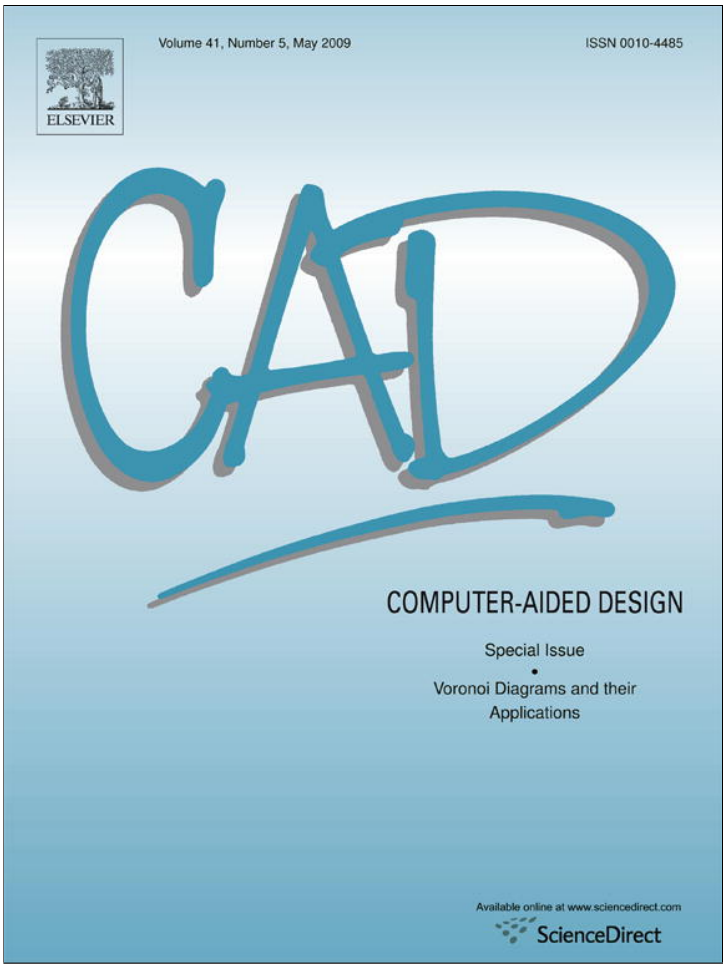
Volume 41, Number 5, May 2009
ISSN 0010-4485
ELSEVIER
CAD
COMPUTER-AIDED DESIGN
Special Issue
Voronoi Diagrams and their Applications
Available online at www.sciencedirect.com
ScienceDirect

Computer-Aided Design 41 (2009) 327–338

Contents lists available at ScienceDirect

Computer-Aided Design

journal homepage: www.elsevier.com/locate/cad

Topology-oriented incremental computation of Voronoi diagrams of circular arcs and straight-line segments[☆]

Martin Held[*], Stefan Huber

Universität Salzburg, FB Computerwissenschaften, A–5020 Salzburg, Austria

A R T I C L E I N F O

Article history:
Received 7 May 2008
Accepted 21 August 2008

Keywords:
Voronoi diagram
Circular arcs
Straight-line segments
Topology-oriented approach
Computation of Voronoi nodes
Algorithm
Implementation
Experiments
Test data

A B S T R A C T

We introduce an algorithm for computing Voronoi diagrams of points, straight-line segments and circular arcs in the two-dimensional Euclidean plane. Based on a randomized incremental insertion, we achieve a Voronoi algorithm that runs in expected time $O(n \log n)$ for a total of n points, segments and arcs, if at most a constant number of segments and arcs is incident upon every point. Our theoretical contribution is a careful extension of the topology-oriented approach by Sugihara and Iri in order to make the incremental insertion applicable to circular arcs.

Our main practical contribution is the extension of Held's Voronoi code Vroni to circular arcs. We discuss implementational issues such as the computation of the Voronoi nodes. As demonstrated by test runs on several thousands of synthetic and real-world data sets, this circular-arc extension of Vroni is reliable and exhibits the average-case time complexity predicted by theory. As a service to the community, all circular-arc data sets (except for proprietary data) have been made public.

To our knowledge, this enhanced version of Vroni constitutes the first implementation that computes Voronoi diagrams of genuine circular arcs on a standard floating-point arithmetic reliably and efficiently, without resorting to some form of approximation or sampling of the circular arcs.

1. Introduction

1.1. Motivation

Voronoi diagrams of straight-line segments have turned out to be useful in a variety of applications with a geometric flavor. Well-known sample applications comprise, for instance, robot motion planning [1,2], shape representation, conversion and reconstruction [3–6], mesh generation [7–9], curve approximation [10], and tool-path generation [11–14].

Unfortunately, so far Voronoi implementations which are able to cope with real-world data have been restricted to points and line segments (and full circles and ellipses). We note that supporting more general curves than plain polygons and, in particular, supporting circular arcs is important for the practical application of a Voronoi algorithm: Offsetting a polygon introduces circular arcs, and it is generally requested that the result of an offsetting operation can again be used as input for a Voronoi algorithm. Also, circular-arc approximations of free-form curves preserve continuity and tend to yield significantly tighter approximations while using less arcs than a straight-line approximation would use. Handling circular arcs as genuine arcs is imperative in the PCB[1] business, since PCB data may be huge. Typically, one cannot afford to replace every arc by tens or even hundreds of straight-line segments as this would cause the memory footprint of a Voronoi-based application to sky-rocket. When the result is output to router machines, using arcs rather than sequences of straight-line segments has mechanical and machine wear-related advantages.

1.2. Prior and related work

Interestingly, virtually all Voronoi algorithms (and implementations) are restricted to points and straight-line segments as input sites. No implementation and no easy-to-implement efficient algorithm are known for computing Voronoi diagrams of circular arcs.

Lee and Drysdale [15] were the first to generalize Voronoi diagrams to straight-line segments and circles as input sites; for

[☆] A preliminary version of this work was presented at the 24th Europ. Workshop on Computational Geometry [Held M, Huber S. Topological considerations for the incremental computation of Voronoi diagrams of circular arcs. In: Proc. 24th Europ. workshop comput. geom. 2008. p. 217–20]. Work supported by Austrian FWF Grant L43-N12.

[*] Corresponding author. Tel.: +43 662 8044 6304; fax: +43 662 8044 172.
E-mail addresses: held@cosy.sbg.ac.at (M. Held), shuber@cosy.sbg.ac.at (S. Huber).
URLs: http://www.cosy.sbg.ac.at/~held (M. Held), http://www.cosy.sbg.ac.at/~shuber (S. Huber).

[1] Printed-circuit board.

0010-4485/$ – see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cad.2008.08.004

n input sites their algorithm runs in $O(n \log^2 n)$ time. Yap [16] developed a divide-and-conquer algorithm for computing the Voronoi diagram of n points, segments and circular arcs in worst-case optimal time $O(n \log n)$. He also introduced the term "cone of influence", but based his definition of the Voronoi cell of a site on ϵ-neighborhoods. However, Yap's algorithm seems tricky to implement; according to our knowledge, it has never been cast into a reliable implementation. The same optimal worst-case bound is achieved by an elegant sweep-line algorithm due to Fortune [17], which, at least in theory, is also applicable to circular arcs. (But, again, we are not aware of an implementation of Fortune's algorithm for circular arcs.)

More recently, Alt and Schwarzkopf [18] studied Voronoi diagrams of so-called "harmless sites" (which include circular arcs). They first select one point out of the relative interior of each curve and then insert the curves in a randomized order, obtaining an expected running time of $O(n \log n)$. However, their paper focuses mostly on establishing a theoretical basis for the definition of Voronoi diagrams of planar curves, while the actual algorithmic details of the insertion of a curve are only sketched. In any case, no implementation of their algorithm is known.

Sugihara and Iri [19] introduced a topology-oriented approach as the key ingredient for the numerically reliable computation of Voronoi diagrams of points on a standard floating-point arithmetic. (See also the more recent survey by Sugihara et al. [20].) Later on, Imai [21] sketched an extension to straight-line segments. Held [22] also followed the avenues of [19]: By using a topology-oriented randomized incremental insertion he obtained an algorithm (and the implementation "Vroni") for the computation of Voronoi diagrams of points and straight-line segments that runs in $O(n \log n)$ expected time.

Several algorithms are known for computing the Voronoi diagram of full circles (rather than circular arcs). Kim et al. [23, 24] discuss the computation of Voronoi diagrams of circles. Based on the Voronoi diagram of the circle centers an edge-flipping algorithm is applied to obtain the actual Voronoi diagram of the circles. While the worst-case time bound of their algorithm is quadratic, Kim et al. claim that in practice it is comparable to the time consumed by the computation of the point-set Voronoi diagram. Recently, Jin et al. [25] presented a sweep-line algorithm that computes the Voronoi diagram of n circles in $O((n+m) \log n)$ time, where m denotes the number of intersection points of the circles. Anton et al. [26] compute eigenvalues of two-by-two matrices to obtain the Delaunay graph of circles, i.e., the dual graph of the Voronoi diagram of circles. Additively weighted Voronoi diagrams – the dual of the Apollonius diagram – were studied by Karavelas and Yvinec [27] and implemented within CGAL. Very recently, Emiris et al. [28] discussed the computation of Voronoi diagrams of (full) ellipses.

1.3. Our contribution

The tremendous practical success of Held's Voronoi package Vroni [22] motivated us to extend its randomized incremental construction of Voronoi diagrams to points, straight-line segments and genuine[2] circular arcs in the two-dimensional Euclidean plane. The input data is assumed to be disjoint (except for common end points of segments and arcs), but need not be arranged within any specific geometric structure (such as one closed polygon). In the sequel, in Section 2, we analyze how the topology-oriented incremental algorithm of Sugihara and Iri [19] can be adapted to handle circular arcs. The extension to circular arcs causes several important issues to surface which have to be handled properly in order to respect the topological conditions and to ensure the overall correctness of the algorithm. We discuss in detail all topological and graph-theoretical extensions of the incremental insertion needed for handling circular arcs. At the end of this section we discuss the run-time complexity of the algorithm and prove that randomization allows it to run in expected time $O(n \log n)$ for n input points, straight-line segments and circular arcs, if at most a constant number of segments and arcs is incident upon every point.

Our new algorithm has been implemented in ANSI C and integrated into Vroni. Thus, it can rely on Vroni's many important features that make Vroni survive real-world data on a standard floating-point arithmetic, such as on-the-fly local cleaning of the data, an automatic selection of a suitable precision threshold ("relaxation of epsilon thresholds"), and a multi-phase recovery mode that eventually leads to "desperate mode", see [22]. We emphasize, though, that the basic scheme presented in Section 2 for incrementally inserting a circular arc into a Voronoi diagram is not bound to the limits of Vroni. We discuss issues related to an actual implementation, including the computation of the Voronoi nodes, in Section 3.

Extensive tests on synthetic and real-world data demonstrate that the circular-arc extension of Vroni is indeed strong enough to compute Voronoi diagrams of hundreds of thousands of circular arcs reliably and efficiently within real-world applications. In particular, in Section 4 we report run-time tests which make it evident that an $O(n \log n)$ complexity can indeed be expected in practice.

This paper is accompanied by several color plates available on the WWW. Point your browser to the WWW home-page [29] of Vroni. Also, virtually all circular-arc test data has been made public in a data repository [30]. (Of course, some proprietary data sets are excluded from the repository even though we included them into our tests.) It is our hope that this data repository[3] will help foster quality experimental work on algorithms that involve circular arcs.

1.4. Basic definitions

For two points $p, q \in \mathbb{R}^2$, let $d(p, q)$ denote the Euclidean distance between p and q. If $Q \subseteq \mathbb{R}^2$ is a set of points then $d(p, Q)$ is defined as $\inf\{d(p, q) : q \in Q\}$. Similarly, if $\mathcal{Q}$ is a finite family of sets of points, then $d(p, \mathcal{Q}) := \min_{Q \in \mathcal{Q}} d(p, Q)$. (Symmetry of the distance function can be established by defining $d(Q, p)$ and $d(\mathcal{Q}, p)$ accordingly.)

For the input to our Voronoi algorithm we will follow the convention established by Kirkpatrick [31]: We call a disjoint system S of subsets of $\mathbb{R}^2$ a *proper* set of input sites if

(1) S consists of points, open[4] straight-line segments and open circular arcs, and
(2) for every segment and arc $s \in S$ the endpoints of s are included in S as well.

For the sake of simplicity we assume that every arc is oriented counter-clockwise (CCW), and that no arc is greater than a semi-circle.[5] Points, open straight-line segments and open circular arcs

[2] Previous versions of Vroni supported circular arcs via a tangent-based polygonal approximation of circular arcs within Vroni and the subsequent recovery of an approximate Voronoi diagram of the original arcs. This scheme is good enough for many practical applications such as tool-path generation for machining, but breaks down once hundreds of thousands of arcs are to be handled (e.g., in a PCB application) since the memory footprint of Vroni becomes unbearable.

[3] Contribution of data is most welcome!

[4] Straight-line segments and circular arcs without their end points; i.e., open in the relative topology of the supporting line resp. circle.

[5] We split arcs greater than semi-circles. It is beneficial both for establishing the theory as well as for an actual implementation to know that for every circular arc there exists at least one line through its center such that the arc is fully contained in (the closure of) one of the half-planes induced by the line. E.g., without this restriction the proof of Lemma 1 is slightly more complicated, and ensuring the tree structure of the Voronoi edges removed (Section 2.2) becomes much more tedious.

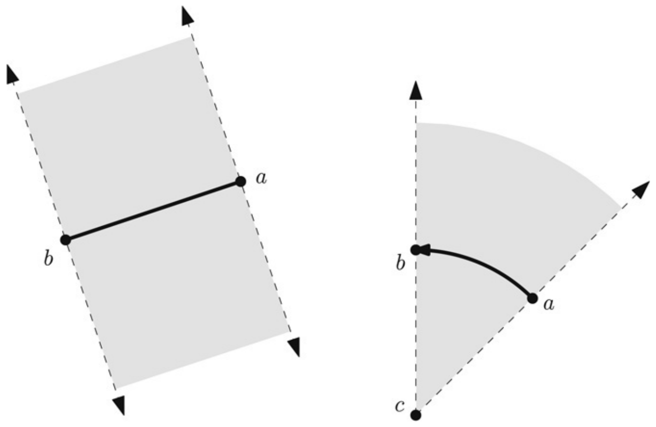

Fig. 1. Cones of influence of a line segment and a circular arc.

are called *sites*. For the rest of this paper we will always assume that we operate on proper set of input sites.

For a vector v and a point p, let $H(p, v)$ be the half-plane $\{q \in \mathbb{R}^2 : q \cdot v \geq p \cdot v\}$. The result of the rotation of v around the origin by 90° is denoted by v^{CCW}, while v^{CW} stands for a rotation by $-90°$. Following Held's terminology [22], the *cone of influence* $\mathcal{CI}(s)$ of a site s is defined as

- $\mathcal{CI}(s) := \mathbb{R}^2$ if s is a point,
- $\mathcal{CI}(s) := H(a, b - a) \cap H(b, a - b)$ if s is a line segment with points a and b, and
- $\mathcal{CI}(s) := H(c, (a - c)^{\mathrm{CCW}}) \cap H(c, (b - c)^{\mathrm{CW}})$ if s is an arc centered at c with start point a and end point b.

In Fig. 1, the cones of influence of a line segment and a circular arc are shown in light grey.

We define the *Voronoi cell* of a site $s \in S$ as

$$\mathcal{VC}(s, S) := \mathrm{cl}\,\{q \in \mathrm{int}\,\mathcal{CI}(s) : d(q, s) \leq d(q, S)\},$$

where $\mathrm{int}\,Q$ denotes the (topological) interior of the set Q and $\mathrm{cl}\,Q$ stands for the closure of Q. The consideration of the interior and exterior in the definition of $\mathcal{VC}(s, S)$ is a technical twist[6] in order to avoid "one-dimensional" portions of a Voronoi cell if two circular arcs meet tangentially in a common end point; all points within the intersection of the boundaries of their cones of influence are equidistant to both arcs! (See also the technicalities needed in Section 2.3.1 in order to choose proper seed nodes.)

As usual, the *Voronoi polygon* $\mathcal{VP}(s, S)$ is given by the boundary of $\mathcal{VC}(s, S)$, and the *Voronoi diagram* $\mathcal{VD}(S)$ of S is defined as

$$\mathcal{VD}(S) := \bigcup_{s \in S} \mathcal{VP}(s, S).$$

For two sites $s_1, s_2 \in S$, the *bisector* $b(s_1, s_2)$ is defined as the loci of points out of $\mathcal{CI}(s_1) \cap \mathcal{CI}(s_2)$ which are equidistant to s_1 and s_2. A *Voronoi edge* between s_1, s_2 is a connected portion of $\mathcal{VP}(s_1, S) \cap \mathcal{VP}(s_2, S)$; it lies on $b(s_1, s_2)$. *Voronoi nodes* are points where three or more Voronoi edges meet. (Thus, they are equidistant to the corresponding sites defining the edges.) The *clearance disk* $\mathcal{CD}(p, S)$ of a point $p \in \mathbb{R}^2$ is the closed disk centered at p with *clearance* radius $r := d(p, S)$.

We conclude the introduction of the terminology used with three facts that are easy to prove.

Lemma 1. *Let S be a proper set of input sites and let $s \in S$. Then $\mathcal{VC}(s, S)$ is generalized star-shaped with nucleus s.*

Corollary 2. *All Voronoi cells of proper input sites are (simply) connected.*

Lemma 3. *The bisectors between pairs of proper sites are given by portions of conics. (That is, all bisectors are elliptic, hyperbolic, or parabolic arcs, including circular arcs and straight-line segments in the limit.)*

2. Incremental construction

2.1. Survey of the algorithm

We employ the incremental topology-oriented approach by Sugihara and Iri [19] in order to enable a reliable implementation of our new algorithm on a standard floating-point arithmetic. Starting with an initially empty set of processed sites, the final Voronoi diagram is obtained by incrementally adding one new site at a time to the set of processed sites and updating the Voronoi diagram accordingly. We first compute the Voronoi diagram of all points, then proceed with inserting the segments, and finally[7] insert the arcs. The insertion of the points, segments and arcs is done in a randomized order (within each group of sites).

Four dummy points (outside of the bounding box of the input sites) are added to the input sites in order to define the initial Voronoi diagram. These four points are given by the corners of a scaled copy of the bounding box of the input sites. (The scale factor may be selected by the user.) Thus, the initial Voronoi diagram consists of four rays that originate from the centroid of these four dummy points.

Beside defining the initial Voronoi diagram, the addition of those four dummy points allows us to get rid of special cases that would otherwise be required for handling the insertion of the first sites and for handling unbounded bisectors. In particular, all original sites inserted afterwards are guaranteed to have bounded Voronoi cells, since only the four dummy points span the convex hull of the input sites. Of course, the disadvantage is that the Voronoi diagram obtained is not universally correct within the entire Euclidean plane, and operations like outwards offsetting can only be performed within an offset distance that depends on the scale factor used.

Every update of the Voronoi diagram due to an insertion of a site is performed by deleting old Voronoi nodes and edges (and creating new Voronoi nodes and edges) according to topological properties that the Voronoi graph has to fulfill. (One important topological property to fulfill is that the set of Voronoi nodes and edges which are deleted during one update forms a tree.) Numerical predicates are used to select those nodes that are the best candidates for deletion.

Provided that every incremental update is performed successfully, we are bound to obtain a final Voronoi diagram which also conforms to the set of topological properties. The obvious crux is to guarantee that every incremental update is performed successfully. As in the case of segment Voronoi diagrams, care has to be taken in order to prevent the complete removal of Voronoi edges that ought to be partly preserved while deleting Voronoi nodes during an incremental update. However, the insertion of circular arcs causes problems to surface that do not occur for segment Voronoi diagrams. Amendments are also necessary in order to handle end points that are shared by two or more sites if one of those sites is an arc.

6 Yap [16] resorts to ϵ-neighborhoods. Alt and Schwarzkopf [18] consider cells that are partially open; as a consequence, the intersection between adjacent cells may be empty.

7 This specific order allows a simplification of the implementation. It is by no means mandatory, and any insertion sequence would be possible, provided that the two end points of a segment or arc are inserted prior to the actual insertion of that segment/arc. Of course, in our implementation the Voronoi diagram of the points and segments is constructed by directly applying the original Vroni code [29,22].

The incremental updates of the Voronoi diagram make it necessary to employ a data structure for storing the Voronoi diagram which supports an efficient traversal of the diagram. We use (a variant of) a winged-edge data structure; see, for instance, [12]. (However, virtually any other "standard" data structure used in computational geometry for storing a planar graph would be equally fine.) Note that this allows us to obtain all edges of a Voronoi cell or all edges that are incident upon a Voronoi node in output-sensitive time. As in VRONI [22], we deal only with degree-three Voronoi nodes. Any higher-degree node corresponds to a bunch of nodes that coincide and that are interconnected by zero-length Voronoi edges. Thus, high-degree nodes do not require any special treatment.

Let S be a set of sites and consider an open arc $s \notin S$ that is to be inserted into $\mathcal{VD}(S)$. Let $S^+ := S \cup \{s\}$. We assume that S^+ is a proper input set and that $\mathcal{VD}(S)$ is known. Recall that S^+ being a proper set implies that the end points of s are present in $\mathcal{VD}(S)$.

We require the Voronoi diagram under construction to fulfill the following topological conditions[8]:

- Every site has its own Voronoi cell.
- Every Voronoi cell is connected.
- The Voronoi cell of an open line segment or open circular arc is adjacent to the Voronoi cells of its end points.

In order to insert the arc s into $\mathcal{VD}(S)$ we proceed as follows (see Fig. 2):

(1) We mark a Voronoi node (*seed node*) of $\mathcal{VD}(S)$, whose clearance disk is intersected by s, by scanning the Voronoi polygon of each end point of s.
(2) We recursively scan $\mathcal{VD}(S)$ and mark all further nodes of $\mathcal{VD}(S)$ whose clearance disks are intersected by s.
(3) Let us call *marked edges* those edges of $\mathcal{VD}(S)$, where both end nodes are marked, and *semi-marked edges* those edges, where exactly one node is marked. Then $\mathcal{VD}(S)$ is adapted as follows: We remove marked edges and marked nodes. On each semi-marked edge e we compute a new Voronoi node (replacing the removed node), defined by the site s and the two defining sites of e. The Voronoi polygon of s is generated by successively connecting new Voronoi nodes by means of Voronoi edges.

In order to prevent destroying the topological conditions we never mark Voronoi nodes which coincide with input points [22]; neither do we select them as seed nodes. While Step (2) in the above list boils down to a simple graph traversal that is identical to the case of segment Voronoi diagrams [22], Steps (1) and (3) require more intensive attention. In particular, to ensure the correctness of the algorithm, we need to explore the following two issues:

- There always has to exist an appropriate seed node.
- Those and only those edges of $\mathcal{VD}(S)$ are marked (and deleted) which are completely in the future Voronoi cell $\mathcal{VC}(s, S^+)$.

2.2. Tree structure of edges removed

For technical reasons we start with discussing the second issue. Obviously, the marked nodes cannot survive in $\mathcal{VD}(S^+)$ since those nodes are clearly closer to s than to any other site. But since we remove each marked edge of $\mathcal{VD}(S)$ between marked nodes, we have to guarantee that the whole edge between two marked nodes is completely in the future Voronoi cell $\mathcal{VC}(s, S^+)$ as well.

8 In other words, these topological conditions form loop invariants in our algorithm.

Lemma 4. *If $\mathcal{VC}(s, S^+)$ contains two nodes $v_1, v_2 \in \mathcal{VD}(S)$, then $\mathcal{VC}(s, S^+)$ contains a set of edges of $\mathcal{VD}(S)$ that form a path connecting v_1 and v_2.*

Proof. Corollary 2 tells us that there exists a curve C in $\mathcal{VC}(s, S^+)$ connecting the nodes v_1 and v_2. We observe that the edges of $\mathcal{VD}(S)$ "cut off" from the Voronoi cells of $\mathcal{VD}(S)$ by C form the path sought. □

Lemma 5. *Let C be a cycle in the graph arising from $\mathcal{VD}(S)$, such that C forms a Jordan curve. We denote the area enclosed by C with $A \subseteq \mathbb{R}^2$. Then there exists a site $s \in S$ such that $s \subseteq A$.*

Proof. The area A contains the interior of a Voronoi cell of $\mathcal{VD}(S)$ either completely or not at all. Furthermore, there exists at least one $s' \in S$ such that A contains at least one point of int $\mathcal{VC}(s', S)$. It follows that $s' \subseteq \mathcal{VC}(s', S) \subseteq A$. □

Theorem 6. *Let T denote the graph comprising the edges of $\mathcal{VD}(S)$ which completely lie in $\mathcal{VC}(s, S^+)$ but which do not intersect with $(\mathrm{cl}\, s)\backslash s$. Then T forms a tree.*

Proof. Lemma 4 implies that T is connected. We see that T is cycle-free as follows. Suppose T contains a cycle C. Lemma 5 states that there exists a site $s' \in S$ such that $s' \subseteq \mathcal{VC}(s, S^+)$. This is not a contradiction if and only if s' is an end point of s and C contains the edge separating s' and s. But then we conclude that C contains a point of $(\mathrm{cl}\, s)\backslash s$, where $\mathrm{cl}\, s$ denotes the (topological) closure of s, which yields a contradiction. □

Corollary 7. *Let T be the graph comprising all marked edges of $\mathcal{VD}(S)$. Then T contains a cycle if and only if T contains an edge e with $e \not\subseteq \mathcal{VC}(s, S^+)$.*

Corollary 7 forms the basic building block in the arguments that follow. Obviously, if an edge e should be removed (resp. at least portions of it that include the end nodes) then it is contained in T. But there are three questions to be answered: (i) Can it happen that T contains an edge e which should be partly preserved; (ii) if yes, how can we detect these cases; and (iii) how can we fix this problem?

The first question is easily answered with "yes": Held [22] already presented arrangements of straight-line segments and points for which this problem occurs. These examples can be easily extended to circular arcs. The second question can be answered using Corollary 7: If we marked an edge which should be partly preserved (and hence, $e \subseteq \mathcal{VC}(s, S^+)$ does not hold), then T contains a cycle. So we can search for cycles in T and find edges e which should be partly preserved. Regarding the third question, we can proceed as follows: Since e should be partly preserved there is at least one point $p \in e$, such that $p \notin \mathcal{VC}(s, S^+)$ holds. If we split e at p by placing a (dummy) degree-two node then we break up the cycle of T. (Note that such a degree-two node is not marked.)

Hence, if we can always find a *proper split point p* on *pathological* edges – which need to be preserved partly although both of its nodes are (rightfully) marked for deletion – then we obtain a tree T of marked edges. Moreover, every edge e is marked if and only if e is completely contained in the future Voronoi cell $\mathcal{VC}(s, S^+)$ and therefore has to be completely removed. In other words, the problem has been broken down to finding proper split points on pathological edges. We distinguish two cases: (i) there is an marked edge e reaching outside of $\mathcal{CI}(s)$ or (ii) there is a marked edge $e \subseteq \mathcal{CI}(s)$ and a point $p \in e$, with $p \notin \mathcal{VC}(s, S^+)$. But before attacking these two cases we first review the technique of apex splitting.

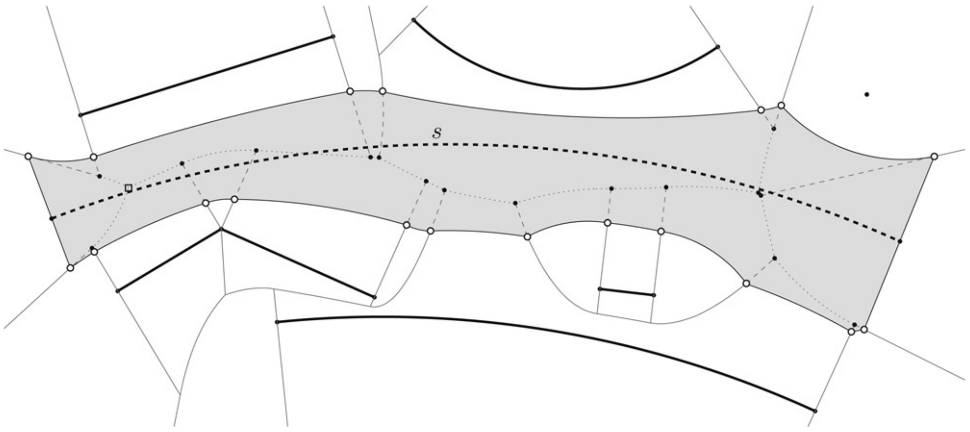

Fig. 2. The dashed arc s is inserted into the Voronoi diagram $\mathcal{VD}(S)$ of the bold sites. In Step (1), the seed node (box) is selected. In Step (2), further nodes (disks) are marked. In Step (3), the new Voronoi cell of s (shaded) is constructed. The marked edges are shown as dotted curves, semi-marked edges are dashed and new Voronoi nodes are depicted by small circles.

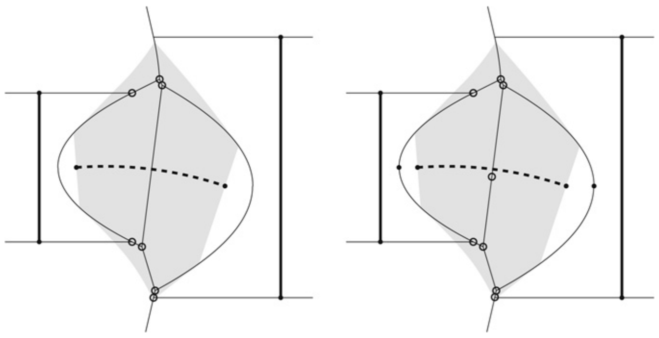

Fig. 3. Left: The insertion of the dashed circular arc causes two Voronoi edges to be marked although they should be partly preserved, resulting in cycles of marked edges. The area shaded in light grey corresponds to the Voronoi cell of the dashed circular arc. Nodes that are marked for deletion are depicted by circles. Right: By splitting the edges at their apices (by inserting the degree-two nodes depicted by solid disks) we break up the cycles.

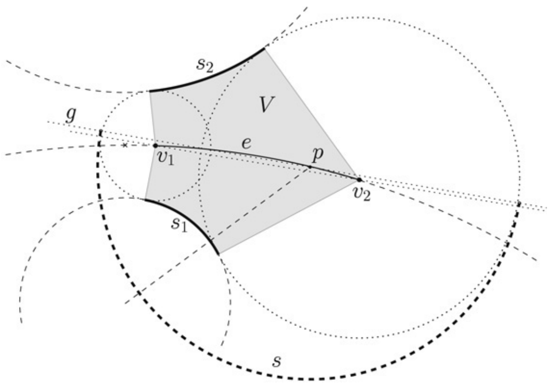

Fig. 4. The insertion of the arc s causes both nodes of e to be marked although some portion of e lies outside of $CI(s)$.

2.2.1. *Apex splitting*

Consider a conic Voronoi edge and assume that the apex of its supporting conic lies on the edge. As suggested in [22], we insert a dummy degree-two node in order to split a conic Voronoi edge at its apex. (See Fig. 3.) Hence, for the sequel we may assume that no Voronoi edge has the apex of its supporting conic in its (relative) interior.

Besides helping to avoid cycles among the edges marked for deletion, the insertion of apex nodes also has a beneficial side-effect: It guarantees that the clearance increases (resp. decreases) monotonically as one moves along a Voronoi edge from one node to the other node. This property helps to identify bottlenecks among the input sites and is of practical importance when computing offset curves based on the Voronoi diagram [32].

2.2.2. *Splitting edges outside of cone of influence*

If all Voronoi edges are split at their apices then, in the case of line segments, one can prove that every Voronoi edge of a segment Voronoi diagram either does not have both nodes marked for deletion or is completely contained in the current cone of influence of the new segment to be inserted. Thus, for segment Voronoi diagrams it is not possible that both nodes of a Voronoi edge are marked while some portion of it lies outside of the current cone of influence and, therefore, needs to be preserved.

Unfortunately, once we deal with circular arcs the insertion of apex nodes is of limited help for avoiding cycles: Even if all Voronoi edges are split at the apices of their supporting conics, it still is possible that both nodes of an edge e are marked for deletion while e is not completely contained in $CI(s)$, as illustrated in Fig. 4.

This sample arrangement of input sites is constructed as follows: We consider the hyperbolic bisector between two circles that are disjoint. We choose two arcs s_1, s_2 on these circles and the corresponding Voronoi edge e on their bisector such that e does not contain the apex in its interior and such that $\mathcal{VC}(s_1, S)$ contains all secants of e. We denote by v_1 (resp. v_2) that node of e which has smaller (resp. larger) clearance. We want to insert an arc s such that the clearance disks of v_1 and v_2 are intersected by s, even though e is not completely contained in $\mathcal{CI}(s)$.

Let V be the union of all line segments resulting from the normal projection of points of e onto s_1 and s_2. (Since every Voronoi cell is generalized star-shaped with respect to its defining site, V is a subset of $\mathcal{VC}(s_1, S) \cup \mathcal{VC}(s_2, S)$.) Recall that the end points of s are already part of $\mathcal{VD}(S)$. Thus, they cannot lie in V or in the clearance disks of v_1 and v_2. Now consider the supporting line g of a secant of e: We get that $g' := g \backslash (V \cup \mathcal{CD}(v_1) \cup \mathcal{CD}(v_2))$ consists of two parts because the set $g \cap \text{int}\,(V \cup \mathcal{CD}(v_1) \cup \mathcal{CD}(v_2))$ is connected. Within each part of g' we choose a point close enough to the neighboring clearance disk such that the line through this point orthogonal to g intersects this clearance disk. All that remains to do is to use these two points as the end points of a semi-circle (which has to lie on that side of g which contains s_1). We note that the three arcs s_1, s_2, s and their end points form a proper input set. By construction, s intersects the clearance disks of v_1 and v_2. Also by construction, some portion of e does not lie on the side of v_1, v_2 relative to g. Thus, this portion of e is outside of $CI(s)$!

Obviously, we can perturb the end points of s slightly such that s is strictly smaller than a semi-circle. This construction scheme

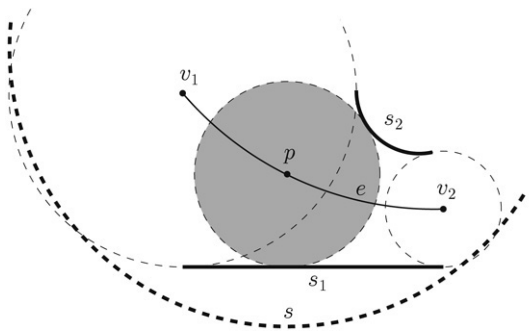

Fig. 5. Both nodes v_1, v_2 of the edge e are marked even though some portion of e has to be preserved after inserting s.

can be adapted to nearly every combination of input sites s_1, s_2 as long as e does not take on the form of a straight-line[9] segment. (If e is a line segment then the marking of both nodes of e implies $e \subseteq \mathcal{CI}(s)$.) Furthermore, one can prove that the arc s cannot be replaced by a segment such that the same scenario occurs. (Hence, this problem does not arise for segment Voronoi diagrams.)

We resolve this problem by inserting a dummy degree-two node p that breaks up the cycle: We always find a proper point p on e by considering the normal projection of the center of s onto s_1, and by intersecting the resulting projection line with e. The resulting dummy node p need not lie outside of $\mathcal{CI}(s)$ but one can prove that it will never lie in the future Voronoi cell $\mathcal{VC}(s, S^+)$. This construction works no matter whether s_1 is a point, segment or arc.

2.2.3. Splitting edges inside of cone of influence

Now suppose that e is a Voronoi edge of $\mathcal{VD}(S)$ which does not lie completely within $\mathcal{VC}(s, S^+)$ although both of its nodes are marked and although it is contained completely in $\mathcal{CI}(s)$. For segment Voronoi diagrams this can only happen if the site enclosed by the cycle is a point, and one can again use a normal projection of that point onto e to split e and break up the cycle [22].

When dealing with arcs this once again is no longer true. As Fig. 5 illustrates, the site s_1 enclosed by the cycle that contains the Voronoi edge e may be a segment or arc. Both nodes of e are marked even though a point p exists on e whose clearance disk is not violated by s.

Fortunately, the same strategy that we used in Section 2.2.2 to break up a cycle is applicable once more: We consider the normal projection of the center of s onto s_1 and intersect the resulting projection line with e in order to obtain a split point p.

2.2.4. Procedure for breaking up cycles

Summarizing, we get a uniform strategy for breaking up cycles and do not need to distinguish different cases in an actual implementation of the algorithm. Recall that such a pathological edge e occurs if and only if a corresponding cycle exists. Thus, we do not need to explicitly (and possibly even repeatedly) check whether a cycle exists within the structure T of edges marked for deletion. Rather, as we scan $\mathcal{VD}(S)$ and mark both nodes of an edge e, we also check whether a split point p computed as outlined above happens to lie on e. If p does indeed lie on e then we insert p as a dummy degree-two node on e and break up a cycle. (Of course, we do not mark p for deletion!) In order to avoid an increase in the number of Voronoi edges, we remove all dummy nodes created during the insertion of s once $\mathcal{VD}(S^+)$ has been obtained.

[9] Thus, s_1 and s_2 may not both be line segments or points, and if s_1, s_2 both are circular arcs then their radii have to differ.

2.3. Selecting a seed node

We will now explain how to determine a proper seed node. A *seed node* is a node of $\mathcal{VD}(S)$ which lies in $\mathcal{VC}(s, S^+)$. Thus, its clearance disk is intersected (or at least touched) by s and it needs to be removed. To fulfill the topological conditions, we do not mark nodes as seed nodes if they coincide with input points. The following lemma tells us that we can restrict the search for a seed node to $\mathcal{VP}(p, S) \cap \mathcal{CI}(s)$, where p is an end point of s. (Recall that no edge $e \in \mathcal{VD}(s)$ contains the apex in its relative interior.)

Lemma 8. *Let $p \in S$ be an end point of s. Then there exists a node $v \in \mathcal{VP}(p, S)$ such that $v \in \mathcal{CI}(s)$. Hence, $d(v, s) \le d(v, p)$.*

Proof. Proof by contradiction: Suppose that no node of $\mathcal{VP}(p, S)$ is contained in $\mathcal{CI}(s)$. Then there exists an edge e of $\mathcal{VP}(p, S)$ such that the end nodes of e are outside of $\mathcal{CI}(s)$ but $e \cap \mathcal{CI}(s) \neq \emptyset$. Therefore, e has to be a non-degenerate parabolic, hyperbolic or elliptic edge with p as its focal point. Let H be an arbitrary half-plane through p such that the intersection with the supporting conic results in a continuous bounded curve.[10] Note that this curve contains the apex as well. Hence, we conclude that e contains the apex. We get a contradiction since no Voronoi edge contains the apex in its interior due to the apex splitting. □

We emphasize that this lemma per se does not imply that a proper seed node is guaranteed always to exist, since we do not mark nodes coinciding with input points! Again, introducing arcs as input sites requires a more extensive analysis. In the following we distinguish two cases:

- Suppose that there exists a node $v \in \mathcal{VP}(p, S)$ which is even in the interior of $\mathcal{CI}(s)$. One can show that all further nodes in $v \in \mathcal{VP}(p, S) \cap \operatorname{int} \mathcal{CI}(s)$ will be marked in the recursive scan. Due to Lemma 4 we know that these (marked) nodes are connected by marked edges.

  In other words, if there exists a node $v \in \mathcal{VP}(p, S) \cap \operatorname{int} \mathcal{CI}(s)$ then we can mark any node with this property as seed node, since the other candidates get marked in the recursive scan anyhow. In an attempt to improve the reliability of the algorithm on a standard floating-point arithmetic, we select the node whose clearance disk is violated the most: We pick the node v such that $d(v, S) - d(v, s)$ is minimized.
- In the other case, we conclude that all nodes of $\mathcal{VP}(p, S) \cap \mathcal{CI}(s)$ have to lie on $\operatorname{bd} \mathcal{CI}(s)$. This can only be the case when s meets other sites of S in a common end point. We can distinguish two sub-cases: (i) the arc s meets exactly one segment or arc $s' \in S$ tangentially in a common end point, or (ii) the arc s meets several sites $s_1, s_2, \ldots s_k \in S$ in a common end point. These two sub-cases are discussed in the sequel.

In our implementation, a second seed node is selected within the Voronoi polygon of the other end point of s. This allows us to run a sanity check during the computation: We verify that the second seed node was indeed marked by the recursive scan. (Recall Lemma 4.)

2.3.1. Selecting a seed node in the presence of tangential sites

Suppose that s meets exactly one other site $s' \in S$ tangentially in the common end point p. (Since we do not allow sites to overlap, at least one of s, s' has to be a circular arc, and this constellation cannot occur for segment Voronoi diagrams.) Let e_1, e_2 be the two Voronoi edges that emanate from p, see Fig. 6. We note that e_1 and e_2 lie on the same supporting line g through p.

[10] The half-plane H locally models the cone of influence, while the conic curve models $e \cap \mathcal{CI}(s)$.

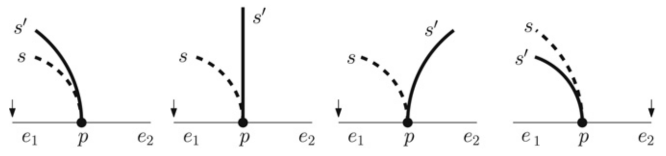

Fig. 6. Selecting a seed node if sites meet tangentially.

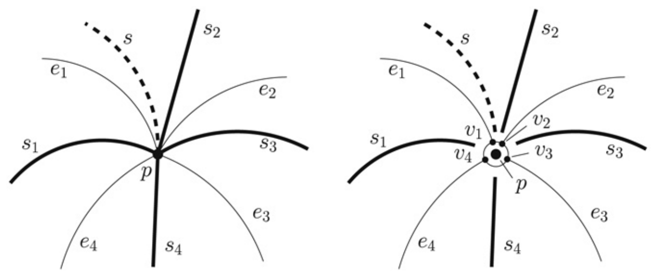

Fig. 7. Selecting a seed node when multiple sites meet in a common end point. Left: Geometric view. Right: Topological view.

The start node shared by e_1 and e_2 is excluded from further consideration because we do not select as seed node a node that coincides with an input point. Since $\mathcal{VP}(p,S)\cap\mathcal{CI}(s)\subset g$, the two other nodes on e_1 and e_2 are the only nodes of $\mathcal{VP}(p,S)$ that could be selected as seed node. However, although possibly both nodes have clearance radii that are identical to their distances from s and s' (and p), only one of these two nodes is admissible as seed node. Suppose that the center of s is on the side of e_1 relative to g and p. We base our decision on the relative order of the sites incident upon p:

- Case: s' is an arc.
  If the center of s' is on the side of e_2 then the node on e_1 is admissible as seed node. If the center of s' is on the side of e_1 and the radius of s' is greater than the radius of s, then the node on e_1 is admissible; otherwise, the node on e_2 is admissible.
- Case: s' is a segment.
  The node on e_1 is admissible.

In Fig. 6, the little arrows point to the edge on which we select the seed node. A symmetric case analysis holds if the center of s is on the side of e_2. If, however, more sites are incident upon p then we need to handle spikes; see below.

2.3.2. Selecting a seed node in the presence of spikes

Suppose that several segments or arcs meet in a common end point p. Likely, this means that the Voronoi cell $\mathcal{VC}(p,S)$ is collapsed to a single point and has zero area, as shown in Fig. 7. (The cell $\mathcal{VC}(p,S)$ need not have zero-area, though.) We scan all nodes of the Voronoi polygon $\mathcal{VP}(p,S)$. If $\mathcal{VP}(p,S)$ has a node v with a clearance greater than zero then v does not coincide with p. In this case we proceed as normal: If v lies on $\mathrm{bd}\,\mathcal{CI}(s)$ then we evaluate $d(v,S)-d(v,s)$. If some sites meet tangentially at p we also have to check whether v is admissible; see above. Again, we select that (admissible) node as seed node whose clearance disk is violated the most.

Otherwise, if no such (admissible) node exists, then we scan the Voronoi edges that are incident upon the nodes which coincide with p. (In Fig. 7, these nodes are numbered $v_1,\ldots,v_4$.) For such a node v_i we consider the Voronoi edge e_i incident upon v_i whose second node does not belong to $\mathcal{VP}(p,S)$. (If no such edge is incident upon v_i then we originate a recursive search in $\mathcal{VD}(S)$, starting at v_i.) For every such edge e_i it is tested whether s intersects the clearance disk of its second node. (One can prove that such a suitable node always exists.)

We emphasize that nodes which coincide with an input point are never deleted during an incremental update. Thus, it is guaranteed that every point site will have a Voronoi cell associated with it in the final Voronoi diagram; it may have zero area, though.

2.4. Complexity

We now analyze the complexity of the insertion of an arc s into the Voronoi diagram $\mathcal{VD}(S)$, with $n := |S^+|$. Since $\mathcal{VD}(S)$ is a planar graph, standard arguments (based on Euler's formula) yield that $\mathcal{VD}(S)$ has $O(n)$ Voronoi edges and nodes. Note that splitting Voronoi edges at their apices at most doubles the number of Voronoi edges and nodes, and, thus, does not change the asymptotic complexity.

Hence, it is obvious that determining a seed node, recursively marking nodes, breaking up cycles, and recursively updating semi-marked edges can be done in $O(n)$ time. We conclude that obtaining $\mathcal{VD}(S^+)$ from $\mathcal{VD}(S)$ takes at most $O(n)$ time. This yields a trivial $O(n^2)$ worst-case bound on the complexity of computing the Voronoi diagram $\mathcal{VD}(S^+)$ incrementally via a sequence of $O(n)$ site insertions. In the sequel we focus on the expected-time complexity.

Lemma 9. *Let N, E and L denote the numbers of marked nodes, marked edges, and semi-marked edges of $\mathcal{VD}(S)$, as caused by the insertion of the arc s. Then, when disregarding all degree-two nodes, the identities*

$$N = L - 2 \quad \textit{and} \quad E = L - 3$$

hold.

Proof. We start with observing that $\mathcal{VD}(S)$ contains only nodes of degree three (and nodes of degree two which can be ignored for the complexity considerations, as noted above). Let T denote the graph consisting of the marked nodes and marked edges. Since T forms a tree, we obtain

$$E = N - 1.$$

We now restrict the L semi-marked edges to $\mathcal{VC}(s,S^+)$ and interpret them as L leaf-edges appended to the tree T, thus obtaining an extended tree T'. We conclude that the number B of edges of $\mathcal{VP}(s,S^+)$ equals L, since two consecutive edges of $\mathcal{VP}(s,S^+)$ are incident to one of the L leaf-edges in T'. If we add the edges of $\mathcal{VP}(s,S^+)$ to T' then we get a planar graph, where every node is of degree three. This yields

$$2(E + L + B) = 3(N + L),$$

that is, $2E + L = 3N$. Combining this result with $E = N - 1$ proves the identities claimed. □

Corollary 10. *The number of nodes (resp. edges) marked during the insertion of s into $\mathcal{VD}(S)$ is linear in the complexity of the new Voronoi polygon $\mathcal{VP}(s,S^+)$.*

Recall that we use randomized insertion. The last corollary is the tool which enables us to apply backwards analysis [33] to prove an $O(n\log n)$ expected-time complexity. We denote by $C(o,O)$ the number of nodes of $\mathcal{VP}(o,O)$, for a proper set O of sites and $o\in O$. Obviously, the complexity of determining a seed node in $\mathcal{VP}(p,S)$ is linear in $C(p,S)$. By Corollary 10, the complexity of recursively marking nodes, breaking up cycles, and recursively updating semi-marked edges is linear in $C(s,S^+)$.

Theorem 11. *If at most a constant number of segments and arcs are incident upon a common end point then the randomized incremental computation of the Voronoi diagram of a proper set of n points, straight-line segments and circular arcs runs in expected time $O(n\log n)$.*

Proof. Let $S := \{p_1, \ldots, p_{n_1}, s_1, \ldots s_{n_2}, a_1, \ldots, a_{n_3}\}$ be a proper set of input sites, with n_1 points, n_2 segments and n_3 arcs $a_1, \ldots, a_k, \ldots, a_{n_3}$. Furthermore, let $S_m := S \backslash \{a_{m+1}, \ldots, a_{n_3}\}$, and $n := n_1 + n_2 + n_3$. Since several $O(n \log n)$ randomized incremental algorithms – e.g., [34,22,35] – are known for computing the Voronoi diagram of $O(n)$ points and straight-line segments, we assume that $\mathcal{VD}(S_{m-1})$ is available and focus on the insertion of the arc a_m into the Voronoi diagram $\mathcal{VD}(S_{m-1})$. As we apply randomized insertion we investigate the expectation of the complexity of inserting an arc among all possible insertion sequences. Let X_m denote the random variable associated with the complexity of inserting a_m into $\mathcal{VD}(S_{m-1})$, after randomly permuting $\{a_1, \ldots, a_m\}$. For $1 \le k \le m$, let p_1^k and p_2^k denote the end points of a_k.

The cost of determining a seed node is linear in $C(p_1^m, S_{m-1}) + C(p_2^m, C_{m-1})$, and we obtain

$$E(X_m) = \frac{1}{m} \sum_{k=1}^{m} C(a_k, S_m) + C(p_1^k, S_{m-1}) + C(p_2^k, S_{m-1}).$$

The complexity of Voronoi cells of points in $\mathcal{VD}(S_{m-1})$ can be bounded relative to $\mathcal{VD}(S_m)$:

$$C(p_1^k, S_{m-1}) + C(p_2^k, S_{m-1}) \le C(p_1^k, S_m) + C(p_2^k, S_m) + 2C(a_k, S_m).$$

Let c denote the maximum number of segments and arcs incident upon a point of S. We conclude that

$$\begin{aligned} E(X_m) &\le \frac{1}{m} \sum_{k=1}^{m} \left(3C(a_k, S_m) + C(p_1^k, S_m) + C(p_2^k, S_m)\right) \\ &\le \frac{1}{m} \left(3 \sum_{k=1}^{m} C(a_k, S_m) + 2c \sum_{i=1}^{n_1} C(p_i, S_m)\right) \\ &\le \frac{3c}{m} \underbrace{\left(\sum_{k=1}^{m} C(a_k, S_m) + \sum_{i=1}^{n_1} C(p_i, S_m)\right)}_{\in O(n)}, \end{aligned}$$

thus obtaining that $E(X_m) \in O(cn\frac{1}{m})$. Since $\sum_{m=1}^{n_3} \frac{1}{m} \in O(\log n_3)$, summing over all expectations yields

$$O(cn \log n_3)$$

as the expected complexity of inserting all arcs $a_1, \ldots, a_{n_3}$. The complexity bound claimed follows immediately for $c \in O(1)$. □

As a side remark, we note that for real-world data one may expect the complexity of "most" Voronoi polygons to be "almost constant", which suggests a nearly linear overall complexity. Indeed, this is confirmed by experiments; see Section 4.

3. Computation of Voronoi nodes

When implementing a Voronoi algorithm for straight-line segments and circular arcs one will quickly discover that the computation of the Voronoi nodes is a major stumbling block: Attempts to compute Voronoi nodes by means of brute-force intersections of the supporting conics of the Voronoi edges are bound to derail, due to the high (algebraic) complexity and inherent numerical instability of these computations on a standard floating-point hardware. (Kim et al. [36] use rational quadratic Beziér curves for parameterizing the bisectors, but they also report that the computation of the Voronoi nodes poses serious (numerical) problems.)

Hence, a smarter approach is needed. Rather than relying on some numerical representation of the bisectors, we compute the Voronoi nodes directly by resorting to the original input (segments and arcs). Only upon the complete construction of the Voronoi diagram we fit parameterizations to the bisectors. (As in VRONI, we use the parameterization detailed in [22].)

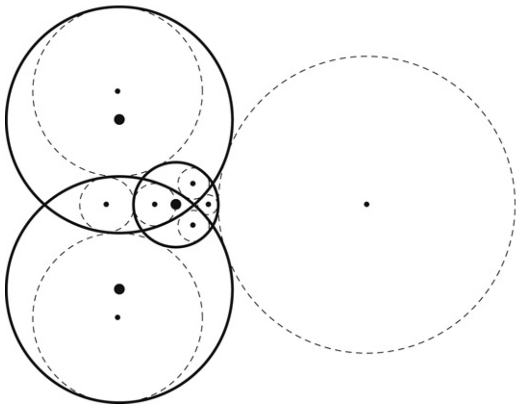
Fig. 8. Eight points (and their dashed offset circles) for three input circles depicted by thick solid curves.

To simplify matters, let us pretend that we have infinite straight-lines and full circles rather than straight-line segments and circular arcs. (Points can be treated as circles with zero radius.) Obviously, computing Voronoi nodes among segments and arcs boils down to computing a point that is equidistant to three objects, where an object can be a straight-line or a circle (with non-negative radius). One can prove easily that any arrangement of three lines and circles, with no pair of objects coinciding, admits at most eight points that are equidistant to all three objects. Thus, we can compute Voronoi nodes by (1) determining all points that are equidistant to three objects, and (2) choosing the proper equidistant point as Voronoi node.

3.1. Points equidistant to three circles and/or lines

We start with considering points equidistant to three circles $C(c_1, r_1)$, $C(c_2, r_2)$ and $C(c_3, r_3)$, with centers $c_i = (x_i, y_i)$ and radii $r_i \ge 0$ for $i \in \{1, 2, 3\}$. The following lemma is easy to prove.

Lemma 12. *A point p is equidistant to $C(c_1, r_1)$, $C(c_2, r_2)$ and $C(c_3, r_3)$ if and only if there exists an offset $t \ge 0$ and three constants $k_1, k_2, k_3 \in \{-1, 1\}$ such that the three offset circles $C(c_1, r_1 + k_1 \cdot t)$, $C(c_2, r_2 + k_2 \cdot t)$ and $C(c_3, r_3 + k_3 \cdot t)$ intersect at p.*

This lemma implies that all points $p = (p_x, p_y)$ equidistant to the three circles are solutions of the system of equations

$$(p_x - x_1)^2 + (p_y - y_1)^2 = (r_1 + k_1 t)^2, \tag{1}$$

$$(p_x - x_2)^2 + (p_y - y_2)^2 = (r_2 + k_2 t)^2, \tag{2}$$

$$(p_x - x_3)^2 + (p_y - y_3)^2 = (r_3 + k_3 t)^2, \tag{3}$$

for specific $k_1, k_2, k_3 \in \{-1, 1\}$ and offsets $t \ge 0$. Fig. 8 depicts all eight points equidistant to three circles.

Note that $k_1^2 = k_2^2 = k_3^2 = 1$. We obtain a linear system in p_x and p_y by subtracting Eq. (2) from Eqs. (1) and (3):

$$\begin{aligned} &2p_x(x_2 - x_1) + 2p_y(y_2 - y_1) \\ &\quad = r_1^2 + 2r_1k_1t - r_2^2 - 2r_2k_2t - x_1^2 + x_2^2 - y_1^2 + y_2^2 \\ &2p_x(x_2 - x_3) + 2p_y(y_2 - y_3) \\ &\quad = r_3^2 + 2r_3k_3t - r_2^2 - 2r_2k_2t - x_3^2 + x_2^2 - y_3^2 + y_2^2. \end{aligned}$$

Solutions of this system are of the form $p_x = (a_0 + a_1 \cdot t)/d$ resp. $p_y = (b_0 + b_1 \cdot t)/d$, where

$$d = 2(x_1(y_2 - y_3) + y_1(x_3 - x_2) + x_2y_3 - x_3y_2),$$

and where the coefficients a_0, a_1, b_0, b_1 depend only on parameters of the three circles. By plugging p_x, p_y into Eq. (1) we get a quadratic equation in t, and, thus, up to two offset values t_1, t_2. (Of course, only real non-negative values of t need to be considered.) The coordinates p_x and p_y are easily obtained by plugging t_1, t_2 into the expressions of p_x and p_y, provided that the denominator d is non-zero. It can be shown that d is zero if and only if c_1, c_2, c_3 are collinear. In this case we have to compute p_x and p_y directly by intersecting two of the three offset circles. Carrying out these simple computational steps for all (eight) combinations of possible values for k_1, k_2, k_3 produces all points equidistant to the three circles.

Of course, in an actual implementation one has to take care of special cases. For instance, if two of the three arcs lie on the same supporting circle then we have only two (rather than three) independent equations. However, in this case their cones of influence intersect either only at the common center or in a ray that extends from the center. In the first case, the center is the only candidate for the point p. In the second case, p is constrained to that ray, and it suffices to intersect that ray (parameterized relative to t) with the third offset circle. In any case, the computation of points equidistant to the three circular arcs is simpler than for the general set-up.

Exactly the same approach[11] works for two circles and a line, respectively, for a circle and two lines. The only difference is that one (resp. two) of the three initial equations are linear. Furthermore, the denominator d gets zero if only if (i) the supporting line of the two centers is orthogonal to the line (in the case of two circles), resp. (ii) the two lines are parallel (in the case of two lines).

3.2. Selecting the correct solution

The approach described above yields a set of points that are equidistant to (the supporting lines and circles of) three sites s_1, s_2, s_3, thus forming the candidate set for a new Voronoi node defined by s_1, s_2, s_3. With out loss of generality, we assume that s_3 is the arc that is to be newly inserted. It remains to choose the proper candidate as new Voronoi node. In order to weed out inappropriate candidates, we apply the following rationale.

(1) Recall that s_1 and s_2 define a semi-marked edge e of $\mathcal{VD}(S)$, and that the new Voronoi node has to lie on e. Thus, we can discard candidates that do not lie on e.
(2) Since e is constrained to $\mathcal{CI}(s_1)$, it is easy to see that e can lie on only one side of s_1 if s_1 is a line segment. Analogously if s_1 is an arc. Thus, k_1 and k_2 in Eqs. (1)–(3) are fixed.
(3) Recall that s_3 is a circular arc rather than a full circle. Thus, we can discard all candidates which do not lie in $\mathcal{CI}(s_3)$.
(4) Finally, if the node v of e which was not marked lies within $\mathcal{CI}(s_3)$, then the new Voronoi node has to lie on the same side of s_3 as v.

A rather lengthy proof based on a case analysis shows that the application of these rules does indeed always select the proper candidate as Voronoi node.

4. Experimental results

The algorithm described was implemented in ANSI C, as an extension to Vroni. (However, we emphasize that the basic algorithm and implementational issues described in the Sections 2 and 3 are independent of the specific implementation.) To the best of our knowledge, this makes Vroni the first implementation of a Voronoi algorithm which is able to cope with genuine circular arcs, without resorting to (straight-line) approximations or some form of sampling. For this reason we cannot compare the new Vroni with other Voronoi implementations. (For Voronoi diagrams of line segments, comparisons of the old Vroni with other Voronoi implementations are given in [22].)

The following performance tests were carried out on an Intel Core 2 Duo E6700 processor which is clocked at 2.66 GHz. It uses a 4 MB Level 2 cache. We note that Vroni is single-threaded and, thus, does not gain by running on a multi-core machine. Furthermore, the test machine runs a 32-bit Linux system and has 4 GB of physical RAM. (However, a single process can allocate only up to approximately 3 GB of memory.)

A natural goal of our work was to provide experimental evidence that the new Vroni is both reliable and efficient. In order to avoid jumping to false conclusions due to idiosyncrasies of the test data, we ran our tests on a large number of synthetic and real-world data sets. As in [22], the synthetic polygonal data was generated by means of RandomPolygonGenerator (RPG), cf. [37], which is a tool designed for the machine generation of pseudo-random polygonal test data. (See [22] for sample figures of polygons tested.)

The real-world data was obtained from companies, colleagues, and the web. Our data sets include GIS maps of roads and river networks, PCB data, boundaries of work-pieces for NC machining or stereo-lithography, outlines of fonts, and NC tool paths. Using Vroni's built-in offsetting function, we also generated offset patterns of some data sets, and re-applied Vroni to those offset patterns. As far as this was meaningful, we also ran PowerApx, cf. [10], on purely polygonal synthetic or real-world data in order to generate data that contains circular arcs. All circular-arc data that is non-proprietary has been made public [30].

The CPU-time consumption of Vroni was obtained by using the C system function "getrusage()". We report both the system and the user time. Of course, any file I/O is not included in the timings reported. All CPU times are given in milliseconds. To avoid systematic patterns in the plots for small data sets due to the finite resolution of the clock, we determined the average CPU-time consumption per data set by running Vroni up to 50 times or until the sum of the CPU times of the individual runs exceeded 50 ms.

Fig. 9 shows the total CPU-time consumption of the new Vroni on about 18 600 data sets, for both synthetic and real-world data. Nearly 3800 of these data sets contain circular arcs. (Since no apparent difference in the CPU-time consumption occurs between real-world data and virtually all synthetic data, we refrain from presenting two plots.) The plot shows the total CPU time consumed (in milliseconds) divided by the number n of input sites. That is, the y-axis shows the total CPU time consumed by the insertion of one site relative to the total number of input sites (which are given in logarithmic scale on the x-axis).

The only synthetic data not included in the plot of Fig. 9 is the "spikes" data discussed in [22]. The "spikes" data is highly contrived data that was specifically designed to explore the dependence of the incremental insertion on the maximum number of segments (or arcs) incident upon a single point. As predicted by the complexity analysis, the new Vroni shows a roughly quadratic overall CPU-time consumption if all n input arcs are incident upon a single input point.

The speed-up of the new Vroni relative to the old Vroni (which handled circular arcs via a Vroni-internal approximation and subsequent recovery of the arcs) can hardly be specified: It depends (i) on the relative number of arcs in the input data, (ii) on the error bound imposed on the approximation of the arcs,

[11] This approach is sketched in [12]; it can be traced back to communication among Gàbor Lukács, Hasse Persson, Tamás Várady and Martin Held nearly 20 years ago.

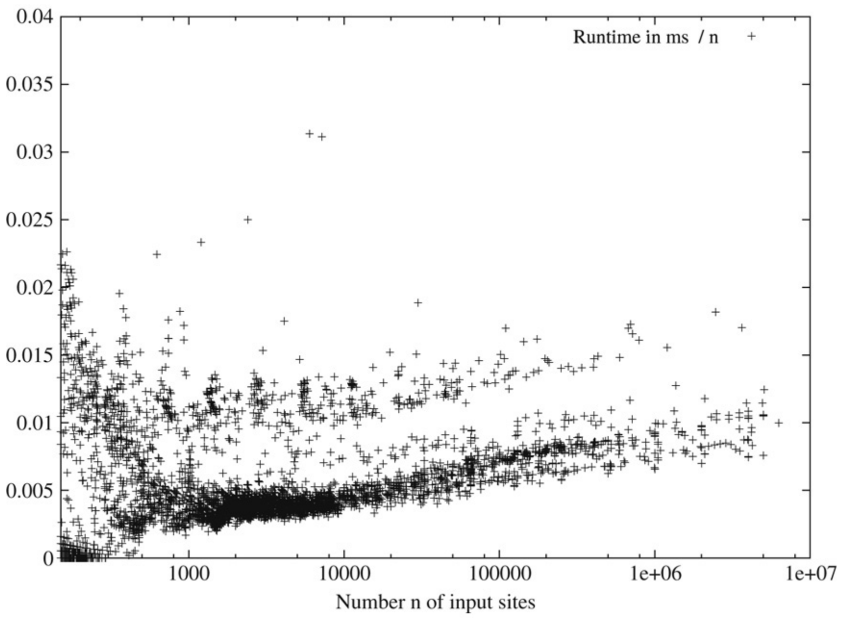

Fig. 9. The total CPU-time consumption (per site) of the new Vroni on about 18 600 data sets.

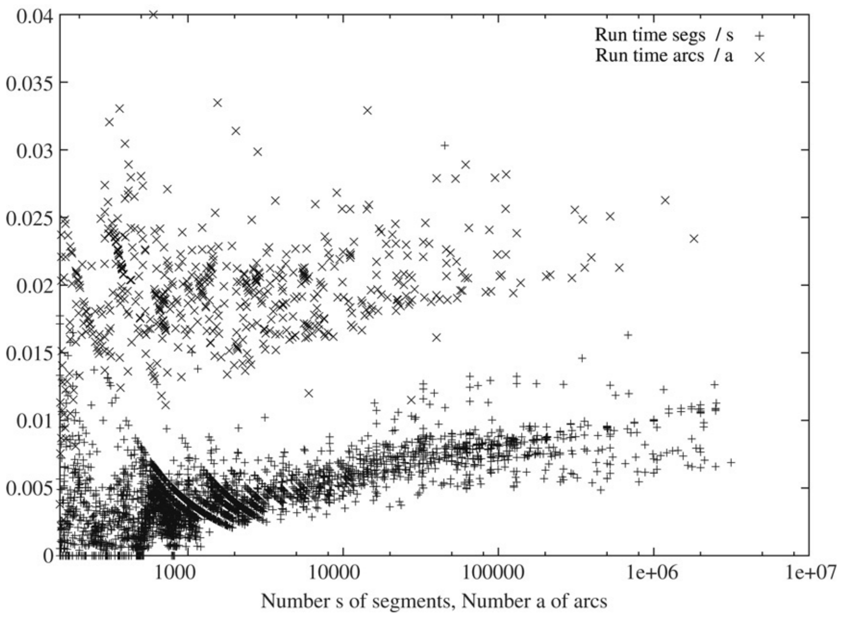

Fig. 10. The CPU-time consumption of the insertion of one circular arc or straight-line segment.

and (iii) on the size of the arcs relative to the size of the bounding box of the data. However, Fig. 10 shows the average CPU times consumed by the insertion of one circular arc and one straight-line segment. (This plot excludes the time spent on the preprocessing prior to the insertion of the segments and arcs as well as the time spent on the computation of the point Voronoi diagram.) As the plot shows, inserting a single arc is about as expensive as inserting three segments. In other words, the new Vroni can be expected to run faster than the old Vroni if every arc is approximated by at least about three segments. Obviously, for reasonable real-world applications the number of approximating segments per arc is significantly higher,[12] leading to a significant speed-up! Besides, the memory footprint is reduced accordingly.

Figs. 9 and 10 clearly show that the incremental insertion manages to construct the Voronoi diagram of n points, segments and arcs in $O(n \log n)$ time for virtually all data sets. Actually, for $n < 10^7$, on our platform n sites tend to be handled within $0.02n$ milliseconds or less. Since our tests cover both real-world and synthetic data sets, including fairly contrived data, it is fair to assume that this complexity bound will also hold for other real-world data.

5. Conclusion

We introduced a randomized incremental algorithm for computing Voronoi diagrams of circular arcs (and points and line segments) in 2D. The algorithm is based on the topology-oriented approach by Sugihara et alii. While the basic incremental scheme is identical to the one used in the line segment Voronoi code Vroni,

12 Typically an arc is approximated by tens or even hundreds of segments.

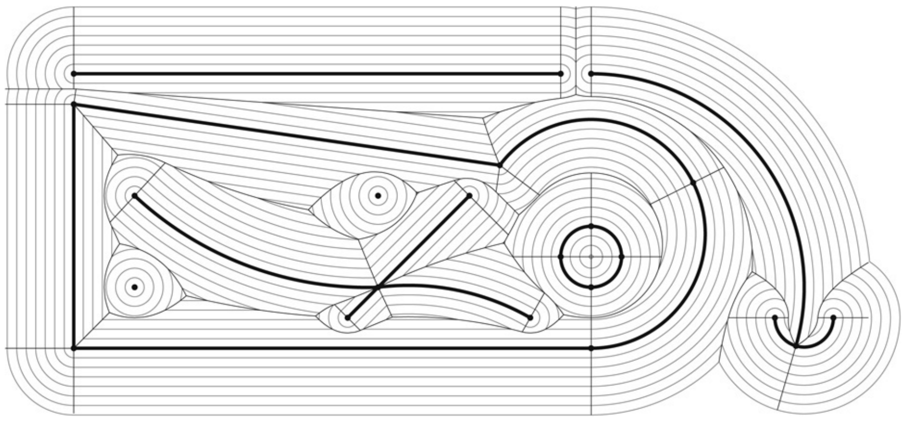

Fig. 11. Voronoi diagram (depicted by thin solid curves) and family of offset curves (grey) for a sample input (thick solid curves and isolated points).

substantial modifications of the incremental insertion were necessary for handling circular arcs. The mere fact that circular arcs are curved objects does complicate matters more than one might assume without going over the (formal) details and attempting a proof of correctness. However, it seems likely that this incremental insertion scheme could cope with other curved primitives (such as elliptic or parabolic arcs) after only minor modifications. (Of course, the basic numeric functions would have to be adapted.) This will be a worthwhile direction for future work.

We have implemented our incremental Voronoi algorithm as an extension to Vroni. Hence, we can rely on Vroni's features, such as the relaxation of epsilon thresholds and a multi-level recovery process. Furthermore, standard functionality of Vroni (such as extracting a medial axis or computing families of offset curves) is now also available for circular arcs. (See Fig. 11.) Extensive experimental tests with both synthetic and real-world test data have shown that the new Vroni runs reliably indeed on a standard floating-point arithmetic. Those data sets among our test data that contain circular arcs and that are not proprietary have been made publically available on the WWW. We expect this data to help foster future experimental research on the algorithmic handling of circular arcs.

To our knowledge, the new Vroni is the first (industrial-strength) implementation of a Voronoi algorithm that can handle genuine circular arcs. Since a major portion of our implementational efforts for this circular-arc extension of Vroni was spent on the routines needed for computing Voronoi nodes, we report on the mathematical foundations of the Voronoi node generation.

Our experiments also made it apparent that the new Vroni easily manages to stay within the time bounds provided by our theoretical analysis: We get $O(n^2)$ resp. $O(n \log n)$ as the worst-case and average-case complexities, and practical tests on thousands of data sets indicated that the average-case bound does indeed hold in practice. Of course, it has to be understood that these timings are not meant to suggest that the $O(n^2)$ worst case could never occur. (And, indeed, the highly contrived "spikes" data forces Vroni to spend $O(n^2)$ time.) Still, for most real-world data sets, on our platform we can expect the Voronoi diagram of n points, segments and arcs to be computed within at most $0.02n$ milliseconds. Since we used both synthetic and real-world test data and since our test data covers a wide range of shapes, there is good reason to assume that our performance statistics will remain valid for other data within some real-world application of Vroni.

References

[1] O'Dunlaing C, Yap C. A retraction method for planning the motion of a disc. J Algorithms 1985;6:104–11.

[2] Wein R, van den Berg J, Halperin D. The visibility-Voronoi complex and its applications. Comput Geom 2007;36(1):66–87.

[3] Giblin P, Kimia B. On the intrinsic reconstruction of shape from its symmetries. IEEE Trans Pattern Anal Mach Intell 2003;25(7):895–911.

[4] Mayya N, Rajan V. An efficient shape-representation scheme using Voronoi skeletons. Pattern Recognit Lett 1995;16(2):147–60.

[5] Oliva J, Perrin M, Coquillart S. 3D reconstruction of complex polyhedral shapes from contours using a simplified generalized Voronoi diagram. Comput Graph Forum 1996;15(3):397–408.

[6] Vermeer P. Medial axis transform to boundary representation conversion. Ph.D. thesis. West Lafayette (Indiana) 47907-1398. USA: CS Dept., Purdue University; 1994.

[7] Ang P, Armstrong C. Adaptive shape-sensitive meshing of the medial axis. Eng Comput 2002;18:253–64.

[8] Gürsoy H, Patrikalakis N. An automatic coarse and fine surface mesh generation scheme based on medial axis transform: Part I algorithm. Eng Comput 1992;8:121–37.

[9] Srinivasan V, Nackman L, Tang J-M, Meshkat S. Automatic mesh generation using the symmetric axis transform of polygonal domains. Proc IEEE 1992; 80(9):1485–501.

[10] Heimlich M, Held M. Biarc approximation, simplification and smoothing of polygonal curves by means of Voronoi-based tolerance bands. Internat J Comput Geom Appl 2008;18(3):221–50.

[11] Elber G, Cohen E, Drake S. MATHSM: Medial axis transform toward high speed machining of pockets. Comput Aided Des 2005;37(2):241–50.

[12] Held M. On the computational geometry of pocket machining. In: Lecture notes comput. sci., vol. 500. Springer-Verlag; 1991.

[13] Held M, Lukács G, Andor L. Pocket machining based on contour-parallel tool paths generated by means of proximity maps. Comput Aided Des 1994;26(3): 189–203.

[14] Held M, Spielberger C. A smooth spiral-like tool path with controlled cutting width for pocket machining. In: Proc 5th int symp Voronoi diagrams in science & engineering. 2008. p. 88–100.

[15] Lee D, Drysdale R. Generalization of Voronoi diagrams in the plane. SIAM J Comput 1981;10:73–87.

[16] Yap C. An $O(n \log n)$ algorithm for the Voronoi diagram of a set of simple curve segments. Discrete Comput Geom 1987;2(4):365–93.

[17] Fortune S. A sweepline algorithm for Voronoi diagrams. Algorithmica 1987; 2(2):153–74.

[18] Alt H, Schwarzkopf O. The Voronoi diagram of curved objects. In: Proc. 11th annu. ACM sympos. comput. geom. 1995. p. 89–97.

[19] Sugihara K, Iri M. Construction of the Voronoi diagram for 'one million' generators in single-precision arithmetic. Proc IEEE 1992;80(9):1471–84.

[20] Sugihara K, Iri M, Inagaki H, Imai T. Topology-oriented implementation — an approach to robust geometric algorithms. Algorithmica 2000;27(1):5–20.

[21] Imai T. A topology oriented algorithm for the Voronoi diagram of polygons. In: Proc. Canad. conf. comput. geom.. Ottawa, Canada: Carleton University Press; 1996. p. 107–12.

[22] Held M. VRONI: An engineering approach to the reliable and efficient computation of Voronoi diagrams of points and line segments. Comput Geom 2001;18(2):95–123.

[23] Kim D-S, Kim D, Sugihara K. Voronoi diagram of a circle set from Voronoi diagram of a point set: I. Topology. Comput Aided Geom Des 2001;18(6): 541–62.

[24] Kim D-S, Kim D, Sugihara K. Voronoi diagram of a circle set from Voronoi diagram of a point set: II. Geometry. Comput Aided Geom Des 2001;18(6): 563–85.

[25] Jin L, Kim D, Mua L, Kim D-S, Hu S-M. A sweepline algorithm for euclidean Voronoi diagram of circles. Comput Aided Des 2006;38(3):260–72.

[26] Anton F, Mioc D, Gold C. The Voronoi diagram of circles made easy. In: Proc. 4th ann. int. symp. Voronoi diagrams in Science & Engineering. 2007, p. 15–24.

[27] Karavelas M, Yvinec M. Dynamic additively weighted Voronoi diagrams in 2D. In: Proc. 10th annu. Europ. symp. algorithms. 2002, p. 586–98.

[28] Emiris I, Tsigaridas E, Tzoumas G. Voronoi diagram of ellipses in CGAL. In: Proc. 24th Europ. workshop comput. geom. France: Nancy; 2008. p. 87–90.
[29] Held M. www.cosy.sbg.ac.at/~held/projects/vroni/vroni.html.
[30] Held M. www.cosy.sbg.ac.at/~held/projects/repository/repository.html.
[31] Kirkpatrick D. Efficient computation of continuous skeletons. In: Proc. 20th annu. IEEE sympos. found. comput. sci. 1979, p. 18–27.
[32] Held M. Voronoi diagrams and offset curves of curvilinear polygons. Comput Aided Des 1998;30(4):287–300.
[33] Seidel R. Backwards analysis of randomized geometric algorithms. In: Pach J, editor. New trends in discrete and computational geometry. Springer-Verlag; 1993. p. 37–68.
[34] Boissonnat J-D, Devillers O, Schott R, Teillaud M, Yvinec M. Applications of random sampling to on-line algorithms in computational geometry. Discrete Comput Geom 1992;8:51–71.
[35] Klein R, Mehlhorn K, Meiser S. Randomized incremental construction of abstract Voronoi diagrams. Comput Geom 1993;3(3):157–84.
[36] Kim D-S, Hwang I-K, Park B-J. Representing the Voronoi diagram of a simple polygon using rational quadratic Beziér curves. Comput Aided Des 1995;27(8): 605–14.
[37] Auer T, Held M. Heuristics for the generation of random polygons. In: Proc. Canad. conf. comput. geom. Ottawa, Canada: Carleton University Press; 1996. p. 38–44.