4$ vertices, where n is the number of sites.

Keywords: Voronoi diagrams; oriented projective plane; incremental algorithm

1 Introduction

One of the problems in designing and implementing algorithms for Voronoi diagrams is the manipulation and representation of the edges extending to infinity. Although this may not be a difficult problem, solutions are artificial. For explicitly construction of Voronoi diagrams we list three common solutions: bounding the diagram with a window or a polygon [5]; introducing "virtual edges" connecting adjacent infinite edges [15, 13] and introducing a "virtual vertex" connecting all infinite edges [12, 6]. The first solution rules out infinite edges. The other two unify the topological representation of finite and infinite edges but introduce meaningless data.

We consider Voronoi diagrams constructed on the oriented projective plane $\mathbb{T}^2$ [14] which is comprised of two copies of $\mathbb{E}^2$ plus one line at infinity. With the definition of Euclidean concepts like distance and perpendicularity, the $\mathbb{T}^2$ is called the two-sided Euclidean plane, where we can treat all edges of the diagram uniformly. Furthermore, when the sites are contained in one side of the plane, the part of the closest site diagram contained in the other side of the plane happens to be the furthest site diagram (as we know it in $\mathbb{E}^2$) and vice-versa. In Section 2 we introduce the basic concepts of $\mathbb{T}^2$ and show how Voronoi diagrams appear on it.

In Section 3 we present an on-line incremental algorithm for the additively weighted diagram. Previous algorithms for the closest site diagram can be found in [8, 13, 4, 15, 2, 12] and for the furthest site diagram in [12, 11]. This diagram has an additional benefit from $\mathbb{T}^2$: it is always connected and every face is simply connected, whereas in $\mathbb{E}^2$ the closest site diagram may have $O(n)$ connected components [13, 4] and the furthest site diagram may have faces with $O(n)$ connected components [11]. Our algorithm is, as any incremental algorithm for planar Voronoi diagrams, not worst case optimal, but it is of practical interest for being on-line and very simple to implement. Moreover, it shows that we can compute this construction without defining different procedures for each side of the plane.

2 Oriented Projective Plane and Voronoi Diagrams

In $\mathbb{T}^2$ a point with homogeneous coordinates $[x, y, w]$ is *not* identical to $[-x, -y, -w]$. They are called antipodal points. The antipode of point p is denoted by $\neg p$. The set of points with $w > 0$ is called the front side of the plane, and those with $w < 0$ are the back side. The set of points with $w = 0$ (except for the invalid triplet $[0, 0, 0]$), which are referred to as improper points, is the line Ω at infinity. In the figures we will use two geometric models for $\mathbb{T}^2$: the flat model, with the usual mapping $[x, y, w] \mapsto (x/w, y/w)$ to Cartesian coordinates, and the spherical model with the mapping $[x, y, w] \mapsto (x, y, w)/\sqrt{x^2 + y^2 + w^2}$. These two models are related by central projection.

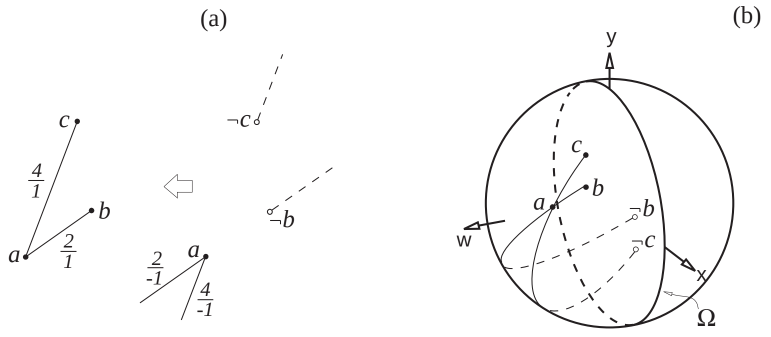

Figure 1: Comparing distances in $\mathbb{T}^2$

The distance between two proper points is defined by the expression: $\mathrm{d}_{\mathbb{T}^2}(a, b) = \sqrt{(x_a w_b - x_b w_a)^2 + (y_a w_b - y_b w_a)^2}/w_a w_b$. This formula yields negative numerical values when used with points in different sides of the plane, and positive values for points in the same side. However, in order to define the diagrams we need to operate and compare these values consistently. This is possible when we avoid the division and associate each value x/w to a point $[x, w]$ of the oriented projective line $\mathbb{T}^1$, $\mathrm{d}_{\mathbb{T}^2} : \mathbb{T}^2 \times \mathbb{T}^2 \to \mathbb{T}^1_+$, defined as $\mathrm{d}_{\mathbb{T}^2}(a, b) = [\sqrt{(x_a w_b - x_b w_a)^2 + (y_a w_b - y_b w_a)^2}, w_a w_b]$. Note that the numerator is always positive, so that the image of the function corresponds to a half of each side of $\mathbb{T}^1$. We call this set $\mathbb{T}^1_+$.

To compare two points a,$b \in \mathbb{T}^1$ we use the rule: $a <_{\mathbb{T}^1} b$ if and only if a_w and b_w have opposite signs and $a_w > 0$; or a_w and b_w have the same sign and $a_x b_w < b_x a_w$. Note that $\mathrm{d}_{\mathbb{T}^2}(a, b) = \mathrm{d}_{\mathbb{T}^2}(b, a) = [x, w]$ and $\mathrm{d}_{\mathbb{T}^2}(\neg b, a) = \mathrm{d}_{\mathbb{T}^2}(b, \neg a) = [x, -w]$. Distances to improper points can also be compared, as described in [10]. Figure 1 shows an example of four segments in the flat and spherical models. Note that the spherical model allows a better visualization for the segments, and their relative lengths. This figure also introduces our drawing conventions: points are open dots and lines are dashed when they are in the back side, for the flat model, and when they are not visible for the spherical model.

We will need a function to add a point in $\mathbb{T}^1$ and a real value, $+ : \mathbb{T}^1_+ \times \mathbb{R} \to \mathbb{T}^1$, defined as $a + v = [a_x + va_w, a_w]$. Qualitatively, we get a bigger point if $v > 0$ and a smaller one if $v < 0$. We hasten to say that we will never compute distances in the algorithm. It will be completely based in the usual determinant predicate to decide the orientation of three points $a, b, c \in \mathbb{T}^2$:

Function 1 `counterClockWise(a, b, c)` *returns* `true` *if* `c` *lies to the left of the oriented line defined by* `a` *and* `b` *and* `false` *otherwise:*

```
return ( (xa(ybwc-ycwb) + ya(wbxc-wcxb) + wa(xbyc-xcyb)) > 0 );
```

2.1 Voronoi Diagrams

We will first discuss the Voronoi diagram of unweighted points, which is a special case of the weighted diagram. Let $\mathcal{S}$ be a finite set of points in $\mathbb{T}^2$, called sites. For each p,$q \in \mathcal{S}$ let

$$H_{pq} = \{x \in \mathbb{T}^2 | \mathrm{d}_{\mathbb{T}^2}(x,p) \leq_{\mathbb{T}^1} \mathrm{d}_{\mathbb{T}^2}(x,q)\}.$$

The Voronoi region of a site p is

$$R_p = \bigcap_{q \in \mathcal{S} \setminus \{p\}} H_{pq}.$$

We follow [13] and define the Voronoi diagram DVor($\mathcal{S}$) to be the set of points which belongs to more than one Voronoi region. A Voronoi edge is the intersection of two regions and a Voronoi vertex is the intersection of more than two regions.

When the sites are contained in the front side of $\mathbb{T}^2$, the part of the diagram contained in the back side of $\mathbb{T}^2$ is equivalent to the furthest site diagram. This is due to the fact that $\mathrm{d}_{\mathbb{T}^2}(x,p) \leq_{\mathbb{T}^1} \mathrm{d}_{\mathbb{T}^2}(x,q)$ if and only if $\mathrm{d}_{\mathbb{T}^2}(\neg x,p) \geq_{\mathbb{T}^1} \mathrm{d}_{\mathbb{T}^2}(\neg x,q)$:

$$\begin{array}{rcl} [x_1, w_1] & \leq_{\mathbb{T}^1} & [x_2, w_2] \;\; \textit{implies} \\ x_1(w_2) & \leq & x_2(w_1) \\ x_1(-w_2) & \geq & x_2(-w_1) \\ [x_1, -w_1] & \geq_{\mathbb{T}^1} & [x_2, -w_2]. \end{array}$$

Figure 2(a) shows the DVor of $\mathcal{S} = \{p, q\}$. Part (b) shows the diagram after three other sites were added to $\mathcal{S}$. The bold edges are the boundary of R_p.

Let $n = |\mathcal{S}|$, and let e and v be, respectively, the number of edges and vertices of DVor($\mathcal{S}$). Assuming that $\mathcal{S}$ contains no four cocircular points, we have the following properties, which reveal the similarities between DVor and the Voronoi diagram on a sphere of $\mathbb{E}^3$[1]:

1. for $n = 2$: the diagram is a line of $\mathbb{T}^2$;

2. for $n = 3$: $v = 2$ and $e = 3$. The vertices are antipodes, the edges are *not* segments of $\mathbb{T}^2$ and the regions are *not* convex (they contain the antipodal vertices);

3. for $n > 3$: every edge is a segment of $\mathbb{T}^2$ and every region is convex.

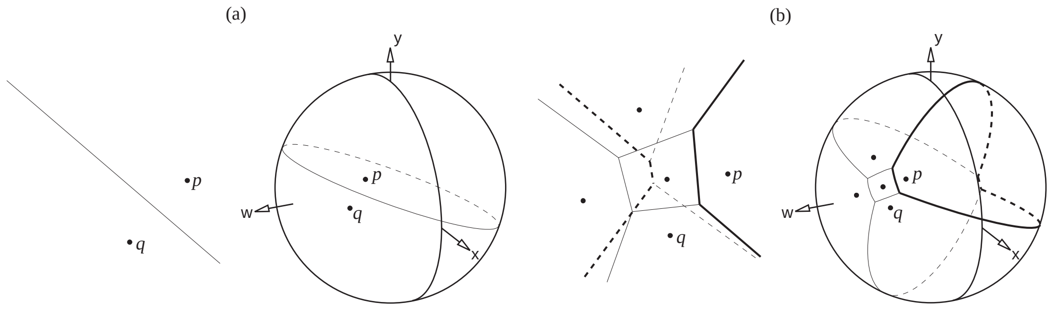

Figure 2: Voronoi diagram in $\mathbb{T}^2$

As we have n equal to the number of faces of the induced planar subdivision and every vertex has valence 3, $v = 2n - 4$ and $e = 3n - 6$.

Let us turn to the additively weighted diagram. Now, each site p has an associated weight $w(p)$. We assume, without loss of generality, that all weights are positive and regard the site as a *closed disk* with radius equal to its weight. For each p,$q \in \mathcal{S}$ we define

$$H_{pq} = \{x \in \mathbb{T}^2 | \mathrm{d}_{\mathbb{T}^2}(x,p) - w(p) \leq_{\mathbb{T}^1} \mathrm{d}_{\mathbb{T}^2}(x,q) - w(q)\},$$

and call DVorW the diagram generated by H. Some of the previous papers on this diagram use $dist(x,p)+w(p)$ instead. The definitions are equivalent in the sense that we can multiply the weights of the sites by -1 to obtain the diagram generated by the other definition. This is interesting to note because the part of the diagram contained in the back side, actually, is equivalent to the furthest site diagram generated by the other definition, as $\mathrm{d}_{\mathbb{T}^2}(x,p) - w(p) \leq_{\mathbb{T}^1} \mathrm{d}_{\mathbb{T}^2}(x,q) - w(q)$ if and only if $\mathrm{d}_{\mathbb{T}^2}(\neg x,p) + w(p) \geq_{\mathbb{T}^1} \mathrm{d}_{\mathbb{T}^2}(\neg x,q) + w(q)$:

$$\begin{array}{rcl}
[x_1, w_1] + (-v_1) & \leq_{\mathbb{T}^1} & [x_2, w_2] + (-v_2) \\
[x_1 - v_1 w_1, w_1] & \leq_{\mathbb{T}^1} & [x_2 - v_2 w_2, w_2] \;\; \textit{implies} \\
(x_1 - v_1 w_1)(w_2) & \leq & (x_2 - v_2 w_2)(w_1) \\
(x_1 - v_1 w_1)(-w_2) & \geq & (x_2 - v_2 w_2)(-w_1) \\
[x_1 - v_1 w_1, -w_1] & \geq_{\mathbb{T}^1} & [x_2 - v_2 w_2, -w_2] \\
[x_1, -w_1] + (v_1) & \geq_{\mathbb{T}^1} & [x_2, -w_2] + (v_2).
\end{array}$$

The intersection of two Voronoi regions is known to be one branch of an algebraic hyperbola opened towards the smallest disk and the regions are star-shaped with respect to the sites. This diagram is more complex than the DVor in the sense that it may have finite regions with only two edges and two regions may share more than one edge [13]. Figure 3(a) shows the DVorW of $\mathcal{S} = \{p, q\}$. Part (b) shows the diagram after three other sites were added to $\mathcal{S}$. Note that this example would be disconnected in $\mathbb{E}^2$.

Sharir [13] showed that, when the diagram is disconnect, every connected component is unbounded: If there could be a bounded connected component K, the portion E of a sufficiently small neighborhood of K, not contained in K, would belong to some site m. Clearly, there would be a point x in E such that the segment $\overline{xm}$ crosses K, contradicting the fact that the region of m is star-shaped.

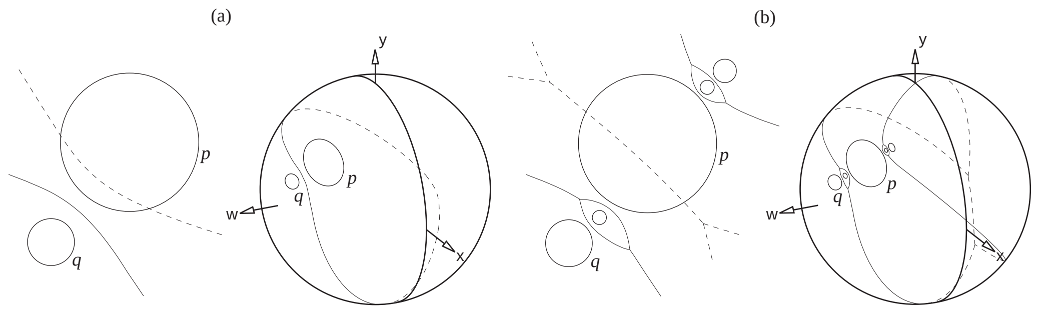

Figure 3: Additively weighted Voronoi diagram in $\mathbb{T}^2$

This argument is enough to show that DVorW is always connected, because there can be no unbounded component in the compact topology of $\mathbb{T}^2$. However, we need to distinguish two situations. When the antipode of the site m is not contained in K, the above argument applies without modifications. But when $\neg m$ belongs to K, the segments $\overline{xm}$, $x \in E$, do not cross K. Nevertheless, they cover all the $\mathbb{T}^2 \setminus K$, such that there would be a segment $\overline{xm}$ crossing some other component of the diagram, again contradicting the fact that the region of m is star-shaped.

In the next section we will present an incremental algorithm to construct this diagram. The algorithm is based on two primitive procedures: the calculation of the two *circuncenters* of three given sites; and the counterclockwise circular order predicate of three points b, c and d with respect to a fourth point a. The latter procedure can be implemented with three calls to the function `counterClockWise` as follows:

Function 2 `counterClockWiseOrder(b, c, d, a)` *returns* `true` *if the counterclockwise circular order of the points* `b`,`c` *and* `d` *around* `a` *is* `bcd` *and* `false` *otherwise:*

```
bool abc = counterClockWise(a, b, c);
bool abd = counterClockWise(a, b, d);
bool acd = counterClockWise(a, c, d);
return ((abc && (!abd || acd)) || (!abd && acd));
```

3 An Incremental Algorithm for the Additively Weighted Diagram

We assume, for simplicity, that no two disks of $\mathcal{S}$ intersect and that there is no four cocircular sites in $\mathcal{S}$. The algorithm, in fact, works for intersecting disks and for negative weights as well, but it needs some modifications if we allow one disk to be entirely contained in another disk. In the conclusions we will discuss these modifications.

Given these conditions, we describe now the other primitive procedure of the algorithm: the calculation of the two circuncenters a and b of three given sites p, q and r. A circuncenter is a point equidistant to the three sites according to the definition of H. In $\mathbb{E}^2$ three disks may have $0, 1$ or 2 circuncenters (see [12] and Fig. 4(a)). In $\mathbb{T}^2$ there are always two such points as Fig. 4(b) shows, for the three possible configurations.

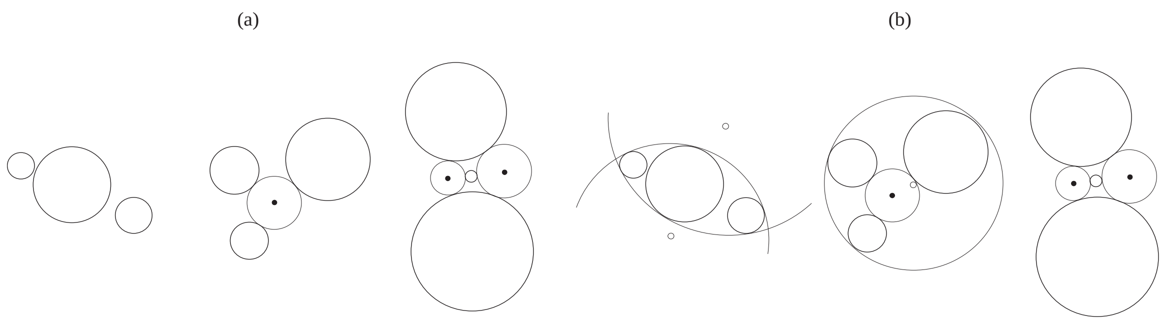

Figure 4: The circuncenters of three sites in $\mathbb{E}^2$ and $\mathbb{T}^2$

Function 3 `circunCenters(p, q, r, a, b)` *calculates the two circuncenters of the sites* `p,q` *and* `r` *storing them in points* `a` *and* `b`*.*

This function amounts to solving a system of three quadratic equations as suggested in [7]. This calculation can also be viewed as the intersection of three right cones in $\mathbb{E}^3$ as noted in [12]. The details can also be found in [9].

3.1 Contributor edges

We call *clearance disk* a closed disk tangent to two or more disks of $\mathcal{S}$. Two closed disks are tangent if their intersection is a single point. The Voronoi diagram can then be defined as the set of all centers of clearance disks of $\mathcal{S}$ (see [15]). When a clearance disk is tangent to more than two sites we call its center a Voronoi vertex. Given the diagram DVorW($\mathcal{S}$) and a new site k, the idea of the algorithm is to find the vertices of R_k and then update the diagram. Clearly, every vertex of R_k must lie on some edge of DVorW($\mathcal{S}$) as it is the center of a clearance disk tangent to k and to two sites of $\mathcal{S}$. We denote the edges of DVorW($\mathcal{S}$) which contain at least one vertex of R_k by *contributor edges*. One edge e may contribute, at most, two vertices as any vertex lying on e must be the center of a clearance disk tangent to k and to the two sites of $\mathcal{S}$ which generate e and any three given disks may have at most two such points (the circuncenters).

Let e_{left} and e_{right} be the sites of the left and right regions of the oriented edge e. The preceding discussion implies that a circuncenter of k, e_{left} and e_{right} is a vertex of R_k if, and only if, it lies on e. Let e_{orig} and e_{dest} be the vertices at the origin and destination of e. We know that the supporting hyperbola of e is star-shaped with respect to both e_{left} and e_{right}. This implies that a circuncenter a of k, e_{left} and e_{right} (a always lies on the hyperbola) is on e if, and only if, the counterclockwise circular order of e_{orig}, a and e_{dest} with respect to e_{left} is $e_{orig} a e_{dest}$ (see Fig. 5).

At this point we could come up with a brute force algorithm finding all vertices using the function `counterClockWiseOrder` to test every edge of DVorW($\mathcal{S}$). Then we could circular sort the vertices around k (R_k is star-shaped) and update the diagram. Fortunately if we define a more specific concept of *oriented contributor edge* we can find the vertices consecutively in the correct order with a very simple traverse procedure if we are given

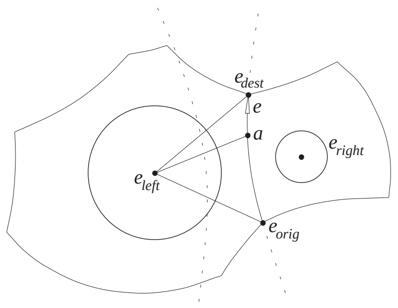

Figure 5: Testing an edge with the orientation predicate

an initial oriented contributor edge. This concept and this procedure are presented in section 3.3.

We note that this characterization through contributor edges cannot be used in $\mathbb{E}^2$, as R_k may have no vertex, that is, its boundary may be not connected to DVorW($\mathcal{S}$) at all.

3.2 The base case

The base case $n = 3$ has three possible configurations shown in Fig. 6. The diagram is always the same: the two circuncenters are connected by three edges separating the three sites in three regions. The following functions construct the base case almost mechanically:

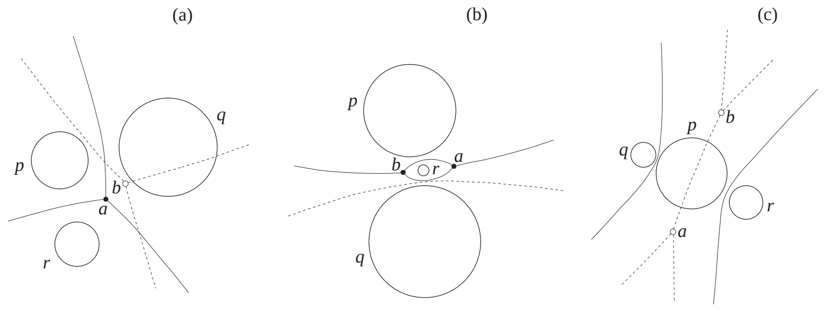

Figure 6: Base cases $n = 3$ for DVorW

Function 4 `createEdge(a, b, p, q)` *creates one edge with origin* `a`*, destination* `b`*, left and right regions* `p` *and* `q` *and returns a pointer to it.*

Function 5 `baseCase(p, q, r)` *constructs* DVorW($\mathcal{S}$) *for* $\mathcal{S} = \{p, q, r\}$ *and returns a pointer to one of the edges of the diagram:*

```
point a, b;
circunCenters(p, q, r, a, b);
if ( counterClockWiseOrder(p, q, r, a) ) swap(a, b);
```

```
createEdge(a, b, p, q);
createEdge(a, b, q, r);
return createEdge(a, b, r, p);
```

The `if` test is due to a subtle problem: as the regions have only two edges, we cannot circular sort the vertices with respect to the sites. Our test to find the vertices of R_k, already sketched in the previous section, that will be used in the insertion of the other sites, relies on the orientation of the edges. As we choose one of the circuncenters for the origin and create edges having left and right regions p and q, q and r, r and p, there is a *correct* circuncenter for the origin. Consider one of the edges e. If we choose the wrong circuncenter, the counterclockwise circular order of the e_{orig}, a generic point x lying on the edge and e_{dest} will not be $e_{orig}xe_{dest}$. It is interesting to note that, as the edges separate the sites and the regions are star-shaped with respect to them, the circular order of the sites with respect to one circuncenter is opposite to the other. The correct origin for the function `createEdge` must be the circuncenter that perceives the counterclockwise circular order prq.

3.3 The traverse procedure

Figure 7(a) introduces the edge functions used in the traverse procedure, which is defined for oriented edges. The names come from the well-known *quad-edge* data structure [6]. For a given oriented edge e: $onext(e)$ is the next edge in counterclockwise direction with the same origin; $rprev(e)$ is the previous edge in counterclockwise direction with the same face; and $sym(e)$ is the opposite oriented edge.

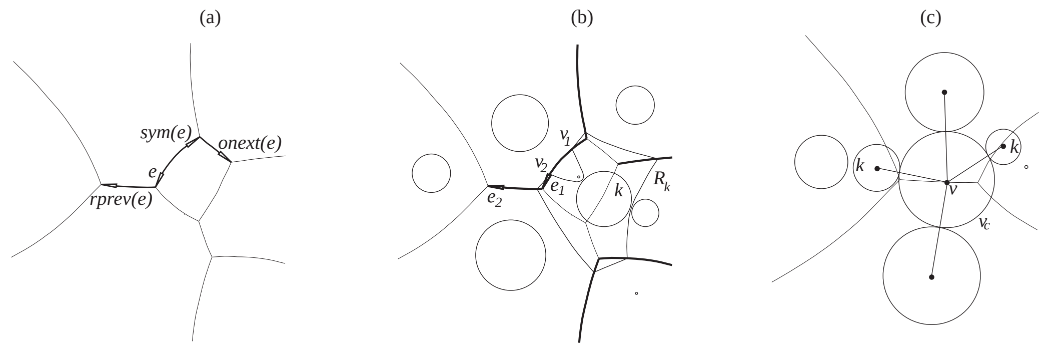

Figure 7: Oriented contributor edges for the traverse procedure

The traverse procedure to be presented needs one first contributor edge. Figure 7(b) shows the insertion of site k in the DVorW($\mathcal{S}$), for $n = 7$. The contributor edges are shown in bold lines. Note that the edges contributing one vertex have one of their endpoints inside R_k an the other outside. The edges contributing two vertices have either both endpoints inside R_k (edge e_1 in the figure), or both outside R_k (this case will be discussed in section 3.3.2). The first contributor edge must be one oriented such that its origin is inside R_k. If this is not true, the procedure will fail to find all vertices. We say, then, that edge e_2 is a contributor, but $sym(e_2)$ is not. If an edge contributes one vertex, clearly, the clearance disk with center in the endpoint contained in R_k must intersect k and the one with center in the endpoint outside R_k must not. We need not test the intersection of k and these clearance disks to decide which edge is oriented from inside R_k to outside. Suppose edge e contributes one vertex v as shown in Fig. 7(c). The boundary of the clearance disk v_c with center in v and tangent to k, e_{left} and e_{right} is divide into two arcs by the tangency points in e_{left} and e_{right}. Site k is tangent to v_c in one of these arcs (the figure shows two possible locations for k). It is easy to see that the clearance disks between v_c and the clearance disk with center in the endpoint not contained in R_k do not intersect k and those between v_c and the one contained in R_k do intersect. We can then use the circular order predicate to decide which is the correct oriented contributor edge. It is the one such that `counterClockWiseOrder(` e_{left}, k, e_{right}, v `) == true`.

For the edges contributing two vertices we also use the circular order predicate to assign one vertex to each oriented edge. An oriented edge contributes the first encountered vertex when going from its origin to its destination. As shown in Fig. 7(b), for e_1 `counterClockWiseOrder(` $e_{1_{orig}}$, v_1, v_2, $e_{1_{left}}$ `) == true`, and for $sym(e_1)$ `counterClockWiseOrder(` $sym(e_1)_{orig}$, v_1, v_2, $sym(e_1)_{left}$ `) == false`. Then e_1 contributes v_1 and $sym(e_1)$ contributes v_2.

Our traverse procedure has the same goal as the one in [5]. The vertex contributed by an edge e is the point where the boundary of R_k crosses e leaving the region of e_{right} and entering the region of e_{left} (in a counterclockwise direction). We need to find the next vertex (which is the point where the boundary leaves the region of e_{left}). The procedure tests the edges of e_{left} in clockwise direction from e:

Function 6 `edgeNotContributor(e)` *returns* `true` *if edge* `e` *do not contributes to* R_k *and* `false` *otherwise:*

```
point a, b;
circunCenters(e_left, e_right, k, a, b);
return ( !( counterClockWiseOrder( e_orig, a, e_dest, e_left) ||
            counterClockWiseOrder( e_orig, b, e_dest, e_left) ) );
```

Function 7 `traverseProcedure`

```
firstEdge = e = findSomeContributorEdge();
do {
   processContributorEdge( e );
   e = onext( e );
   while ( e != firstEdge && edgeNotContributor( e ) )
         e = rprev( e );
} while ( e != firstEdge );
```

The function `findSomeContributorEdge` does not affect the worst-case complexity of the algorithm ($O(n^2)$), as R_k may have $O(n)$ edges. In practice, however, it may be the major factor in the running time. The simple "walking" method suggested in [5] can be

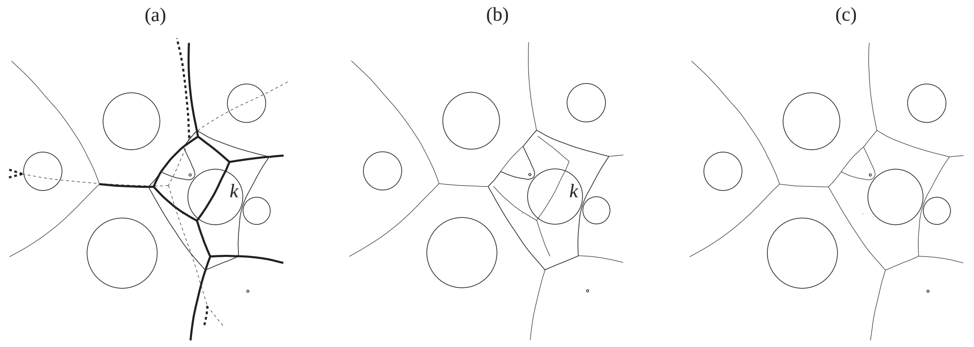

Figure 8: The traverse procedure updating the diagram

used, because the region of the site of $\mathcal{S}$ closest to k must have a contributor edge. More elaborated schemes exist (see [3] for instance).

Figure 8(a) shows, in bold lines, the edges traversed by the procedure for the example of Fig. 7(b). The function `processContributorEdge` is called for every oriented contributor edge and updates the diagram. The update is very simple (see Fig. 8(b) and (c)): the origins of the oriented contributor edges are changed to the corresponding vertices; consecutive vertices are joined by a new edge; and, in the end, some edges of DVorW($\mathcal{S}$) may be deleted.

3.3.1 Correctness

The correctness of the procedure is based on the star-shaped property of the regions of DVorW($\mathcal{S}$). The function `findSomeContributorEdge` returns a contributor edge e. We have already shown that there must be contributor edges in DVorW($\mathcal{S}$). The vertex contributed by the edge e is the point where the boundary of R_k crosses e leaving the region of e_{right} and entering the region $R_{e_{left}}$ of e_{left} (in a counterclockwise direction). The `traverseProcedure` will test all edges of $R_{e_{left}}$ in clockwise direction. Denote these edges by $e_1, e_2, \ldots, e_i$ as in the Fig. 9(a). The boundary of R_k must leave $R_{e_{left}}$, thus, the procedure will eventually find another vertex. Note that the boundary of R_k may enter and leave one region several times, so that, it is not obvious that the procedure always finds the correct vertex where the boundary leaves the region.

The argument, however, is indeed very simple. If the boundary leaves $R_{e_{left}}$ through e_1 then the procedure correctly finds the vertex as it is the first tested edge. But suppose e_1 contributes two vertices v_1 and v_2 (see Fig. 9(a)). We saw that v_1 is assigned to e_1. This is correct because the boundary cannot leave the region through v_2 as it would have to enter it sometime again through v_1 and would not "be able to leave" it.

Generally, if the boundary leaves $R_{e_{left}}$ through e_k then none of the edges $e_1, e_2, ..., e_{k-1}$, which are tested before e_k, can be a contributor edge and the procedure will reach e_k. This is due to the fact that every part of the boundary crossing $R_{e_{left}}$ is a part of the bisector of k and e_{left}. If any of $e_1, e_2, ..., e_{k-1}$ could be a contributor edge, then $R_{e_{left}}$ would become disconnect, which is impossible (see Fig. 9 (b)). This also implies that every traversed edge, except the contributor edges, is entirely contained in R_k.

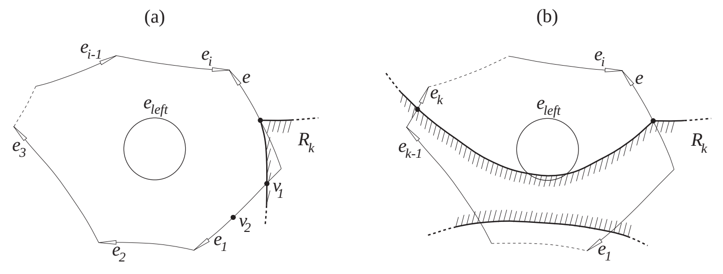

Figure 9: Demonstration of the correctness of the traverse procedure

When we defined oriented contributor edges, we said that their origins must be contained in R_k. We invite the reader the follow the `traverseProcedure`, for the example of Fig. 7(b), if the first contributor edge was $sym(e_2)$ (which has its origin outside R_k). The vertices v_1 and v_2, in that figure, are just not found. In this case, the arguments above can not be applied because the traversed edges are not inside R_k. We could, alternatively, traverse the regions in counterclockwise direction to find the vertices in clockwise direction. The requirement would, then, be that the contributor edges always have their destinations inside R_k.

3.3.2 A special case

When R_k has only two edges (and two vertices), there is only one edge e of DVorW($\mathcal{S}$) contributing the two vertices. This is a special case, as the two oriented contributor edges have their origins outside R_k. However, the traverse procedure correctly finds the two vertices. Figure 10(a) shows the edges traversed by `traverseProcedure` in an example.

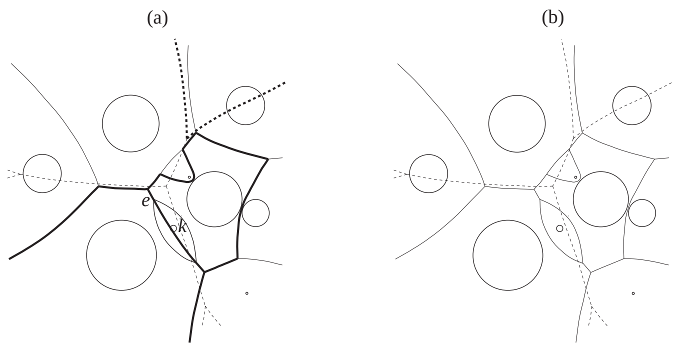

Figure 10: Special case when updating the diagram

The update differs from the general case: choose one oriented contributor edge and change its destination to the corresponding vertex; creates a new edge from the origin to the corresponding vertex of the other oriented contributor edge; and join the vertices by two edges separating k from e_{left} and e_{right}. Figure 10(b) shows the resulting diagram. This case is similar to our base case, but here the update is entirely mechanical as the orientation of the contributor edges are already consistent.

3.4 Comments on the back part of the diagram

The subtlety of this algorithm is the fact that the primitives encapsulate all concepts of $\mathbb{T}^2$. The function `circunCenters` returns two points of $\mathbb{T}^2$ which are used by the function `edgeNotContributor` in simple orientation tests (which are defined for all points in $\mathbb{T}^2$).

The traverse procedure and the update procedures never know whether they are working with finite or infinite edges, or with front or back edges. The procedure for insertion of the site k in the convex-hull of $\mathcal{S}$ is exactly the same as for k not in the convex-hull.

It is clear that any topological data structure able to represent partitions of the sphere can be used to represent the DVorW. The geometry of the edges of the simple DVor can be represented by the coordinates of the endpoints, because they are segments of $\mathbb{T}^2$. Fortunately, the geometric representation of the edges of DVorW, which are arcs of hyperbolas in $\mathbb{T}^2$, also have a simple and uniform representation. This subject has independent interest and appears in [10].

4 Conclusion

We have successfully implemented this algorithm. The details and some images of the visualization of the diagram in both the flat and spherical models of $\mathbb{T}^2$ can be found in [9]. We note that the cost of the exact computation of the test `edgeNotContributor` may be high, as the coordinates of the circuncenters are, in general, irrational numbers.

The algorithm assumes that no site is entirely contained in another. If this restriction cannot be applied, and we still want an on-line algorithm, some modifications are needed:

1. The function `circunCenters` must be prepared to handle three sites that do not have a circuncenter. Then function `edgeNotContributor` must return `true` in these cases.

2. The function `findSomeContributorEdge` may find no contributor edge. This can be due to two cases: site k is contained in some site of $\mathcal{S}$; or site k contains $n-1$ sites of $\mathcal{S}$. The implementation must be able to distinguish these cases.

3. From the preceding item we see that the insertion k may reduce the diagram to the base case $n = 3$ or even to $n = 2$ and $n = 1$ which have no vertices. This also needs special treatment.

This characterization through contributor edges proved very useful for DVorW but cannot be used, for instance, for the Voronoi diagram of line segments, because three line segments may have no circuncenter even in $\mathbb{T}^2$. Nevertheless, it would still be interesting to construct this diagram in $\mathbb{T}^2$ for the uniform representation of the edges.

One interesting generalization if this construction is the higher order diagram. For instance, the order 2 diagram can be defined by $H_{pq,rs} = \{x \in \mathbb{T}^2 | \mathrm{d}_{\mathbb{T}^2}(x,p) + \mathrm{d}_{\mathbb{T}^2}(x,q) \leq_{\mathbb{T}^1} \mathrm{d}_{\mathbb{T}^2}(x,r) + \mathrm{d}_{\mathbb{T}^2}(x,s)\}$. Then, in general, the order k diagram is antipodal to the order $(n-k)$ diagram.

References

[1] J. M. Augenbaum and C. S. Peskin, On the construction of the voronoi mesh on a sphere, J. Comput. Phys., 59 (1985) 177–192.

[2] F. Aurenhammer, Power diagrams: properties, algorithms and applications, SIAM J. Comput. 16 (1) (1987) 78–96.

[3] J-D. Boissonnat, O. Devillers, R. Schott, M. Teillaud and M. Yvinec, Applications of Random Sampling to On-line Algorithms in Computational Geometry. Discrete & Comp. Geometry 8 (1992) 51–71.

[4] S. Fortune, A sweepline algorithm for Voronoi diagrams, Algorithmica 2 (1987) 153–174.

[5] P. J. Green and R. Sibson, Computing Dirichlet tessellations in the plane, Computer Journal 21 (1997) 168–173.

[6] L. Guibas and J. Stolfi, Primitives for the manipulation of general subdivisions and the computations of Voronoi diagrams, ACM Transc. on Graphics 4 (2) (1985) 74–123.

[7] M. Held, On the Computational Geometry of Pocket Machining, Lecture Notes on Computer Science 500 (1991).

[8] D. T. Lee and R. L. Drysdale, Generalization of Voronoi diagrams in the plane, SIAM J. Comput. 10 (1) (1981) 73–86.

[9] G. A. Pinto, Generalizações do Diagrama de Voronoi construídas através de Cônicas no Plano Projetivo Orientado e suas Visualizações, Master's Thesis, Institute of Computing, UNICAMP (1998).

[10] G. A. Pinto and P. J. de Rezende, Representation of conics in the oriented projective plane, Proc. of the X SIBGRAPI, published by IEEE, Campos do Jordão, Brazil (1997) 71–78.

[11] D. Rappaport, A convex hull algorithm for discs and applications, Computational Geometry: Theory and Applications 1 (1992) 171–187.

[12] H. Rosenberger, Order-k Voronoi diagrams of sites with additive weights in the plane, Algorithmica 6 (1991) 490–521.

[13] M. Sharir, Intersection and closest-pair problems for a set of planar discs, SIAM J. Comput. 14 (2) (1985) 448–468.

[14] J. Stolfi, Oriented Projective Geometry: A Framework for Geometric Computations, 1st edition, Academic Press, Inc. (1991).

[15] C. K. Yap, An $O(n \log n)$ Algorithm for the Voronoi Diagram of a Set of Simple Curve Segments. Discrete & Comp. Geometry 2 (1987) 365–393.