Computation of Voronoi Diagrams of Circular Arcs and Straight Lines

Magisterarbeit

zur Erlangung des Diplomingenieurgrades an der
Naturwissenschaftlichen Fakultät der Universität Salzburg

eingereicht von
Stefan Huber

eingereicht bei
Ao. Univ.-Prof. Dipl.-Ing. Dr. Martin Held

Februar 2008

To my parents,
Gabriele and Stefan

Abstract

VRONI is one of few existing implementations for the stable computation of Voronoi diagrams of line segments. A topology-oriented approach in combination with double-precision floating-point arithmetic makes VRONI also the fastest and most reliable implementation available. Up to now, Voronoi diagram algorithms used in industrial applications process input data consisting of points and straight-line segments. Since circular arcs are important in various applications like CAD/-CAM, printed circuit boards, etc., so far circular arcs have been approximated by a reasonable number of straight-line segments.

Extending the algorithm to support circular arcs has several advantages over an approximated solution, including higher performance and lower memory consumption. In this diploma thesis we extend VRONI to genuine circular arcs, pursuing the strategy of topological constraints and carefully implemented numerical procedures based on double-precision floating-point arithmetic. To the best of our knowledge, this makes VRONI the first implementation supporting genuine circular arcs. We provide an extensive mathematical analysis including proofs of correctness, computation of Voronoi nodes and compare the new implementation with the pre-genuine-arc version of VRONI.

Acknowledgment

I would like to thank my supervisor Prof. Martin Held for his all-around support. In particular, he provided the original version of VRONI, always had time for valuable discussions, and spent much time with proof-reading the thesis.

Further, I would like to thank Gerhard Mitterlechner and Roland Kwitt for proof-reading major parts of the thesis and valuable hints in linguistic issues.

Last but not least, I would like to thank my parents who supported me my whole life and made it possible for me to pursue my studies.

Contents

Chapter 1

Introduction

1.1 Overview and motivation

Computational geometry deals with algorithms and data structures for solving geometrical problems. Fundamental concepts in computational geometry are *convex hull*, *triangulation*, and *Voronoi diagram*. The reason for the importance of Voronoi diagrams arises in the first instance from its properties regarded to nearest-neighbor questions.

Voronoi diagrams have been applied to a wide range of problems in computational geometry. In particular, they lead to elegant solutions to problems in CAD/CAM[1], as well as motion planning. However, for a long time they have been of little practical use due to the lack of industrial-strength implementations. To our knowledge, VRONI is the fastest industrial-proven implementation around; it has been implemented and maintained by Martin Held. The corner points of success are, among others, the usage of ordinary double-precision floating-point arithmetic for high performance and topology-oriented methods for stability.

All implementations of Voronoi algorithms, which are usable on real-world data, process input data consisting of points and straight-line segments[2]. But since circular arcs are frequently used primitive objects in CAD/CAM, algorithms for computing the Voronoi diagram either rely on the approximation of circular arcs by a sufficient number of straight-line segments or they do not support them at all. Obviously, the approximation leads to problems: determining the sufficient number of straight lines, numerical accuracy, performance, memory consumption, etc.

This thesis extends the implementation VRONI to the handling of genuine circular arcs, pursuing a strategy of topological constraints and carefully implemented

[1] Computer Aided Design, Computer Aided Manufacturing

[2] There are further implementations processing circles only.

numerical procedures, using ordinary double-precision floating-point arithmetic. The remainder of this thesis is organized as follows:

The current chapter introduces the notion of Voronoi diagrams of points, and the extension to line segments and circular arcs, using the concept of cone of influences [Hel91]. Some basic lemmas and properties will be introduced. After that, we give an overview of theoretical investigations and implementations. The second chapter contains the theoretical part: at first, we motivate the ideas behind the topology-oriented Voronoi algorithm for points and extend these ideas to segments and circular arcs. Several problems arise from the introduction of arcs, regarded to topological and graph-theoretical constraints necessary for the correctness of the algorithm. Solutions to these problems are worked out and proofs of their correctness are given. The third chapter puts the focus in the numerically stable computation of Voronoi nodes. The last chapter shortly demonstrates an application, namely offsetting. After that, we compare the new VRONI with the pre-genuine-arc version.

1.2 Common definitions and notations

Notation	Description
$b(A, B)$	Bisector between the sets A and B
$D(c, r)$	Closed disk centered at c with radius r: $\{q \in \mathbb{R}^2 \ : \ d(q, c) \leq r\}$
$\mathrm{bd}\, A$	Boundary of A
$\mathcal{CD}(p, S)$	Clearance disk $D(p, d(p, \bigcup_{s \in S} s))$ of point $p \in \mathbb{R}^2$ and set S of input sites s
$\mathrm{ch}\, A$	Convex hull of A
$\mathrm{cl}\, A$	Closure of A
$\mathcal{CI}(s)$	Cone of influence of input site s
v^{CCW}	Vector $v \in \mathbb{R}^2$ rotated 90^o counter clock-wise
v^{CW}	Vector $v \in \mathbb{R}^2$ rotated 90^o clock-wise
$d(p, q)$	Euclidean distance of point p and q, that is $\|\|p - v\|\|$
$d(A, B)$	Extension of d to the infinimum-distance between sets A and B: $\inf\{d(p, q) \ : \ p \in A, q \in B\}$
$H(p, v)$	Half-plane with orthogonal vector v and a point p on the boundary: $\{q \in \mathbb{R}^2 \ : \ q \cdot v \geq p \cdot v\}$
$\mathrm{int}\, A$	Interior of a set A
$\|\|v\|\|$	Euclidean 2-norm of the vector $v \in \mathbb{R}^2$
$\mathrm{relint}\, A$	Relative-interior of a set A where the universal set is given by the context

$\bigcup S$	The union $\bigcup_{s \in S} s$ for a set S of sites
$\mathcal{VC}(s, S)$	Voronoi cell of input site $s \in S$
$\mathcal{VD}(S)$	Voronoi diagram of the set S of input sites
$\mathcal{VP}(s, S)$	Voronoi polygon of input site $s \in S$

1.3 Voronoi diagrams

Voronoi diagrams are named after the Russian mathematician Georgy Voronoy[3]. Their historical roots are quadratic forms, studied by Dirichlet and Gauß. This is the reason for sometimes calling them *Dirichlet tesselations*. Voronoy provided a generalization to higher dimensions (see [Kle89, Aur91] for an overview).

Voronoi diagrams emerged in computer science the first time in the work of Shamos and Hoey [SH75]. They were interested in the Voronoi diagram of n points in the two-dimensional Euclidean space $\mathbb{R}^2$ and presented an $O(n \log n)$ divide-and-conquer algorithm for computing it. Voronoi diagrams have been generalized in several directions (see [LD81, Yap87, Kle89, AS95] for an overview), including

- Voronoi diagrams of curved objects, in particular of line segments and circular arcs;
- Voronoi diagrams in higher dimensions;
- Voronoi diagrams under different metrics, in particular the L_1 and L_∞ metric.

Aurenhammer [Aur91] gives a survey of Voronoi diagrams of points and circles. They have numerous applications in computer science, biology or geography. In computer science problems concerning robotics, CAD/CAM or GIS[4] are addressed. Especially in CAD/CAM and robotics domain, an generalization to Voronoi diagrams of segments and circular arcs has high potential for practical applications, like offsetting and motion planning.

1.3.1 Voronoi diagrams of points

We follow the notations and definitions of [Hel91]. Consider a set S of n pairwise distinct input points $p_1, \ldots, p_n \in \mathbb{R}^2$ in the Euclidean plane with standard Euclidean distance function d. The *Voronoi cell* $\mathcal{VC}(p, S)$ of the point $p \in S$ is given by

$$\mathcal{VC}(p, S) := \{q \in \mathbb{R}^2 \ : \ \forall p' \in S : \ d(q, p) \leq d(q, p')\}.$$

[3]Different transliterations exists, in particular Georgij Voronoj and Georgi Woronoi.
[4]Geographic Information System.

Hence, it consists of points q which are at least as close to the input point p than to all other input points p'. The relation to nearest-neighbor questions is obvious. The Voronoi cell $\mathcal{VC}(p, S)$ has a convex polygonal shape. The polygonal boundary of $\mathcal{VC}(p, S)$ is denoted by $\mathcal{VP}(p, S)$, the so-called *Voronoi polygon*, and therefore given by

$$\mathcal{VP}(p, S) := \{q \in \mathcal{VC}(p, S) \ : \ \exists p' \in S \setminus \{p\} : \ d(q, p) = d(q, p')\}.$$

In other words, the polygonal boundary $\mathcal{VP}(p, S)$ is the locus of points $q \in \mathbb{R}^2$ having equal distance to p and at least one further input point $p' \in S \setminus \{p\}$ such that no third point from S is strictly closer. The union of all boundaries $\mathcal{VP}(p, S)$ for $p \in S$, is called the *Voronoi diagram* of the input points in S and is denoted by $\mathcal{VD}(S)$:

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

Voronoi edges and Voronoi nodes The edges of a Voronoi polygon $\mathcal{VD}(p, S)$ are called *Voronoi edges* and are well defined by two input points separated by the edge. Voronoi edges can intersect each other only at their end points. The intersection point of three or more Voronoi edges is called *Voronoi node*. A Voronoi node is therefore equidistant to all input points which are co-defining a Voronoi edge ending in this node. The number of edges ending in a node is called the *degree* of the node. If no four input points are co-circular then all Voronoi nodes have degree three.

1.3.2 Generalization to segments and arcs

A straightforward generalization of Voronoi diagrams to points, straight-line segments and circular arcs (summarized as input *sites*) addresses the needs in CAD/-CAM. In this domain we have to deal with polygonal and circular objects. We extend the notion of the distance d of points to the infinimum distance between two sets A and B:

$$d(A, B) := \inf\{d(p, q) \ : \ p \in A, q \in B\}.$$

By identifying points p with the single point set $\{p\}$ and re-using the new distance d in the definitions of Section 1.3.1, we get a straight-forward generalization of Voronoi diagrams. By doing so a problem occurs: bisectors between two lines can now be sets of non-zero area. In Figure 1.2 a simple example is illustrated. This is not what we expect for the structure of a Voronoi diagram, which should be more like a skeleton consisting of "lines". Moreover, this generalization does not lead to elegant solutions for problems like offsetting, etc.

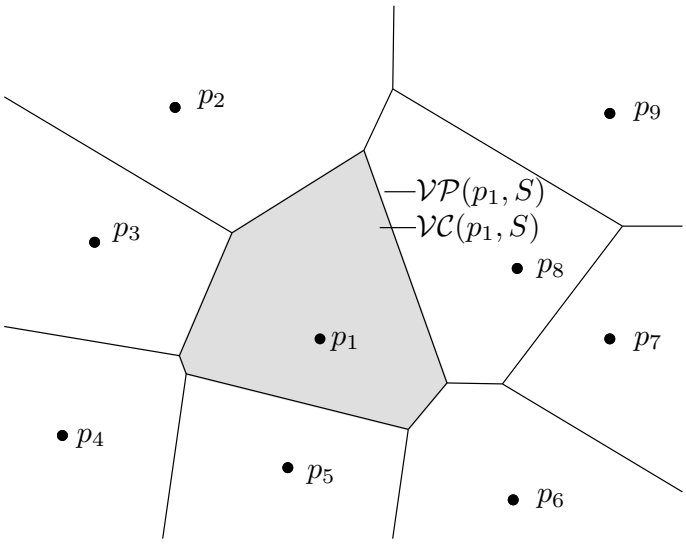

Figure 1.1: A set $S = \{p_1, \ldots, p_9\}$ of input points and the corresponding Voronoi diagram $\mathcal{VD}(S)$.

The introduction of *cones of influence* fixes this problem [Hel91]. We separately define the cone of influence for all three types of input sites, namely points, segments, and arcs. For an input site s we define the cone of influence $\mathcal{CI}(s)$ in the following way. Recall the notation of a half-plane $H(p, v)$ from Section 1.2.

Definition 1.1 (Cone of influence)**.** For an input site s the *cone of influence* $\mathcal{CI}(s)$ is defined as

$$\mathcal{CI}(s) := \begin{cases} \mathbb{R}^2 & \text{for a point } s \\ H(a, b-a) \cap & \text{for a segment } s \text{ with startpoint } a \\ \quad H(b, a-b) & \quad \text{and end point } b \\ H(c, (s-c)^{\mathrm{CCW}}) \cap & \text{for a CCW arc } s \text{ with center } c, \text{ start } s \text{ and end } e, \\ \quad H(c, (e-c)^{\mathrm{CW}}) & \quad \text{if } s \text{ is smaller than a semi-circle} \\ H(c, (s-c)^{\mathrm{CCW}}) \cup & \text{for a CCW arc } s \text{ with center } c, \text{ start } s \text{ and end } e, \\ \quad H(c, (e-c)^{\mathrm{CW}}) & \quad \text{if } s \text{ is greater than a semi-circle} \end{cases}$$

The cone of influence restricts the Voronoi cell of an input site to a specific area, as illustrated in Figure 1.3. Based on this idea we can extend the terms Voronoi cell, Voronoi polygon and Voronoi diagram, as we use them in the remaining work. From now on we declare that for every arc and segment in the set S of input sites, both end points are contained in S as well. For formal reasons of disjoint input

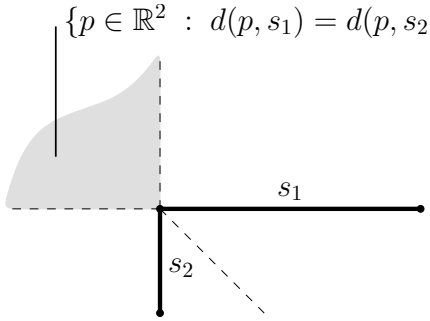

Figure 1.2: A simple example where the set of points equidistant to two segments has non-zero area.

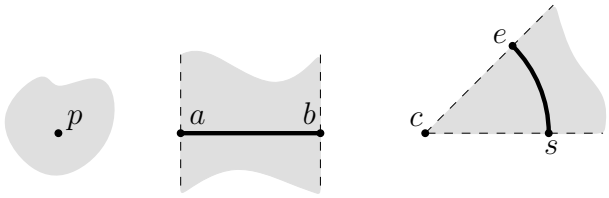

Figure 1.3: The cone of influence of a point, segment and arc.

sites we consider a segment as an open line segment. The same holds for an arc[5].

Definition 1.2 (Voronoi cell, polygon, and diagram)**.** Let us consider a finite set S of disjoint input sites consisting of points, open line segments and open arcs. For every segment and arc in S the end points are elements of S, too. We define a *Voronoi cell* $\mathcal{VC}(s, S)$ of an input site $s \in S$ as

$$\mathcal{VC}(s, S) := \mathrm{cl}\{q \in \mathrm{int}\, \mathcal{CI}(s) \ : \ d(q, s) \leq d(q, \bigcup S)\}. \tag{1.1}$$

Other common terms for $\mathcal{VC}(s, S)$ are Voronoi Region and Voronoi Area. The boundary of the Voronoi cell $\mathcal{VC}(s, S)$ is called Voronoi Polygon $\mathcal{VP}(s, S)$ and denoted as

$$\mathcal{VP}(s, S) := \{q \in \mathcal{VC}(s, S) \ : \ \exists s' \in S \setminus \{s\} : d(q, s) = d(q, s')\}. \tag{1.2}$$

The union of all Voronoi polygons $\mathcal{VP}(s, S)$ for every input site s is called the Voronoi diagram $\mathcal{VD}(S)$ of the input sites $s \in S$ and is defined as

$$\mathcal{VD}(S) := \bigcup_{s \in S} \mathcal{VP}(s, S). \tag{1.3}$$

[5]Open in the sense of relative-open with respect to the supporting straight line or circle, respectively.

Note on the closure-interior definition of $\mathcal{VC}$ On the first sight the definition of a Voronoi cell, using the closure-interior expressions, looks unfamiliar. The common definition of a Voronoi cell $\mathcal{VC}(s, S)$ is more like $\{q \in \mathcal{CI}(s) : d(q, s) \leq d(q, \bigcup S)\}$. At first, the difference seems negligible but Figure 1.4 illustrates a significant reason for using the closure-interior definition (or something equivalent).

Consider two arcs s_1, s_2 that meet tangentially, as depicted in Figure 1.4. All points on the line segment from the center of s_1 to the common end point of the arcs s_1 and s_2 are equidistant to both arcs. In Figure 1.4 a point p on this segment is illustrated. Therefore, these points would be elements of the Voronoi cell $\mathcal{VC}(s_2, S)$ (and of course of $\mathcal{VC}(s_1, S)$ as well) in the sense of the non-closure-interior definition. In contrast to that, the closure-interior definition removes these "needles" from the Voronoi cell. Again, these "needles" are unnatural and hurt the common conception of Voronoi diagrams. Moreover, these "needles" prevent applications like offsetting and destroy many nice properties of Voronoi cells, in particular of the Voronoi polygon. For example, these "needles" form a continuum of points having the property of being equidistant to three sites: both arcs and the common end point. Is this a continuum of Voronoi nodes? This is a fundamental contradiction to a commonly known property of Voronoi diagrams: the number of Voronoi nodes is linear in the number of input sites.

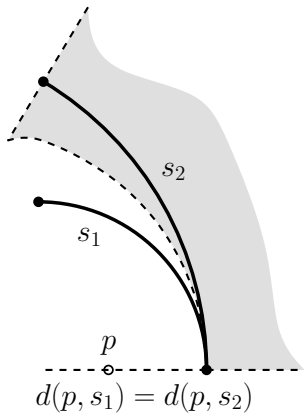

Figure 1.4: The standard definition of the Voronoi cell $\mathcal{VC}(s_2, S)$, without the closure-interior term, gathers the whole dashed line to the center of s_1 as well.

Definition 1.3 (Bisector)**.** The bisector $b(A, B)$ between two sets A and B is the locus of points equidistant to A and B, thus

$$b(A, B) := \{q \in \mathbb{R}^2 \ : \ d(q, A) = d(q, B)\}. \tag{1.4}$$

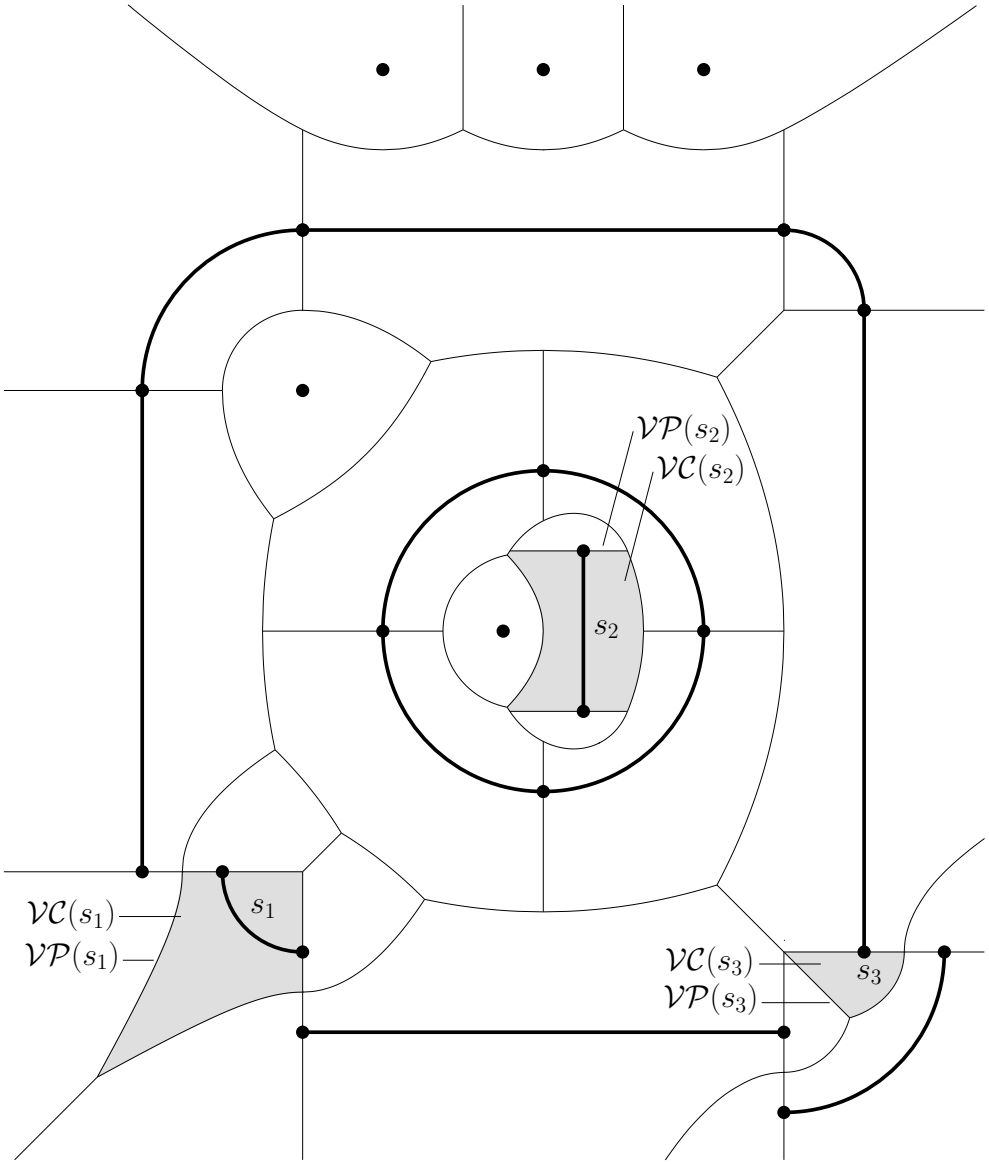

Figure 1.5: A generalized Voronoi diagram of input sites (bold) and three highlighted Voronoi cells.

Note that the definitions are identical with the ones for Voronoi diagrams of points, for the case of S consisting exclusively of points. Many properties of Voronoi diagrams of points are valid for generalized Voronoi diagrams, too. Other properties have to be relaxed to remain valid.

Clearance, Voronoi edges and nodes The term *Voronoi edge* defined by $s_1, s_2 \in S$ is reused in generalized Voronoi diagrams, in the sense of a part of the Voronoi polygon $\mathcal{VP}(s_1)$, or $\mathcal{VP}(s_2)$, where each point is equidistant to s_1 and s_2. It is therefore a section of the bisector $b(s_1, s_2)$. Roughly speaking, a Voronoi polygon $\mathcal{VP}(s)$ consists of Voronoi edges "around" the input site s. A point where two Voronoi edges intersect is called a *Voronoi node*. A Voronoi node at point p is therefore equidistant to all pairs of input sites s_1, s_2 defining a Voronoi edge ending in p. The distance $d(p, \bigcup S)$ of a point $p \in \mathbb{R}^2$ to the set S of all input sites is called the *clearance*. The disk $D(p, d(p, \bigcup S)))$ is known as *clearance disk* $\mathcal{CD}(p, S)$. The interior of $\mathcal{CD}(p, S)$ is disjoint with $\bigcup S$, thus contains no point of an input site $s \in S$. The whole $\mathbb{R}^2$ is tessellated into Voronoi cells. With other words: there is no point which would not belong to a Voronoi cell.

Lemma 1.4 (Bisector forms)**.** *The bisector between two circles (in particular points) consists of ellipses and hyperbolas.*

Proof. This fact is well known and easy to see: consider a circle centered at c_1 with radius r_1 and a circle centered at c_2 with radius r_2. Then, the set of points p equidistant to both circles are those satisfying the equation

$$| \, ||p - c_1|| - r_1| = | \, ||p - c_2|| - r_2|.$$

W.l.o.g., we assume $r_1 \geq r_2$. By distinguishing the cases which arise from the signs of $||p - c_1|| - r_1$ and $||p - c_2|| - r_2$, we obtain the equation for hyperbolas $||p - c_1|| - ||p - c_2|| = r_1 - r_2$ for equal signs and for unequal signs the equation for ellipses $||p - c_1|| + ||p - c_2|| = r_1 + r_2$. □

Lemma 1.5 (Bisector forms)**.** *The bisector between a circle and a line consists of parabolas.*

The following lemma is known and useful. A more general one is proven in [AS95] (Lemma 2) for the Voronoi diagram of "harmless curves", including arcs and segments.

Lemma 1.6. *For every point $p \in \mathcal{VC}(s, S)$ and for a point $q \in \operatorname{cl} s$ with $d(p, s) = d(p, q)$, the whole line $\overline{pq}$ is in $\mathcal{VC}(s, S)$.*

Proof. Consider a point $p \in \mathcal{VC}(s, S)$ and a point $q \in \operatorname{cl} s$ such that $d(s, q) = d(p, q)$. Assume that there is a point $x \in \operatorname{int} \overline{pq}$, such that $x \notin \mathcal{VC}(s, S)$. Then, there exists a site $s' \in S \setminus \{s\}$ with $d(x, s') < d(x, s) = d(x, q)$. But since $d(s', p) \leq d(s', x) + d(x, q) < d(s, x) + d(x, p) = d(s, p)$, due to the triangle inequality, we obtain a contradiction to $p \in \mathcal{VC}(s)$. □

Definition 1.7. A set A is called *generalized star-shaped* with nucleus $N \subseteq A$ if for every point $p \in A$ exists a point $q \in N$ such that A contains the whole segment $\overline{pq}$.

Corollary 1.8. *The Voronoi cell $\mathcal{VC}(s, S)$ of a site $s \in S$ is generalized star-shaped with nucleus s.*

Proof. This is an immediate consequence of Lemma 1.6. □

Note The point $q \in s$ appearing in the proof of Lemma 1.6 is commonly known as *projection* of p on s. In [Hel91] (p. 70, Lemma 5.1) its is shown that the projection q of p on s is uniquely defined for segments, arcs or points s, except for p being the center of an arc s. Note that from a pure formal point of view the projection q can lie outside of s, but has to lie at least in the closure $\operatorname{cl} s$ of s. This is the case if $p \in \operatorname{bd} \mathcal{CI}(s)$.

Corollary 1.9. *Let $s \in S$ denote an input site. Then, $\mathcal{VC}(s, S)$ is connected.*

Proof. Lemma 1.6 and the fact that s is connected itself imply that $\mathcal{VC}(s, S)$ is connected. □

Lemma 1.10. *The Voronoi cell $\mathcal{VC}(s, S)$ of an input site $s \in S$ is simply connected.*

Proof. We distinguish all cases of sites $s \in S$. If s is a point, thus $s = \{p\}$, the lemma holds since the Corollary 1.8 tells us that $\mathcal{VC}(s)$ is generalized star-shaped with nucleus $N = \{p\}$, i.e. star-shaped.

Let $\gamma : [0, 1] \to \mathcal{VC}(s, S)$ be a Jordan-curve in $\mathcal{VC}(s, S)$ and let $\gamma^* := \gamma([0, 1])$ be the image of the curve. We show that every point enclosed by γ^* is in $\mathcal{VC}(s, S)$, too. Let s be a segment and let q be a point enclosed by γ^*. We consider the projection line g of q on s and denote the projection by $p \in \operatorname{cl} s$. The intersection $g \cap \gamma^*$ contains at least one point p' such that $q \in \overline{p'p}$. Since $p' \in \gamma^* \subseteq \mathcal{VC}(s, S)$, p is the projection of p' on s and $q \in \overline{p'p}$, we conclude that $q \in \mathcal{VC}(s, S)$.

If s is a circular arc with center c, we denote by q a point enclosed by γ^*. We consider the ray g from c through q. The Voronoi polygon $\mathcal{VP}(s, S) \subseteq \mathcal{CI}(s)$ must be hit by g and we take a point $p' \in \mathcal{VP}(s, S) \cap g$, such that $d(c, p') \leq d(c, q)$. The intersection $g \cap s$ is the projection of q on s and is denoted by p. We now conclude that $q \in \overline{p'p} \subseteq \mathcal{VC}(s, S)$ due to Lemma 1.6. □

Note Note that this lemma is not true for circles! Consider a circle and a point in the center. Then, the Voronoi cell of the circle has a hole. This counter-example holds for every input site which is homeomorphic to a circle, except for a point.

1.4 Prior and related work

1.4.1 Related theoretical work

The generalization of Voronoi diagrams of points to segments (and circles) has been done by Lee and Drysdale. They presented an $O(n \log^2 n)$ algorithm in [LD81]. Fortune found an $O(n \log n)$ sweep-line algorithm for points and line segments [For86], where the main surprise was that a sweep-line algorithm exists at all. A year later Yap presented an $O(n \log n)$ algorithm for points, segments and arcs [Yap87]. Here the term cone of influence was introduced. However, the definition of the Voronoi diagram in [Yap87] makes no use of that – ϵ-neighborhoods[6] are used instead, to address the problem discussed in Section 1.3.2 – in contrast to the definition in [Hel91], as we already mentioned. Held extended a line segment divide-and-conquer algorithm of Lee [Lee82] to line segments and arcs [Hel91]. In [AS95], Alt and Schwarzkopf presented an $O(n \log n)$ algorithm for specific curved objects which they call "harmless curves". They define harmless curves as curves where no circle touches the curve more than once.

Topology-oriented methods were introduced by Sugihara and Iri in [SI92]. They presented an incremental algorithm for the Voronoi diagram of points, running in $O(n)$ in the average and $O(n^2)$ in the worst case. An extension to segments has been sketched by [Ima96] a few years later. His algorithm has $O(n^2)$ complexity if numerical errors are small enough, such that backtracking is avoided and runs in $O(n^3)$ in the worst case. Held showed missing algorithmic details in [Hel01] and introduced the implementation VRONI.

A related problem to the Voronoi diagram of segments and arcs is the Voronoi diagram of full circles, also known as the Apollonius graph. There is an $O(n \log^2 n)$ algorithm for n, possibly intersecting, circles by Sharir [Sha85]. Kim et al. transfered the concept of topology-oriented algorithms to the Apollonius graph in [KKS01a] and [KKS01b] and presented an $O(n^2)$ algorithm. But the construction is completely different to the algorithms in [SI92] or [Ima96]. The idea is to modify the Voronoi diagram of growing points in a proper way and to handle topological changes (edge flipping) appropriately. Recently, Jin et al. presented an $O((n+m) \log n)$ sweep-line algorithm for n circles, where m is the number of pairwise intersections [JKM+06].

[6] And a property of points, called $*$-close.

1.4.2 Existing implementations

There are only few implementations for the usage in real world. The following citation from [SIII00] states a reason for that issue:

> There is a great gap between theoretically correct geometric algorithms and practically valid computer programs.

When implementing a geometric algorithm, two parts have to be taken into account: firstly, the algorithmic part. Secondly, the numerical, computational part. A few paradigms arised to handle the computational part, where exact arithmetics and topological-oriented algorithms are two of them. The few implementations known can handle either segments or circles, but not segments and arcs simultaneously. We will shortly describe implementations we are aware of:

Name	**Description**	**Input**
AVD	Written by M. Seel. Uses LEDA, an exact arithmetic C++ package. Is known to be relatively slow. Furthermore, points have integer coordinates and the algorithm is an adaption of abstract Voronoi diagrams in [KMM93] to line segments (see preliminaries of [Kar04]).	Points, Segments
CGAL (Voronoi Diagrams)	Described in [Kar04], extends the algorithm in [KY03]. Considers an arrangement of segments for splitting on intersections. Expected $O((n+m)\log^2 n)$ complexity, where $m \in O(n^2)$ is the complexity of the arrangment. Uses floating-point filtering: Filters out degenerate data, i.e. uses double-precision and falls back to exact computation if necessary. According to [Kar04], this is the first "realistically efficient" implementation using exact computation.	Points, intersecting Segments
CGAL (Appolonius)	Written by Karavelas and Yvinec [KY02]. Expected $O(nT(h) + h\log h)$ complexity, where h is the number of sites having a non-empty cell and $T(k)$ is the time to locate the nearest neighbor within k sites. Has its origin in the algorithm from [KMM93] and is fully dynamic (can delete sites).	Circles

PLVOR	Described in [Ima96]. First topology-oriented algorithm for segments: the result is topologically correct. If no backtracking used, PLVOR has $O(n^2)$ complexity and worst-case of $O(n^3)$. Written in Fortran.	Points, Segments
PVD	Described in [SHM01]. Handles straight-line polygons with holes. Seems to be robust for practical purposes and uses topological properties. Based on ear-clipping of the Delaunay triangulation. Uses ordinary floating-point arithmetic. Written in C++.	Line-Polygons

Chapter 2

Insertion of Sites

In this chapter we discusses the basic incremental algorithm, constraints arising from theoretical considerations, sub-routines to fulfill these constraints and finally the correctness of these. The construction of Voronoi diagrams in VRONI is done *incrementally* by using a *topology-oriented approach*. Solving problems in computational geometry by using standard floating-point arithmetic is well known to be critical. An often cited statement by Steven Fortune can be found as the first sentence in [For00]:

> It is notoriously difficult to obtain a practical implementation of an abstractly described geometric algorithm.

Designing these algorithms by using topological properties of the specific problem has been discussed in [SIII00]. The idea is to move complexity to a discrete world[1] of topological properties, e.g. exploit a specific structure of an underlying graph. Here, ordinary integer arithmetic is used and is therefore free of numerical inaccuracy. In other words, the challenge is to find an algorithm which preserves fundamental topological properties – e.g. when inserting an input site in a Voronoi diagram – which are necessary properties for a solution to be correct. A decade before, a Voronoi diagram algorithm for points on a single-precision arithmetic has been presented in [SI92]. There, the authors assume a priori that "numerical errors may arise in the course of computation[...]". A few years later [Ima96] sketched an extension to segments. This principle, combined with others, led to the stable and fast Voronoi diagram implementation VRONI by Held, which has been applied to numerous industrial projects. In [Hel01] the following design principles have been emphasized:

1. Topology-oriented approach;

[1] The authors call it "combinatorial and/or topological part" or "topological part" for short.

2. Carefully implemented geometric primitives;

3. Relaxation of epsilon thresholds;

4. Multi-level recovery process with desperate mode.

The following sections describe the building blocks of VRONI with its extension to circular arcs. We will follow the paper [Hel01], where the pre-genuine-arc implementation has already been described.

2.1 The basic algorithm

Of course, the basic scheme of the incremental insertion in VRONI has remained more or less untouched, except necessary modifications needed to support arcs. As an introduction we will shortly explain the ideas introduced in [SI92] for constructing the Voronoi diagram of points. After that, we will present the actual implementation regarded to segments and arcs.

2.1.1 Motivation: VD of points

Consider n points $p_1, p_2, \ldots, p_n$ in the plane, denoted by the set S of input sites, and assume that the Voronoi diagram has already been computed for these points. By $S^+ = S \cup \{p_{n+1}\}$ we denote the set S including a new point $p_{n+1} \notin S$. When inserting the new point p_{n+1} in this Voronoi diagram, a straight-forward strategy would be:

1. Determine the point p_i whose Voronoi cell $\mathcal{VC}(p_i, S)$ has been hit by p_{n+1}.

2. Insert an edge $b(p_i, p_{n+1}) \cap \mathcal{VC}(p_i, S)$ in the Voronoi cell $\mathcal{VC}(p_i, S)$ of p_i.

3. Take one edge e' of the two edges of $\mathcal{VP}(p_i, S)$ intersected by the newly inserted edge. Let us denote by p_i' the site which is defining e' together with p_i. Thus the edge e' is also part of the polygon $\mathcal{VP}(p_i')$. We repeat Step 2 and Step 3 for the point p_i' until the start cell $\mathcal{VC}(p_i, S)$ is reached again. Thus, we walked around the new point p_{n+1} and traced the border of its new cell.

This straight-forward strategy is quite vulnerable to numerical inaccuracy. A simple example (see [SI92]) is illustrated in Figure 2.1, where a small inaccuracy can lead to topologically inconsistent results. This example can be easily modified such that for a given numerical precision the Voronoi cell is not "closed".

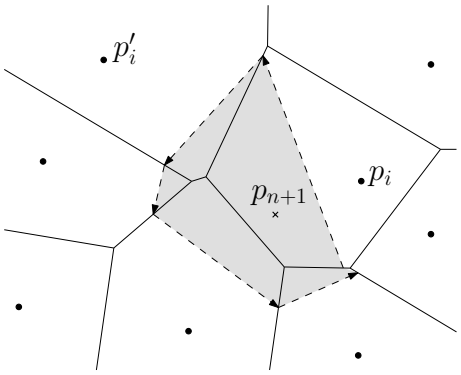

Figure 2.1: An example where numerical inaccuracies lead to topological inconsistencies.

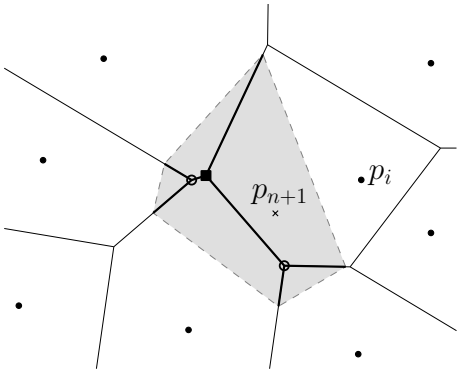

Figure 2.2: An example for a topologically driven insertion. The seed node is labeled by a square. The Voronoi nodes removed are labeled by a circle.

A topology-oriented algorithm exploits the graph structure of the Voronoi diagram. In [SI92] the following necessary requirements for the edge-graph and the topological properties of Voronoi cells to form a Voronoi diagram are stated:

- Every site has its own Voronoi cell;
- Every Voronoi cell is connected;
- Two Voronoi cells share at most one edge.

This is used for the topology-oriented algorithm. The following roughly described[2] algorithm inserts a new point p_{n+1} in the Voronoi diagram. The prop-

[2]See [SI92] for details – the algorithm for points, segments and arcs is described in more detail later. This is just a motivation for the case of points to give an illustration.

erties above are preserved – a loop invariant in other words. In Figure 2.2 this insertion is illustrated.

1. Determine the point p_i whose Voronoi cell $\mathcal{VC}(p_i, S)$ has been hit by p_{n+1}.
2. There exists at least one node in $\mathcal{VP}(p_i, S)$ which is closer to p_{n+1} than p_i. Take the node whose clearance disk is "violated most" by p_{n+1} and mark it. This initial node is known as seed node.
3. For every marked node, we mark an adjacent node if the clearance disk is violated. Repeat this step recursively.
4. The result is a tree structure of edges where both end points are marked. Remove every edge in this tree structure. Every edge where exactly one node is marked has to be adapted: calculate a new node replacing the marked one by considering the defining points of the edge and the new point p_{n+1}.
5. These new nodes are spanning the new Voronoi cell $\mathcal{VC}(p_{n+1}, S^+)$.

This idea of exploiting topological properties is extended to input sites consisting of points and segments in [Ima96]. The missing gaps for a practical implementation were filled by Held [Hel01] and led to the implementation of VRONI. The extension of VRONI to handle circular arcs is worked out in this thesis.

2.1.2 The current algorithm

The input data The current implementation can handle an input set S of $v \in \mathbb{N}_0$ sites consisting of points, segments of straight lines and circular arcs. For every segment or arc $s \in S$, both end points of s are contained in S too. No two input sites intersect each other[3]. As mentioned in Section 1.3.2, the segments and arcs are seen as open segments resp. open arcs. Furthermore, all circles and arcs are split appropriately such that every arc $s \in S$ is less than a semi-circle. Input sets with the properties from above are denoted as *proper* input sets from now on.

The initial Voronoi diagram After reading the input data, all input sites are translated and scaled appropriately such that the sites fit into the unit square. Since VRONI is an incremental algorithm it has to start with a well-defined initial Voronoi diagram. This initial Voronoi diagram results from the four corner points of an appropriately enlarged copy of the unit square. These four points are added to S as initial dummy points, as Figure 2.3 shows. There are four edges in the

[3] Actually, VRONI attempts to handle intersections: If a topological problem arises then VRONI check for intersections, splits two sites at the point of intersection, and restarts.

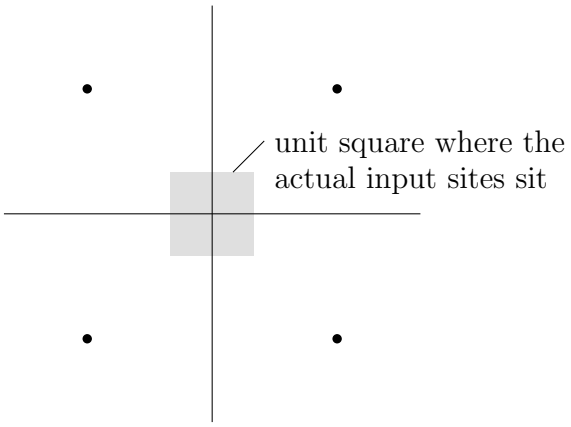

Figure 2.3: Initial Voronoi diagram with four dummy points.

initial Voronoi diagram which are unbounded and each of them is defined by two initial dummy points. The following lemma states that this remains true after inserting all input sites, since the convex hull $\mathrm{ch}(\bigcup S)$ is now the bounding box of S, including the four dummy points.

Lemma 2.1. *Consider a proper set S of input sites and an input site $s \in S$. The Voronoi cell $\mathcal{VC}(s, S)$ is unbounded iff s intersects the boundary $\mathrm{bd}\, C$ of the convex hull $C = \mathrm{ch}(\bigcup S)$.*

Proof. Let us assume that a point $p \in s$ is on the boundary of the convex hull, thus $p \in s \cap \mathrm{bd}\, C$. Then there exists a half-plane H such that $\bigcup S \subset C \subset H$ and $p \in \mathrm{bd}\, H$, due to the convexity of C. We consider a ray r perpendicular to the boundary-line $\mathrm{bd}\, H$ of H starting at p such that $r \subset \mathbb{R}^2 \setminus \mathrm{int}\, H$. Every point on r is closer to p than to a $p' \in H \setminus \{p\}$, hence r is in the Voronoi cell $\mathcal{VC}(s)$ and we conclude that $\mathcal{VC}(s)$ is unbounded.

Conversely, let us now assume that the Voronoi cell $\mathcal{VC}(s, S)$ is unbounded. Since $\mathrm{int}\, \mathcal{VC}(s, S)$ is generalized star-shaped with nucleus s, there exists a point $p \in s$ and an unbounded ray $r \subseteq \mathcal{VC}(s, S)$, starting from p. We consider the half-plane H perpendicular to r, where p is on the boundary $\mathrm{bd}\, H$ of H, such that $r \subset \mathbb{R}^2 \setminus \mathrm{int}\, H$. Since the ray r belongs to the Voronoi cell $\mathcal{VC}(s, S)$, all other sites $s' \in S \setminus \{s\}$ must be in H. Hence, p is part of the boundary $\mathrm{bd}\, H$ of the convex hull and s intersects $\mathrm{bd}\, H$. □

A similar proof of this well-known lemma can be found in [LD81]. Although they consider segments (and circles), the proof is general enough and would apply here too.

Inserting a new site VRONI inserts the input sites in semi-randomized order. At first, all points are inserted. After that VRONI inserts the segments and at last the arcs. The points, segments and arcs themselves are inserted in random order. By doing so, we avoid unfavourable insertion orders. In the remainder of this chapter we assume that $S^+ := S \cup \{s\}$ is a proper set of input sites, where $s \notin S$. Furthermore, we assume that we already know the Voronoi diagram $\mathcal{VD}(S)$ of S and we insert the new site s, where s is an arc if not specified differently. The principal insertion algorithm is sketched in Alg.-1 - Alg.-4. Of course, the actual implementation is more complicated:

- The actual implementation has to maintain a proper data structure. VRONI saves a reference to the nodes on both ends of every edge and the references to the adjacent edges – clock-wise and counter clock-wise. Therefore the update function has to reconstruct the references to the clock-wise and counter-clock-wise edges too. Furthermore, for every input site a reference to a node, resp. edge, of its Voronoi polygon is maintained. Vice versa, the defining sites are remembered for every edge too.

- When determining the seed node while inserting an arc or segment, topological checks are done too: Actually, a seed node is determined in the Voronoi polygons of both end points. The edge structure which is to be removed must reach both seed nodes. Recall that Voronoi cells are connected. This issue will be further discussed in Section 2.3.

  Later we will show that the edge graph of these edges has to form a tree and in Section 2.2 we discuss what will happen if the algorithm marks a whole cycle of edges.

- If there are more candidates for the seed node, we take the one whose clearance disk is violated most: if we denote by $c_v = d(v, S)$ the clearance of the node v and with $d_v = d(v, s)$, then we take the node v where $c_v - d_v$ is minimized.

- We do not select seed nodes which coincide with an input point. Especially, nodes which coincide with the end points of segments and arcs are not selected as seed nodes. Furthermore, these nodes are not marked as well. The issue related to this precaution is discussed in Section 2.3.2.

- VRONI intensively uses epsilon thresholds for various geometric predicates, in particular when determining wheter the clearance disk of a node has been intersected. If topological inconsistencies have been detected, VRONI tries to relax the epsilon thresholds step-by-step. If the "relaxation of epsilon thresholds" reaches an upper bound, VRONI falls back to "desperate mode"

Algorithm 1 Insert a new site s.

$v \leftarrow$ DetermineSeedNode(s)
mark node v
for incident edge e to v **do**
  RecursiveMarkNodes(s, v, e)
end for
for incident edge e to v **do**
  RecursiveUpdate(s, v, e)
end for

Algorithm 2 DetermineSeedNode(s)

if s is a segment or arc **then**
  $s' \leftarrow$ one of both end points of s
else
  $s' \leftarrow$ a site s' with $s \subseteq \mathcal{VC}(s', S)$
end if
return the node $v \in \mathcal{VP}(s', S)$ whose clearance disk is "violated most" by s

Algorithm 3 RecursiveMarkNodes(s, v, e)

$v' \leftarrow$ opposite node of v on e.
if s intersects clearance disk of v' and v' not marked and $v' \in \mathcal{CI}(s)$ **then**
  mark node v'
  for incident edge e' to v' with $e' \neq e$ **do**
    RecursiveMarknodes(s, v', e')
  end for
end if

Algorithm 4 RecursiveUpdate(s, v, e)

$v' \leftarrow$ opposite node of v on e.
if v' is marked **then**
  remove edge e
  for incident edge e' to v' with $e' \neq e$ **do**
    RecursiveUpdate(s, v', e')
  end for
else
  $s_1 \leftarrow$ site left of edge e
  $s_2 \leftarrow$ site right of edge e
  $v^* \leftarrow$ calc node equidistant to s, s_1, s_2 on the old edge e.
  replace v by v^* for edge e
end if

and tries to find any more or less usable solution. Furthermore, VRONI searches for possible intersections of input sites. At first locally and if nothing is found globally. If any intersections have been found, then VRONI splits the input sites at their intersection and restarts from scratch.

An intensive discussion of these issues can be found in [Hel01], "Reliability issues". In particular, the concept of "epsilon relaxation" and "desperate mode" is explained there.

- For a faster neighbor searching (e.g. when searching for the Voronoi cell hit) geometric hashing has been applied. See [Hel01] for details.

A bigger example for inserting an arcs s in the Voronoi Diagram of points, segments and arcs is illustrated in Figure 2.4. In the following sections of this chapter we discuss the insertion of segments and arcs. In particular we discuss the tree structure of marked edges and how to handle situations where a cycle of edges is marked. After that we consider the determination of seed nodes. The computation of the new nodes is described in chapter 3.

2.2 Tree structure of edges removed

We want to examine the structure of the removed edges in Algorithm 1. We will show that these edges have to form a tree. Therefore, the question arises, whether Algorithm 1 indeed removes edges which form a tree. Recall that we denote by $S^+ = S \cup \{s\}$ a proper set of input sites, with an arc $s \notin S$, if not declared differently.

Lemma 2.2. *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 forms a path from v_1 to v_2.*

Proof. Since $\mathcal{VC}(s, S^+)$ is connected, there exists a Jordan-curve $\gamma : [0, 1] \to \mathcal{VC}(s, S^+)$, beginning at v_1 and ending at v_2. We denote by γ^* the image $\gamma([0, 1])$. The image γ^* intersects on its way from v_1 to v_2 one or more Voronoi cells of $\mathcal{VD}(S)$. Suppose that γ^* intersects every edge of$\mathcal{VD}(S)$ at moste once (otherwise the proof works virtually the same).

Let us denote the intersected cells with $\mathcal{VC}(s_1, S), \ldots, \mathcal{VC}(s_k, S)$. We say a part of an edge is *cut off* if the projection segments of this part to the corresponding site cross γ^*. The edges which are (possibly partly) cut off from $\mathcal{VC}(s_j, S)$ by γ^*, are successively denoted by $e^j_1, \ldots, e^j_{l_j}$ as we walk on γ, for $j \in \{1, \ldots, k\}$; see Figure 2.5.

Then we know that these edges e^j_i which are completely cut off, lie completely in the future Voronoi cell $\mathcal{VC}(s, S^+)$, since the projection segments of points on the

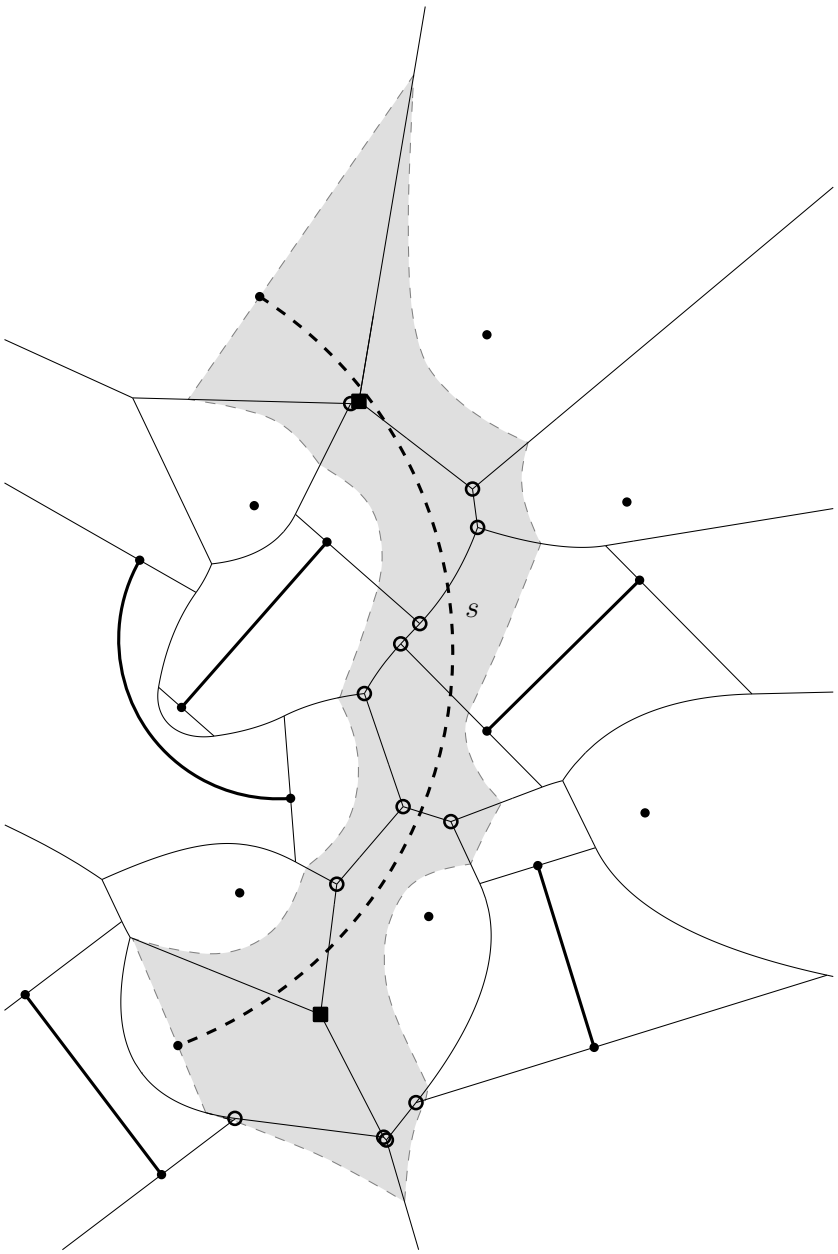

Figure 2.4: Inserting an arc s (dashed arc) in a Voronoi diagram. The marked nodes are labeled by a circle, the seed nodes are labeled by solid boxes. The new Voronoi cell of s is illustrated as the shaded area limited by the dashed lines.

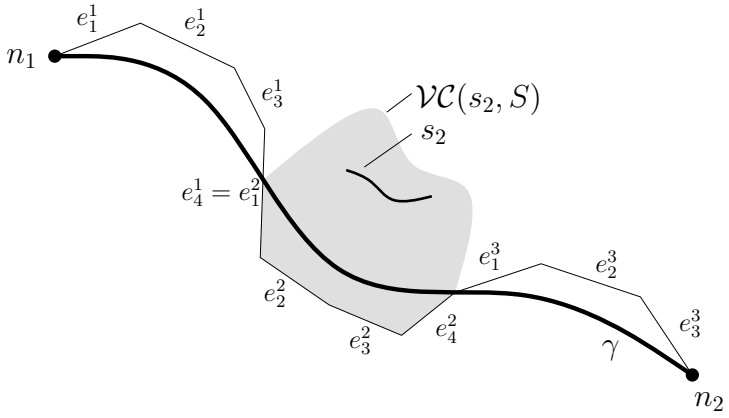

Figure 2.5: The curve γ crosses Voronoi cells of $\mathcal{VD}(S)$. We consider the edges cut off by γ^*. As an example, the shaded area is the Voronoi cell of s_2. The edges $e_1^2, \ldots, e_4^2$ are cut off by γ^*.

edges cross γ^*. If an edge $e_{l_j}^j$ is partly cut off then the edge e_1^{j+1} is partly cut off, too, and the two parts together yield the whole edge $e_{l_j}^j = e_1^{j+1}$. These edges lie completely in $\mathcal{VC}(s, S^+)$ as well due to the previous argument. So we get a whole path of edges of $\mathcal{VD}(S)$, from v_1 to v_2, and lying completely in $\mathcal{VC}(s, S^+)$.

□

Lemma 2.3. *Let S be a proper set of input sites and C 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 consists of a natural number of Voronoi cells. If A would contain only a part of a Voronoi cell there would be an edge through the interior of the cell and there would be points on the bisector which are strictly closer to the corresponding site. We conclude that if A contains a single point $p \in \operatorname{int} \mathcal{VC}(s', S)$, then A contains the whole Voronoi cell $\mathcal{VC}(s', S)$, for $s' \in S$. We take any edge e of C (such that e is not degenerated to a single point) and denote by s and t the sites defining the edge e.

Since C contains e, either points of $\operatorname{int} \mathcal{VC}(s)$ or points of $\operatorname{int} \mathcal{VC}(t)$ are members of A. Therefore, A contains the whole Voronoi cell of s or t, in particular the corresponding site itself. □

Theorem 2.4. *Let $S^+ = S \cup \{s\}$ be a proper set of input sites, with a site $s \notin S$. We denote by T the graph arising from the edges e from $\mathcal{VD}(S)$ which completely lie in $\mathcal{VC}(s, S^+)$, but do not intersect with $(\operatorname{cl} s) \setminus s$. Then T forms a tree.*

Proof. First, we see that T is connected, since two nodes of T can be connected by path due to Lemma 2.2. Assume that T contains a cycle C and A is the area enclosed by C. Then A contains at least one site $s' \in S$ i.e., $s' \subseteq A$. Since A is a subset of $\mathcal{VC}(s, S^+)$ because Voronoi cells are simply connected we conclude that $s' \subseteq \mathcal{VC}(s, S^+)$.

The only case where this does not lead to a contradiction is that s is an arc or segment, and s' is one of its end points. But since we excluded edges intersecting $(\operatorname{cl} s) \setminus s$ by assumption, there can not be a cycle. □

Let us now examine the deletion of edges (Alg.-1) in more detail. Edges are deleted if both end-nodes are marked. We refer to these edges also as *marked edges*. It is obvious that all edges in the tree T which are completely removed are marked edges. Unfortunately, this does not hold conversely (see the examples later). In other words, there are edges where both end-points are marked by the algorithm, even though they should not be removed completely. Assume the graph of marked edges contains a cycle, then an edge has been marked which should be partly preserved, thus reaches outside the future Voronoi cell $\mathcal{VC}(s, S^+)$. (The other direction is also true and shown in the proof of Theorem 2.5.)

In [Hel01] we can find two cases which must be handled to break up accidentally created loops. These two cases are referred as "Preserving Voronoi regions" and "Breaking up cycles". We make the following very similar distinction: firstly, an marked edge could reach outside of the cone of influence of s. Secondly, an edge could be marked, although e is completely contained in $\mathcal{CI}(s)$, anyhow a portion of this edge has to be preserved. Thus, there is at least one point on this edge whose clearance disk is not violated by s. In the following we discuss these cases (including arcs as input sites). The idea to fix these situations is always the same: we find a point on the pathological edge which is outside the cone of influence of s or whose clearance disk is not violated by s and split the edge at this point by inserting a degree-two node. By doing so we have broken up the cycle in the graph of marked edges since the splitting node must not be marked.

2.2.1 Breaking up cycles outside of CI

In Figure 2.6 an example is shown where an arc is inserted in a Voronoi diagram and a cycle has been marked. A first look on this example shows that a split of edges at their apices[4] would solve this problem. Since this splitting has additional nice side-effects (described later) we split every edge at its apex, if the apex is in its relative interior. Another example is listed in [Hel01] for inserting a segment. The

[4]Straight-line edges separating two input points are split where the line connecting the points intersects the edge. Elliptic edges are split where the ("longer") main axis of the ellipse intersects the edge.

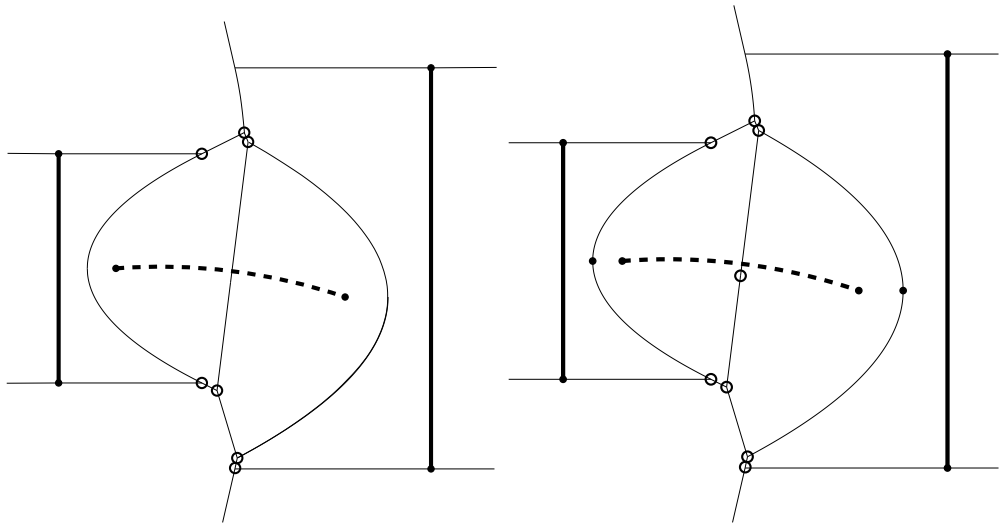

Figure 2.6: An example where parts of two edges must be preserved when inserting the dashed arc, because the edges reach outside the CI. Left: Before inserting degree-two node at apex. Right: After inserting degree-two nodes.

question is whether this precaution solves all these problems which are discussed in this section. In other words, can it happen that both end points of an edge are marked even though e reaches outside of $\mathcal{CI}(s)$? Unfortunately it can. In Figure 2.7 an example is illustrated. In the following we describe these pathological examples and find a solution for a break-up of the corresponding edges. Note that no edge contains the apex in the relative interior.

Constructing pathological examples

We can find all pathological examples by the following construction schema (cf. Figure 2.7). Let us assume that the edge e is not degenerated[5] and is on a hyperbolic bisector between two distinct circles, such that s_1 and s_2 are the two arcs on these circles having e on their Voronoi polygons. Let s_1 be the site whose Voronoi cell contains the secants of e and s_2 the other. Since $\operatorname{relint} e$ does not contain the apex, the clearance of the points of e changes monotonically on e. By v_1 we denote the node of e with the smaller clearance and the other by v_2. We want to place a segment or arc s in this figure such that v_1 and v_2 are marked and $e \not\subseteq \mathcal{CI}(s)$.

Keep in mind that the end points of s have already been inserted. In particular, the end points of s can not be in the interior of the clearance disks $\mathcal{CD}(v_1, S) \cup \mathcal{CD}(v_2, S)$. Let V be the union of projection segments when projecting each point

[5]Neither degenerated to a segment, nor to a single point.

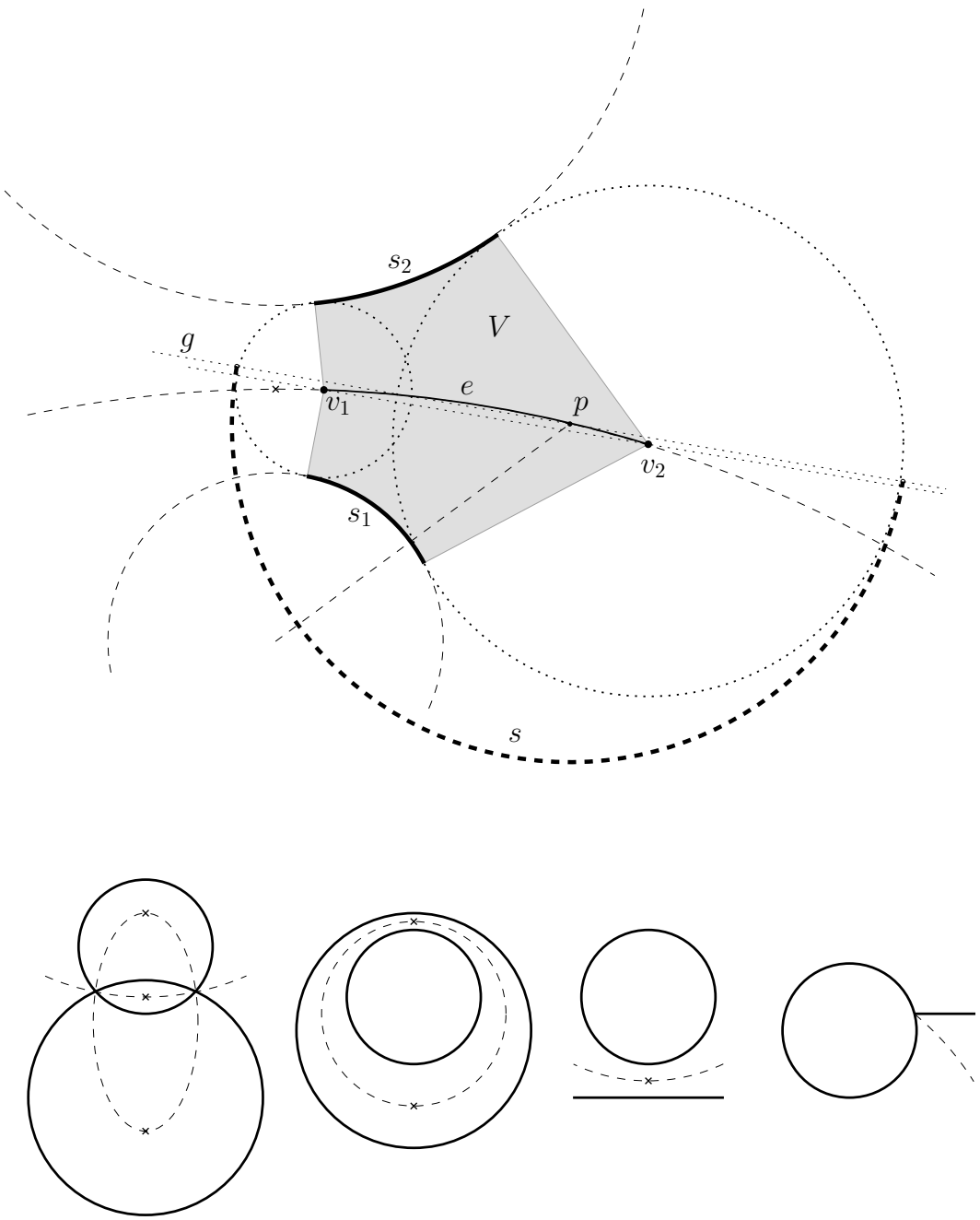

Figure 2.7: Top: first prototype-case where the end points v_1 and v_2 of the edge e are marked when inserting s, even though e reaches outside of $\mathcal{CI}(s)$ and should be partly preserved. Bottom: all equivalent cases of bisectors where the site s_1 – whose VC contains the secants of e – is an arc resp. a point. The apices are labeled by a cross.

of e on s_1 resp. s_2. Thus, $V \subseteq \mathcal{VC}(s_1, S) \cup \mathcal{VC}(s_2, S)$. Since e already exists, the end points of s cannot be in V. Therefore we search for arcs and segments s whose end points do not lie in $\text{int}(V \cup \mathcal{CD}(v_1, S) \cup \mathcal{CD}(v_2, S))$, even though $v_1, v_2 \in \mathcal{CI}(s)$ but $e \not\subseteq \mathcal{CI}(s)$.

These properties cannot be fulfilled by a straight-line segment s. Assume there is such a segment s. Then one of both boundary lines of $\mathcal{CI}(s)$ has to form a secant of e. Since $\mathcal{CD}(v_1, S)$ and $\mathcal{CD}(v_2, S)$ must be intersected by s, both points will be part of $\mathcal{VC}(s, S^+)$, as well as the whole projection segments of v_1, v_2 on s. But since a part of e remains as an edge between $\mathcal{VC}(s_1, S^+)$ and $\mathcal{VC}(s_2, S^+)$ there exists at least one point $p \in e$ which belongs to $\mathcal{VC}(s_1, S^+)$. But the projection segment of p on s_1 – which belongs to $\mathcal{VC}(s_1, S^+)$ too – intersects the projection segments of v_1 resp. v_2 on s and there would be points which would lie in the interior of different Voronoi cells $\mathcal{VC}(s_1, S^+)$ and $\mathcal{VC}(s, S^+)$. This is a contradiction.

So we search for arcs s fulfilling these properties. For arcs we can construct an example by taking a secant g of e and continuing it in both directions. The section $g \cap \text{int}(\mathcal{CD}(v_1, S) \cup \mathcal{CD}(v_2, S) \cup V)$ is connected, thus $g \setminus \text{int}(\mathcal{CD}(v_1, S) \cup \mathcal{CD}(v_2, S) \cup V)$ consists of two connected components. In both of these components we define an end point of s, such that an orthogonal line of g through every end point intersects the clearance disk $\mathcal{CD}(v_1, S)$ resp. $\mathcal{CD}(v_2, S)$. Then a semi-circle s, having these end points, fulfills all constraints wanted. It is clear that this picture can be easily modified to get arcs that are strictly smaller than a semi-circle, too.

This construction schema can be applied to all bisector forms shown in the bottom of Figure 2.7 (s_1 is an arc) and Figure 2.8 (s_1 is a segment). Furthermore, the cases where either s_1 or s_2 is a point are handled, too, by the corresponding cases for arcs whose radius is thought to be zero. In other words, for nearly every possible bisector-form there exists a pathological example. The only exceptions are bisectors in the form of a straight line[6]. In this case the marking of both end nodes implies the inclusion to $\mathcal{CI}(s)$ automatically.

Solving pathological examples

A simple approach to solve these pathological examples is to find a point $p \in e$ which is outside of $\mathcal{CI}(s)$ by a straight-forward method and split e at p. By doing so we construct a degree-two node which is surely not marked. However, this approach increases code complexity noticeably, since all possible bisector forms have to be taken into account. In the following we discuss a more convenient solution and distinguish between the cases where s_1 is an arc (cf. Figure 2.7) and s_1 is a segment (cf. Figure 2.8).

- Consider the prototype example in Figure 2.7. We intersect the line through

[6] Which means that s_1, s_2 being both segments or points or s_1, s_2 being arcs with equal radii.

the centers of s and s_1 with $b(s_1, s_2)$ and denote the point[7] by p. Assume that p is indeed on e. (We will show this right after.) We can distinguish two cases:

- $p \notin \mathcal{CI}(s)$: Then we found a point outside of $\mathcal{CI}(s)$ and p is a proper split-point on e.
- $p \in \mathcal{CI}(s)$: Let c be the center of s and r be its radius. Firstly, we note that $s_1 \subseteq \mathcal{CI}(s) \cap D(c, r)$. Otherwise $\mathcal{VC}(s_1, S^+)$ would not be connected (nor generalized star-shaped), since points on e have to remain for $\mathcal{VC}(s_1, S^+)$ and the projection lines to s_1 would cross s. Secondly, the line we constructed is the projection of p on s and since $s_1 \subset \mathcal{CI}(s) \cap D(c, r)$ we conclude that $d(p, s_1) < d(p, s)$. Therefore $p \notin \mathcal{VC}(s, S^+)$, thus p is again a proper split-point on e.

The question which remains is whether p is always on e. Assume we got a split-point $p' \notin \mathcal{CI}(s)$ somehow and the algorithm continues. Then two Voronoi nodes are calculated: a node $m_1 \in e$ between v_1 and p', and a node $m_2 \in e$ between v_2 and p'. Thus, the section of e between m_1 and m_2 is preserved as an edge between s_1 and s_2 and the other two parts of e lie in $\mathcal{VC}(s, S^+)$. But if the constructed point p would not lie on e between m_1 and m_2, then p would be in the interior of $\mathcal{VC}(s, S^+)$ and would be at the same time closer to s_1 than to s, which is a contradiction.

The argumentation above makes no distinction of the type of input site of s_2 and therefore can be applied to all bisector forms listed in the bottom of Figure 2.7. Furthermore, the argumentation can be again extended to s_1 being a point instead of an arc.

- Consider the example in Figure 2.8. We can not apply the approach outlined above since s_1 is a segment and has therefore no center. Instead, we project the center of s on the supporting line of s_1 and denote the projection point by q. Furthermore, we denote by p the intersection of this projection line with $b(s_1, s_2)$.

  We assume that q is on s_1 and show that later. If $p \notin \mathcal{CI}(s)$ then p is a proper split point on e. If $p \in \mathcal{CI}(s)$ then $d(p, s_1) < d(p, s)$ since the line $\overline{pq}$ is the projection line of p on s_1 and p is again a proper split point on e.

  The fact that q must be on s_1 can be seen as described in the previous case: If q is not on s_1, then the center c of s would not be in $\mathcal{CI}(s_1)$. Let a be the end point of s_1 such that $v_2 \in b(s_1, a)$ and let b be the other end

[7] There is only one such point on the half of the bisector (apex-splitting) on which e lies since the center of s_1 is a focal point of $b(s_1, s_2)$.

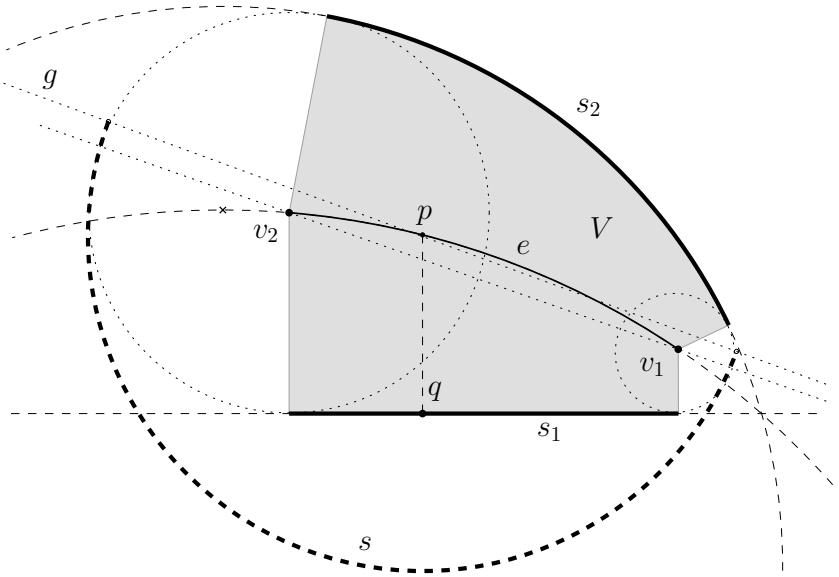

Figure 2.8: Second prototype-case where the end points v_1 and v_2 of the edge e are marked when inserting s, even though e reaches outside of $\mathcal{CI}(s)$ and should be partly preserved. Here s_1 is a segment.

point. W.l.o.g., we assume that c is in the half-space $H(a, a-b)$. Then we conclude that $d(v_2, a) < d(v_2, s)$, which is a contradiction since v_2 has been marked: We project v_2 on s and denote the intersection of the projection line with the supporting line of s_1 with q'. Then the three points v_2, a, q' form a right-angled triangle and triangle-inequality shows the statement.

Summary

In Section 2.2.1 we demonstrated that there are examples, where the algorithm marks an edge, even though the edge should partly remain. We further showed that with the introduction of circular arcs, the splitting at apices does not suffice to solve these cases. After giving a construction scheme for all pathological examples we discussed how to solve this problems by splitting the corresponding edge at a proper split point. The split point is found by projecting the center of the inserted arc on the site whose Voronoi cell contains the secants of the pathological edge.

Note on apex-splitting and breaking up long edges

A nice side-effect of splitting edges at their apices is that the clearance of points on edges is monotonically increasing or decreasing when moving from one end of the

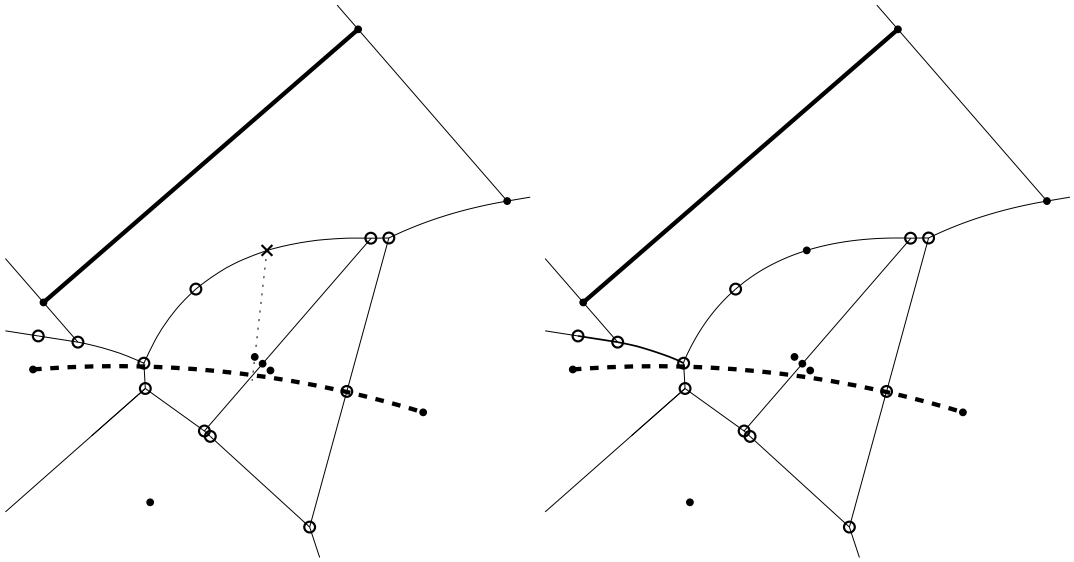

Figure 2.9: Example where a part of an edge must be preserved when inserting the dashed arc – inside CI. Left: The cross on the edge is closer to the point than to the arc. Right: After breaking up the loop we obtain a tree of edges to be deleted.

edge to the other. This is an important property when calculating offset curves based on Voronoi diagrams. Furthermore, the splitting simplifies the computation of the points on the edges for a graphical user interface output.

Beside apex-splitting we split edges between a segment/arc and its end points if the edge contains the end point in its interior. In other words, we split these edges extending on "both sides of the cone of influence" of the segment/arc. We refer to these edges as "long edges". This is necessary for offsetting because we need offset-monotonic edges for that.

2.2.2 Breaking up cycles inside of CI

We discuss situations where Alg.-1 marks a whole cycle C of edges. In Section 2.2.1 we assumed that C contained an edge which partly extended outside of $\mathcal{CI}(s)$. In this section we assume that C is completely in $\mathcal{CI}(s)$. Thus, there exists an edge e which contains a point p for which $d(p, \bigcup S) < d(p, s)$ holds. Hence, some portion of e should be preserved, even though the end points of e have been marked, thus are closer to s than to $\bigcup S$.

At first sight, it is unclear if this situation even exists at all. An example is given in Figure 2.9. The corresponding example for segments can be found in [Hel01]. First of all, we discuss the insertion of segments into a Voronoi diagram

of points and segments. And after that we will discuss the insertion of arcs in a Voronoi diagram of points, segments and arcs.

Inserting a segment

This situation is described in [Hel01], "Breaking up cycles". When replacing the arc in Figure 2.9 by a proper segment a cycle of Voronoi edges is marked. Suppose that we have marked a whole cycle C which encloses the area $A \subseteq \mathbb{R}^2$. As described in [Hel01] we scan all edges e and search for points $s' \in S$ which are contained in A and define e. If the projection line, when projecting s' on s, is continued to a straight line and intersects e at the point p, then p is closer to s' than to s. Thus, we can split e and have therefore broken up the cycle C.

The question arises whether such a point s' always exists. Since there must be an edge e which is partly preserved we can ask: can it be that the site $s' \in A$, defining the edge e to be broken up, is a segment? The answer is no. Consider a cycle C of marked edges and an edge e which should be partly preserved. Let us assume that e is defined by the segment $s' \subseteq A$. Let us denote by $p \in e$ a point for which $d(p, s') < d(p, s)$ holds. Then s has to overlap with the clearance disks of both end points v_1, v_2 of e but not the clearance disk of p. Figure 2.10 illustrates this situation. Then we can project one of both end points of s' on s and the projection line continued to a straight line intersects C at least two times. There is an intersection q which is clearly closer to s' than to s. Therefore C could be split into at least two parts[8]. This is a contradiction since the graph of marked edges must finally result in a tree and therefore has to be connected.

To sum up, if we detect a cycle C of marked edges we check all edges e of C which are defined by point $s' \in S$ enclosed by C and make sure that the projection line of s' on s does not intersect with e (when continuing the line to a straight line). If it does, we split e at the intersection point and the cycle C is broken up.

Inserting an arc

Assume we are inserting an arcs s and Alg.-1 marked edges resulting in a cycle C. Again we want to find the edge e which should be split up. Unfortunately we can not restrict the searching for the corresponding sites $s' \in S$ to points. In Figure 2.11 an example is constructed, where the site $s' \in A$, defining the edge e to be broken up, is a segment.

[8] Actually there could be other cycles too, holding the two parts together. But then we could proceed these cycles before the current one, such that the current one is the last cycle in the graph of marked edges and the argument holds again. So if there would by any cycle which would be split in two parts we could reorder the proceedings such that this cycle is processed at last.

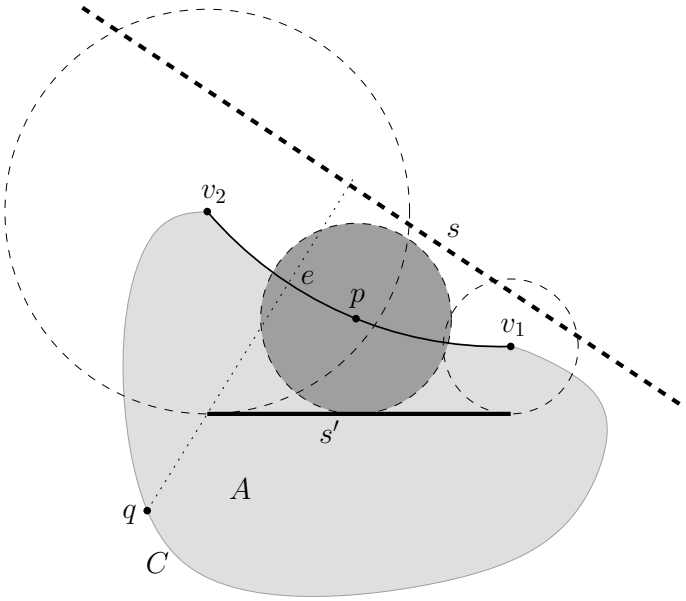

Figure 2.10: If a segment s' would define the edge e which should be partly preserved then C would disintegrate into at least two parts.

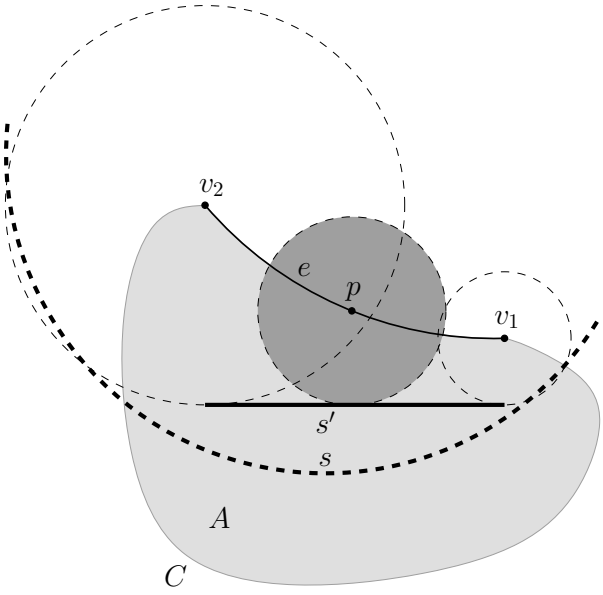

Figure 2.11: The defining site s' of the edge e in the cycle C is a segment. The arc s has to be inserted.

So, let us assume we insert the arc s and there is a cycle C of marked edges. Let $A \subseteq \mathbb{R}^2$ denote the area enclosed by C. There is an edge e of C which should be partly preserved. We denote by $s' \in A$ the site defining e, with $s \subseteq A$, such that e contains a point $p \in e$, with $d(p, s') < d(p, s)$. Note that $A \subseteq \mathcal{CI}(s)$ holds.

We now know that s can not lie in an arbitrary position: firstly, s has to intersect $\mathcal{CD}(v_1, S)$ and $\mathcal{CD}(v_2, S)$ where v_1, v_2 are the end points of e. Secondly, the end points of s are not allowed to intersect $\mathcal{CD}(v_1, S)$, $\mathcal{CD}(v_2, S)$, or any other clearance disk of a point of e. Thirdly, s must not lie within $\mathcal{CD}(p, S)$. Fourthly, we consider any point $q \in s'$ and the corresponding projection of q on s. The continued projection line intersects C. Then any intersection closer to s' than to s must lie on e. Otherwise C could be split in two disjoint parts which is a contradiction since the graph of marked edges must finally result in a tree. Very roughly speaking, the arc s has to "surround" the site s'. By the way, this is the reason why there are no segments s fulfilling the constraints.

The recipe for finding the point $p \in e$ is the same as in Section 2.2.1. We project the center of s on s' and the intersection of the projection line with e is denoted by p. The argumentation that p exists is the same, too.

2.2.3 Final routine for breaking up cycles

The complete routine for breaking up cycles when inserting an arcs s is listed in Alg.-5. The ideas have been collected from Section 2.2.1 and Section 2.2.2. The following notes give further details:

- Theory tells us that for every cycle there is an edge e which can be (and has to be) split such that the cycle is broken up. If e is the edge, then the corresponding intersections lead to a proper split point. If we test all edges e in C and check if there is an intersection p with e and its clearing disk $\mathcal{CD}(p, S)$ is not intersected by s, we will locate a proper split point.

- Note that we do not check whether the solution is outside of $\mathcal{CI}(s)$. Remember the argumentation in Section 2.2.1 that in both cases – $p \in \mathcal{CI}(s)$ resp. $p \notin \mathcal{CI}(s)$ – p is a proper split point, assuming e is the correct edge. Therefore we do not check for containment in the cone of influence.

- In the first instance, we discuss the extension of VRONI to circular arcs. However, the algorithm Alg.-5 holds for segments, too. The difference is that the branch for s' being an arc is never used and the branch for s' being a segment has to be ignored.

Algorithm 5 Tree'ify graph T of marked edges when inserting arc s

```
 1: c ← center of s
 2:
 3: while ∃ cycle C in T do
 4:   for edge e in C do
 5:     s' ← site of e enclosed by C
 6:
 7:     if s' is a point then
 8:       c' ← s'
 9:     else if s' is an arc then
10:       c' ← center of s'
11:     else
12:       c' ← projection of c on s'
13:     end if
14:
15:     if c'c ∩ e ≠ ∅ then
16:       p ← c'c ∩ e
17:       if d(p, S) < d(p, s) then
18:         split e at p
19:         leave for-loop
20:       end if
21:     end if
22:   end for
23: end while
```

Theorem 2.5 (Correctness of edge-marking algorithm)**.** *Let S be a proper set of input sites and $S^+ := S \cup \{s\}$ for a segment or arc $s \notin S$. The algorithms Alg.-3 and Alg.-5 together mark an edge of $\mathcal{VD}(S)$ iff the edge completely lies in the future Voronoi cell $\mathcal{VC}(s, S^+)$.*

Proof. The algorithm Alg.-3 marks all edges of $\mathcal{VD}(S)$ which completely lie in $\mathcal{VC}(s, S^+)$ and under certain circumstances more. These marked edges where both end-nodes are marked but the edge does not completely lie in $\mathcal{VC}(s, S^+)$ lead to a cycle in the edge-graph:

If we split these edges at a proper split-point we get two nodes, the end-nodes of the edge, which belong to the future cell $\mathcal{VC}(s, S^+)$ and are therefore connected by a path of edges of $\mathcal{VD}(S)$ due to Lemma 2.2. So there has been a cycle before the break-up.

We search for cycles and if we found one there are two possibilities discussed in Section 2.2.1 and Section 2.2.2. Both cases lead to the breakup strategy stated above. □

2.3 Selecting a seed node

In this section we discuss the selection of the seed node in Alg.-2 when inserting a segment or an arc s with end points $p_1, p_2 \in S$. We start with discussing the necessary constraints for selecting the seed node and after that we discuss the steps to fulfill these constraints. Again, we will see that the introduction of arcs complicates one or the other issue. However, we

- select a seed node on both Voronoi polygons $\mathcal{VP}(p_1, S)$ and $\mathcal{VP}(p_2, S)$ of the end points of s,
- check that the tree T of marked edges reaches both seed nodes, and
- select (if possible) a seed node shared by both Voronoi polygons.

If there is an edge e which is shared by the Voronoi polygons of both end points of s then the first and second constraint above are fulfilled automatically. Thus, we assume in the following that there is no such shared node. In general, we select the Voronoi node $v \in \mathcal{VP}(p_1, S)$ resp. $v \in \mathcal{VP}(p_2, S)$, such that $v \in \mathcal{CI}(s)$. In most cases there are more than one candidate, therefore we take the one whose clearance disk $\mathcal{CD}(v, \bigcup S)$ is violated most by s. That is, where $d(v, S) - d(v, s)$ is minimized.

However, in some cases the selection of seed nodes is dictated by the arrangement of already inserted input sites in S. The problem arises from the fact that

we do not mark nodes if they coincide with an input point from S. This will be discussed later in Section 2.3.2. The reason is that Voronoi nodes whose Voronoi cells are collapsed to a single point would be completely removed. Here geometric properties do not suffice – we need graph-theoretical considerations. Furthermore, we used the property that nodes coinciding with input points are not marked, in the proof of Theorem 2.4. However, at first the question arises whether a seed node always exists.

Lemma 2.6 (Existance of seed node)**.** *Let S be a proper set of input sites and $S^+ := S \cup \{s\}$ for a segment or arc $s \notin S$. Furthermore, let $p \in S$ be an end point of s and recall that no edge $e \in \mathcal{VD}(s)$ contains the apex in its relative interior. Then there exists a node $v \in \mathcal{VP}(p, S)$ such that $v \in \mathcal{CI}(s)$, hence $d(v, s) \leq d(v, p)$.*

Proof. For every node $v \in \mathcal{VP}(p, S) \cap \mathcal{CI}(s)$ the property $d(v, s) \leq d(v, p)$ holds and for $v \in \mathcal{VP}(p, S) \cap \operatorname{int} \mathcal{CI}(s)$ even the strict inequality $d(v, s) < d(v, p)$. In other words, a seed node exists iff there exists a node $v \in \mathcal{VP}(p, S)$ for which $v \in \mathcal{CI}(s)$ holds.

From now on we talk about nodes $v \in \mathcal{VP}(p, S)$. Let us assume there is no such node $v \in \mathcal{CI}(s)$. Then there exists an edge e whose end points are outside of $\mathcal{CI}(s)$, even though $e \cap \mathcal{CI}(s) \neq \emptyset$. Otherwise p would not lie in its Voronoi cell. So e has to be a non-degenerate[9] parabolic, hyperbolic or elliptic edge with p as a focal point. We are asking for $v \in \mathbb{R}^2$ such that the intersection of the half-space $H(p, v)$ with an parabolic, hyperbolic or elliptic bisector results in a continuous piece of e, touching the straight line $\operatorname{bd} H$. This the case iff this piece contains the apex in its interior. Contradiction. □

As we saw in the proof, there is always a node $v \in \mathcal{VP}(p, S)$ such that v is also in $\mathcal{CI}(s)$. We have to clarify now if indeed every node $v \in \mathcal{VP}(p, S) \cap \mathcal{CI}(s)$ is a potential candidate for being a seed node. Let us assume that there exists a node $v \in \operatorname{int} \mathcal{CI}(s)$ and we select v as seed node. We can move along the edges of $\mathcal{VP}(p, S)$ and every edge lying completely in $\mathcal{CI}(s)$ will be marked by Alg.-3. And therefore every further potential seed node on $\mathcal{VP}(p, S)$ is marked too. So in the case where a node $v \in \operatorname{int} \mathcal{CI}(s)$ exists, every node $v \in \mathcal{CI}(s) \cap \mathcal{VP}(p, S)$ can be used as the seed node and we take the one whose clearance disk is violated most.

2.3.1 Handling tangential sites

Let us now assume that there is no such node $v \in \operatorname{int} \mathcal{CI}(s)$. Thus, all potential seed nodes v are on the boundary straight line $\operatorname{bd} \mathcal{CI}(s)$. If the Voronoi cell $\mathcal{VC}(p, S)$ is collapsed to a single point, one or more segments or arcs are already incident to p.

[9] Not degenerate to a single point or a straight line.

This case is discussed in Section 2.3.2. We assume that $\mathcal{VC}(p, S)$ is not collapsed to a single point. In the following we distinguish two cases:

- We consider the insertion of segments s. Assume that all nodes $v \in \mathcal{VP}(p, S) \cap \mathcal{CI}(S)$ lie on the boundary $\operatorname{bd}\mathcal{CI}(s)$. Then, there exists already a segment $s' \in S$ which is incident to p such that the potential seed nodes lie on $\mathcal{VP}(p, S) \cap \mathcal{VP}(s', S)$. But this is only the case if s' and s overlap, which does not comply with a proper set of input sites. In other words, when inserting a segment s any node $v \in \mathcal{VP}(p, S) \cap \mathcal{CI}(s)$ works as seed node, as long as $\mathcal{VP}(p, S)$ is not collapsed to a single point.

- We consider the insertion of arcs s. Again, we can argue that arcs and segments $s_1, \ldots, s_m \in S$ are already incident to p, such that $s_1, \ldots, s_m$ are tangential in the point p and $s_1, \ldots, s_m$ lie on the same half-space $H(p, v)$, with $v \in \mathbb{R}^2$ pointing in the tangential direction of s from p. Furthermore, there can only be one segment with this property, otherwise two segments would overlap. We consider the situation from the point of view of p in direction of v. There is an edge on the "left side" and an edge on the "right side", cf. Figure 2.12. We have to choose a seed node on either the one or the other edge, not coinciding with p. If we decide to choose one side, Alg.-3 does not reach the other side since nodes coinciding with points are not marked.

  We have to make the correct choice. We denote by e_1 the edge on the same side as the center of s and with e_2 the other. Furthermore, let s_1 be the site where e_1 belongs to its Voronoi polygon and let s_2 be the site to which e_2 belongs. Note, that s_1 and s_2 can denote the same site if only one such site is incident to p. The following rules are applied to determine the correct seed node:

  - If s_1 is an arc and the center of s_1 is on the side of e_1 and the radius of s_1 is bigger than the radius of s, then we take the node on e_1. Same holds if the center of s_1 is on the side of e_2 or if s_1 is a segment.
  - If s_2 is an arc and the center of s_2 is on the side of e_1 and the radius is smaller than the radius of s, we take the node on e_2.
  - If none of these two rules is applied, neither the node on e_1 nor the node on e_2 is a proper seed node. This issue is described in Section 2.3.2.

2.3.2 Handling spikes

As we saw in the last section, there can be situations where a geometrical decision for the selection of the seed node does not suffice. Especially when the Voronoi

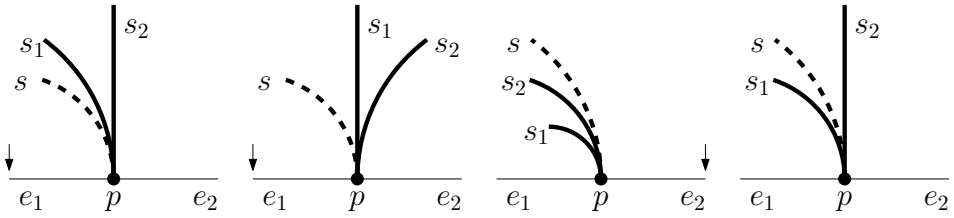

Figure 2.12: Selecting the seed node when at least one other site $(s_1, s_2, \ldots, s_m)$ tangential to s is already incident to p. The edge e_1 is the edge on the side of the center of s, e_2 the other edge. The site s_1 is the site to which e_1 belongs, s_2 the site to which e_2 belongs. The arrow indicates the side on which we look for a seed node.

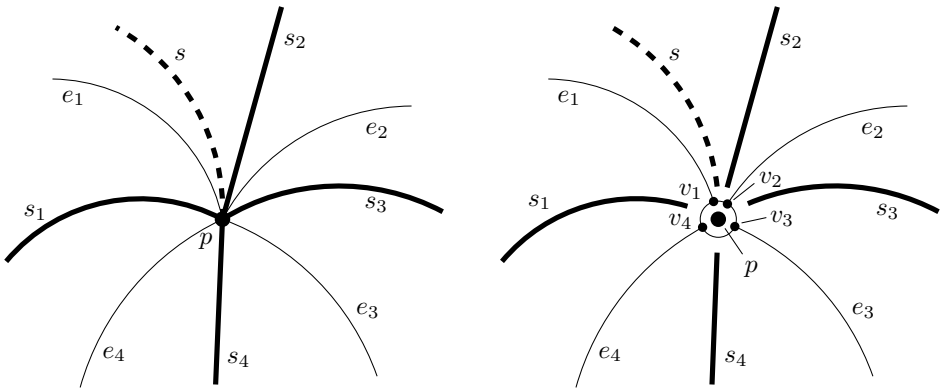

Figure 2.13: Several arcs and segments meeting at a common end point p. Left: Geometrical view with sites drawn as thick curves. Right: Topological, graph theoretical view.

cell $\mathcal{VC}(p, S)$ of the end point p is collapsed to a single point. This can be the case when three or more segments or arcs meet in the common end point p.

The reason why geometrical predicates fail here is that all nodes $v \in \mathcal{VP}(p, S)$ have distance zero to the segment or arc s which is to be inserted. In [Hel01] we can find in the section "Handling spikes" the corresponding strategy when inserting segments. The same holds for arcs too. Let us denote by $v_1, v_2, \ldots v_k$ the nodes of $\mathcal{VP}(p, S)$ and with $e_1, \ldots, e_k$ the edges with one end point $v_j, 1 \leq j \leq k$ and another end point $w_j, 1 \leq j \leq k$ not coinciding with p. Furthermore let $s_1, \ldots, s_k$ be the sites incident to p such that e_l separates s_l and s_{l+1}, when defining $s_{k+1} := s_1$. Then we check for all nodes w_j whether their clearance disk $\mathcal{CD}(w_j, S)$ is intersected by s and select such a node as seed node. When identifying $w_{k+1} := w_1$ and s lies "between" s_l and s_{l+1} then w_l (and only w_l) fulfills this condition.

Note on tangential sites Note that here we have to be aware of tangential sites as well. If we consider Figure 2.12 and think of further tangential sites going downwards from p then the Voronoi cell of p can be collapsed to a single point. In this case we have to handle spikes and disregard certain nodes of edges lying on $\mathrm{bd}\,\mathcal{CI}(s)$. We do not further discuss these cases because they are exactly the same as in the last section.

2.4 Correctness and complexity

2.4.1 Correctness of the algorithm

Theorem 2.7. *Let us denote by S a proper set of input sites. Furthermore, let $s \notin S$ be a segment or arc and $S^+ := S \cup \{s\}$. Algorithm 1 computes the Voronoi diagram $\mathcal{VD}(S^+)$ by inserting s into $\mathcal{VD}(S)$ and adapting $\mathcal{VD}(S)$ accordingly.*

Proof. Let us recapitulate Alg.-1 when inserting a segment or an arc s. The construction of the Voronoi diagram of the points is well known and not discussed here. At first we select a proper seed node and we showed that there always exists such a node. We discussed in Section 2.3 the different cases to find a possible one. Actually, we select a seed node on the Voronoi cells of both end points. Both seed nodes are in the new Voronoi cell $\mathcal{VC}(s, S^+)$ by construction. After that we recursively mark nodes whose clearance disk has been violated by s, beginning with one of both seed nodes.

Theorem 2.5 showed that Alg.-1 and Alg.-5 together mark these and only these edges[10] which completely lie in $\mathcal{VC}(s, S^+)$. The corresponding edge-graph is denoted by T and is a tree. All edges in T have to be deleted (and, of course, the corresponding nodes). Furthermore, we check whether both seed nodes have been marked. (Otherwise something went wrong due to numerical inaccuracy.)

The edges where exactly one end-node is marked are the edges where a new Voronoi node has to be determined on, because $\mathcal{VP}(s, S^+)$ intersects these edges. The new nodes for these edges are defined by the defining sites of the edge and the new site s. The new Voronoi cell $\mathcal{VD}(s, S^+)$ is spanned by the newly calculated Voronoi nodes. The edges defining $\mathcal{VP}(s, S^+)$ can be obtained by walking (counter) clock-wise on the leafs of the tree T and successively connecting newly created nodes by edges, as it is done by the pre-genuine-arc algorithm too. By doing so, we obtain a new generalized star-shaped Voronoi cell $\mathcal{VC}(s, S^+)$. □

[10] Recall that marked edges are edges whose end-nodes are marked.

The topological constraints

As mentioned in the beginning of this chapter, VRONI implements an algorithm using topological constraints. Therefore VRONI preserves topological and graph-theoretical properties when inserting a new input site (see [Hel01]):

- Every site has its own Voronoi cell: In other words, it never happens that a whole Voronoi cell disappears since cycles in the edge graph of marked edges are broken up. The input site which is inserted gets its own Voronoi cell because at least the seed node is marked.
- Every Voronoi cell is connected: This holds since the edge-graph of marked edges is a tree, in particular connected.
- The Voronoi cell of a segment or an arc s is adjacent to the Voronoi cells of its end points: After inserting s this holds because the seed nodes are on the Voronoi polygons of the points. After that the property is preserved since nodes coinciding with input points are never marked.

2.4.2 Run time complexity

We denote by $S^+ = S \cup \{s\}$ a proper set of input sites, with $n := |S^+|$ the number of input sites of S^+ and with $s \notin S$ the site which is inserted into $\mathcal{VD}(S)$. Since VRONI implements an incremental algorithm we focus on the insertion procedure (for a segment or arc s), consisting of these four steps:

Determining seed node Finding an end point p of s is a single query, since the corresponding references are saved. In most cases the ordinary scan around the Voronoi polygon $\mathcal{VP}(p, S)$ is done in expected constant time, but in any case in $O(n)$. Recall that a Voronoi diagram holds $O(n)$ many edges and nodes.

In the case of sites meeting tangential in a common end point, we have to find the corresponding edges e_1 and e_2 of Section 2.3.1 by scanning $\mathcal{VP}(p, S)$ and the same holds in the case of spikes. This yields to a worst case complexity of $O(n)$ in the presence of tangential sites (or spikes).

Recursively marking nodes There is an upper bound of $O(n)$ for the number of nodes. The concrete number of marked nodes strongly depends on the "size" of s. If s extends within a significant portion of the bounding box and there are many other sites along of s, such that s crosses many Voronoi cells, a big number of nodes will be marked. But in real-world data, especially in polygonal shaped figures, this might not be the case for many sites.

Breaking up cycles An obvious upper bound of Alg.-5 is $O(n^2)$. However the construction of these pathological examples is tricky and it is very unlikely that the graph T from Alg.-3 contains two cycles or even more in real-world data.

Anyhow, we can modify Alg.-3 to have $O(n)$ complexity: Instead of looping over all cycles and after that looping over all edges in these cycles we could just loop over all marked edges. By doing so, we split the edges which have to be split and further splittings do not[11] occur.

The standard argument using Euler's formula yields an $O(n)$ complexity for the number of edges of $\mathcal{VD}(S)$. This complexity is not hurt by the introduction of edge splitting in Alg.-3 since these nodes could be removed afterwards. The apex splitting does not hurt the complexity either because the number of edges is at most doubled. To sum up, we obtain an $O(n)$ breaking up strategy by modifying Alg.-3, as mentioned above.

Recursive update of edges The recursive update of edges is in any cases done in $O(n)$ and again strongly depends on the "size" of s.

The worst-case complexity of VRONI is therefore $O(n^2)$, when considering the modifications mentioned above. However, since VRONI applies randomized insertion of input sites, we can investigate the expected run time complexity.

Expected run time complexity

We apply backward analysis on the incremental insertion of arcs as mentioned for similar problems in [Sei93]. Shortly speaking, we consider the randomized insertion and describe the complexity of an insertion by terms of its result. After that, we argue that all possible insertion sequences occur with the same probability and obtain an expression for the expectation of the complexity of an insertion. The detailed argumentation is given below. However, first of all, we need an expression for the complexity of the insertion of an arc in terms of the resulting Voronoi diagram.

Lemma 2.8. *Let $S^+ = S \cup \{s\}$ be a proper set of input sites with an arc (or segment) $s \notin S$. We denote by N the number of nodes and by E the number of edges of $\mathcal{VD}(S)$ which are marked by Alg.-3. Furthermore, we denote by L the number of edges of $\mathcal{VD}(S)$ which are adapted by Alg.-4. Then, the equations*

$$N = L - 2, \qquad E = L - 3$$

hold.

[11] They would not hurt the correctness anyway.

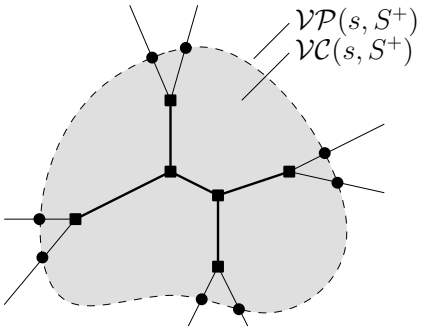

Figure 2.14: The graph structure of $\mathcal{VD}(S)$ in the future Voronoi cell $\mathcal{VC}(s, S^+)$. The tree T consists of bold edges. The tree T' consists of all edges restricted to $\mathcal{VC}(s, S^+)$. The number of nodes marked by boxes is N. The number of nodes marked by disks is L.

Proof. First of all, we recall that VRONI handles nodes of degree three (and nodes of degree two which can be ignored for complexity considerations as already mentioned). We consider the nodes and edges of $\mathcal{VD}(S)$ which are marked by Alg.-3. We already know, that this subgraph T has to form a tree. This observation leads to the equation

$$E = N - 1.$$

Furthermore, we consider the L edges of $\mathcal{VD}(S)$ which are adapted by Alg.-4 and restrict them to $\mathcal{VD}(s, S^+)$ for further considerations. We interpret them as L leaf-edges appended to the tree T and obtain an extended tree T', see Figure 2.14. Let us consider the number B of edges of $\mathcal{VP}(s, S^+)$. Since the number L of nodes of $\mathcal{VP}(s, S^+)$ is the number of edges of $\mathcal{VP}(s, S^+)$, we obtain $B = L$. If we add the edges of $\mathcal{VP}(s, S^+)$ to T' then we get a new planar graph where every node has degree three. The degree-three property leads to the equation

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

respectively to $2E + L = 3N$. The last equation combined with $E = N - 1$ implies the two equations to prove. □

Corollary 2.9. *Let $S^+ = S \cup \{s\}$ be a proper set of input sites with an arc (or segment) $s \notin S$. The complexity of Alg.-3, Alg.-5, and Alg.-4, when inserting s into $\mathcal{VD}(S)$, is linear in the complexity (of the number of nodes) of $\mathcal{VC}(s, S^+)$.*

Proof. The statement is trivial when modifying Alg.-5 as discussed in the beginning of Section 2.4.2, due to Lemma 2.8. □

Let us consider a proper set of input sites $S := \{p_1, \ldots, p_{n_1}, s_1, \ldots, s_{n_2}, a_1, \ldots, a_{n_3}\}$ with points p_i, segments s_j and arcs a_k, where $i \in \{1, \ldots, n_1\}, j \in \{1, \ldots, n_2\}, k \in \{1, \ldots, n_3\}$. Furthermore, let $S_m := S \setminus \{a_{m+1}, \ldots, a_{n_3}\}$, and $n := n_1 + n_2 + n_3$. Let us consider the insertion of the arc a_m into the Voronoi diagram of S_{m-1}. Since we apply randomized insertion we could ask for the expectation of the complexity of inserting the m-th arc among all possible insertion sequences. Let us denote by X the random variable of the complexity of inserting the arc a_m into S_{m-1}, when before randomly permuting $\{a_1, \ldots, a_m\}$. We denote by $C(s, S)$ the complexity of $\mathcal{VP}(s, S)$, i.e. the number of nodes of $\mathcal{VP}(s, S)$ in $\mathcal{VD}(S)$. Furthermore, we denote by p_1^k and p_2^k the endpoints of the arc a_k.

The complexity of Alg.-2 (finding a seed node in both endpoints of a_m), when inserting the arc a_m into $\mathcal{VD}(S_{m-1})$, is linear to $C(p_1^m, S_{m-1}) + C(p_2^m, S_{m-1})$ and we therefore obtain

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

Let us denote by c the maximal number of segments and arcs incident to a endpoint. The inequality

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

holds due to Lemma 2.8. Therefore, we conclude that

$$\begin{aligned} E(X) &\leq \frac{1}{m} \sum_{k=1}^{m} 3C(a_k, S_m) + C(p_1^k, S_m) + C(p_2^k, S_m) \\ &\leq \frac{1}{m} \left(3 \sum_{k=1}^{m} C(a_k, S_m) + 2c \sum_{i=1}^{n_3} C(p_i, S_m) \right) \\ &\leq \frac{3c}{m} \underbrace{\left(\sum_{k=1}^{m} C(a_k, S_m) + \sum_{i=1}^{n_3} C(p_i, S_m) \right)}_{\in O(n)} \end{aligned}$$

and we have shown that $E(X) \in O(\frac{nc}{m})$ holds. The expected complexity of inserting all arcs $a_1, \ldots, a_m$ is therefore

$$O\left(cn \cdot \sum_{m=1}^{n_3} \frac{1}{m} \right) \subseteq O(cn \log n).$$

Corollary 2.10. *If the maximal number of segments and arcs incident to a common endpoint is in $O(1)$ then the expected run time of* VRONI *is $O(n \log n)$.*

Chapter 3

Computation of Nodes

The computation of Voronoi nodes forms the numerical core of a Voronoi diagram algorithm. Since numerical accuracy and stability mainly depend on a robust computation of Voronoi nodes this chapter devotes the main attention to the numerical quality of the algorithm presented.

A new Voronoi node is determined by three input sites. The so-far implementation of VRONI deals with points and straight-line segments. Therefore the procedure for computing new Voronoi nodes has to deal with the following combinations[1] of input sites:

- Point-Point-Point,
- Segment-Point-Point,
- Segment-Segment-Point,
- Segment-Segment-Segment.

This thesis adds circular arcs as possible input site type. Therefore the number of combinations would rise to $\binom{3-3+1}{3} = 10$ combinations – in other words 6 cases have to be added. Firstly we are interested in determining all points equidistant to three input sites of a specific type. For this task a circular arc can be treated as a full circle and line segments can be treated as straight lines in the first instance. By doing so points can be seen as circles with zero radius as well. By this simplification the following cases have to be added in VRONI:

- Circle-Circle-Circle,
- Circle-Circle-Line,

[1] The number of combinations with repetition of k elements out of n types is given by $\binom{n-1+k}{k}$.

- Circle-Line-Line.

In the following, the computation of points equidistant to circles and lines is given. We will see that the solutions are far away of being unique. For situations of infinitely many solutions, special cases have to be introduced. Here the abstraction to full circles and straight lines goes too far. By doing so, we handle numerical bottlenecks too. The last section discusses how to select the correct solution out of many possible ones.

3.1 Determining equidistant points

The calculations in the following sections are based on a common idea of finding equidistant points to three sites by intersecting their offset sites. In [Hel91] (page 100-101) this idea is sketched. Due to a footnote in [Hel91] the origin of this approach can be traced back to Gàbor Lukács, Hasse Persson, Tamás Várady, and Martin Held.

Knowing these formulae was essential for implementing this extension of VRONI. Note that this approach works so nicely because the site type is invariant when building the offset sites. With other words: the offset site of circles and lines are circles and lines again. A rule of thumb for the numerical stability is: the more circles are involved, the less stable are the formulae.

3.1.1 Circle-Circle-Circle

We consider three circles centered at $c_1 = (x_1, y_1), c_2 = (x_2, y_2), c_3 = (y_3, y_3)$ with radii r_1, r_2, r_3, respectively. Recall the notation of a disk $D(.,.)$ in Section 1.2. The determination of the points equidistant to these three circles is based on the following idea:

Lemma 3.1. *A point p is equidistant to the three circles iff a common offset $t \geq 0$ and three constants $k_1, k_2, k_3 \in \{-1, 1\}$ exist such that the three offset circles with radii $r_j + k_j \cdot t$ for $j \in \{1, 2, 3\}$ meet at p.*

Proof. Let us assume a point $p = (p_x, p_y)$ is equidistant to the three circles with distance t. We define three constants

$$k_j := \begin{cases} -1 & \text{for } p \in D(c_j, r_j) \\ 1 & \text{otherwise} \end{cases}$$

for $j \in \{1, 2, 3\}$. Then the three offset circles centered at c_j with radii $r_j + k_j \cdot t$ meet at the point p, obviously. Vice versa, consider three offset circles with same

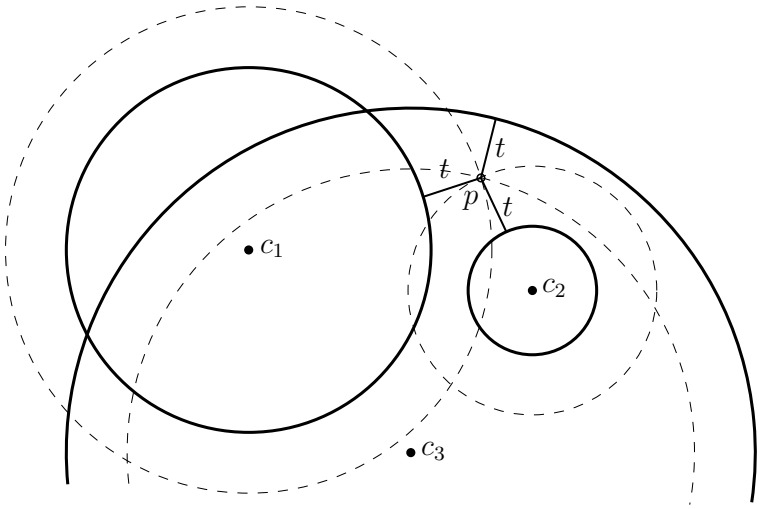

Figure 3.1: Offset circles intersect at the equidistant point p. The corresponding constants are $k_1 = 1, k_2 = 1, k_3 = -1$.

offset t (but with maybe different values for $k_1, k_2, k_3 \in \{-1, 1\}$) coinciding at a common point p. Then the point p is equidistant to each input circle.

□

Hence, we can determine all equidistant points to three circles by searching for offsets $t \geq 0$ and constants $k_1, k_2, k_3 \in \{-1, 1\}$ such that the offset circles with radii $r_j + k_j \cdot t$, for $j \in \{1, 2, 3\}$, meet in a common point. With other words, we search for solutions $t \geq 0$, $k_1, k_2, k_3 \in \{-1, 1\}$ and $(p_x, p_y) \in \mathbb{R}^2$ such that the system

$$\begin{aligned}
(p_x - x_1)^2 + (p_y - y_1)^2 &= (r_1 + k_1 t)^2 && (3.1)\\
(p_x - x_2)^2 + (p_y - y_2)^2 &= (r_2 + k_2 t)^2 && (3.2)\\
(p_x - x_3)^2 + (p_y - y_3)^2 &= (r_3 + k_3 t)^2 && (3.3)
\end{aligned}$$

is fulfilled. In general this system can not be solved uniquely – the number of solutions will be discussed later. Linear equations in p_x, p_y can be obtained by subtracting Eqn.-3.2 from Eqn.-3.1 and Eqn.-3.3. By using $k_1^2 = k_2^2 = k_3^2 = 1$ we get:

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

These equations can be solved easily for p_x, p_y. Note that the solutions depend only linearly on t:

$$p_x = \frac{a_0 + a_1 \cdot t}{d}, \tag{3.4}$$

$$p_y = \frac{b_0 + b_1 \cdot t}{d}, \tag{3.5}$$

where

$$\begin{aligned}
a_0 &:= y_1((x_3^2 + y_3^2 - r_3^2) - (x_2^2 + y_2^2 - r_2^2)) + \\
&\quad y_2((x_1^2 + y_1^2 - r_1^2) - (x_3^2 + y_3^2 - r_3^2)) + \\
&\quad y_3((x_2^2 + y_2^2 - r_2^2) - (x_1^2 + y_1^2 - r_1^2)), \\
a_1 &:= 2(k_1 r_1(y_3 - y_2) + k_2 r_2(y_1 - y_3) + k_3 r_3(y_2 - y_1)), \\
b_0 &:= x_1((x_2^2 + y_2^2 - r_2^2) - (x_3^2 + y_3^2 - r_3^2)) + \\
&\quad x_2((x_3^2 + y_3^2 - r_3^2) - (x_1^2 + y_1^2 - r_1^2)) + \\
&\quad x_3((x_1^2 + y_1^2 - r_1^2) - (x_2^2 + y_2^2 - r_2^2)), \\
b_1 &:= 2(k_1 r_1(x_2 - x_3) + k_2 r_2(x_3 - x_1) + k_3 r_3(x_1 - x_2)), \\
d &:= 2(x_1(y_2 - y_3) + y_1(x_3 - x_2) + x_2 y_3 - x_3 y_2).
\end{aligned}$$

By plugging the formulae for p_x, p_y into Eqn.-3.1 we get a second-degree polynomial in t:

$$0 = (a_0 + a_1 t - d x_1)^2 + (b_0 + b_1 t - d y_1)^2 - d^2 (r_1 + k_1 t)^2.$$

That is

$$0 = \alpha_0 + \alpha_1 t + \alpha_2 t^2,$$

with

$$\begin{aligned}
\alpha_0 &:= a_0^2 + b_0^2 - 2d(x_1 a_0 + y_1 b_0) + d^2(x_1^2 + y_1^2 - r_1^2) \\
\alpha_1 &:= 2(a_0 a_1 + b_0 b_1 - d(x_1 a_1 + y_1 b_1) - d^2 k_1 r_1) \\
\alpha_2 &:= a_1^2 + b_1^2 - d^2
\end{aligned}$$

Recipe for computing offsets For every configuration $k_1, k_2, k_3 \in \{-1, 1\}$ the equation $\alpha_0 + \alpha_1 t + \alpha_2 t^2 = 0$ is solved and non-negative roots are solutions t for offsets. The intersections of the three offset circles with radii $r_j + k_j t, j \in \{1, 2, 3\}$ are the corresponding equidistant points.

Determining equidistant points from an offset As mentioned above the intersection of the offset circles leads to the equidistant points. Plugging the offset t into the equations Eqn.-3.4, Eqn.-3.5 – the expressions for p_x and p_y – yields the equidistant point p as well. But this approach only works if the denominator d is non-zero. In a practical application d must be even perceptibly non-zero.

Lemma 3.2. *The denominator d is zero iff the centers c_1, c_2, c_3 are collinear.*

Proof. The denominator d can be expressed as $d = 2(\det(c_2, c_3) + \det(c_1, c_2 - c_3))$ where $c_j, j \in \{1, 2, 3\}$ are seen as column-vectors. Firstly, we examine that d is move-invariant:

$$\begin{aligned} & \det(c_2 + \Delta, c_3 + \Delta) + \det(c_1 + \Delta, c_2 - c_3) \\ = \; & \det(c_2, c_3) + \det(\Delta, c3) - \det(\Delta, c2) + \det(\Delta, \Delta) \\ & + \det(c_1, c_2 - c_3) + \det(\Delta, c_2 - c_3) \\ = \; & \det(c_2, c_3) + \det(\Delta, c_3 - c_2) + \det(c_1, c_2 - c_3) + \det(\Delta, c_2 - c_3) \\ = \; & \det(c_2, c_3) + \det(c_1, c_2 - c_3) \end{aligned}$$

Secondly, we move c_j by $-c_1$ and get new centers $c'_j = c_j - c_1, j \in \{1, 2, 3\}$ such that d results in

$$d = 2(\det(c'_2, c'_3) + \det(0, c'_2 - c'_3)) = 2\det(c'_2, c'_3).$$

Now its easy to see that d is zero iff c'_2, c'_3 are linearly dependent. Recall that c'_2, c'_3 arose from c_2, c_3 by a movement of $-c_1$ and therefore the lemma is shown. □

Corollary 3.3. *The equidistant point p can be obtained by plugging t into Eqn.-3.4, Eqn.-3.5 iff the centers c_1, c_2, c_3 are not collinear.*

In the case where the three centers c_1, c_2, c_3 are collinear, the approach of intersecting offset circles with known offset t und known constants k_1, k_2, k_3 must be taken. But since the centers are collinear, the solutions arise pairwise symmetrically with respect to the line through the centers. By the way, in this case the intersection of two offset circles suffices.

Lemma 3.4. *The maximum number of equidistant points to three circles is eight (excluding the case of infinitely many solutions), and this bound is sharp.*

Proof. Let us assume three circles C_1, C_2, C_3 are given. We are interested in the bisector $b_2 = b(C_1, C_2)$ between C_1 and C_2 (hyperbola and/or ellipse) and in the bisector $b_3 = b(C_1, C_3)$ between C_1 and C_3. The intersections between the two bisectors b_2, b_3 are the points equidistant to all three circles C_1, C_2, C_3[2]. We can find an upper bound on the number of possible intersections by considering all possible combinations of different bisector forms for b_2 and b_3. Note that an ellipse/hyperbola has not more than two intersections with another ellipse/hyperbola when a focal point is shared (except for the case of infinitely many solutions).

[2]This is an alternative method for retrieving all equidistant points, but not of practical use

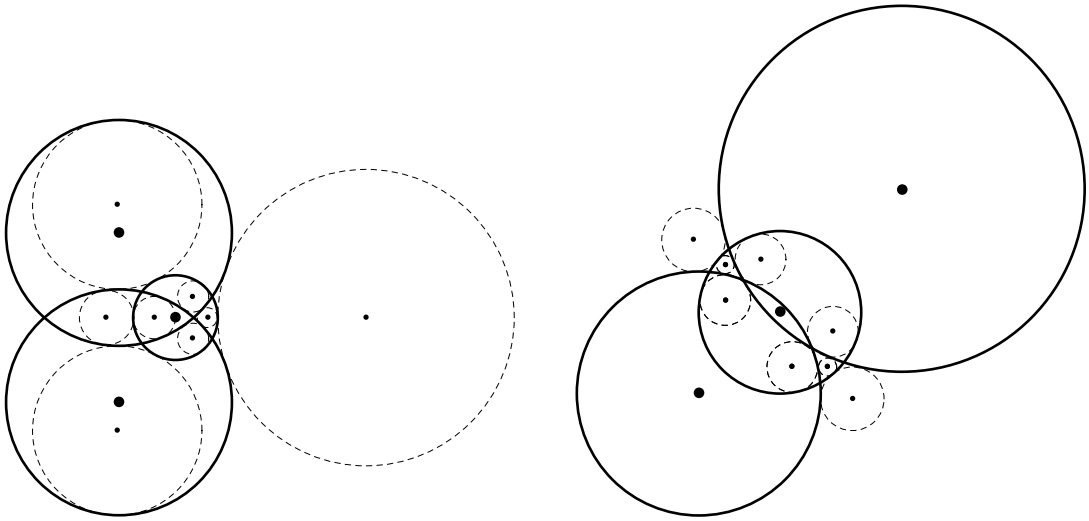

Figure 3.2: Equidistant points to three circles: The bold circles are the input circles. The centers of the dashed circles are the equidistant points searched for. The left figure shows input circles where every triple $(k_1, k_2, k_3) \in \{-1, 1\}^3$ results in a solution. The right example shows collinear centers.

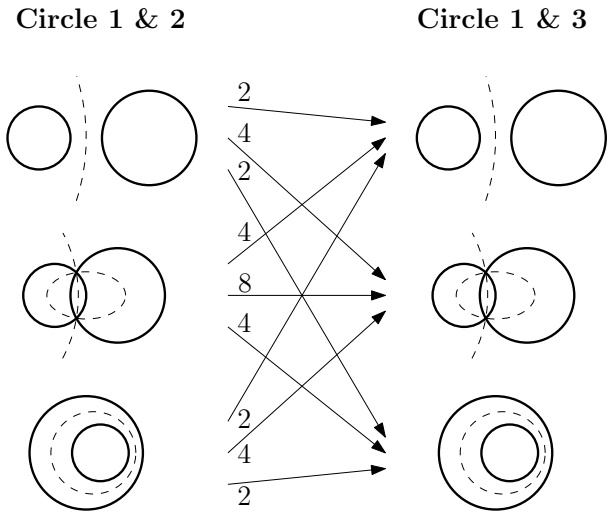

Figure 3.3: All possible bisector forms and the maximum number of intersections for every pair.

In Figure 3.3 all combinations are illustrated. The maximum number of intersections can only occur, if C_1 and C_2 intersect each other and C_1 and C_3 intersect as well. In this case b_1 and b_2 consists of an ellipse and a hyperbola. If the hyperbola and ellipse of b_1 intersect both hyperbola and ellipse of b_2 twice the maximum number of eight equidistant points is reached. □

Lemma 3.5. *The number of points equidistant to three circles is infinite iff two circles are identical.*

Proof. We use the notation of the last proof, namely $b_2 = b(C_1, C_2), b_3 = b(C_1, C_3)$. For the case of three identical circles the lemma is trivial. Let us assume there exists a circle C_1 which is not identical to the remaining two circles C_2, C_3. The number of equidistant points is infinite iff a hyperbola or ellipse of b_2 is identical to a hyperbola or ellipse of b_3. This is the case iff C_2 and C_3 are identical. □

3.1.2 Circle-Circle-Line

We consider two circles centered at $c_1 = (x_1, y_1), c_2 = (x_2, y_2)$ with radii r_1, r_2 and a straight line with orthonormal vector $o_3 = (u_3, v_3)$ and (signed) orthogonal distance $w_3 \in \mathbb{R}$ to the origin. The Hessian Form of the straight line is therefore $u_3x + v_3y = w_3$, where $||o_3|| = 1$. We are interested in all points equidistant to both circles and the straight line. The following motivation corresponds to Lemma 3.1:

Lemma 3.6. *A point p is equidistant iff there exists a common offset $t \geq 0$ and constants $k_1, k_2, k_3 \in \{-1, 1\}$ such that the offset circles with radii $r_j + k_j \cdot t$ for $j \in \{1, 2\}$ and the offset line with orthogonal distance $w_3 + k_3 \cdot t$ meet in p.*

Analogous to the case with three circles, the set of equidistant points can be determined by searching for proper offsets $t \geq 0$ and constants $k_1, k_2, k_3 \in \{-1, 1\}$ such that the following system is fulfilled:

$$\begin{aligned} (p_x - x_1)^2 + (p_y - y_1)^2 &= (r_1 + k_1 t)^2 && (3.6)\\ (p_x - x_2)^2 + (p_y - y_2)^2 &= (r_2 + k_2 t)^2 && (3.7)\\ u_3 p_x + v_3 p_y &= w_3 + k_3 t && (3.8) \end{aligned}$$

By subtracting Eqn.-3.7 from Eqn.-3.6 we get a linear equation in p_x, p_y. Together with Eqn.-3.8 the system results in:

$$\begin{aligned} 2p_x(x_2 - x_1) + 2p_y(y_2 - y_1) &= r_1^2 + 2r_1k_1t - r_2^2 - 2r_2k_2t - x_1^2 + x_2^2 - y_1^2 + y_2^2\\ u_3 p_x + v_3 p_y &= w_3 + k_3 t \end{aligned}$$

By solving the linear system in p_x, p_y we get linear expressions in t:

$$p_x = \frac{a_0 + a_1 t}{d}, \tag{3.9}$$

$$p_y = \frac{b_0 + b_1 t}{d}, \tag{3.10}$$

where

$$\begin{aligned} a_0 &= v_3((x_2^2 + y_2^2 - r_2^2) - (x_1^2 + y_1^2 - r_1^2)) + 2w_3(y_1 - y_2), \\ a_1 &= 2(k_3(y_1 - y_2) + v_3(k_1 r_1 - k_2 r_2)), \\ b_0 &= u_3((x_1^2 + y_1^2 - r_1^2) - (x_2^2 + y_2^2 - r_2^2)) + 2w_3(x_2 - x_1), \\ b_1 &= 2(k_3(x_2 - x_1) + u_3(k_2 r_2 - k_1 r_1)), \\ d &= 2(v_3(x_2 - x_1) - u_3(y_2 - y_1)). \end{aligned}$$

Recipe for computing offsets Equally to the case of three circles, we obtain a second-degree polynomial $\alpha_0 + \alpha_1 t + \alpha_2 t^2$ in t by reintroducing the formulae of p_x and p_y in Eqn.-3.6. The offsets $t \geq 0$ we search for are the non-negative roots of the polynomial for every triple $(k_1, k_2, k_3) \in \{-1, 1\}^3$. The corresponding points p_x, p_y for a specific offset $t \geq 0$ can be found by intersecting the corresponding offset circles and offset line.

Determining equidistant points from an offset Analogously to the case of three circles we can determine the corresponding equidistant point for an offset $t \geq 0$ by reintroducing t in the formulae for p_x, p_y (see Eqn.-3.9, Eqn.-3.10). This can only be done if the denominator d is non-zero.

Lemma 3.7. *The denominator d is zero iff the center-line is orthogonal to the line, i.e. $c_2 - c_1$ and o_3 are linearly dependent.*

Proof. Since d can be expressed as $d = 2\det(c_2 - c_1, o_3)$ the lemma is trivial. □

Corollary 3.8. *The equidistant point p can be obtained by plugging t into Eqn.-3.9, Eqn.-3.10 iff the center-line $\overline{c_1 c_2}$ is not orthogonal to the line.*

Lemma 3.9. *The maximum number of equidistant points to two circles and a line is eight (except for the case of infinitely many solutions), and this bound is sharp.*

Proof. This proof is very similar to the one of three circles. Consider the bisector $b_1 = b(C_1, s)$ between the first circle C_1 and the line s (one or two parabola) and the bisector $b_2 = b(C_2, s)$ between the second circle C_2 and the line s (one or two

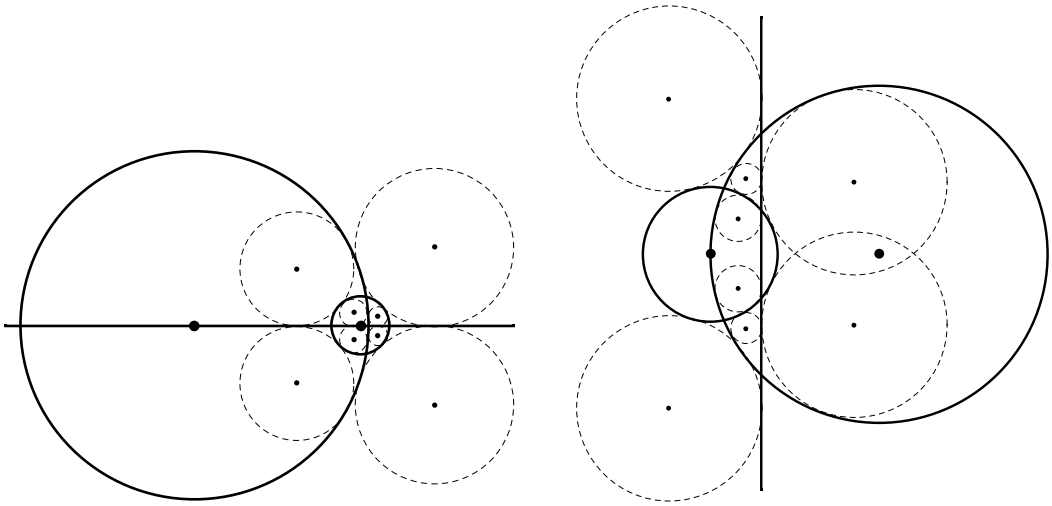

Figure 3.4: Equidistant points to two circles and a line. The bold circles are the input circles. The centers of the dashed circles are the equidistant points searched for. The left figure shows input circles where every triple $(k_1, k_2, k_3) \in \{-1, 1\}^3$ results in a solution. The right example shows a line orthogonal to the center-line.

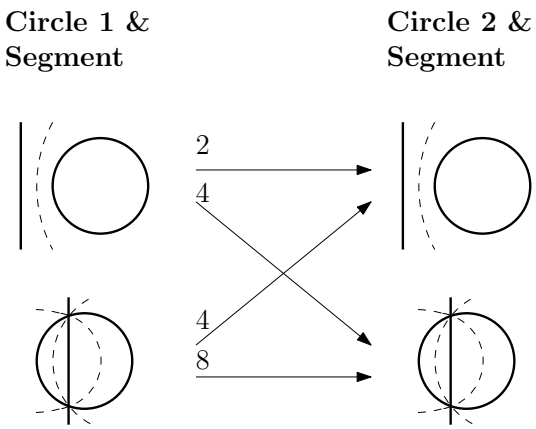

Figure 3.5: All possible bisector forms and the maximum number of intersections for every pair.

parabola). Then the set of equidistant points arise from the intersections of b_1 and b_2. Two parabola have no more than two intersections (excluding the case of infinitely many intersections).

In Figure 3.5 all possible bisector forms and their combinations are illustrated. The maximum number of eight intersections can only be reached when C_1 and s intersect and C_2 and s intersect as well. □

Lemma 3.10. *The number of points equidistant to two circles and a line is infinite iff the two circles are identical.*

Proof. We use the notation of the last proof. The number of equidistant points is infinite iff the bisectors b_1 and b_2 are identical. This is the case iff C_1 and C_2 are identical. □

3.1.3 Circle-Line-Line

We consider a circle centered at $c_1 = (x_1, y_1)$ with radius r_1 and two lines orthonormal vector $o_2 = (u_2, v_2), o_3 = (u_3, v_3)$ and (signed) orthogonal distances $w_2, w_3 \in \mathbb{R}$ to the origin, respectively. The Hessian forms of the two lines are therefore $u_2x + v_2y = w_2$, with $||o_2|| = 1$ and $u_3x + v_3y = w_3$, with $||o_3|| = 1$. The following motivation is analogous to Section 3.1.1 and Section 3.1.2.

Lemma 3.11. *A point p is equidistant to a circle and two lines iff there exists a common offset $t \geq 0$ and three constants $k_1, k_2, k_3 \in \{-1, 1\}$ such that the offset circle with radii $r_1 + k_1 \cdot t$ and the two offset lines with orthogonal distance $w_j + k_j \cdot t$, for $j \in \{2, 3\}$ meet in p.*

The set of equidistant points can be determined by searching for offsets $t \geq 0$ and constants $k_1, k_2, k_3 \in \{-1, 1\}$ such that the following system is fulfilled:

$$
\begin{aligned}
(p_x - x_1)^2 + (p_y - y_1)^2 &= (r_1 + k_1 t)^2 && (3.11)\\
u_2 p_x + v_2 p_y &= w_2 + k_2 t && (3.12)\\
u_3 p_x + v_3 p_y &= w_3 + k_3 t && (3.13)
\end{aligned}
$$

Solving Eqn.-3.12 and Eqn.-3.13 for p_x, p_y we get expressions depending linearly on t:

$$
\begin{aligned}
p_x &= \frac{a_0 + a_1 t}{d}, , && (3.14)\\
p_y &= \frac{b_0 + b_1 t}{d}, . && (3.15)
\end{aligned}
$$

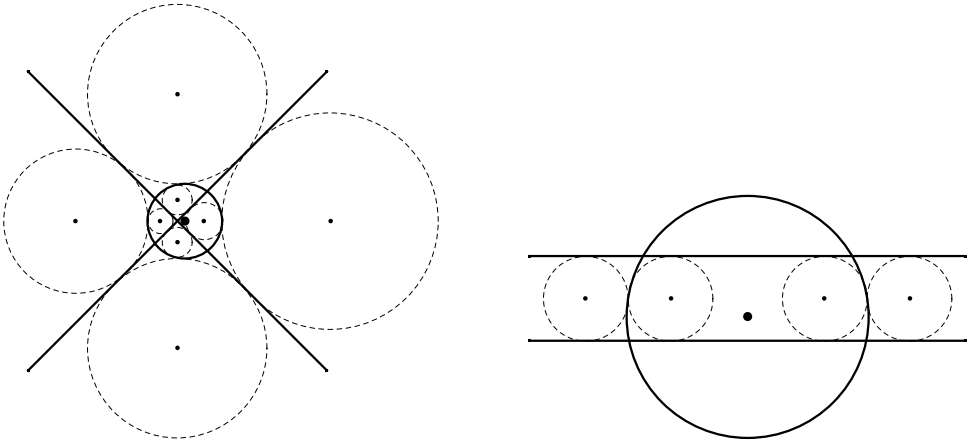

Figure 3.6: Two examples for the results of the procedure described in this section. The bold circles are the input circles. The centers of the dashed circles are the equidistant points sought. The left figure shows input circles where every triple $(k_1, k_2, k_3) \in \{-1, 1\}^3$ results in a solution. The right example shows two parallel lines.

where

$$\begin{aligned} a_0 &:= w_2 v_3 - w_3 v_2, \\ a_1 &:= k_3 v_3 - k_2 v_2, \\ b_0 &:= w_3 u_2 - w_2 u_3, \\ b_1 &:= k_3 u_2 - k_2 u_3, \\ d &:= u_2 v_3 - u_3 v_2. \end{aligned}$$

Plugging Eqn.-3.14, Eqn.-3.15 into Eqn.-3.11 leads to the known formulae from of the last sections Section 3.1.1 and Section 3.1.2. The recipe for retrieving all offsets $t \geq 0$ and the corresponding equidistant points $p = (p_x, p_y)$ is completely identical. The following corollary fills the gap for getting equidistant points p by reintroducing the offset:

Corollary 3.12. *The equidistant point $p = (p_x, p_y)$ for a specific offset $t \geq 0$ can be obtained by plugging t into the formulae Eqn.-3.14, Eqn.-3.15 iff $d = \det(o_2, o_3) \neq 0$ which means that the lines are not parallel.*

Lemma 3.13. *The maximum number of points equidistant to a circle and two lines is eight (except for the case of infinitely many solutions). The boundary is sharp.*

Lemma 3.14. *The number of points equidistant to a circle and two lines is infinite iff the two lines are identical.*

3.2 Special cases

In Section 3.1 we discussed how to obtain the set of points equidistant to circles and lines. These procedures fail if the number of points is infinite. We obtained that this is the case when two circles or lines are identical. This corresponds to two concentric arcs or two segments being on a line and having a common end point[3]. Note that the corresponding second-degree polynomials in the offset t are the zero-polynomials in this special cases. Hence, the abstraction of arcs to full circles and segments to lines goes to far. There is a second reason to look for special cases: It turns out that two arcs being tangential or an arc being tangential to a segment can be numerically inaccurate or instable. Since two arcs being concentric with a common end point are tangential too, we can summarize the following special cases:

- Two tangential arcs with common end point.
- Two co-circular arcs without common end point.
- Two arcs where both centers and an end point of each arc are collinear.
- Tangential arc and segment with common end point.

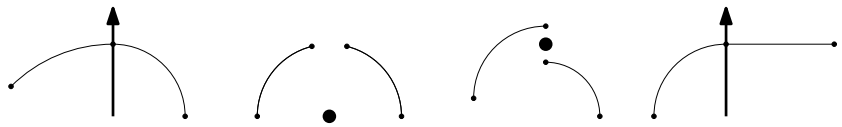

Figure 3.7: The special cases and the resulting bisectors as bold points or rays. F.l.t.r: Tangential arcs. Co-circular disjoint arcs without common end point. Concentric arcs where centers and two end points are collinear. Tangential arc and segment.

All cases have in common that the corresponding bisector is a ray or even a single point and can be easily handled by similar methods as in Section 3.1. The

[3]Two non-overlapping segments on a line without common end point have disjunct cone of influences. Whereas two concentric arcs without common end point have their common center as equidistant point in both cone of influences.

points on the ray can be parametrized straight-forwardly by a starting point, a unit direction vector, and the distance to the start point. This parametrization can be introduced into the equation originating from the third defining site, which finally yields the offsets and points sought.

For example, let us assume the third site is a segment with orthonormal vector $o_2 = (u_2, v_2)$ and orthogonal distance w_2 to the origin. The Hessian Form of this straight line is therefore $u_2x + v_2y = w_2$, where $||n_2|| = 1$. Furthermore, let $c_1 = (x_1, x_2)$ be the start point of the ray and let $o_1 = (u_1, v_1)$ be the direction vector, with $||o_1|| = 1$. We search for a point $(p_x, p_y) \in \mathbb{R}$, a distance $t \geq 0$ and constants $k_1, k_2 \in \{-1, 1\}$, such that the following system is fulfilled:

$$\begin{aligned} p_x &= x_1 + k_1 t \cdot u_1 \\ p_y &= y_1 + k_1 t \cdot v_1 \\ p_x x_2 + p_y y_2 &= w_2 + k_2 t \end{aligned}$$

By introducing the first and second equation (ray-parametrization) into the third equation we get solutions of t, similar to the approach of Section 3.1. After reintroducing t into the first and second equation, we get the equidistant points sought.

3.3 Selecting the correct solution

We saw that the number of equidistant points to three sites is far away of being unique. Therefore we can ask for the correct solution out of many possible ones. The abstraction of arcs to full circles and segments to straight lines leads in general to a higher number of equidistant points. Firstly, all points not in the cone of influences can be dropped. Secondly, VRONI implements an incremental algorithm with topological constraints. A new node has to be computed when adapting an edge by taking the two defining sites and the new inserted site. Therefore the new node has to lie on the old edge. Among others, these two informations can be used to find the correct solution.

Since VRONI uses double-precision arithmetic we have to deal with numerical instability. For this reason a "black or white" procedure can be problematic. Instead of that a punishment-oriented procedure has been introduced: Every potential solution is a value 0 assigned. If the solution lies outside of a cone of influence the value is increased by a proper value. If the solution is not in a valid region of the old edge, then a value rating the deviation is added. At the end we take the solution with the lowest value assigned.

Chapter 4

Results

In the sequel of this thesis, we present a classical application of Voronoi diagrams, namely offsetting. We will shortly describe the computation of contour-parallel offset curves of curvilinear[1] polygons, once the Voronoi diagrams is known. But there exists a few other problems which can be solved easily by using the Voronoi diagram, such as computing the medial axis or determining the maximum inscribed circle. Finally, we present experimental results, in particular runtime comparisons considering the pre-genuine-arc version of Vroni and CGAL Apollonius.

4.1 An application: Offsetting

We will not exactly describe the algorithm for getting the contour parallel offset curves from the Voronoi diagram. It is more or less a demonstration for what can be done, once the Voronoi diagram has been calculated. Furthermore, the algorithm for this problem is more or less the same as the one for the pre-genuine-arc version of Vroni. However, offsetting has been of great scientific interest in past and is still these days. In [HLA94] a short survey on contour parallel offsetting in combination with Voronoi diagrams is given, where we can find the following statement:

> A fundamental NC-machining problem is the clearing of areas within specified boundaries from material.

Furthermore, there is a note "that Persson first proposed the use of Voronoi diagrams for efficient offsetting". However, a precise, theoretical consideration has been presented the first time in [Hel91].

[1] Consisting of straight lines and circular arcs.

The idea can be roughly described as follows: Let us assume a input figure consisting of segments and circular arcs as boundary elements is given. The conventional algorithm shifts all boundary elements by a specific offset and removes those parts which do not belong to the offset curve. Removing the overhanging parts can be very complex. In particular, determining all intersections is very time consuming. Furthermore, its necessary to add trimming arcs to get a closed curve, see [HLA94]. Note that offsetting can result in a topological change. That is, the offset curves can fall apart into several components.

A more elegant and easy method to compute these offset curves makes use of Voronoi diagrams. Let us consider the Figure 4.1. The idea is that the offset is monotonically changing when walking on a Voronoi edge[2]. Consider an arbitrary end point of a segment or arc of the outer contour. If we walk from this point on an edge to the inner of the polygon then the offset monotonically increases. If we take any point p on this edge then we know that the offset curve with the same offset as the clearance of p contains p. This idea can be used to compute a whole offset curve with a specific offset: We start at a point of the polygon and walk along an edge, until we reach the wanted offset or we continue at the succeeding edge in counter clock-wise direction. If we reach a position on an edge with the specific offset we place a point and continue in counter clock-wise direction as shown in Figure 4.1. For every further point, with a specific offset, we place an offset site connecting the last two points. The offset curves, we obtain, are illustrated in Figure 4.2.

This method is very easy to implement, fast and stable. We only have to determine for all edges whether they belong to the inner of the polygon or to the outer as a preprocessing. More examples, showing offset curves, are illustrated in the appendix.

4.2 Experimental results

The genuine-arc version of VRONI has been tested with a large number of datasets. They can be separated into two parts:

- Special data for testing algorithmic details, like handling tangential sites, handling spikes, testing the breaking-up code or just for testing the correct plotting all possible bisector forms.

- Approximately 700 datasets consisting at one hand of industrial examples and at the other of synthetic examples like spirals, fractals like Sierpinski or

[2]Recall the note on apex splitting.

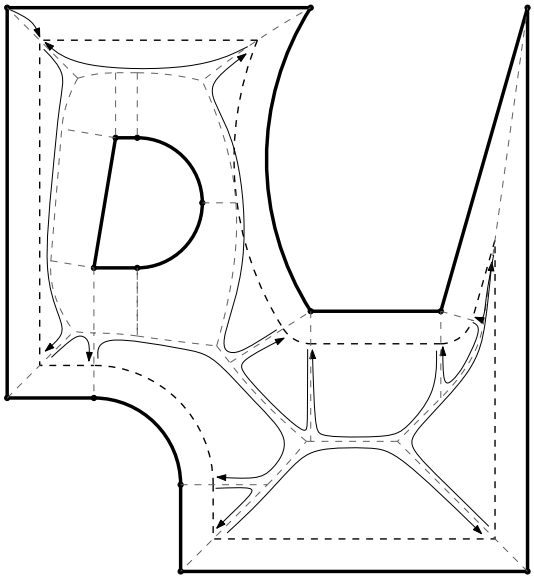

Figure 4.1: Given an input figure with holes (bold), consisting of segments and arcs, we compute a single offset curve (dashed) by walking (arrows) along the edges of the Voronoi diagram (grey, dashed).

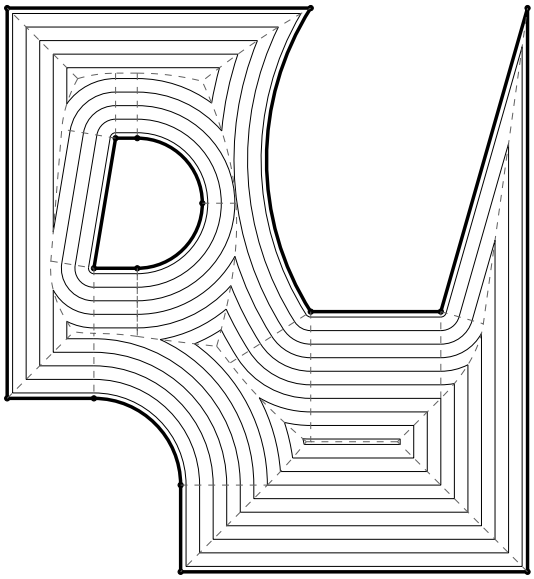

Figure 4.2: Given an input figure with holes (bold), consisting of segments and arcs, we compute the offset curves of the inner area based on the Voronoi diagram (grey, dashed) counter clock-wise.

Koch snowflake, closed curves consisting of tangential-continues arcs, curvilinear[3] polygons with holes, or the offset curves of previous datasets.

In the following we will compare the run time of the pre-genuine-arc version of VRONI with the new one. We do not compare VRONI with other Voronoi codes since all other implementations we are aware of do not natively support circular arcs. A comparison of other Voronoi codes with (the pre-genuine-arc version of) VRONI can be found in [Hel01].

The hardware setup We performed the comparison on an Intel Core 2 Duo E6700 processor. Each core is clocked at 2.66 Ghz and uses 4MB of Level 2 cache. The Linux kernel rates the CPU at 5333.90 bogomips[4] per core. Furthermore, the system has 4GB of RAM. Note that a single process can not allocate more than approximately 3GB of RAM in user space on our 32-bit Linux system. Furthermore it should be noted that VRONI does not make use of multiple cores, i.e. is not multi-threaded.

Setup of old Vroni The old (pre-genuine-arc version of) VRONI supports circular arcs by approximating them by tangential segments. The number of segments used can be determined by setting relative[5] or absolute error bounds. By default a heuristic approach is used. For the comparison, we have set an absolute error bound of $5 \cdot 10^{-7}$. This was a suitable[6] bound such that all data sets worked correctly.

We have compared the new and old VRONI on a few data sets. The corresponding run time tables can be found in the Appendix A. In the following, we will describe the data sets we consider for the comparison.

Biarcs The biarcs data set consists of curvilinear polygons without holes. The polygons consist of tangential arcs and segments but the number of arcs is about 24-times bigger than the number of segments. The data set consists of 313 examples, where each examples consists of about 900 input sites. This data set is not listed in the appendix.

Curvilinear The curvilinear data set is similar to the biarcs data set and has been constructed by creating random polygons using the RPG [AH96] package and after that applying a biarc-approximation [HH08]. An example can be found in the appendix, see Figure B.3.

[3] Consisting of straight lines and circular arcs.
[4] A heuristic rating of the "speed" of a processor.
[5] With respect to the radii of the arcs.
[6] For example, 10^{-6} was not sufficient.

Koch snowflake, Sierpinksi, space filling The Koch snowflake and the Sierpinski data sets consist of "circular" examples of the famous iterative fractal pattern construction. Note that the Koch snowflake data sets consists only of arcs whereas the Sierpinski has an equal number of segments and arcs. The space filling data set results from the Sierpinski data set by applying a shearing-function to the input data. An example from the Sierpinski data set is illustrated in the appendix, see Figure B.1.

The "n/a" entries in the run time tables indicate that memory usage was too high to allow the old VRONI to finish the computation. Note that the number of arcs is an order of several hundred-thousand.

Spiral The spiral data set consists of closed curvilinear polygons forming a spiral. The spiral data set is of special interest. Since the "outer" arcs of the spiral are very large, compared to other data sets, the old VRONI has to approximate them with a higher number of segments. This leads to performance gains up to 70, as listed in Table A.5. An example is given in the appendix, see Figure B.2.

Apollonius The Apollonius data set consists of n random circles, constructed in the following way: First, we choose a random radius r uniformly distributed on $[0.05 \cdot r_{max}, r_{max}]$ and after that we choose a random center uniformly distributed on $[r, 1-r]^2$. By choosing the constant $r_{max} \in (0, 0.5)$ we can justify the mean[7] size of the circles. Finally, we extract an intersection-free arrangement of arcs as the input for VRONI. Furthermore, if a circle does not intersect with any other circle then the circle is split into four arcs with equal chord-length. See Figure B.4 for an example.

The data set has been constructed with the following parameters: For every $n \in \{10.000, 10.500, \cdots, 30.000\}$, we set $r_{max} = 10/n$ and generated test data according to the description above. The corresponding run times are illustrated in Table A.6.

The run times of the data sets described above are illustrated in Figure 4.3. All examples with $n < 100$ have been dropped since the run time of the new VRONI is either zero or a small inaccurate value. However, we can see in Figure 4.3 that the run time is more or less independent of the type of input data. More over, in most cases the run time is between $0.015n$ and $0.025n$.

In contrast to the new VRONI, the run times of the old VRONI vary more significantly. The reason is that the approximation of arcs strongly depends on their size. Hence, different types of data sets tend to result in different run time

[7] In exact stochastic terms, the expectation of r is of course $r_{max} \cdot (0.05 + 0.95/2)$.

behaviours. However, an overall performance gain of about $10-20$ can be reached in most examples. Furthermore, a trend of old VRONI getting faster with larger input sets can be experienced. This can be explained by considering the arcs involved in the larger data sets: they get smaller relative to the bounding box with increasing size of input sets and are therefore approximated by fewer segments.

In Figure 4.4 we can see the specific run times of inserting arcs and segments. As we can see, inserting a single arc consumes about 0.03 seconds, whereas inserting a single segment consumes about 0.01 seconds.

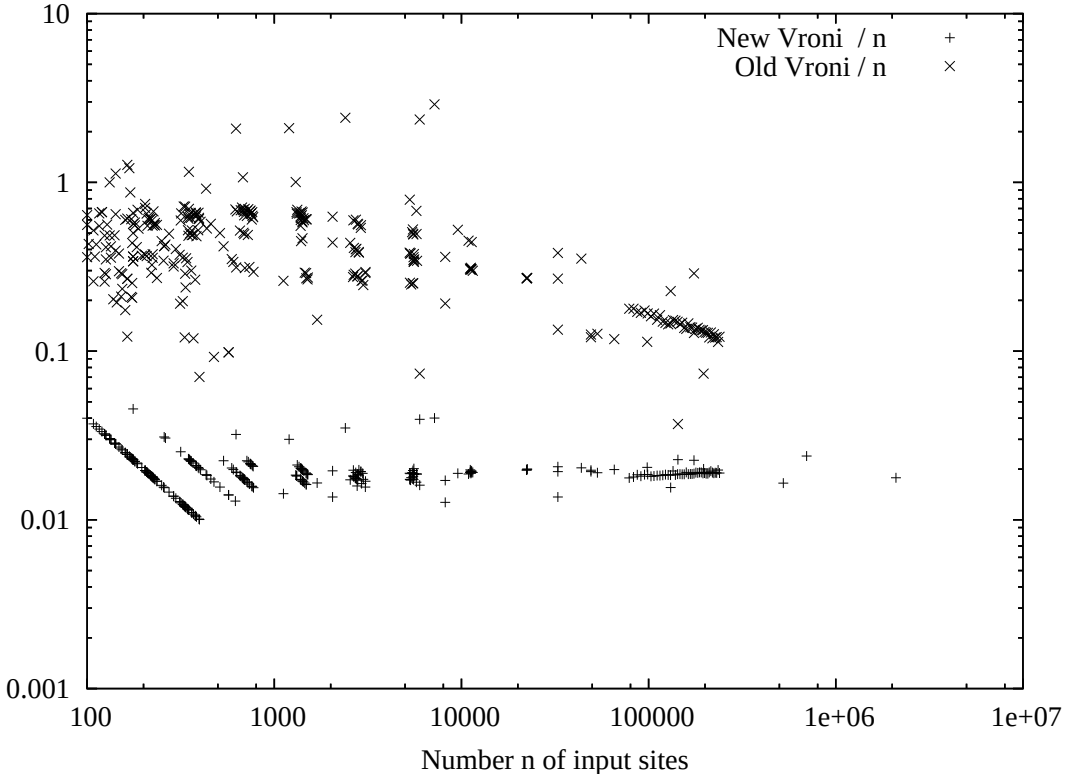

Figure 4.3: The run times of the new and old VRONI in milliseconds are shown on 435 different examples. The run times have been divided by the number n of sites.

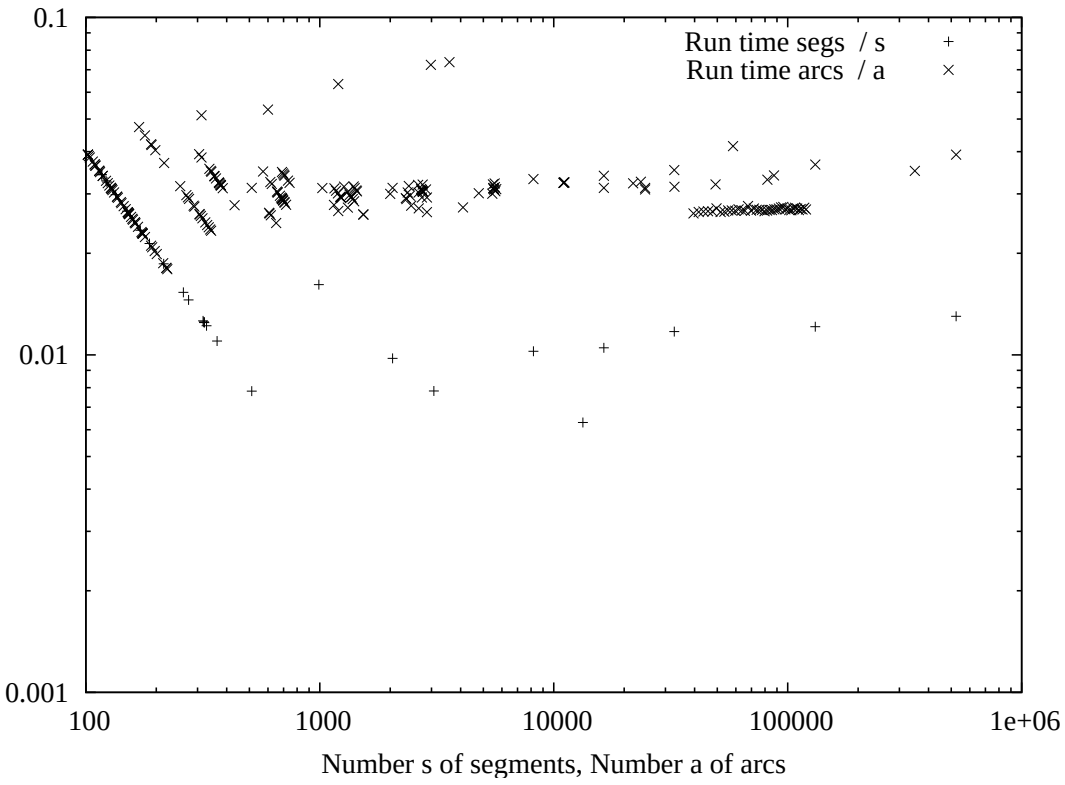

Figure 4.4: The run times of inserting (i) segments and (ii) arcs of the new VRONI on 435 different examples.

Bibliography

[AH96] Thomas Auer and Martin Held. Heuristics for the Generation of Random Polygons. In *Proceedings of the 8th Canadian Conference on Computational Geometry*, pages 38–44. Carleton University Press, 1996.

[AS95] Helmut Alt and Otfried Schwarzkopf. The Voronoi Diagram of Curved Objects. In *Annual Symposium on Computational Geometry*, pages 89–97, 1995.

[Aur91] Franz Aurenhammer. Voronoi Diagrams – A survey of a Fundamental Geometric Data Structure. *ACM Computing Surveys*, 23:345–405, 1991.

[For86] Steven Fortune. A sweepline algorithm for Voronoi diagrams. In *SCG '86: Proceedings of the second annual symposium on Computational geometry*, pages 313–322, New York, NY, USA, 1986. ACM.

[For00] Steven Fortune. Introduction. *Algorithmica*, 27:1–4, 2000.

[Hel91] Martin Held. *On the Computational Geometry of Pocket Machining*. Springer-Verlag, Berlin Heidelberg, 1991.

[Hel01] Martin Held. VRONI: An engineering approach to the reliable and efficient computation of Voronoi diagrams of points and line segments. *Computational Geometry*, 18:95–123, 2001.

[HH08] Martin Heimlich and Martin Held. Biarc Approximation, Simplification and Smoothing of Polygonal Curves by Means of Voronoi-Based Tolerance Bands. *IJCGA*, to appear 2008.

[HLA94] Martin Held, Gábor Lukács, and László Andor. Pocket machining based on contour-parallel tool paths generated by means of proximity maps. *Computer-Aided Design*, 26(3):189–203, 1994.

[Ima96] Toshiyuki Imai. A Topology Oriented Algorithm for the Voronoi Diagram of Polygons. In *Proceedings of the 8th Canadian Conference on Computational Geometry*, pages 107–112. Carleton University Press, 1996.

[JKM+06] Li Jin, Donguk Kim, Lisen Mu, Deok-Soo Kim, and Shi-Min Hu. A sweepline algorithm for Euclidean Voronoi diagram of circles. *Computer-Aided Design*, 38:260–272, 2006.

[Kar04] Menelaos I. Karavelas. A robust and efficient implementation for the segment Voronoi diagram . In *International Symposium on Voronoi Diagrams in Science and Engineering*, pages 51–62, 2004.

[KKS01a] Deok-Soo Kim, Donguk Kim, and Kokichi Sugihara. Voronoi diagram of a circle set from Voronoi diagram of a point set: I. Topology. *Computer Aided Geometric Design*, 18(6):541–562, July 2001.

[KKS01b] Deok-Soo Kim, Donguk Kim, and Kokichi Sugihara. Voronoi diagram of a circle set from Voronoi diagram of a point set: II. Geometry. *Computer Aided Geometric Design*, 18(6):563–585, July 2001.

[Kle89] Rolf Klein. *Concrete and Abstract Voronoi Diagrams*. Springer-Verlag, Berlin Heidelberg New York, 1989.

[KMM93] Rolf Klein, Kurt Mehlhorn, and Stefan Meiser. Randomized incremental construction of abstract Voronoi diagrams. *Comput. Geom. Theory Appl.*, 3(3):157–184, 1993.

[KY02] Menelaos I. Karavelas and Mariette Yvinec. Dynamic Additively Weighted Voronoi Diagrams in 2D. In *10th European Symposium on Algorithms (ESA 2002)*, pages 586–598, 2002.

[KY03] Menelaos I. Karavelas and Mariette Yvinec. The Voronoi Diagram of Planar Convex Objects. In *11th European Symposium on Algorithms (ESA 2003)*, pages 337–348, 2003.

[LD81] Der-Tsai Lee and Robert L Drysdale III. Generalization of Voronoi Diagrams in the plane. *SIAM Journal of Computing*, 10(1):73–87, 1981.

[Lee82] Der-Tsai Lee. Medial Axis Transformation of a Planar Shape. *IEEE Trans. Pattern Analysis and Machine Intelligence*, PAMI-4(4):363–369, 1982.

[Sei93] Raimund Seidel. Backwards Analysis of Randomized Geometric Algorithms. In J. Pach, editor, *New Trends in Discrete and Computational Geometry*, pages 37–68. Springer-Verlag, 1993.

[SH75] Michael I. Shamos and Dan Hoey. Closest Point Problems. In *16th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 151–162, 1975.

[Sha85] Micha Sharir. Intersection and Closest-Pair Problems for a Set of Planar Discs. *SIAM Journal on Computing*, 14(2):448–468, 1985.

[SHM01] Saurabh Sethia, Martin Held, and J. S. B. Mitchell. PVD: A Stable Implementation for Computing Voronoi Diagrams of Polygonal Pockets. In *3rd Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 105–116, 2001.

[SI92] Kokichi Sugihara and Masao Iri. Construction of the Voronoi Diagram for "One Million" Generators in Single-Precision Arithmetic. *Proceedings of the IEEE*, 80:1471–1484, 9 1992.

[SIII00] Kokichi Sugihara, Masao Iri, Hiroshi Inagaki, and Toshiyuki Imai. Topology-Oriented Implementation – An Approach to Robust Geometric Algorithms. *Algorithmica*, 27:5–20, 2000.

[Yap87] Chee K. Yap. An $O(n \log n)$ Algorithm for the Voronoi Diagram of a Set of Simple Curve Segments. *Discrete and Computational Geometry*, 2:365–393, 1987.

Appendix A

Run time tables

The following run time tables are organized as follows. The first three columns list the number of points, segments and arcs in the input data. The following two columns contain the run time in milli-seconds, as given by VRONI. These timings are measured by using the `rusage` C-command. The sixth columns illustrates the performance gain, i.e. the quotient of the run time of old VRONI to new VRONI.

If a "n/a" appears as run time of old VRONI then memory usage was too high to allow old VRONI to finish the computation. However, it does not indicate a crash. If the "Speed-up" is "n/a" then one of both run time values is "n/a". If the "Speed-up" is "inf" than the run time of new VRONI was zero.

Table A.1: Curvilinear data set.

Pnts	**Segs**	**Arcs**	**New Vroni**	**Old Vroni**	**Speed-up**
Number of			ms	ms	
218	4	214	8.0	232.0	29.0
371	1	370	16.0	500.0	31.2
657	1	656	24.0	876.1	36.5
690	2	688	24.0	908.1	37.8
693	10	683	24.0	872.1	36.3
708	3	705	24.0	660.0	27.5
715	7	708	24.0	876.1	36.5
1273	123	1150	44.0	1116.1	25.4
1317	161	1156	48.0	724.0	15.1
1322	7	1315	52.0	1576.1	30.3
1328	154	1174	48.0	752.0	15.7
1355	159	1196	48.0	780.0	16.2

Pnts	Segs	Arcs	New VRONI	Old VRONI	Speed-up
1357	14	1343	48.0	1048.1	21.8
1373	150	1223	48.0	1108.1	23.1
1386	187	1199	44.0	768.0	17.5
1393	11	1382	52.0	1584.1	30.5
1415	6	1409	56.0	1084.1	19.4
1484	215	1269	56.0	780.0	13.9
1496	261	1235	52.0	736.0	14.2
2644	11	2633	100.0	2028.1	20.3
2653	320	2333	92.0	1348.1	14.7
2665	21	2644	100.0	2012.1	20.1
2676	328	2348	92.0	2032.1	22.1
2728	19	2709	104.0	2764.2	26.6
2749	321	2428	96.0	1388.1	14.5
2755	28	2727	108.0	2900.2	26.9
2756	24	2732	104.0	2732.2	26.3
2764	364	2400	100.0	1392.1	13.9
2768	36	2732	104.0	1916.1	18.4
2775	317	2458	96.0	2000.1	20.8
2783	30	2753	104.0	1892.1	18.2
2787	34	2753	112.0	2096.1	18.7
2798	38	2760	104.0	1956.1	18.8
2893	24	2869	108.0	1984.1	18.4
5557	45	5512	212.0	3432.2	16.2
5570	70	5500	220.0	3392.2	15.4
5596	58	5538	216.0	3408.2	15.8
5601	79	5522	216.0	3424.2	15.9
5619	50	5569	212.0	3504.2	16.5
5639	52	5587	220.0	3460.2	15.7
5667	66	5601	220.0	3452.2	15.7
5670	43	5627	216.0	5024.3	23.3
5754	74	5680	220.0	3444.2	15.7

Table A.2: Koch snowflake data set.

Pnts	**Segs**	**Arcs**	**New Vroni**	**Old Vroni**	**Speed-up**
Number of			ms	ms	
1534	0	1534	48.0	892.1	18.6
1536	0	1536	52.0	904.1	17.4
4093	0	4093	140.0	1568.1	11.2
16380	0	16380	676.0	4388.3	6.5
24576	0	24576	948.1	6156.4	6.5
24578	2	24576	968.1	5940.4	6.1
26628	3071	23557	1012.1	6740.4	6.7
32775	1	32774	1300.1	7724.5	5.9
49154	0	49154	2012.1	11156.7	5.5
71681	13315	58366	3256.2	5300.3	1.6
98304	16385	81919	3940.2	14456.9	3.7

Table A.3: Sierpinski data set.

Pnts	**Segs**	**Arcs**	**New Vroni**	**Old Vroni**	**Speed-up**
Number of			ms	ms	
4	0	4	0.0	16.0	inf
16	8	8	0.0	20.0	inf
64	32	32	0.0	72.0	inf
256	128	128	8.0	256.0	32.0
1024	512	512	28.0	900.1	32.1
4096	2048	2048	104.0	2964.2	28.5
16384	8192	8192	448.0	8812.6	19.7
65536	32768	32768	2036.1	29733.9	14.6
262144	131072	131072	8648.5	n/a	n/a
1048576	524288	524288	37210.3	n/a	n/a

Table A.4: Space-filling data set.

Pnts	**Segs**	**Arcs**	**New Vroni**	**Old Vroni**	**Speed-up**
Number of			ms	ms	
4	0	4	0.0	12.0	inf
20	0	20	4.0	48.0	12.0
84	0	84	4.0	204.0	51.0
340	0	340	12.0	728.0	60.7
1364	0	1364	48.0	1640.1	34.2
5460	0	5460	204.0	4932.3	24.2
21844	0	21844	888.1	15429.0	17.4
87380	0	87380	3940.2	50399.2	12.8
349524	0	349524	16693.0	n/a	n/a

Table A.5: Spiral data set.

Pnts	**Segs**	**Arcs**	**New Vroni**	**Old Vroni**	**Speed-up**
Number of			ms	ms	
8	0	8	0.0	24.0	inf
16	0	16	0.0	40.0	inf
48	0	48	4.0	120.0	30.0
312	0	312	20.0	1300.1	65.0
600	0	600	36.0	2512.2	69.8
1199	1	1198	84.0	5792.4	69.0
2991	7	2984	236.0	14116.9	59.8
3591	7	3584	288.0	20837.3	72.4

Table A.6: Apollonius data set generated from random circles.

Pnts	**Segs**	**Arcs**	**New Vroni**	**Old Vroni**	**Speed-up**
Number of			ms	ms	
39190	5	39595	1396.1	14040.9	10.1
41300	2	41657	1488.1	14800.9	9.9
43278	5	43629	1608.1	14840.9	9.2
45252	2	45622	1648.1	15325.0	9.3
47359	1	47680	1760.1	16541.0	9.4
49269	3	49633	1836.1	16537.0	9.0
51256	5	51628	1864.1	16553.0	8.9
53221	2	53608	1944.1	17829.1	9.2
55308	2	55652	2032.1	17013.1	8.4
57266	1	57634	2104.1	18769.2	8.9
59380	3	59685	2188.1	17785.1	8.1
61321	5	61664	2280.1	18081.1	7.9
63408	6	63707	2344.1	18333.1	7.8
65302	3	65640	2412.2	19093.2	7.9
67341	4	67672	2628.2	20609.3	7.8
69365	4	69667	2568.2	21073.3	8.2
71309	3	71649	2664.2	21321.3	8.0
73429	5	73715	2744.2	21269.3	7.8
75322	5	75660	2812.2	22249.4	7.9
77306	8	77642	2884.2	21277.3	7.4
79354	0	79678	3040.2	21529.3	7.1
81278	3	81624	3036.2	23793.5	7.8
83358	4	83664	3120.2	23181.4	7.4
85261	2	85623	3192.2	23537.5	7.4
87202	3	87592	3280.2	22389.4	6.8
89276	6	89631	3412.2	24213.5	7.1
91361	3	91667	3472.2	25169.6	7.2
93345	4	93648	3572.2	24997.6	7.0
95367	4	95684	3648.2	25325.6	6.9
97321	2	97659	3676.2	25969.6	7.1
99332	2	99656	3772.2	25817.6	6.8
101323	8	101623	3812.2	26125.6	6.9
103370	4	103678	3888.2	26577.7	6.8

Pnts	Segs	Arcs	New VRONI	Old VRONI	Speed-up
105265	2	105635	4024.3	25417.6	6.3
107257	8	107589	4176.3	27693.7	6.6
109384	7	109672	4176.3	26097.6	6.2
111277	10	111592	4212.3	28061.8	6.7
113334	5	113638	4312.3	27305.7	6.3
115331	4	115646	4380.3	27965.7	6.4
117368	0	117686	4640.3	26629.7	5.7
119408	1	119700	4532.3	29101.8	6.4

Appendix B

Examples

In the following we illustrate a few examples. The offset curves of curvilinear polygons (consisting of segments and arcs, probably with holes) have been restricted to the inner

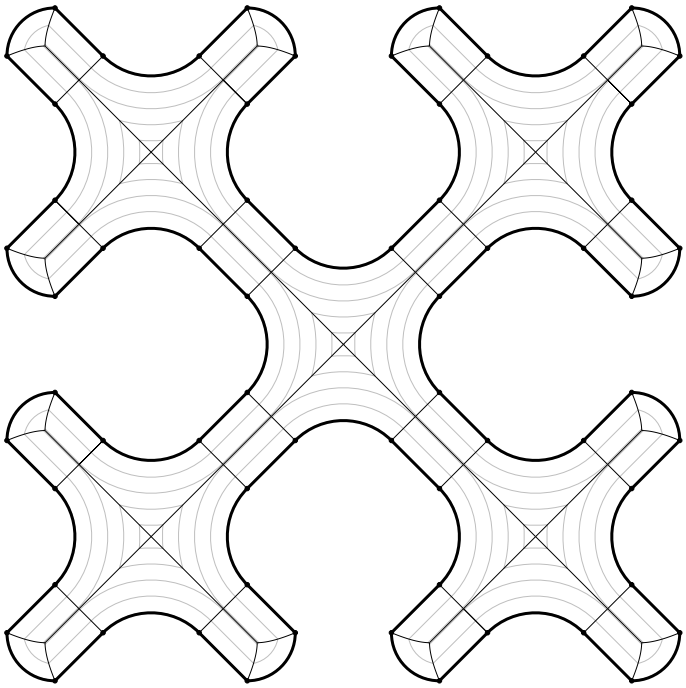

Figure B.1: A Sierpinski dataset with 64 points, 32 segments and 32 arcs.

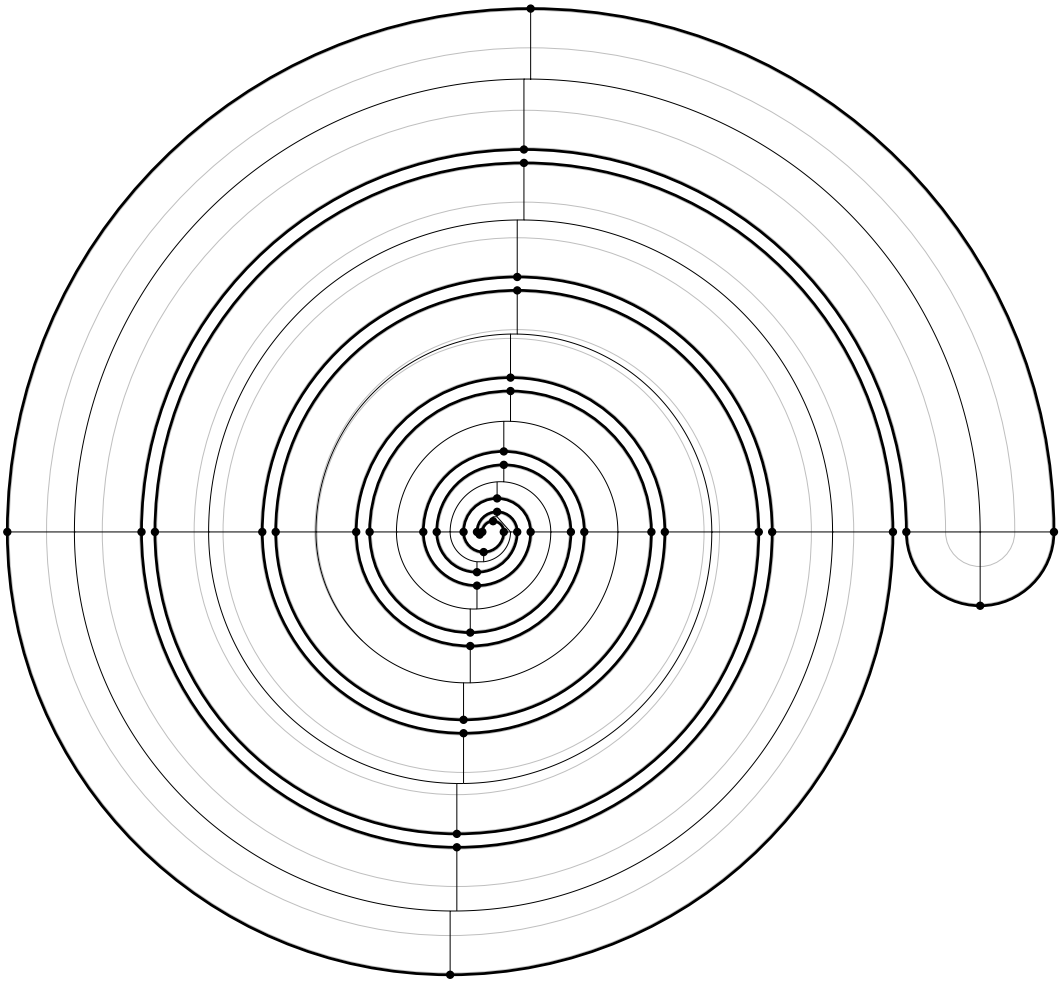

Figure B.2: A spiral dataset with 48 points, 0 segments and 48 arcs.

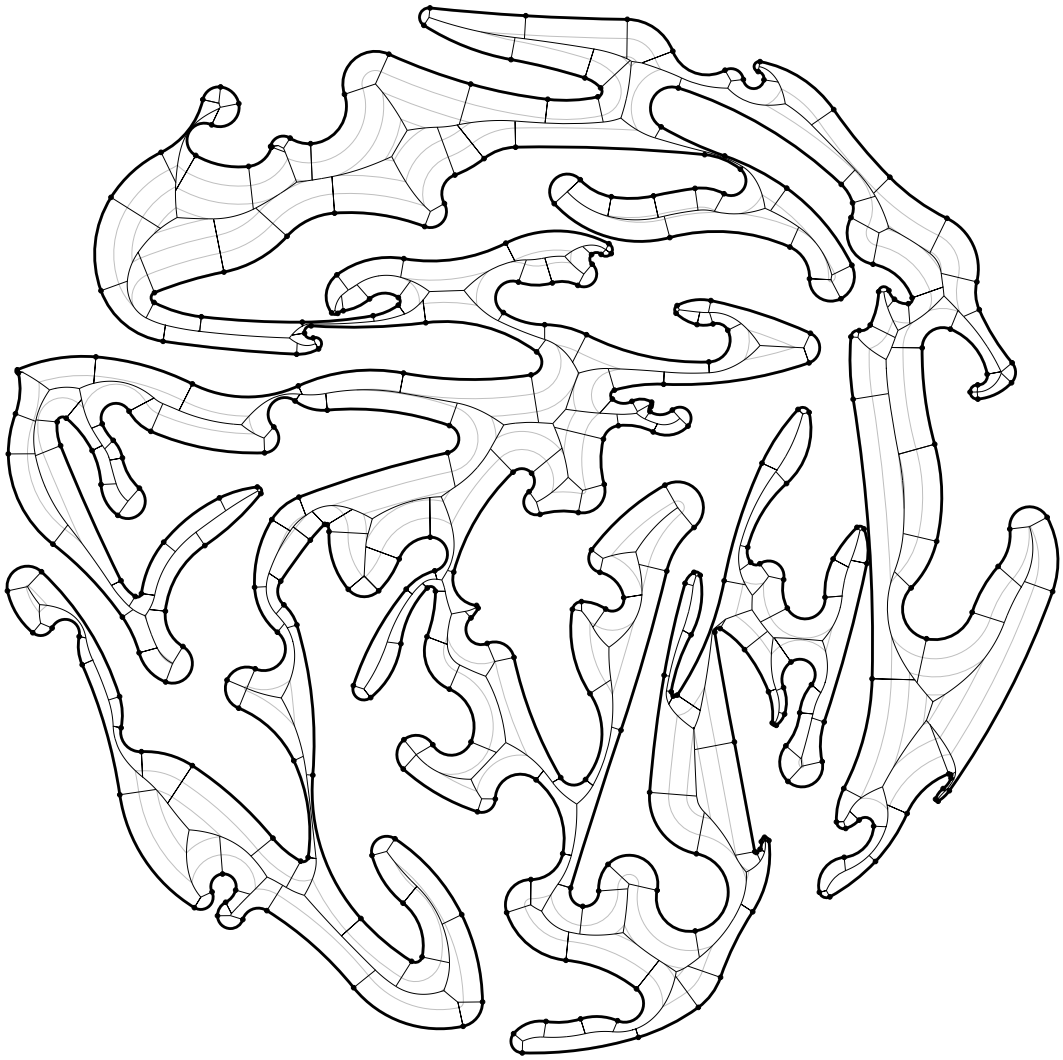

Figure B.3: A curvilinear dataset with 371 points, 0 segments and 371 arcs. The whole data set is a closed polygon consisting of tangential arcs.

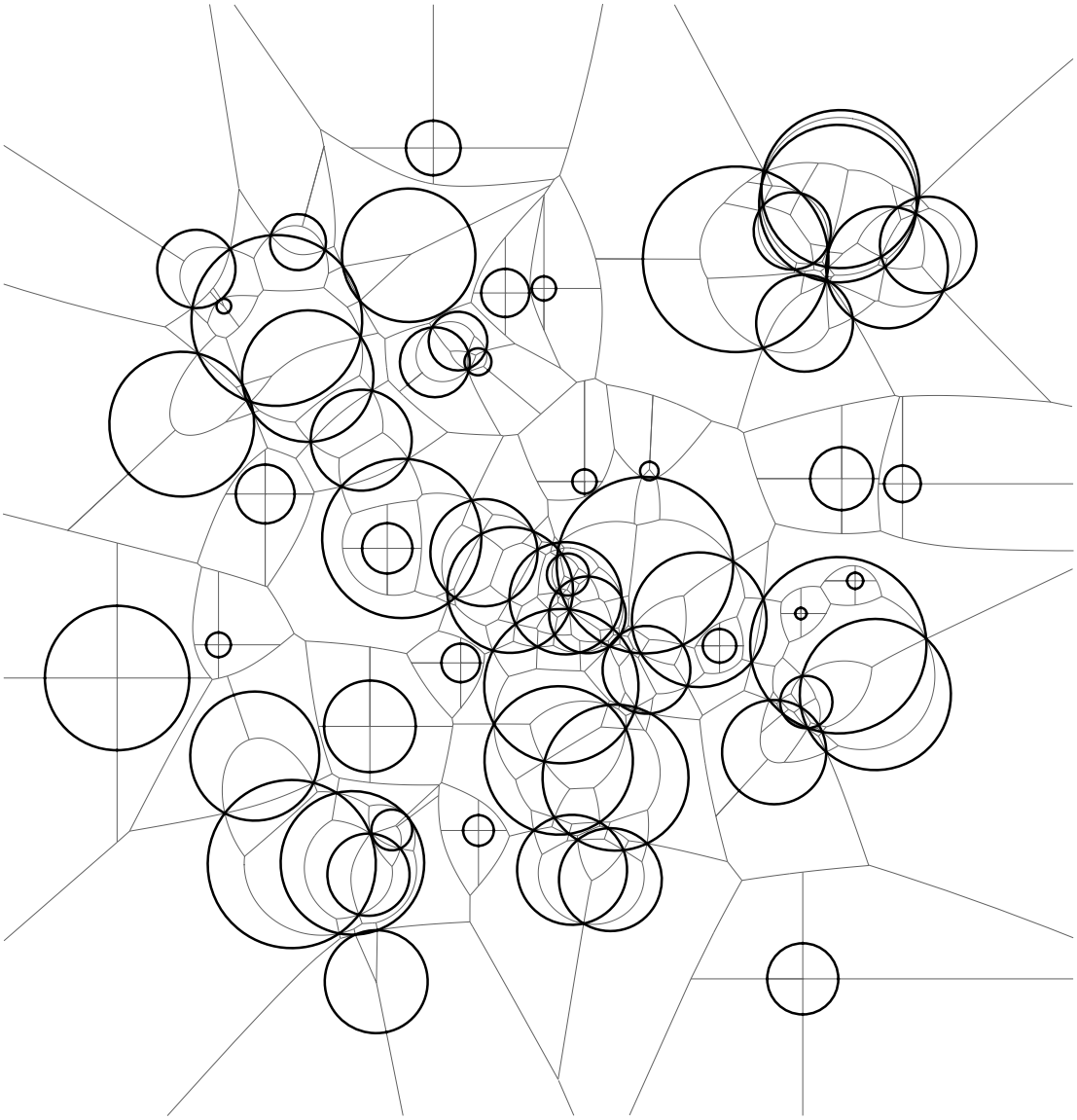

Figure B.4: The Voronoi diagram of random circles. A number of 60 random circles with uniformly distributed radii up to 0.1 and uniformly distributed center have been put in the unit square. As a preprocessing, the intersection-free overlap (arrangement), consisting of 352 arcs, has been extracted.

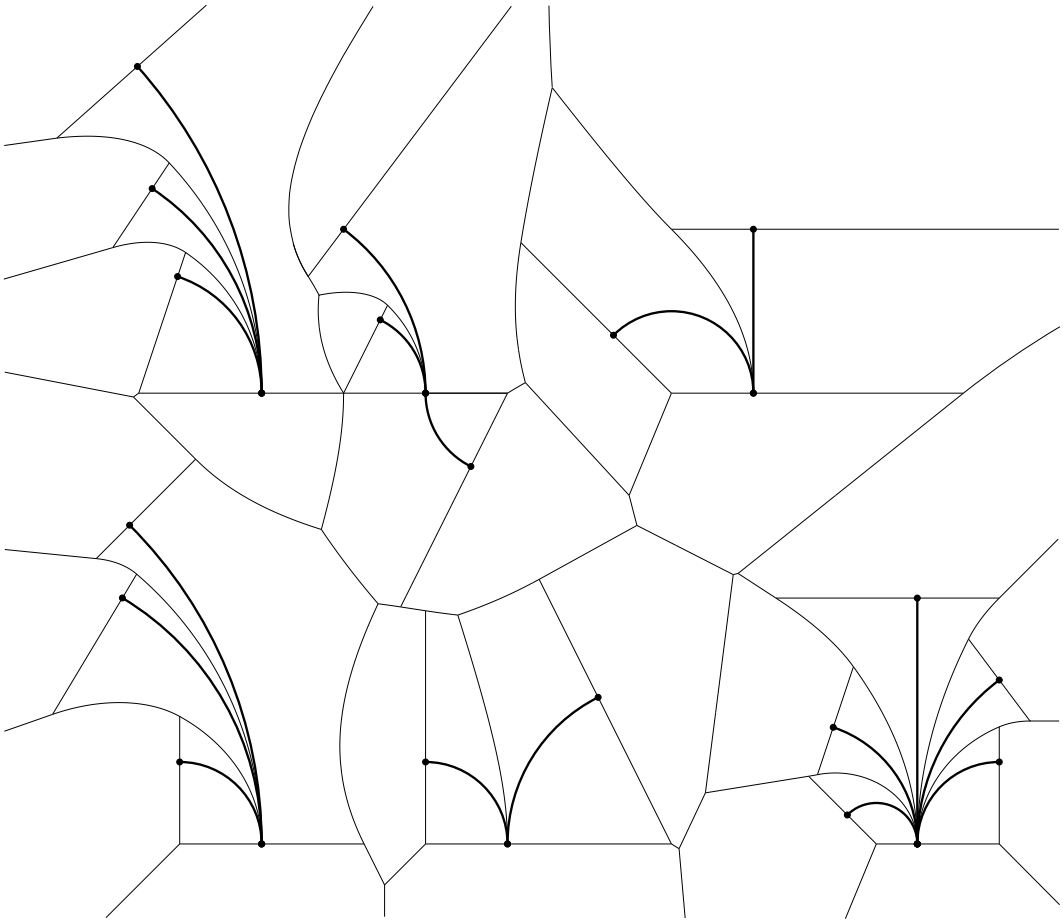

Figure B.5: A data set with tangential sites.

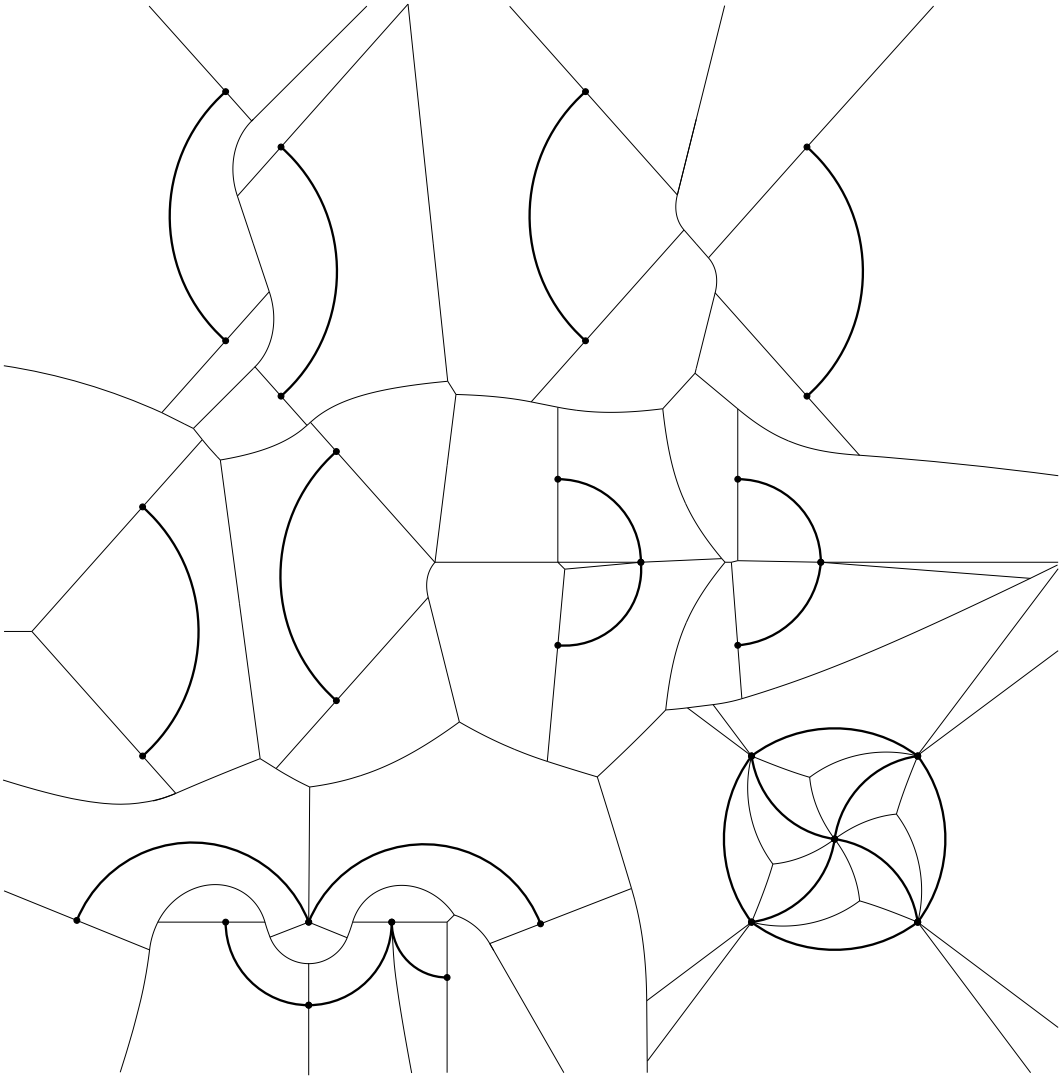

Figure B.6: A data set with all kinds of hyperbolic and elliptic bisector forms.

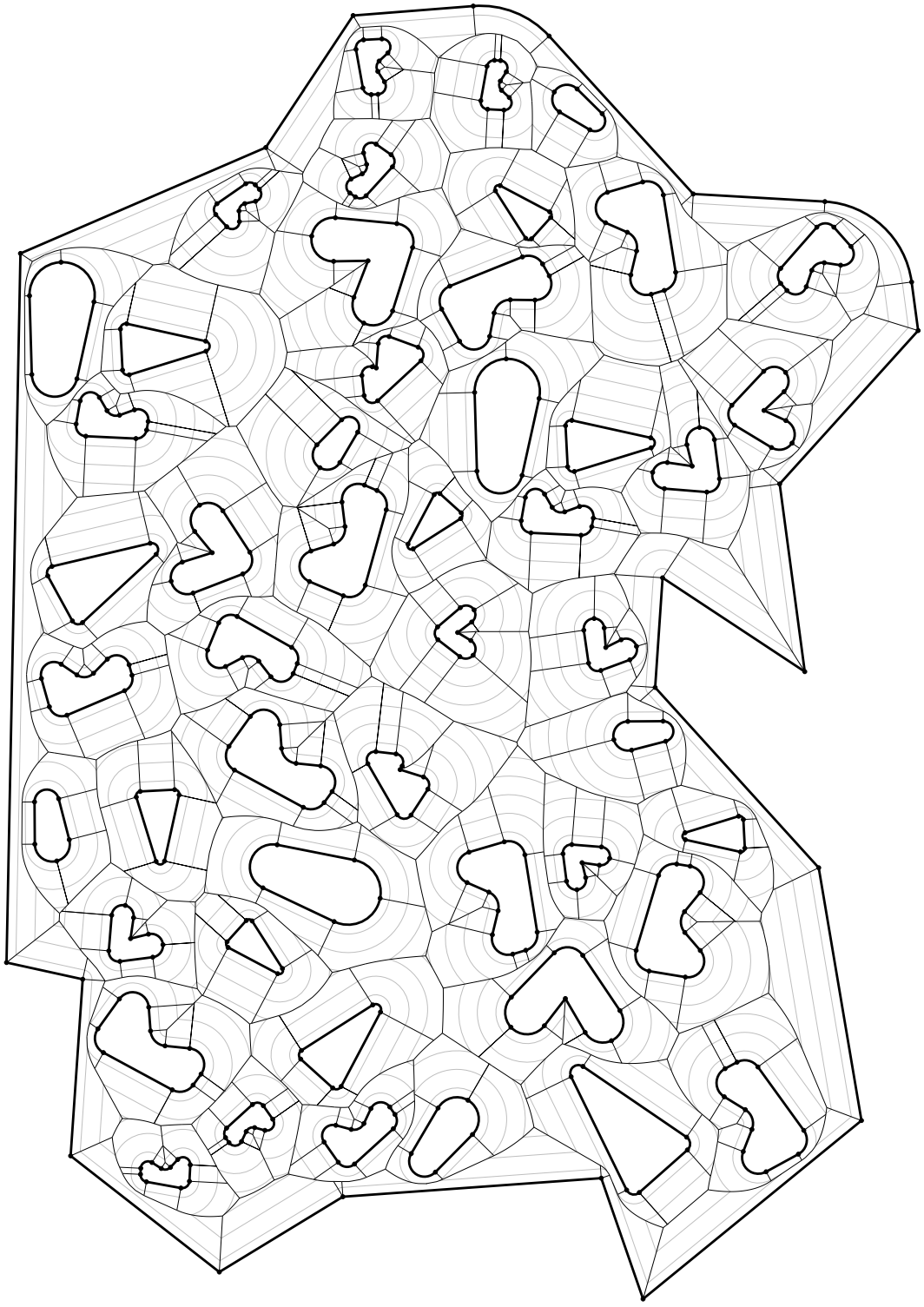

Figure B.7: A data set showing a polygon containing holes.

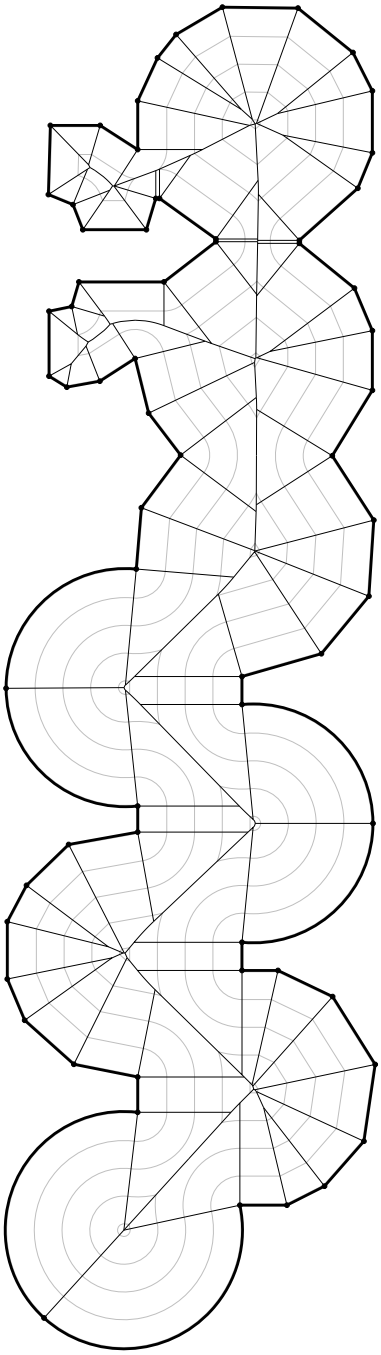

Figure B.8: Another data set showing a polygon without holes.

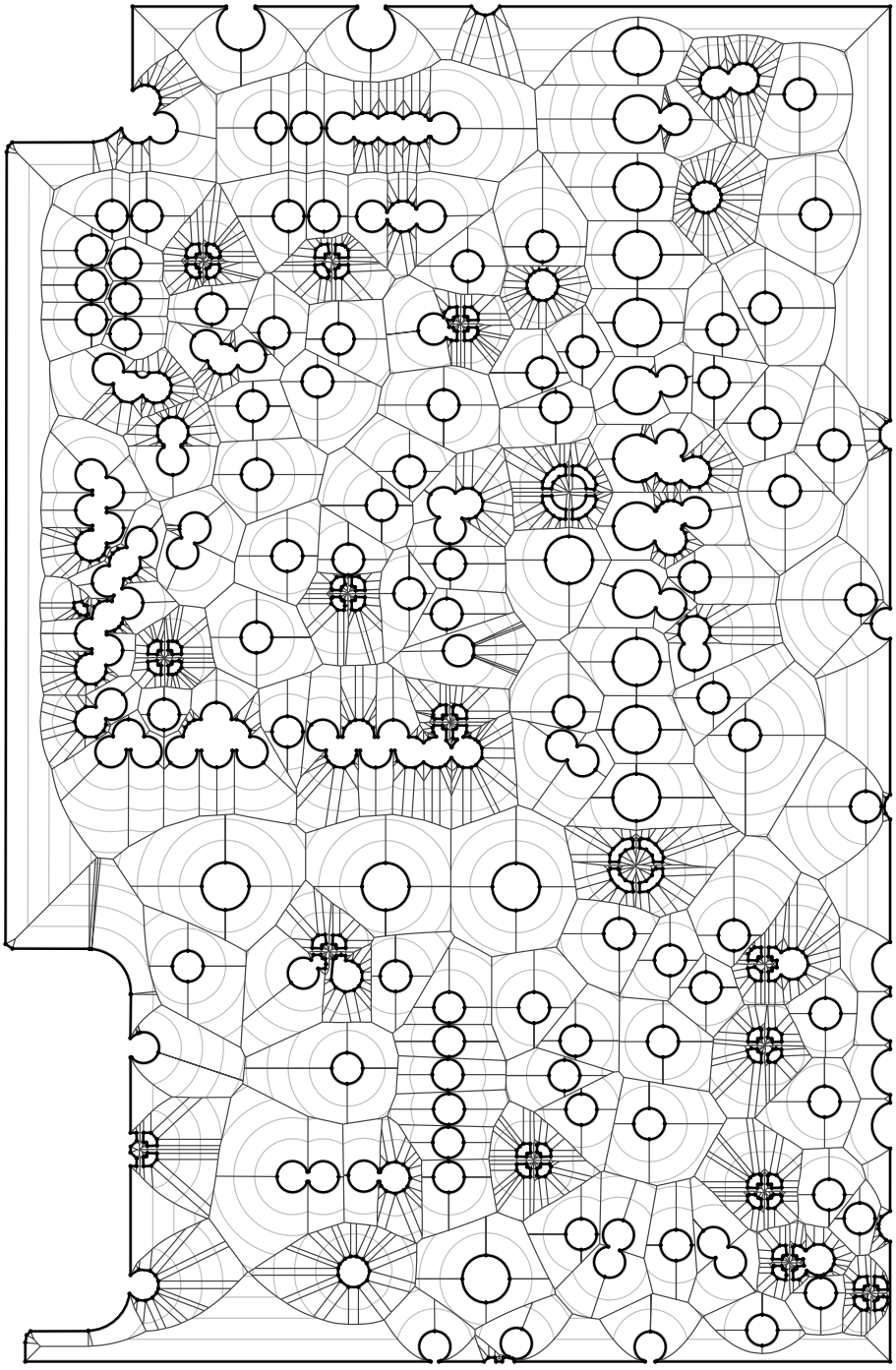

Figure B.9: A printed-circuit board (PCB) demonstrating the high occurrence of circular arcs.

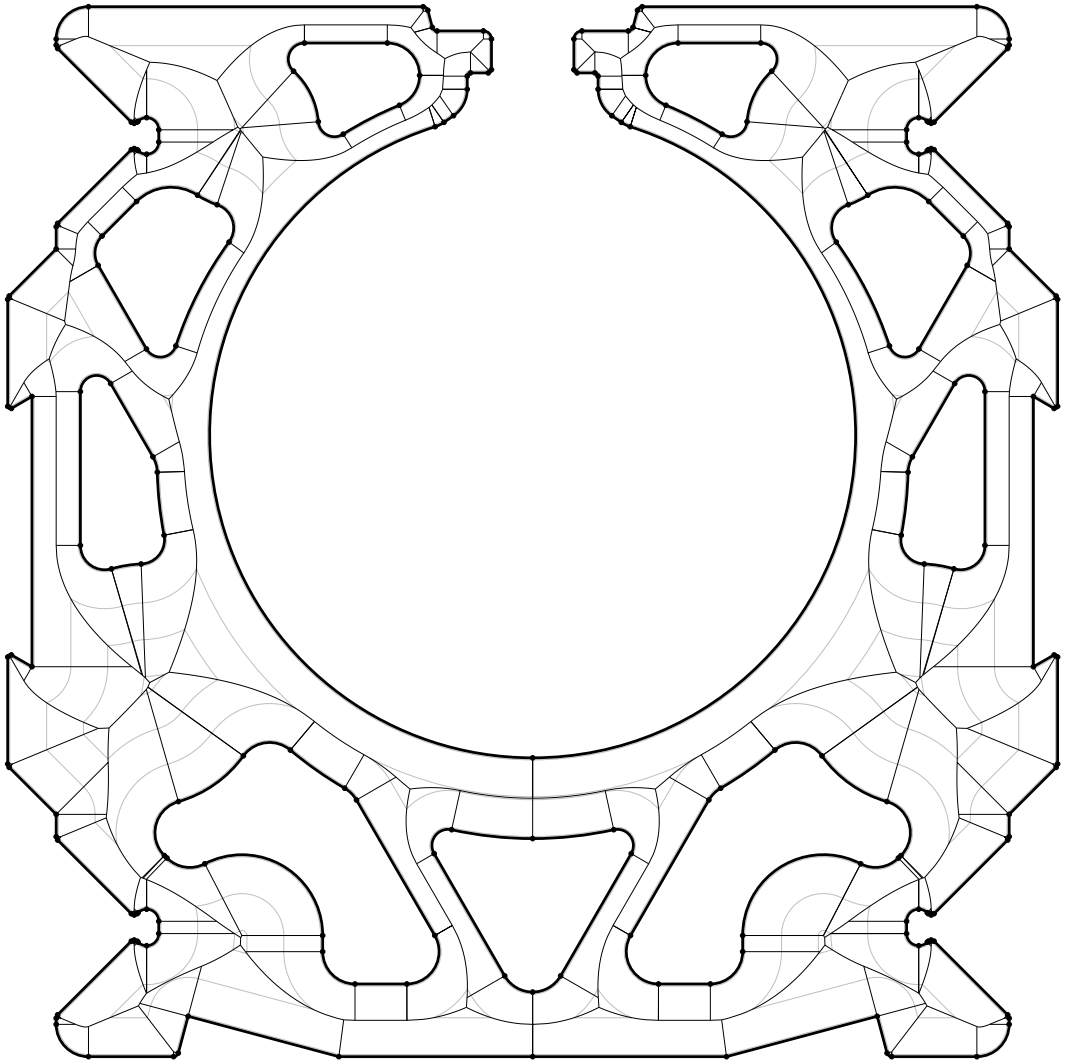

Figure B.10: An NC-machining data set.