A Constructive Approach to Sweep Algorithms for Voronoi Diagrams

by

Philipp A. Heuberger

Diploma, Swiss Federal Institute of Technology (ETH), Zürich, Switzerland, 1988

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science
in the Department
of
Computer Science

We accept this thesis as conforming
to the required standard

[redacted]
Dr. H. A. Müller, Supervisor

[redacted]
Dr. G. C. Shoja, Departmental Member

[redacted]
for Dr. R. Illner, Outside Member

[redacted]
Dr. K. Li, External

Supervisor: Dr. H. A. Müller

Abstract

Constructing the Voronoi diagram where the given sites are points in the Euclidean space veils the discrete problem of determining the finite set of *siteless subsets* of the original sites. The function *siteless* incorporates all geometric and the function *subset* all combinatorial concerns. The set of all siteless subsets is called the *Voronoi set.* Independent of dimension, our precise definition of the Voronoi diagram is based on the Voronoi set. Thus, the finite Voronoi set is a convenient abstraction of the infinite Voronoi diagram separating geometric from combinatorial concerns.

Furthermore, this thesis describes a constructive approach to Voronoi sweep algorithms. In a first attempt, we derive a brute-force algorithm demonstrating our methodology of strengthening program conditions. Then, two efficient algorithms for the plane are presented: the *sweephull* and the *sweepcircle* Voronoi algorithm. The *sweepcircle* algorithm is a companion of Fortune's sweepline algorithm, where the sweepline has been replaced by a *sweepcircle.* The *sweephull* algorithm, a generalized form of both sweepline and *sweepcircle* algorithms, is based on our improved understanding of the relationship between the convex hull and the Voronoi diagram.

Keywords: Voronoi diagram, sweep algorithms, well-founded sets, partial order.

Examiners:

[redacted]

Dr. H. A. Müller

[redacted]

Dr. G. C. Shoja

[redacted] (Oral chair)

for Dr. R. Illner

[redacted]

Dr. K. Li

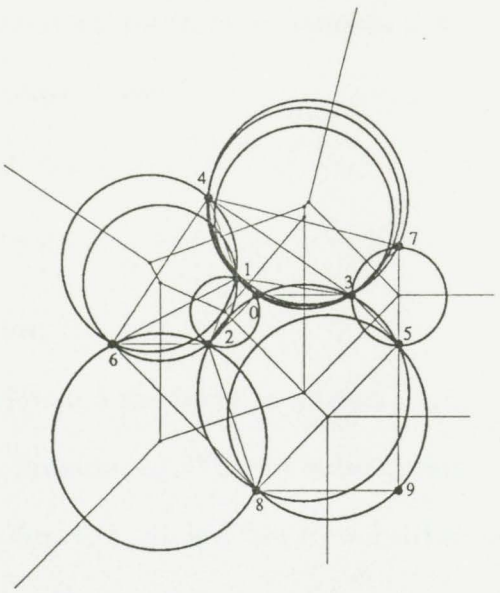

Figure 0.1: Empty-circle, Voronoi and Delaunay diagram

Contents

List of Tables

List of Figures

Acknowledgements

I wish to express my appreciation to my supervisor Dr. Hausi Müller for his guidance and support throughout my studies at UVic. For the discussions before and at my thesis defence, I want to thank my commitee, Dr. G. C. Shoja, Dr. R. Illner, Dr. K. Li, and the chair Dr. van den Driessche. For reading and commenting an early version of my thesis, I would also like to thank Dr. F. Ruskey, Dr. M. van Emden, Dr. N. Horspool, Jan Vitek, Paul Strooper, Rod Byrne, and Michael Whitney. In particular, I am indebted to Jim Uhl for his help in my struggle with English and Kenny Wong for many fruitful discussions about Voronoi diagrams and its fast computation in the plane. Special thanks go to Professor E. W. Dijkstra who took time to listen to my ideas during his visit in Victoria. And, lastly, I would like to thank all my friends and colleagues for all the good times I had during my stay on beautiful Vancouver Island.

"THANK YOU" to

- My office mates for their patience: Jim Uhl, Brian Corrie, Craig Sinclair, Scott Tilley, and Graeme Jones.
- The over-nighter chess, bicycle and tennis club: Mahbub Hassan, Anand Srinivasan, and Marco Escalante.
- The sushi gourmets: Yasu Koda and Kim Cheung.
- The intramural soccer team: Hausi Müller, Mehmet Orgun, Scott Tilley, Grant Marven, Peter Walsh, Zhen Liang Shi, and many more.
- The Vancouver Island section of the alpine club Canada: Sandy Briggs, John Pratt, and Dennis Manke.
- The engineering people in the Petchlab: Shankar Pennathur, Chintha Tellaumbura, Anwarul Hasan, and many more.
- The departemental joggers and marathon runners: Mike Whitney, Paul Strooper, Rod Byrne, Mehmet Orgun, Jimmy Lee, and Randal Tomczuk.
- The campers: Xiaoling Sun, Mehmet Orgun, Mahbub Hassan, Hassan Janoowalla, and Kim Cheung.
- The cinecenta addicts: Marco Escalante and Meenu Bhargava.
- The discussion partner: Eugene Neufeld.
- The Swiss Jass club: Ruth and Rennie Warburton, Johanna Held, Darlene Burnell, and Liz and Gunther Grambart.

Chapter 1

Introduction

The *Voronoi diagram* is an infinite geometric object that reflects proximity relations among a set of sites, usually points in Euclidean space. A two-dimensional example is depicted in Figure 1.1. The diagram divides the space into *Voronoi regions* such that all points in a region are closer to its site than to any other site. The vertices of the diagram are called *Voronoi vertices*, and its line segments are called *Voronoi edges* or, if unbounded, *Voronoi rays*. In the conventional view of Preparata and Shamos, a Voronoi region is regarded as the intersection of half-planes [15].

This thesis addresses three different areas of interest: computational geometry, program design theory, and programming environments.

- For the reader interested in computational geometry, Chapter 2 offers an alternative definition of Voronoi diagrams deviating from the conventional view. Based on our different understanding of the problem, we present two competitive algorithms called *sweephull* and *sweepcircle* for computing the Voronoi, Delaunay, and empty-circle diagram (see Figures 1.1, 1.2, and 1.3).

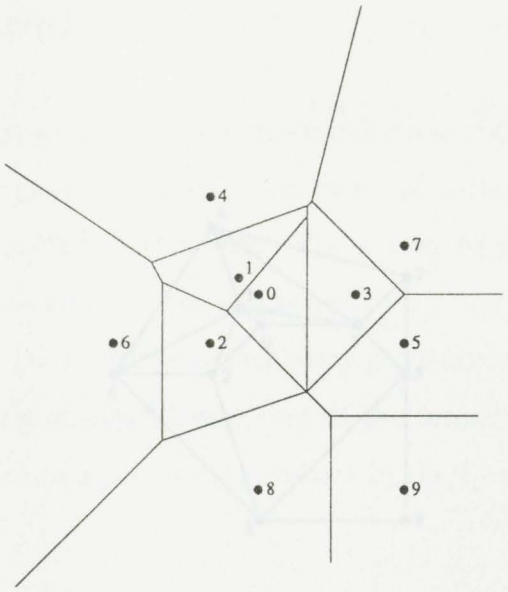

Figure 1.1: Voronoi diagram

- Software engineers might appreciate the underlying program design theory advocated by the books [29] and [30]. Their strong influence manifests itself in the structure of this thesis, separating problem specification (Chapter 2), algorithm derivation (Chapter 3) and program implementation (Chapter 4).

- The *sweephull* algorithm has been implemented in a new programming environment called Oberon [31]. The environment allowed a smooth realization of the algorithm.

The importance of Voronoi diagrams has been discussed extensively in the literature; however, we briefly summarize the algorithmic history of the Voronoi diagram.

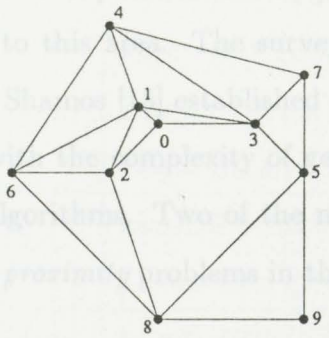

Figure 1.2: Delaunay diagram

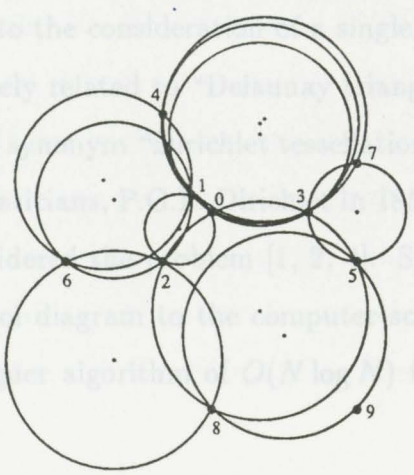

Figure 1.3: Empty-circle diagram

1.1 Background

Computational geometry, as an area of research in its own right, emerged in the early seventies. Since Shamos gave the discipline its name [5], a large number of computer scientists have been attracted to this area. The survey by Lee and Preparata [14] and the text by Preparata and Shamos [15] established computational geometry as a new field of study that deals with the complexity of geometric problems within the framework of the analysis of algorithms. Two of the major areas in computational geometry are *convex hulls* and *proximity* problems in the Euclidean space.

In this section, we review publications concerning the Voronoi diagram, which is related to proximity problems. Table 1.1 classifies the publications from an algorithmic point of view.

In a 1975 paper, Shamos and Hoey discussed a number of proximity problems [6] and reduced their treatise to the consideration of a single geometric structure called the Voronoi diagram. Closely related to "Delaunay triangulation," the Voronoi diagram is also known by the synonym "Dirichlet tessellation" and has its roots in the past century. The mathematicians, P.G.L. Dirichlet in 1850, G. Voronoi in 1907 and B. Delaunay in 1934, considered the problem [1, 2, 4]. Shamos and Hoey have not only introduced the Voronoi diagram to the computer science community, but also sketched a divide-and-conquer algorithm of $O(N \log N)$ for its computation in two dimensions.

An alternative to divide-and-conquer originates from the mathematical application of function approximation. Lawson, interested in surface interpolation to irregular-

Algorithmic approach	Reference
divide-and-conquer	Shamos and Hoey 1975, [6]
incremental	Lawson 1977, [7] Green and Sibson 1978, [8] Bowyer 1981 and Watson 1981, [11, 12] Ohya, Iri, and Murota 1984, [13]
divide-and-conquer versus incremental	Lee and Schachter 1980, [10] Guibas and Stolfi 1985, [16]
transformation from the plane to 3-D space and back	Brown 1979, [9] Edelsbrunner and Seidel 1986, [17]
transformation within the plane	Fortune 1986, [18]

Table 1.1: Algorithmic classification

ly located data, computes a triangular grid prior to calculating an estimate of the function value at a query point [7]. He proposes an $O(N^2)$ incremental algorithm for constructing the grid which is, in fact, a Delaunay triangulation. In the sequel, Green and Sibson succeeded in efficiently implementing the algorithm [8]. Independently, Bowyer and Watson were able to adapt the incremental method to three dimensions [11, 12] and to produce stereo-scopic pictures of the three dimensional diagram. Experimental results of Ohya, Iri, and Murota indicate that their algorithm, which uses a refined incremental method in two dimensions, has a linear average-time complexity.

Lee and Schachter [10], and later Guibas and Stolfi, reviewed both algorithmic approaches with emphasis on divide-and-conquer. Not until a decade after Shamos sketched the divide-and-conquer algorithm, Guibas and Stolfi finally specified it in

sufficient detail that implementation became practical [16].

Of theoretical interest is Brown's solution to the problem [9]. His fundamental result is that a D-dimensional Euclidean Voronoi diagram of N points can be constructed by projecting the points onto a $D+1$-dimensional hypersphere, constructing the $D+1$-dimensional convex hull of the transformed points, and then transforming the convex hull back to D-space resulting in the Voronoi diagram. Edelsbrunner and Seidel further pursued the idea of relating Voronoi diagrams to convex hulls of higher dimension [17].

One of the latest discoveries is Fortune's novel sweepline algorithm [18], as surveyed by O'Rourke [21]. Fortune distorts the plane with a transformation so that the deformed Voronoi regions are first entered when their site is encountered. Based on this clever planar transformation, the sweepline algorithm achieves $O(N \log N)$ worst-case performance while avoiding the complex merge step of the divide-and-conquer algorithm.

Applications of the Voronoi diagram include the already mentioned function interpolation which has practical use particularly in terrain fitting or in other areas such as geology, geography, and meteorology. This explains Watson's geophysical background and his effort to implement a three-dimensional algorithm [12]. Principally, applications are found in every science which is somehow related to Euclidean space and proximity (e.g., two extremes: astronomy and crystallography). Another application area concerns the decomposition of polygons into convex sets, yielding a research field called morphology [22]. Researchers in that field expect to bring forth advances in pattern recognition. A more complete list of applications with detailed

examples and references may be found in [10].

Various generalizations besides higher dimensions have been proposed. One question of interest is: what does the Voronoi diagram on the surface of a sphere look like? This leads to the use of metrics other than the Euclidean one. Another generalization is to consider weighted sites, line segments, or objects with even more complicated shapes. Finally, we mention the definition of the k-order Voronoi diagram. For any point inside a region of the k-order Voronoi diagram, the k nearest neighbors are precisely the k sites associated to the region. In this thesis, however, we limit the discussion to the one-order Voronoi diagram.

1.2 Specifying the Voronoi diagram

To open the discussion, this section contrasts the thesis to existing knowledge and current research in computational geometry. We draw the reader's attention to flaws in the prevalent theory of Voronoi diagrams and then try to resolve the provoked controversy.

In the conventional view of Voronoi diagrams, there exist points that do not belong to a particular region. In Figure 1.1, for example, all points on a line segment belong either to no region at all or to several. We remove this ambiguity and understand the Voronoi diagram in a more general sense, such that it actually partitions the space. A Voronoi edge in Figure 1.1 is regarded as a separate region in the next lower dimension.

Restricted to two dimensions, the Delaunay triangulation is usually called the

"straight-line dual" of the Voronoi diagram [15]. In the presence of degeneracies, however, the one-to-one correspondence (implied by the notion of duality) of a Voronoi edge in the Voronoi diagram and a Delaunay edge in the Delaunay triangulation does not hold. Therefore, Preparata and Shamos take the necessary precaution in assuming that "no four sites are cocircular." We eliminate this "big" assumption and do not force a Delaunay triangulation, preserving the one-to-one correspondence. As shown in the Delaunay diagram of Figure 1.2, four or more cocircular sites outline a convex polygon and not several arbitrary triangles. Consequently, we emphasize the property of the empty circles [4], illustrated in Figure 1.3, or more generally, the identity of a Voronoi vertex and the center of an "empty hypersphere" (i.e., a hypersphere[1] containing no other site).

We further avoid the term "straight-line dual." It is ill-defined in this context and suggests that there exists an isolated function (i.e., a function with a single edge parameter) and its inverse that can transform a Voronoi edge into a Delaunay edge and vice versa. Even if (by the "big" assumption) a one-to-one correspondence between Voronoi and Delaunay edges holds, such a function with a single edge as its argument does not exist.

In Chapter 2, we precisely define two boolean functions *equi* and *siteless* on subsets of the original set of sites. Our focus is on the discrete space of all the subsets, first constrained by *equi* and then by *siteless*. A subset for which the function *equi* evaluates to true is called an *equiset* and represents one or more "hyperspheres." A *siteless equiset* represents one or more "empty hyperspheres." Similar ideas are found in a paper by Watson [12]. Our approach differs from his in that a *set* of sites (rather

[1]Subsequently, the prefix "hyper" denotes independence of dimensions.

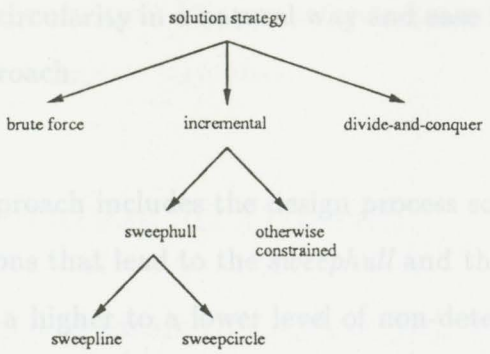

Figure 1.4: Derivation tree

than a site-tuple) denotes a hypersphere. This small change has a consequence which relates the geometric notion of "convex hull" in Euclidean space to the combinatorial notion of "subset" in the space of *siteless equisets*.

The partitioning property of the Voronoi diagram and the relation between "convex hull" and "subset" are captured in two conjectures, which intuitively hold for the one-, two-, and three-dimensional Euclidean space. Parallel to our work, Elbaz and Spehner developed a similar theory [25]. However, their approach is less general and more complicated than the one presented here.

1.3 Constructive program design

Many algorithms have been proposed for the computation of the Voronoi diagram. We distinguish three categories: divide-and-conquer [6], incremental [8], and sweep algorithms [18] as depicted in Figure 1.4. Although efficient, the implementation of a divide-and-conquer algorithm tends to be complex and error-prone due to degeneracies. Thus, we narrow our focus to incremental and sweep algorithms, which deal

with collinearity and cocircularity in a natural way and ease the understanding of the divide-and-conquer approach.

Our constructive approach includes the design process so that the reader can appreciate the considerations that lead to the *sweephull* and the *sweepcircle* algorithms. All derivations go from a higher to a lower level of non-determinism and every level is associated with one or more conditions which are strengthened in the course of a derivation. A derivation is rooted at a general solution strategy which evolves from the problem specification. The tree in Figure 1.4 reflects the dependence among the algorithms.

An initial combinatorial brute-force derivation demonstrates the methodology used. A more sophisticated derivation reveals the *sweephull* algorithm. It is motivated by relaxing the condition on the sequence of sites processed by the sweepline and *sweepcircle* algorithms. That is, every newly encountered site is required to be outside the convex hull of the processed sites. Informally, the algorithm applies the technique of "Dirichlet tessellation" near the convex hull of the sites already processed so that every new Voronoi region is unbounded. The Voronoi diagram is represented by a graph where the nodes correspond to *siteless equisets*. This graph, called the *Voronoi graph*, is shown to be well-founded and the equisets associated with hyperspheres that intersect at least one edge of the convex hull form a partially ordered set. Therefore, all *siteless equisets* that might be affected by an update form a partially ordered set. Thus, the termination argument of the sweephull algorithm is based on the theory of well-foundedness as presented in [30].

The derivation of the mathematical formulas necessary for the *sweepcircle* algo-

rithm are presented in Appendix A. The presentation is similar to Fortune's style for readers familiar with his sweepline algorithm.

1.4 Implementation in Oberon

The sweephull algorithm is implemented in Oberon [31], the successor of Modula-2 and Pascal. The implementation is convenient, simple, and fast.

The idea of strengthening conditions on the program state also applies to the implementation. A derivation step, however, is now dictated by properties of the used hardware architecture and programming environment and not by properties of the problem. The high-level Oberon language and environment imposes few artificial constraints. It also facilitates this last step before practical use with its powerful type system. Our simple implementation of the sweephull algorithm models the Voronoi graph by a dynamic data structure based on circular lists. The need for a priority queue, present in its counterparts, sweepline and sweepcircle, is avoided and floating-point arithmetic is not necessary, even though it is currently used.

Chapter 2

Specification and solution

This chapter specifies the problem of constructing the Voronoi diagram, separates geometric and combinatorial concerns, and presents the general solution strategy.

In this specification, two different spaces are considered: the geometric space of points and the combinatorial space of site subsets. A restricted set of subsets, called the *Voronoi set*, contains all relevant information of the Voronoi diagram. The motivation for our specification is the ease of handling degeneracies (i.e., collinearity and cocircularity) and independence of dimension. An example illustrates the presented definitions for the special case of two-dimensional Euclidean space and demonstrates the validity of our specification. Our general solution strategy then concentrates on finding the *Voronoi base* from which the *Voronoi set* can be generated.

Also, we feel free to experiment with our notation. The reason why we deviate from traditional set notation is to improve our skill of manipulating logical formulae and to gain more freedom in precisely formulating our ideas. We carefully designed this

chapter so that the readers can familiarize themselves with our notational conventions gleaned from a recent monograph by Dijkstra and Scholten [30].

2.1 Preliminary definitions and notation

Let P be a sequence of N distinct points in the D-dimensional Euclidean metric space $\mathcal{R}^D$ and let S be the set of indices to the points in P (i.e., $0 \ldots N-1$). Subsequently, the set S is referred to as the *sites*.

In the following, we use a function notation, advocated by Dijkstra. The infix period explicitly indicates function application.

Ex.	$p.j$	function p applied to parameter j
	$dst.(p, P.s)$	function dst applied to parameter pair p and $P.s$
	$DST.D.(p, q)$	function DST applied to parameter D yielding a new function which is applied to the parameter pair (p, q)

Function application is left-associative. Thus, $DST.D.(p, q)$ is equal to $(DST.D).(p, q)$.

Function $dst.(p, q)$ denotes the Euclidean distance between two points p and q in $\mathcal{R}^D$. A coordinate j of point p is referred to as $p.j$ where p is interpreted as a function which returns the point's jth coordinate.

$$DST.D.(p, q) = \sqrt{\sum_{j=0}^{D-1} (q.j - p.j)^2} \tag{2.1}$$

Since the dimension D is rarely altered, we write *dst* instead of $DST.D$. Thus, *dst* hides the fact that the distance is dependent on the dimension. Similar to the function application $p.j$, which returns a coordinate, the function application $P.s$ returns a point. Thus, $dst.(p, P.s)$ denotes the distance between point p and point $P.s$, or more conveniently, between point p and site s.

The final remark regarding notation concerns the equality symbol $=$. Equality being applied to two boolean operands is denoted by the equivalence symbol $\equiv$. Our main reason for introducing this symbol is to ease the type association of the equality operands, thereby improving the readability of boolean expressions. Also, the equivalence symbol has the lowest binding power of all logical connectives and parentheses can be saved. Parentheses are exclusively used to delineate scopes of local variables or parameter tuples.

2.2 Characteristic functions

This section introduces two functions that are used in the subsequent definition of the Voronoi set and the Voronoi diagram (i.e., *siteless* and *sitelessPoints*).

The boolean function *equi* applied to a subset nS of S returns true, if there exists a point p such that its distance k to each site in nS is constant.

$$equi.nS \equiv (\exists p\colon\ p \in \mathcal{R}^D\colon\ (\exists k\colon\ k \in R\colon\ (\forall s\colon\ s \in nS\colon\ dst.(p, P.s) = k)))$$

The variables p, k, and s are local to their matching scopes which are delineated by corresponding parenthesis pairs.

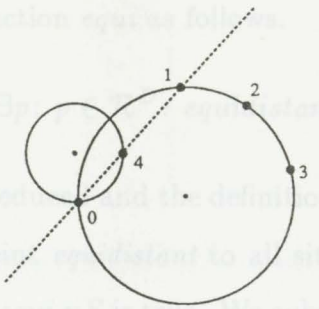

Figure 2.1: *equi*: {04}, {0123} and ¬*equi*: {1234}, {014}

Informally, the function *equi* applied to nS, returns true, if all the members (sites) of nS lie on some hypersphere, and false otherwise. The local variables p and k represent the center and the radius of such a hypersphere. Moreover, the function *equi* is a generalization of non-collinearity and cocircularity (e.g., *equi* holds for three non-collinear sites or four cocircular sites). For example, assume the site configuration in Figure 2.1. Then, the function applications, *equi*.{04} and *equi*.{0123}, both evaluate to true. However, *equi*.{1234} and *equi*.{014} both evaluate to false. (For convenience, we omit commas in subsets if possible).

Note that the *equi* function is asymmetric with respect to p and k. For a given radius k, there might be multiple centers. However, for a given center p, there is at most one radius k. Hence, k is given its own scope within p's scope. Furthermore, we emphasize k's dependence on p and introduce the boolean function *equidistant*. Applied to a point set nS and a point p, *equidistant*.(nS, p) returns true, if all sites in nS have equal distance to p.

$$equidistant.(nS, p) \equiv (\exists k\colon\ k \in \mathcal{R}\colon\ (\forall s\colon\ s \in nS\colon\ dst.(p, P.s) = k))$$

Hence, we can rewrite the function *equi* as follows.

$$equi.nS \equiv (\exists p\colon\ p \in \mathcal{R}^D\colon\ equidistant.(nS, p))$$

Thus, the importance of k is reduced and the definition now reads: "$equi.nS$ is equivalent to the existence of a point *equidistant* to all sites in nS." The subset of sites nS is said to be an *equiset*, if $equi.nS$ is true. We subsequently write parameter e for an equiset instead of nS.

An equiset can be interpreted as a hypersphere. The function $equiPoints$ provides another interpretation of an equiset. Let the value of function $equiPoints.e$ denote the set of points p equidistant to the sites in the equiset e. The defining equivalence of $equiPoints$ is obtained by simply omitting p's existential quantifier, resulting in an expression which denotes a set of points rather than a single boolean value.

$$equiPoints.e = (p\colon\ p \in \mathcal{R}^D\colon\ equidistant.(e, p))$$

Note that the center p of a hypersphere in the definition of the function *equi* is not necessarily unique. For example, given two sites in $\mathcal{R}^D$, there is an infinite number of possible hyperspheres; their centers are on a $D-1$-dimensional hyperplane as illustrated in Figure 2.2. Thus, the interpretation of an equiset provided by $equiPoints$ is the hypervolume of points consisting of all possible centers of the hyperspheres mentioned above. Moreover, $equi.e \equiv equiPoints.e \neq \phi$.

Let $SITELESS$ be a function which, if applied to the sites S, yields the function *siteless*.

$$siteless = SITELESS\,.\,S$$

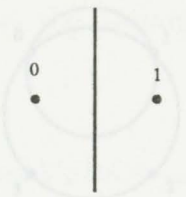

Figure 2.2: $equiPoints.\{01\}$ is a $D-1$-dimensional hyperplane

Applied to an equiset e, *siteless* itself returns true, if there exists a point p equidistant to the sites in e such that the Euclidean distance between p and any of the remaining sites in S is strictly greater than the distance between p and any site in e.

$$siteless.e \equiv (\exists p\colon p \in \mathcal{R}^D\colon equidistant.(e,p) \wedge (\forall s\colon s \in (S-e)\colon dst.(p,P.(anySite.e)) < dst.(p,P.s)))$$

$S - e$ is the complement of e and $anySite$ returns a random site of e. Note that S is a global parameter to *siteless*.

Observe that a hypersphere exists, if equidistant is true; furthermore, this hypersphere is empty, if siteless is true. For example, *equi* holds for the set of sites $\{0123\}$ in Figure 2.1 but *siteless* does not. Note that *siteless.e* returns false if e is not maximal. For example, let S be the set of the four sites $\{0123\}$ in Figure 2.3. Then, $siteless.\{0123\}$, $siteless.\{01\}$, $siteless.\{12\}$, $siteless.\{23\}$, and $siteless.\{30\}$ evaluate to true, but $siteless.\{012\}$, $siteless.\{123\}$, $siteless.\{230\}$, and $siteless.\{301\}$ do not because of the strict inequality. Thus, an empty hypersphere is uniquely represented by one siteless equiset.

Similar to the function $equiPoints$, we define $sitelessPoints$ to return all the

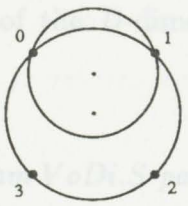

Figure 2.3: *siteless*: $\{0123\}, \{01\}, \{12\}$ and $\neg$*siteless*: $\{012\}, \{123\}$

points p for which *siteless* evaluates to true.

$$sitelessPoints.e \;=\; (p\colon p \in \mathcal{R}^D\colon equidistant.(e,p) \wedge (\forall s\colon s \in (S-e)\colon dst.(p,P.(anySite.e)) < dst.(p,P.s)))$$

Thus, $sitelessPoints.e$ is a subset of $equiPoints.e$. Note that $sitelessPoints.e$ can also be an infinite set of points. For example in Figure 2.2, $sitelessPoints.\{01\}$, which is equal to $equiPoints.\{01\}$, represents the infinite set of points on the hyperplane perpendicular to the line segment between sites 0 and 1.

2.3 The Voronoi set and its properties

In this section, we present the definition of the *Voronoi set* together with three conjectures. The validity of the specification and its relevance to our problem is conjectured because a strictly formal proof is beyond the scope of this thesis.

The set of all siteless equisets $Vs.S$ is referred to as the *Voronoi set*.

$$Vs.S = (e\colon e \subseteq S\colon siteless.e)$$

The Voronoi diagram $VoDi.S$ is defined as the collection of all hypervolumes specified by $sitelessPoints.e$.

$$VoDi.S = (sitelessPoints.e\colon e \in Vs.S)$$

Given this exact characterization of the D-dimensional Voronoi diagram, we now claim our first conjecture.

Conjecture 1 *The Voronoi diagram $VoDi.S$ partitions $\mathcal{R}^D$.*

Each siteless equiset in the finite space of site subsets corresponds to one hypervolume, which consists of points in Euclidean space composing one part of the Voronoi diagram. Moreover, we claim that these hypervolumes are disjoint and that their union is $\mathcal{R}^D$.

A two- or three-dimensional Voronoi diagram is depicted by assigning a color $c.|e|$ to each point p in the hypervolume $sitelessPoints.e$. In stereo-scopic pictures as shown in [11, 12] or in Figure 1.1, we define $c.n$ in terms of dimension D.

$$c.n = \begin{cases} transparent & \text{if } n < D \\ black & \text{if } n \geq D \end{cases}$$

In preparation for Conjectures 2 and 3, we define the functions *convexPoints* and *BOUNDARY*. The function *convexPoints* is applied to a finite set of points fs of $\mathcal{R}^D$ and returns an open point set which does not include the boundary and therefore not all points of fs.

$$\begin{aligned} convexPoints.fs \;=\; & (p\colon p \in \mathcal{R}^D\colon (\exists \lambda\colon \text{function of type } (\mathcal{N} \to \mathcal{R})\colon \\ & (\forall i\colon 0 \leq i < |fs| - 1\colon 0 < \lambda.i) \wedge \sum_{i=0}^{|fs|-1} \lambda.i = 1\colon \\ & p = \sum_{i=0}^{|fs|-1} \lambda.i \times fs.i)) \end{aligned}$$

The function *convexPoints* differs slightly from the notion *convex set* used in set theory or the analogous function *conv* in computational geometry. The coefficients $\lambda.i$,

which generate *convex combinations* within the function $convexPoints$, are strictly positive. Note that $convexPoints.fs$ returns an infinite point set, if $|fs| > 1$ or a singleton set, if $|fs| = 1$.

In computational geometry, we compute the convex boundary of a finite point set instead of its content. Therefore, let us introduce the function $BOUNDARY$ which, if applied to the function $convexPoints$, yields a function returning the convex boundary of a finite point set fs. The following formula implicitly defines the function $BOUNDARY$.

$$(\forall S\colon S \subset \mathcal{R}^D\colon (\forall fs\colon S \subseteq convexPoints.fs\colon$$
$$BOUNDARY.convexPoints.S \subseteq (convexPoints.fs - convexPoints.S)))$$

Note that the function $BOUNDARY$ applies to a function of type $\alpha = (fs \rightarrow ps)$ where fs denotes a finite and ps a possibly infinite point set. Then, the type of function $BOUNDARY$ is described by $(\alpha \rightarrow \alpha)$.

Preparata and Shamos introduced the function CH informally as "the complete description of the boundary $CH(S)$." According to common use, both, CH and $conv$, are referred to as the *convex hull* of S. In the context of this thesis, this usage is rather misleading. We therefore emphasize the intuitive meaning of "hull" and if we use the term "convex hull," we mean $CH(S)$, the boundary.

Using $BOUNDARY$, we precisely define the function CH.

$$CH = BOUNDARY.convexPoints$$

The main reason for introducing $BOUNDARY$, however, is to apply it to the function $sitelessPoints$ to formulate Conjecture 2.

Conjecture 2

$$(\forall e\colon e \in Vs.S\colon$$
$$BOUNDARY.sitelessPoints.e = (sitelessPoints.t\colon t \in Vs.S\colon e \subset t))$$

We claim that the disjoint hypervolumes are convex and that the convex hull of one such hypervolume is itself composed of convex hypervolumes where each one corresponds to one of the siteless equisets. Furthermore, note that the composition on the right-hand side of the equation is based on the subset operator. Thus, this conjecture relates "convex hull" on the left-hand side of the equation to "subset."

In addition, we define the D-dimensional Delaunay diagram. The definition relates the Delaunay diagram $DeDi.S$ to the Voronoi diagram $VoDi.S$. Moreover, it shows the dual nature of the functions $convexPoints$ and $sitelessPoints$.

$$DeDi.S = (convexPoints.e\colon e \in Vs.S)$$

Similar to the Voronoi diagram, we claim that the hypervolumes $convexPoints.e$ of the Delaunay diagram are disjoint.

Conjecture 3 *The Delaunay diagram $DeDi.S$ partitions the union of $convexPoints.S$ and $BOUNDARY.convexPoints.S$.*

In drawing a two- or three-dimensional Delaunay diagram, the color black is assigned to each point p in the point set $convexPoints.e$, only for siteless equisets e of cardinality one and two.

In one, two, and three dimensions, all three conjectures hold intuitively; informal or partial proofs have been given before [15, 25]. Based on the Euclidean metric,

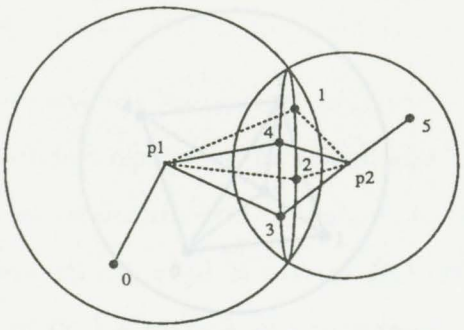

Figure 2.4: Four cocircular sites as a common subset

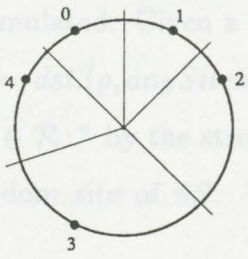

Figure 2.5: Set {03} is a subset but not siteless

predicate calculus, and the presented definitions, we are reasonably confident that all three conjectures can be proven formally for any dimension. Section 2.5 presents an example illustrating these conjectures.

2.4 Generating the Voronoi set from its Voronoi base

Conjecture 2 allows a simplified solution strategy whereby only the Voronoi base, a subset of the Voronoi set, has to be computed to obtain the Voronoi diagram.

In the function *equidistant*, introduced earlier, the dependence of k on p is actu-

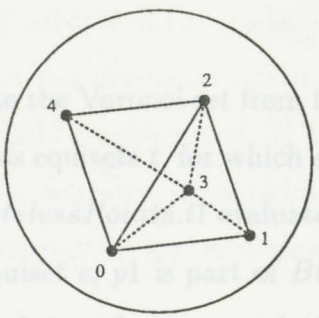

Figure 2.6: Five sites on a sphere

ally stronger than previously formulated. Given a point p, there exists at most one distance k with the property $k = dst.(p, anySite.nS)$. That is, we can replace k's existential quantification "$\exists k\colon\ k \in \mathcal{R}\colon$" by the stronger "$k = dst.(p, anySite.nS)\ ::$" where $anySite.nS$ returns a random site of nS. This manipulation results in the following formula.

$$equidistant'.(nS, p) \equiv (k = dst.(p, P.(anySite.nS)) :: (\forall s \in nS\colon\ dst.(p, P.s) = k))$$

Note that both expressions are of type boolean and that the domain (i.e., $k \in \mathcal{R}$) can be omitted, since it is clear that *dst* returns a real number. However, there is a small difference between *equidistant* and *equidistant'*: if nS is the empty set, the function *anySite* is not defined. Thus, we restrict nS to be a nonempty subset of S.

Evaluating the function *siteless* is impossible in general (i.e., for an infinite number of equidistant points). Let us therefore focus on the situation where there is a unique equidistant point. The *Voronoi base* $Vb.S$ is the subset of siteless equisets for which there is a unique equidistant point p.

$$Vb.S = (e\colon\ e \subseteq S\colon\ siteless.e \wedge |e| > D)$$

Thus, $sitelessPoints.e$ is a singleton set, if e is a member of the Voronoi base $Vb.S$.

Let us show how to complete the Voronoi set from the Voronoi base assumed that we are able to find all the siteless equisets t, for which $sitelessPoints.t$ evaluates to a single point. For example, let $sitelessPoints.t1$ evaluate to the set $\{p1\}$. Conjecture 2 claims that for some siteless equiset e, $p1$ is part of $BOUNDARY.sitelessPoints.e$, where e is a proper subset of $t1$. Let us find an e such that e is of maximal cardinality and $sitelessPoints.e$ forms a line. Then, $p1$ is a boundary point of the line. With luck, we might find another t, say $t2$, and another p, say $p2$, with $t2$ containing the same subset e. Then, $convexPoints.\{p1, p2\}$ is equal to $sitelessPoints.e$. We say that e is a *common subset* of $t1$ and $t2$.

$$(e \text{ is a } \textit{common subset} \text{ of } (t1, t2)) \equiv e \subset t1 \wedge e \subset t2$$

Consider the three-dimensional example in Figure 2.4.

$$\{1234\} \text{ is a } \textit{common subset} \text{ of } (\{01234\}, \{12345\})$$

Although the above process can be continued recursively to generate the Voronoi set, the siteless equisets e with cardinality greater than D are of special interest. First, only the points in $sitelessPoints.e$ are actually drawn (i.e., colored in black). Second, there are at most two siteless equisets $t1$ and $t2$ which are supersets of e. Third, they are the basis for the well-foundedness of the Voronoi base (see Chapter 3).

Unfortunately, there is a drawback since the generated set is incomplete. From the interpretation of $sitelessPoints$, equisets related to unbounded hypervolumes, or, from the interpretation of $convexPoints$, equisets related to hypervolumes of the convex hull of S are discarded.

One might therefore take all subsets of the equisets in the Voronoi base, but this does not work either. Consider the degenerate case depicted in Figure 2.5. Non-siteless equisets such as {03} would be generated. In two dimensions, the solution is to consider the set {01234} as a sequence in clockwise (or counterclockwise) order. This method does not apply, however, in higher dimensions.

The question therefore is: what is the analog of "clockwise" in higher dimensions? The answer is simple, if we slightly extend the meaning of clockwise. Consider Figure 2.5 again. What we want to generate from {01234} is {01}, {12}, {23}, {34}, and {40} (just another way of representing the sequence). Note, that the generated sets {01}, {12}, {23}, and {34} describe the convex hull of {01234}. Similarly, for the three-dimensional case depicted in Figure 2.6, the sets {012}, {013}, {024}, {034}, {123}, and {234} describe the convex hull of the set {01234}. Thus, the generalization of "clockwise" is "convex hull." In other words, the Voronoi set is generated from the base by forming all possible subsets of the siteless equisets in the base. For a degenerate equiset, its convex hull has to be found. Note that the Delaunay diagram (whose definition is based on the function *convexPoints*) nicely displays these observations.

2.5 Example in two dimensions

This section illustrates the presented definitions and conjectures. Figures 1.1, 1.2 and 1.3 display the location of the sites $S = \{0,1,2,3,4,5,6,7,8,9\}$ in two-dimensional

Euclidean space. The following set of siteless equisets represents the Voronoi set.

$$
\begin{aligned}
Vs.S = \{\ & \{985\},\{82035\},\{862\},\{612\},\{641\},\{431\},\{473\},\{753\},\{301\},\{021\},\\
& \{59\},\{98\},\{85\},\{82\},\{86\},\{62\},\{61\},\{64\},\{41\},\{43\},\\
& \{47\},\{73\},\{75\},\{53\},\{20\},\{21\},\{10\},\{13\},\{30\},\\
& \{0\},\{1\},\{2\},\{3\},\{4\},\{5\},\{6\},\{7\},\{8\},\{9\}\ \}
\end{aligned}
$$

The siteless equisets of cardinality greater than two constitute the Voronoi base $Vb.S$. For a set e in the Voronoi base $Vb.S$, there exists only a single point p in $equiPoints.e$. This justifies the special status of the equisets in $Vb.S$. In Figure 2.7, the points $p1$ and $p2$ arise from the equisets $\{862\}$ and $\{82035\}$ and the circles illustrate that no other site is as close as or closer to $p1$ than 8, 6, or 2 (similarly for $p2$ and 8, 2, 0, 3, or 5). That is, these equisets are siteless.

Let us now check whether the equiset $\{82\}$, not in $Vb.S$, is *siteless* according to our definition. In contrast to the previous two equisets, we now have an infinite number of choices for p (i.e., $equiPoint.\{82\}$ contains any point on the bisector between 8 and 2). However, only points between $p1$ and $p2$ (i.e., $sitelessPoints.\{82\}$) are valid choices so that $siteless.\{82\}$ evaluates to true. Figure 2.8 shows such a point $p3$ and similarly $p4$ for $siteless.\{2\}$. Thus, $\{82\}$ and $\{2\}$ are siteless.

Conjecture 1 implies that if we draw all the empty circles through 8 and 2, collect all their centers and repeat this procedures for all other equisets in the Voronoi set, then we select each point of the plane exactly once.

Conjecture 2 applied to the equiset $\{2\}$ implies that the union over the point sets

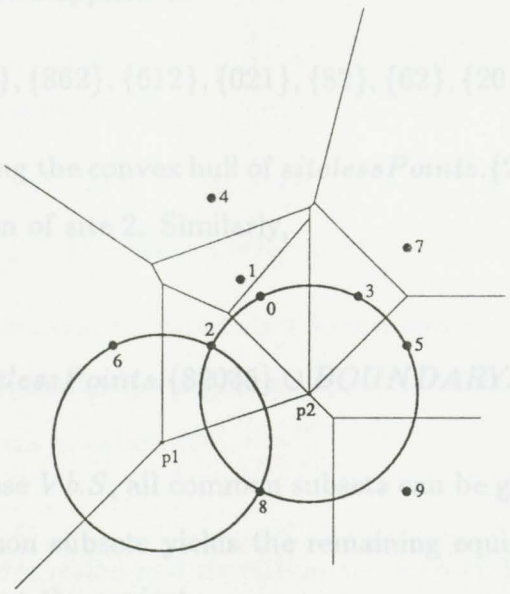

Figure 2.7: Sets {862} and {82035} are elements of the Voronoi base

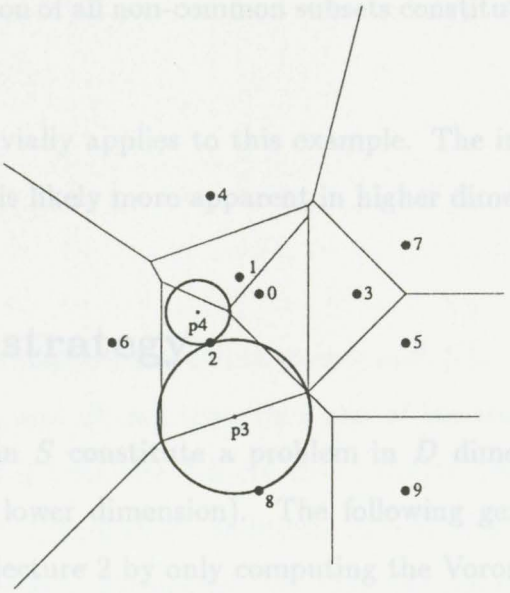

Figure 2.8: Sets {2} and {82} are elements of the Voronoi set

returned by *sitelessPoints* applied to

$$\{82035\}, \{862\}, \{612\}, \{021\}, \{82\}, \{62\}, \{20\}, \{21\}$$

is equivalent to computing the convex hull of $sitelessPoints.\{2\}$, which is, the boundary of the Voronoi region of site 2. Similarly,

$$sitelessPoints.\{82\} =$$
$$BOUNDARY.sitelessPoints.\{82035\} \cup BOUNDARY.sitelessPoints.\{862\}$$

From the Voronoi base $Vb.S$, all common subsets can be generated. In our example, the set of all common subsets yields the remaining equisets in the Voronoi set (i.e., $Vs.S - Vb.S$) except the equisets

$$\{59\}, \{98\}, \{86\}, \{64\}, \{47\}, \{75\} .$$

Note that the enumeration of all non-common subsets constitutes the convex hull of S.

Thus, our theory trivially applies to this example. The importance of the regularity of these concepts is likely more apparent in higher dimensions.

2.6 Solution strategy

Assume that the sites in S constitute a problem in D dimensions (otherwise, the problem is solved in a lower dimension). The following general solution strategy takes advantage of Conjecture 2 by only computing the Voronoi base. The complete Voronoi set, Voronoi diagram, and Delaunay diagram are easily generated from $Vb.S$. We therefore focus on finding $Vb.S$. In other words, our goal is to write a program with

state space $(D, S, base)$ which asserts the following predicate, called the postcondition of the program, without changing D or S. (D is the dimension, S the set of sites and, *base* the Voronoi base.)

$$Vor.(D, S, base) \equiv (\forall e \subseteq S: |e| > D: \quad (2.2)$$
$$(e \in base \wedge siteless.e) \vee (e \notin base \wedge \neg siteless.e))$$

We avoid any overspecification to maintain a broad spectrum of possible programs. This is possible by maximizing the degree of non-determinism, which is one reason for our choice of Dijkstra's notation [28].

Non-deterministic alternation and repetition are denoted by the **if-fi**- and **do-od**-constructs.

if $B_1 \rightarrow SL_1$
| $B_2 \rightarrow SL_2$
fi

do $B_1 \rightarrow SL_1$
| $B_2 \rightarrow SL_2$
od

B_1 and B_2 are boolean expressions called *guards* and SL_1 and SL_2 are statement lists. If both guards B_1 and B_2 are true, then any of the sequence lists SL_1 or SL_2 is executed. If no guard is true, the **if-fi**-construct aborts execution whereas the **do-od**-construct is skipped. Established conditions such as preconditions, invariants, or postconditions are given in curly braces. We also use the usual set operators (i.e., $\cup$, $-$, $\exists$, $\forall$, $\in$, ϕ). The boolean operator **cand** is the conditional and-operator. The function $powerSet.(l, S)$ computes all subsets nS of S with $|nS| > l$.

program Voronoi solution strategy

const D, N : positive Integer; S : set of N sites in D dimensions;

var *base* : set of subsets of S; e : subset of S;

begin

$base :=$ any subset of $powerSet.(D, S)$;

do $\exists e \in (powerSet.(D, S) - base)$ **cand** $siteless.e \rightarrow base := base \cup \{e\}$

| $\exists e \in base$ **cand** $\neg siteless.e \rightarrow base := base - \{e\}$

od

{ **Postcondition** $Vor.(D, S, base)$ }

end Voronoi solution strategy.

Note that, depending on the value of *siteless.e*, e is either added to or removed from *base*. The conjunction of the negated guards establishes the postcondition $Vor.(D, S, base)$. That is, when the **do-od**-construct is exited, the postcondition is satisfied. Observe that the following loop invariant ensures that *base* is a set of subsets of S.

$$\textbf{Invariant} \quad base \subseteq powerSet.(D, S) \tag{2.3}$$

The small information content or weakness of the invariant is characteristic of a general solution strategy. We do not specify how e is chosen in each iteration step to maintain a high degree of non-determinism (i.e., the schedule for choosing e is not determined). Note that the subsets, initially present in *base*, could be randomly added to *base* during the execution of the loop. A fair (e.g., non-repeating) choice of e, so that all possible e are eventually chosen, guarantees termination. Since we do not specify how e is selected, proving termination (besides showing the establishment of the postcondition) is part of our program correctness proof.

In the subsequent chapter, the geometric aspects represented by *siteless* and the combinatorial aspects exemplified by *powerSet* are pursued separately and then merged to derive a simple and efficient algorithm.

2.7 Summary

The functions *equidistant*, *equi*, and *siteless* precisely characterize the Voronoi diagram. The Voronoi set, the set of all siteless equisets, provides a discrete abstraction of the Voronoi diagram and separates geometric and combinatorial concerns. Our general definition of the D-dimensional Voronoi diagram readily includes degeneracies.

By means of the function $BOUNDARY$, we presented a novel approach distinct from the functions CH and *conv* used in computational geometry with a pleasing nuance to set theory.

Several existing theorems are unified into two conjectures. Each siteless equiset corresponds to a convex hypervolume, where all hypervolumes partition the infinite Euclidean space (Conjecture 1). The convex hull of these objects is composed of convex hypervolumes where each volume again corresponds to one of the siteless equisets (Conjecture 2).

The Voronoi set can be generated by the Voronoi base, resulting in a simplified solution strategy.

Chapter 3

Algorithm derivation

This chapter makes extensive use of the separation between geometric and combinatorial concerns. Section 3.1 focuses entirely on the geometric aspects of the problem and the brute-force derivation presented in Section 3.2 examines the combinatorial aspects of our solution strategy. Although algorithms for three and higher dimensions are within reach, particularly in the first two sections, the presentation is limited to two dimensions. In Section 3.3, a more sophisticated derivation leads, in a first refinement step, to the *sweephull* algorithm and, in a second refinement step, to the *sweepline* and *sweepcircle* algorithms.

3.1 The circle criterion and the function siteless

The boolean function *siteless* is defined using the functions *dst* (Euclidean distance), *equidistant*, *anySite*, and logical quantification. In this section, we replace these three abstractions by the operations *addition*, *subtraction*, *multiplication*, and *repetition* (an iterative construct).

In Subsections 3.1.1 and 3.1.2, we focus on the functions *equi* and *siteless* and substitute the squared Euclidean distance for *dst* in their definitions. As we proceed, we derive the necessary conditions for the existence of a solution and formulate it in terms of the functions *equiTest* and *sitelessTest*. Subsection 3.1.3 formulates procedures *Equi* and *Siteless* based on the *equiTest* and *sitelessTest* functions. After deriving *sitelessTest* formally and mechanically, we bridge the gap to Guibas and Stolfi's "Incircle Test" [16] in Subsection 3.1.4. The reward for this tedious work is better insight on how to correctly formulate the circle criterion and derive *siteless* for any dimension and metric.

3.1.1 Deriving function equiTest by substitution

In Chapter 2, the distance function *dst* is used in our specification formulae only in comparisons. It is therefore not necessary to take the square root for comparing Euclidean distances. Therefore, we redefine the function *dst* to be $(DST.2)^2$.

$$dst.(a,b) \;:=\; (a_x - b_x)^2 + (a_y - b_y)^2 = \\ a_r^2 + b_r^2 - 2a_xb_x - 2a_yb_y \tag{3.1}$$

Note that a_r^2 and b_r^2 abbreviate $a_x^2 + a_y^2$ and $b_x^2 + b_y^2$, respectively and $a_r^2 = dst.(o,a)$ where o stands for the origin.

Since our solution strategy concentrates on finding all siteless equisets e ($|e| > D$), and since we have restricted our derivation to two dimensions, we assume e to be of cardinality at least three (i.e., $\exists\{a,b,c\} : \{a,b,c\} \subseteq e$). In addition, we assume that e is *equi*. That is, a, b and c are cocircular and *equiPoints.e* consists of one

distinct point $p := (p_x, p_y)$, the center of a unique circle. Moreover, k, used in the definition of *equi*, is the square of this circle's radius r. Let us rewrite the function $equidistant'.(\{a, b, c\}, p)$ by evaluating the universal quantification for $nS = \{a, b, c\}$.

$$r^2 := dst.(p, a) = dst.(p, b) = dst.(p, c)$$

Substituting the definition of *dst* yields equations (3.2) - (3.4).

$$r^2 - p_r^2 = a_r^2 - 2p_x a_x - 2p_y a_y = \tag{3.2}$$

$$b_r^2 - 2p_x b_x - 2p_y b_y = \tag{3.3}$$

$$c_r^2 - 2p_x c_x - 2p_y c_y \tag{3.4}$$

Combining these equations by subtracting (3.3) from (3.2) yields equation (3.5); and similarly, (3.4) from (3.2) yields (3.6); (3.4) from (3.3) yields (3.7)—-thereby eliminating r and p_r.

$$a_r^2 - b_r^2 - 2p_x(a_x - b_x) - 2p_y(a_y - b_y) = 0 \tag{3.5}$$

$$a_r^2 - c_r^2 - 2p_x(a_x - c_x) - 2p_y(a_y - c_y) = 0 \tag{3.6}$$

$$b_r^2 - c_r^2 - 2p_x(b_x - c_x) - 2p_y(b_y - c_y) = 0 \tag{3.7}$$

Multiplying (3.5) by (a_y - c_y) and (3.6) by (a_y - b_y) and subtracting equation (3.6) from (3.5) yields equation (3.8).

$$\begin{aligned}(a_y - c_y)(a_r^2 - b_r^2) - 2p_x(a_x - b_x)(a_y - c_y) - \\ (a_y - b_y)(a_r^2 - c_r^2) + 2p_x(a_x - c_x)(a_y - b_y) = 0\end{aligned} \tag{3.8}$$

Solving equation (3.8) for p_x finally yields equation (3.9).

$$p_x = \frac{(a_y - b_y)(a_r^2 - c_r^2) - (a_y - c_y)(a_r^2 - b_r^2)}{2\left[(a_x - c_x)(a_y - b_y) - (a_x - b_x)(a_y - c_y)\right]} \tag{3.9}$$

The denominator of equation (3.9) can be written in terms of a determinant as equation (3.10) exhibits.

$$det.(a,b,c) := (a_x - b_x)(a_y - c_y) - (a_x - c_x)(a_y - b_y) = \begin{vmatrix} a_x & a_y & 1 \\ b_x & b_y & 1 \\ c_x & c_y & 1 \end{vmatrix} \tag{3.10}$$

Thus, p_x can be rewritten as in equation (3.11).

$$p_x = \frac{(a_y - b_y)(a_r^2 - c_r^2) - (a_y - c_y)(a_r^2 - b_r^2)}{-2det.(a,b,c)} \tag{3.11}$$

Similarly, solving for p_y yields equation (3.12).

$$p_y = \frac{(a_x - b_x)(a_r^2 - c_r^2) - (a_x - c_x)(a_r^2 - b_r^2)}{2det.(a,b,c)} \tag{3.12}$$

Thus, the solution (p_x, p_y) is the set $equiPoints.e$ for $e = \{a,b,c\}$. The cross-product $(b-a) \times (c-a)$ in $det.(a,b,c)$ is the well-known left-turn-test (counterclockwise-test) in computational geometry. For the sake of completeness, we define *equiTest* to be

$$equiTest.\{a,b,c\} := \begin{cases} \text{“}c \text{ left”} & \text{if } \det.(a,b,c) > 0 \\ \text{“}c \text{ on”} & \text{if } \det.(a,b,c) = 0 \\ \text{“}c \text{ right”} & \text{if } \det.(a,b,c) < 0 \end{cases}$$

The coordinates p_x and p_y are defined only if $det.(a,b,c)$ is non-zero (i.e., a, b, and c are non-collinear). Therefore, $equi.e$ is equivalent to $equiTest.\{a,b,c\} \neq$ “c on”. Note further that in deriving p_x and p_y, only equations (3.5) and (3.6), and not (3.7), have been used. Site a occurs in both equations (3.5) and (3.6), distinguishing a from b or c. Let us return to the function *anySite* introduced in our specification of function *equidistant'*. Because of the special rôle of a, function *anySite* is defined to return the “first” site of its set parameter e (i.e., site a).

3.1.2 Deriving function sitelessTest by substitution

In this section, we derive the formulae for *siteless* if its argument e is $\{a, b, c\}$, an equiset, and S is $\{a, b, c, s\}$. Using the unique equidistant point p (i.e., $\{(p_x, p_y)\}$) obtained in the preceding section, we formulate *siteless* using the function *sitelessTest*, which applies to the restricted equisets e of cardinality three. The function *sitelessTest* tests whether a given site s, an element of $S - e$, is inside, outside, or on the circle through sites a, b, and c. Guibas and Stolfi call this "Incircle test" [16]. Our function *sitelessTest* differs from the "Incircle test" by returning a non-boolean value, taking into account that the site s might be found on the circle. Let

$$\Delta := dst.(s, p) - dst.(a, p) \tag{3.13}$$

then

$$sitelessTest.(\{a, b, c\}, s) := \begin{cases} \text{“}s \text{ outside”} & \text{if } \Delta > 0 \\ \text{“}s \text{ on”} & \text{if } \Delta = 0 \\ \text{“}s \text{ inside”} & \text{if } \Delta < 0 \end{cases} \tag{3.14}$$

Substituting *dst* and p in equation (3.13) yields equation (3.15).

$$\Delta = s_r^2 - a_r^2 - 2p_x(s_x - a_x) - 2p_y(s_y - a_y) \tag{3.15}$$

Substituting p_x and p_y in equation (3.15) and multiplying by $det.(a, b, c)$ yields equation (3.16).

$$\begin{aligned} \Delta det.(a, b, c) \;=\; & (s_r^2 - a_r^2) det.(a, b, c) + \\ & (s_x - a_x)\left[(a_y - b_y)(a_r^2 - c_r^2) - (a_y - c_y)(a_r^2 - b_r^2)\right] - \\ & (s_y - a_y)\left[(a_x - b_x)(a_r^2 - c_r^2) - (a_x - c_x)(a_r^2 - b_r^2)\right] \end{aligned} \tag{3.16}$$

Rearranging the terms of the equation yields (3.17).

$$\Delta det.(a, b, c) \;=\; (s_r^2 - a_r^2) det.(a, b, c) +$$

$$(a_r^2 - c_r^2)\,[(s_x - a_x)(a_y - b_y) - (a_x - b_x)(s_y - a_y)] -$$
$$(a_r^2 - b_r^2)\,[(s_x - a_x)(a_y - c_y) - (a_x - c_x)(s_y - a_y)] \qquad (3.17)$$

Identifying occurrences of *det* yields equation (3.18).

$$\begin{aligned} \Delta det.(a,b,c) \;=\; & (s_r^2 - a_r^2) det.(a,b,c) - \\ & (c_r^2 - a_r^2) det.(a,b,s) + \\ & (b_r^2 - a_r^2) det.(a,c,s) \end{aligned} \qquad (3.18)$$

Abbreviating the right-hand side of equation (3.18) by $det.(a,b,c,s)$[1] finally yields equation (3.19).

$$\Delta det.(a,b,c) = det.(a,b,c,s) \qquad (3.19)$$

We again distinguished a over b or c in equation (3.13) and hence, we define *anySite* in the specification of *siteless* to return site a. However, sites b or c are also valid choices. Note that Δ is independent of the choice of the left- or right-turn function for the *det*-function with three parameters.

We omit the trivial formulation of procedure *EquiTest*, but show the procedure *SitelessTest* in the following program fragment. Note that conditions on the program state are given in curly braces. The postcondition indicates the result returned by a procedure. The parameters of procedure *SitelessTest*, an equiset $\{a,b,c\}$ of cardinality three and a site s, are both explicit.

[1] In Subsection 3.1.4, we show that $det.(a,b,c,s)$ is the determinant of a 4×4-matrix.

procedure $SitelessTest.(\{a,b,c\},s)$

begin

{ **Precondition** $det.(a,b,c) \neq 0$ }

if $det.(a,b,c) > 0 \rightarrow \Delta := det.(a,b,c,s);$

| $det.(a,b,c) < 0 \rightarrow \Delta := -det.(a,b,c,s);$

fi

{ **Postcondition** $sitelessTest(\{a,b,c\},s)$ and Δ

according to definition (3.13) and (3.14) }

end $SitelessTest.(\{a,b,c\},s)$.

One reason for including this trivial derivation is to demonstrate that the derivation is forced by our sound specification. Abstractions in the specification formulae are gradually replaced, giving concrete implementations (e.g., *dst* is replaced by the squared Euclidean distance, universal quantification is replaced by repetition, and existential quantification is ensured through a non-zero denominator).

3.1.3 Procedures Equi and Siteless

The repeated invocation of procedure *SitelessTest* yields in a straightforward manner the following two procedures which compute the functions *equi* and *siteless*. The procedure *AnySites* selects $D+1 = 3$ arbitrary sites from the subset nS, the explicit parameter of procedure *Equi*. The parameters of procedure *Siteless* are an equiset e (explicit) and the set of sites S (implicit).

procedure *Equi.nS*

var nS' : set of sites; a, b, c : site;

begin

{ **Precondition** $nS \subseteq S$ }

$nS' := nS$; $\{a, b, c\} := AnySites.nS$;

if $EquiTest.\{a, b, c\} \neq$ "c on" $\rightarrow nS' := nS' - \{a, b, c\}$;

do $\exists s \in nS'$ **cand** $SitelessTest.(\{a, b, c\}, s) =$"$s$ on" $\rightarrow nS' := nS' - \{s\}$ **od**

| $EquiTest.\{a, b, c\} =$ "c on" $\rightarrow$ **skip**

fi

{ **Postcondition** $equi.nS \equiv nS' = \phi$ }

end *Equi.nS.*

Implicit declaration: **const** S : set of sites;

procedure *Siteless.e*

var S' : set of sites; a, b, c : site;

begin

{ **Precondition** $equi.e$ }

$S' := S$;

$\{a, b, c\} := AnySites.e$;

do $\exists s \in S'$ **cand** $SitelessTest.(\{a, b, c\}, s) \neq$"$s$ on" $\rightarrow S' := S' - \{s\}$ **od**

{ **Postcondition** $siteless.e \equiv e = S'$ }

end *Siteless.e.*

3.1.4 Function sitelessTest as a determinant

Linear algebra provides an elegant formulation of the functions *equiTest* and *sitelessTest*. Both functions compute the determinants of specially formed matrices.

This approach of using determinants promises a general derivation of both functions which is independent of dimension and metric being used.

The claim is:

$$det.(a,b,c) = \begin{vmatrix} a_x & a_y & 1 \\ b_x & b_y & 1 \\ c_x & c_y & 1 \end{vmatrix} \tag{3.20}$$

and

$$det.(a,b,c,s) = \begin{vmatrix} a_x & a_y & a_r^2 & 1 \\ b_x & b_y & b_r^2 & 1 \\ c_x & c_y & c_r^2 & 1 \\ s_x & s_y & s_r^2 & 1 \end{vmatrix}. \tag{3.21}$$

Subtracting the first row from all other rows does not change the determinant in equations (3.20) and (3.21), and using the recursive definition of the determinant yields equations (3.22) and (3.23).

$$det.(a,b,c) = \begin{vmatrix} a_x & a_y & 1 \\ b_x - a_x & b_y - a_y & 0 \\ c_x - a_x & c_y - a_y & 0 \end{vmatrix} = \begin{vmatrix} b_x - a_x & b_y - a_y \\ c_x - a_x & c_y - a_y \end{vmatrix} \tag{3.22}$$

$$det.(a,b,c,s) = \begin{vmatrix} a_x & a_y & a_r^2 & 1 \\ b_x - a_x & b_y - a_y & b_r^2 - a_r^2 & 0 \\ c_x - a_x & c_y - a_y & c_r^2 - a_r^2 & 0 \\ s_x - a_x & s_y - a_y & s_r^2 - a_r^2 & 0 \end{vmatrix} = \begin{vmatrix} b_x - a_x & b_y - a_y & b_r^2 - a_r^2 \\ c_x - a_x & c_y - a_y & c_r^2 - a_r^2 \\ s_x - a_x & s_y - a_y & s_r^2 - a_r^2 \end{vmatrix} =$$

$$(s_r^2 - a_r^2) \begin{vmatrix} b_x - a_x & b_y - a_y \\ c_x - a_x & c_y - a_y \end{vmatrix} - (c_r^2 - a_r^2) \begin{vmatrix} b_x - a_x & b_y - a_y \\ s_x - a_x & s_y - a_y \end{vmatrix} +$$

$$(b_r^2 - a_r^2) \begin{vmatrix} c_x - a_x & c_y - a_y \\ s_x - a_x & s_y - a_y \end{vmatrix} \tag{3.23}$$

The determinants of the 2×2-matrices are easily shown to match the corresponding $det.(a, b, c)$, $det.(a, b, s)$, and $det.(a, c, s)$ in equation (3.18).

The structure of the three-dimensional *sitelessTest* is expected to be similar to equation (3.19). Indeed, investigating the one-dimensional analog to our two-dimensional derivation results in equation (3.24).

$$\Delta \begin{vmatrix} a_x & 1 \\ b_x & 1 \end{vmatrix} = \begin{vmatrix} a_x & a_r^2 & 1 \\ b_x & b_r^2 & 1 \\ s_x & s_r^2 & 1 \end{vmatrix} \tag{3.24}$$

In one dimension, site a corresponds to (a_x) and a_r^2 is equal to a_x^2. Based upon the equations for the one- and two-dimensional cases, the three-dimensional *sitelessTest* should be of the form of equation (3.25).

$$\Delta \begin{vmatrix} a_x & a_y & a_z & 1 \\ b_x & b_y & b_z & 1 \\ c_x & c_y & c_z & 1 \end{vmatrix} = \begin{vmatrix} a_x & a_y & a_z & a_r^2 & 1 \\ b_x & b_y & b_z & b_r^2 & 1 \\ c_x & c_y & c_z & c_r^2 & 1 \\ d_x & d_y & d_z & d_r^2 & 1 \\ s_x & s_y & s_z & s_r^2 & 1 \end{vmatrix} \tag{3.25}$$

A further generalization to D dimensions yields equation (3.26). Note that the definition of the distance function hidden in subscript r is replaced by $DST.D$ applied

to the origin o and the sites $P.0$, $P.1$, ... $P.D$.

$$\Delta \begin{vmatrix} P.0.0 & P.0.1 & \ldots & P.0.D-1 & 1 \\ P.1.0 & \ldots & \ldots & \ldots & 1 \\ \ldots & \ldots & \ldots & \ldots & 1 \\ P.D.0 & \ldots & \ldots & P.D.D-1 & 1 \end{vmatrix} =$$

$$\begin{vmatrix} P.0.0 & P.0.1 & \ldots & P.0.D-1 & (DST.D.(o,P.0))^2 & 1 \\ P.1.0 & \ldots & \ldots & \ldots & (DST.D.(o,P.1))^2 & 1 \\ \ldots & \ldots & \ldots & \ldots & \ldots & 1 \\ \ldots & \ldots & \ldots & \ldots & \ldots & 1 \\ P.D.0 & \ldots & \ldots & P.D.D-1 & (DST.D.(o,P.D))^2 & 1 \\ P.s.0 & P.s.1 & \ldots & P.s.D-1 & (DST.D.(o,P.s))^2 & 1 \end{vmatrix} \tag{3.26}$$

Our formulation of *sitelessTest* differs from the "Incircle test" by Stolfi and Guibas [16] shown in equation (3.27).

$$det.(a,b,c,s) = \begin{vmatrix} a_x & a_y & a_r^2 & 1 \\ b_x & b_y & b_r^2 & 1 \\ c_x & c_y & c_r^2 & 1 \\ s_x & s_y & s_r^2 & 1 \end{vmatrix} > 0 \tag{3.27}$$

Let us compare our constructive derivation of *sitelessTest* with the proof of the "Incircle test" presented by Guibas and Stolfi. Our derivation is based on the rules of algebra. A reader familiar with these rules can easily follow the argumentation. A verification of the proof is very mechanical and could even be performed by a theorem prover. Furthermore, every single step is motivated by our specification of the functions *equi* and *siteless*. In comparison, the proof by Guibas and Stolfi depends highly on the reader's intuition of a mapping into three-dimensional space. Their interpretation of "Incircle" might be interesting, but after their proof they

wonder why reversing the orientation of the function parameters flips the sign and therefore the outcome of the "Incircle" test. This important detail is easily explained by our formulation of *sitelessTest*: the value returned by the function *sitelessTest* depends on the sign of $det.(a, b, c)$.

3.2 Brute-force derivation

In this section, we develop a brute-force program for constructing the Voronoi base. The algorithm exploits the non-boolean specification of *sitelessTest* and refines the procedure *Siteless*. The goal of our initial brute-force derivation is to formulate a correct and general program and to demonstrate our methodology. At this early stage of derivation, we ignore performance issues.

Now that procedure *Siteless* is derived, we continue with the solution strategy developed in Section 2.6. A site in S is represented by an integer in the range $0 \ldots N-1$. The idea is to start initially with a program state space $(D, S, base)$, where *base* is empty, and then gradually add siteless equisets. Thus, without explicitly writing the program, the program invariant is strengthened from invariant (2.3) to (3.28).

$$\textbf{Invariant} \quad base \subseteq powerSet.(2, S) \wedge base \subseteq Vb.S \equiv base \subseteq Vb.S \qquad (3.28)$$

A major complication is the ambiguous naming of an equiset (i.e., $\{i, j, k\} = \{k, i, j\} = \ldots$), which is ill-suited for computer representation. Therefore, we enforce uniqueness by an arbitrary rule so that the equisets in *base* are ordered into ascending sequences,

yielding the following strengthened invariant.

$$\textbf{Invariant} \qquad base \subseteq Vb.S \wedge$$
$$(\forall\{\ldots,i,j,\ldots\} : \{\ldots,i,j,\ldots\} \in base\colon i < j) \tag{3.29}$$

where $\{\ldots,i,j,\ldots\}$ denotes a sequence. This greatly simplifies the generation of $powerSet.(2,S)$.

Instead of rejecting an equiset e, because $SitelessTest.(e,s)$ evaluates to "s on" in the procedure *Siteless*, we prefer including s in e. We call this action: "e is *expanded* by s." Thus, we refine the procedure *Siteless* to procedure *SitelessExpand*. In addition to the parameter equiset e, the variable parameter *base* is passed to this procedure. The expanded equiset e' is included in *base*, if appropriate. By this refinement we generate all 3-sequences (i.e., $powerSet.(2,S) - powerSet.(3,S)$).

Implicit declarations:

const $D = 2, N$: positive Integer; $S = \{0,\ldots,N-1\}$: sequence of sites;

procedure $SitelessExpand.(base, e)$
var e' : sequence of sites out of S; a, b, c, s : site;
begin
{ **Precondition** $e \subseteq S$ }
$\{a, b, c\} := AnySites.e$; $e' := e$; $s := 0$;
if $EquiTest.\{a, b, c\} \neq$ "c on" $\rightarrow$
do $s < N$ **cand** $SitelessTest.(\{a, b, c\}, s) =$"$s$ outside" $\rightarrow s := s + 1$
| $s < N$ **cand** $SitelessTest.(\{a, b, c\}, s) =$"$s$ on" $\rightarrow s := s + 1$;
$e' := append.(e', s)$
od ;
{ **Postcondition** $siteless.e' \equiv s = N$ }
if $s = N \rightarrow base := base \cup \{e'\}$ **fi**
fi
end $SitelessExpand.(base, e)$.

The last degree of freedom in the program is the order in which the 3-sequences are generated. Let us therefore define "$e < f$" ($e, f \subseteq S$), if the first three sites of e are lexicographically less than the ones of f. Once more the invariant is strengthened and the result is invariant (3.30).

$$\begin{aligned} \textbf{Invariant} \quad & base \subseteq Vb.S \wedge \\ & (\forall \{\ldots, i, j, \ldots\} : \{\ldots, i, j, \ldots\} \in base : i < j) \wedge \\ & (\forall e \in Vb.S - base : e > max.base) \end{aligned} \tag{3.30}$$

Function max denotes the maximum element of $base$. Instead of repeating the complete invariant at each step, it is more convenient to write only the refining conjunct.

Hence, our derivation proceeds from a high level of non-determinism, invariant (2.3), to a lower level, invariant (3.30), through stepwise-refinement of conditions on the program state (e.g., preconditions, postconditions, and invariants). The strengthening of conditions not only captures the derivation process firmly, but also provides an elegant way of presenting a derivation. Furthermore, it directly yields the following program.

program Voronoi brute-force (deterministic)
const $D = 2, N$: positive Integer; $S = \{0, \ldots, N-1\}$: sequence of sites;
var *base* : set of subsequences of S; i, j, k : positive Integer; *done* : Boolean;
begin
$i, j, k := 0, 1, 2$; $done := FALSE$; $SitelessExpand.(base, \{i, j, k\})$;
do $\neg done \rightarrow$ { **Invariant** (3.30) }
if $k < N - 1 \rightarrow k := k + 1$;
$SitelessExpand.(base, \{i, j, k\})$;
if $k - j > 1 \rightarrow j, k := j + 1, j + 2$;
$SitelessExpand.(base, \{i, j, k\})$;
if $j - i > 1 \rightarrow i, j, k := i + 1, i + 2, i + 3$; $SitelessExpand.(base, \{i, j, k\})$
| $j - i \leq 1 \rightarrow done := TRUE$
fi
fi
fi
od
{ **Postcondition** $Vor.(2, \{0, \ldots, N-1\}, base)$ }
end Voronoi brute-force (deterministic).

Dijkstra's program notation is a specification language rather than an implementation language. Therefore, it is not well suited for writing the deterministic brute-force

program. However, the program can now be easily converted into an imperative language. The only problem left is the choice of data structures for the two sets *base* and *e*.

One immediate practical use of this brute-force program is to produce correct test-cases for testing more sophisticated programs. A version extended to handle three-dimensional sites could be the basis for producing stereo-scopic pictures of Voronoi or Delaunay diagrams.

3.3 Sweep algorithms

In [15], Preparata and Shamos describe the *plane-sweep* technique as follows. A vertical line sweeps the plane from left to right, halting at special points, called *event* points. The intersection of the sweepline with the problem data contains all the relevant information for the continuation of the sweep. A sweep algorithm maintains two basic data structures: the *event point schedule* and the *sweepline status*. The event points are ordered from left to right in the direction of the x-axis and are scheduled according to this order. The line advances to the abscissa of the current event point and the sweepline status is updated to adequately describe the geometric structure at the current position of the sweepline. The complete schedule is not necessarily extracted from the input data, but may be dynamically updated during the execution of the sweepline algorithm.

In this section, we apply a generalized sweep technique to the Voronoi diagram and refer to the above description as the sweepline technique. Instead of sweeping a straight line, we sweep the plane with a "growing" polygon or circle. Thus, a *sweep*

algorithm is not restricted to sweeping a line across the plane. The time metaphor (suggested by the temporal terms such as schedule and event) models the sweepline technique accurately, because of the linear movement of the sweepline in one direction. A more general sweep paradigm, however, sweeps the plane in several directions simultaneously. Therefore, we de-emphasize time and resort to the abstract methodology of strengthening conditions on the program state for describing sweep algorithms. Furthermore, the generalization integrates Fortune's sweepline algorithm into our derivation tree as a branch belonging to the incremental algorithms; the difference between an incremental and a sweep algorithm is reduced to the formulation of different preconditions. Similar to the brute-force derivation, strengthened invariants complete the description of the sweephull, sweepline, and sweepcircle algorithms.

3.3.1 Incremental algorithms

The incremental processing of a sequence of sites is a characteristic property of a Voronoi sweep algorithm. We therefore transform the set of sites S into a sequence of sites, asserted by the precondition of the incremental skeleton of a Voronoi algorithm. In search of a new invariant, we again resume the discussion of the solution strategy. We are looking for a condition which is weaker than postcondition (2.2) and stronger than invariant (2.3). A standard way to weaken the postcondition is to replace a constant by a variable. Since we have a sequence of sites $S = \{0, \ldots, N-1\}$, we replace the constant N by the variable i. This yields the following strengthened invariant.

$$\textbf{Invariant} \quad Vor.(2, \{0, \ldots, i-1\}, base) \tag{3.31}$$

The following program skeleton is easily constructed from the mentioned conditions.

program Voronoi incremental skeleton

const N : positive Integer; $S = \{0, \ldots, N-1\}$: sequence of sites;

var *base* : set of subsequences of S; i : site;

begin

{ **Precondition** S is a sequence of sites}

$base, i := \phi, 1;$

do $i \neq N \rightarrow$ { **Invariant** (3.31) }

(* update statement *)

$i := i + 1$

od

{ **Postcondition** $Vor.(2, \{0, \ldots, N-1\}, base)$ }

end Voronoi incremental skeleton.

3.3.2 Hull order

Fortune's sweepline algorithm uses a sequence of sites lexicographically ordered by Cartesian coordinates. We show that a lexicographical ordering by polar coordinates leads to a *sweepcircle algorithm.* Therefore, let us find a common condition on the site sequence weaker than those imposed by the Cartesian or polar orderings.

Define the site sequence S to be in *hull order*, if every site i is encountered strictly outside the convex hull of $\{0, \ldots, i-1\}$, or equivalently, if every site i is part of the convex hull of $\{0, \ldots, i\}$, but not of $\{0, \ldots, i-1\}$.

$$\text{(site sequence } S = \{0, \ldots, N-1\} \text{ is in hull order)} \equiv \tag{3.32}$$

$$(\forall i\colon\ 0 \leq i < N\colon\ i \in CH.\{0, \ldots, i\} \wedge i \notin CH.\{0, \ldots, i-1\}) \tag{3.33}$$

sequence	origin	Cartesian	polar	hull
{4,7,1,0,3,6,2,5,8,9}	upper left	x	(x)	x
{0,1,2,3,4,5,6,7,8,9}	site 0	-	x	x
{2,5,3,6,8,0,7,1,9,4}	-	-	-	x

Table 3.1: Sequences in Cartesian, polar, and hull order

Note that both Cartesian and polar sorting establish hull order on a site sequence.

Table 3.1 lists the site sequences for the example in Figure 1.1 in Cartesian, polar, and hull order. Note that the sequence in Cartesian order can be regarded as a sequence in polar order with the origin at infinity. Also, the hull-order property is independent of the location of the origin. Therefore, hull order is an important concept in computational geometry.

Remark In [13], Ohya, Iri, and Murota define *serpentine cell* order and *outward spiral cell* order. Both orderings also imply hull order.

Figure 3.1 shows the derivation tree built so far. The derivation from the "solution strategy" to "incremental" is motivated by strengthening invariant (2.3) to (3.31). The precondition of an incremental algorithm simply is the condition: "S is a sequence of sites." We further differentiate incremental algorithms by the stronger precondition (3.33): "S is a sequence of sites in hull order." This refines incremental algorithms into sweep algorithms. The representative sweephull algorithm is discussed in Section 3.3.3. In Figure 3.2, we redraw the derivation tree and label the nodes by

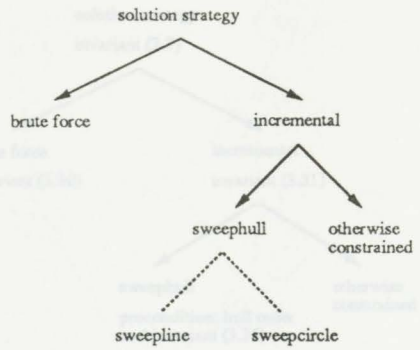

Figure 3.1: Presented derivations

the corresponding preconditions and invariants. The leaf of the tree labeled *otherwise constrained* is introduced, because incremental algorithms exist which impose a different constraint on the site sequence. For example, Guibas and Stolfi present an incremental algorithm assuming that the sites are in some large convex polygon, say a triangle, whose vertices are among the given sites [16].

We discovered hull order by the "bottom-up" argument of weakening the precondition of the sweepline and sweepcircle algorithm. We further motivate the hull-order refinement by a "top-down" argument. The siteless equisets in the Voronoi base are in one-to-one correspondence to the Delaunay polygons (i.e., $convexPoints.e\colon e \in Vb.S$) and each empty circle contains one Delaunay polygon. If the next site to be processed appears inside the convex hull of the proposed sites, the new site is clearly inside or on the border of a Delaunay polygon and therefore inside the corresponding circle. Hence, it invalidates at least one empty circle. In the opposite case where the next site appears outside the convex hull, however, we might perform the update without losing an empty circle.

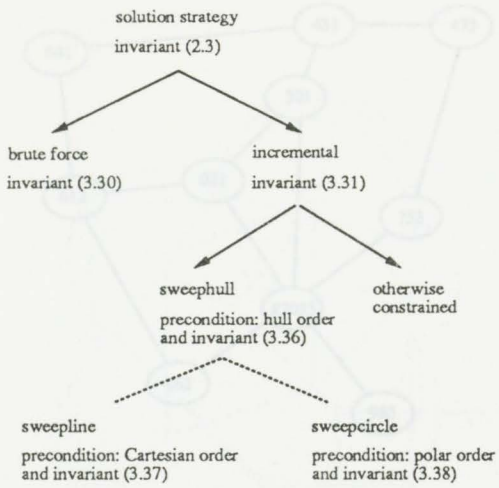

Figure 3.2: Derivation tree labeled with conditions

3.3.3 Sweephull algorithm

In this section, we focus on the update statement in the Voronoi incremental skeleton. We investigate static properties of the Voronoi base and examine the close relationship between the Voronoi diagram and the convex hull using the *Voronoi graph*, which further develops the idea of the common subsets introduced in Chapter 2. The theory of well-foundedness applies to the Voronoi graph and is the basis for the termination argument of the sweephull algorithm. The derived invariant is further strengthened for the sweepline and sweepcircle algorithms, which complete our derivation tree. The access method for the Voronoi graph further differentiates the three algorithms: sweephull, sweepline, and sweepcircle.

3.3.3.1 The Voronoi graph

Let us investigate the static properties of the Voronoi base $Vb.S'$ for one of the intermediate sequences $S' = \{0, \ldots, i-1\}$. The static properties help to update from

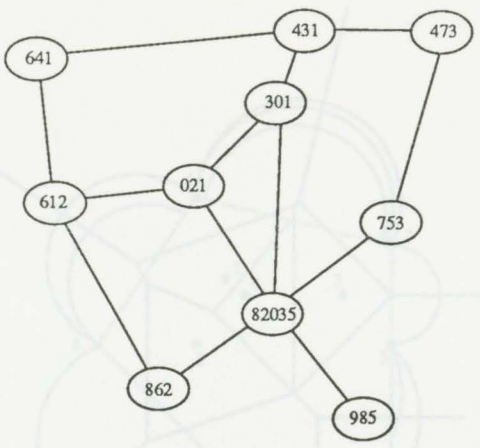

Figure 3.3: Undirected Voronoi graph G

$i-1$ to i while maintaining the invariant for the various sweep algorithms.

We define the *Voronoi graph* $G = (V, E)$ where each member of the Voronoi base $Vb.S'$ corresponds to a node in V and an edge in E between two nodes indicates a common siteless subset of cardinality two.

$$E = ((e, f) \colon e, f \in Vb.S' \colon (\exists t \colon t \in Vs.S' \colon |t| = 2 \wedge t \subset e \wedge t \subset f)) \qquad (3.34)$$

Figure 3.3 shows the undirected graph for the example depicted in Figure 1.1. An edge in the Voronoi graph corresponds to a Voronoi edge.

The precondition (3.33) divides the plane into two parts: the points inside the convex hull ($convexPoints.S$) and the points outside the convex hull ($\mathcal{R}^2 - convexPoints.S$). Let us define a relation between two nodes e and f. A siteless equiset e is said to be "less than" a siteless equiset f, $e < f$, if, outside the convex hull, the circle corresponding to f includes the circle corresponding to e.

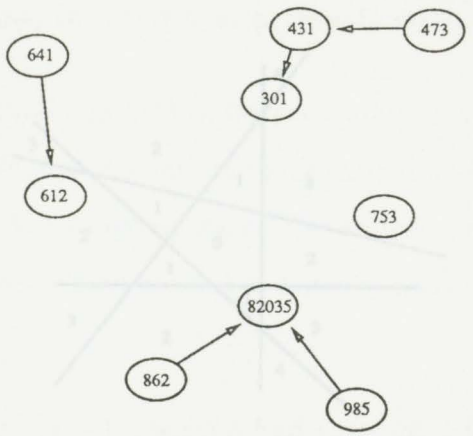

Figure 3.5: Directed Voronoi graph G'

The directed graph in Figure 3.5 corresponds to the example in Figure 1.1. Comparing Figure 3.5 with Figure 3.4, note that every node of the directed graph is a candidate to be removed from the Voronoi graph by an update step. Since part of its associated empty circle lies outside the convex hull, the circle could be a target for the newly encountered site. Note that the decreasing chains may be disconnected.

3.3.3.2 Accessing the Voronoi graph

After introducing the directed Voronoi graph, the question arises: how does an algorithm access the possibly isolated chains of the directed Voronoi graph? Consider the worst case for the location of the sites inside the convex hull: assume that a site s is located very close to the convex hull segment between sites a and b. Then, the segment extended into a line can be regarded as an extreme empty circle corresponding to a siteless equiset $\{a, b, s\}$, a starting node of one or more decreasing chains. Thus, all the starting nodes of the decreasing chains are in one-to-one correspondence with the segments of the convex hull; therefore, the sweephull algorithm keeps track of them. In other words, the sweephull algorithm is embedded into an incremental

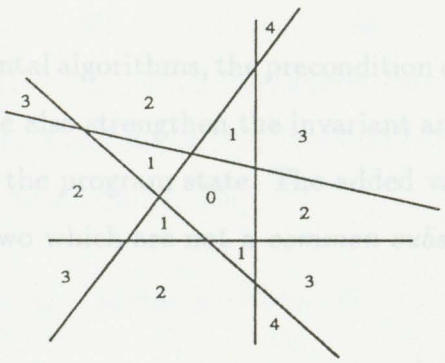

Figure 3.6: Convex hull areas

convex hull algorithm.

We observe that the extended segments subdivide the area outside the convex hull, $\mathcal{R}^2 - convexPoints.S$, where the new site may lie. Different sets of convex hull segments are visible from different areas in the subdivision. Figure 3.6 counts the number of hull segments visible from each area and thus, the number of starting nodes belonging to chains which have to be considered for an update step.

Once the starting nodes are found, the update step of site i removes all nodes e from the decreasing chains for which $sitelessTest.(e, i)$ evaluates to "i inside." The removed nodes form a tree with the starting nodes as the children of an imagined auxiliary root. The edges of the Voronoi graph incident upon the leaves of the tree are expanded by site i to new siteless equisets and therefore to new nodes of the undirected Voronoi graph.

3.3.3.3 Sweep skeleton, invariant, and termination

In contrast to the incremental algorithms, the precondition of sweep algorithms asserts that S is in hull order. We also strengthen the invariant and add a new variable *hull* (short for convex hull) to the program state. The added variable contains all siteless equisets t of cardinality two which are not a *common subset* of two equisets e and f in the base.

$$\textbf{Invariant} \quad Vor.(2, \{0, \ldots, i-1\}, base) \wedge hull = (t\colon |t| = 2 \wedge t \in Vs.\{0, \ldots, i-1\}\colon (\nexists e, f\colon e, f \in Vs.\{0, \ldots, i-1\}\colon t \subset e \wedge t \subset f)) \tag{3.36}$$

The following is the skeleton of a Voronoi sweep algorithm and, in particular, of the sweephull algorithm.

program Voronoi sweep skeleton
const N : positive Integer; $S = \{0, \ldots, N-1\}$: sequence of sites;
var *base* : set of subsequences of S; i : site; *hull* : set of site pairs;
begin
{ **Precondition** S is a sequence of sites in hull order }
$base, i := \phi, 1$;
do $i \neq N \rightarrow$ { **Invariant** (3.36) }
(* locate site i and update *hull*;
update *base* by following the decreasing chains of the Voronoi graph *)
$i := i + 1$
od
{ **Postcondition** $Vor.(2, \{0, \ldots, N-1\}, base)$ }
end Voronoi sweep skeleton.

To prove termination, we claim that $(Vb.S, <)$ forms a partially ordered set and that the decreasing chains in $Vb.S$ are finite. These are the conditions for $Vb.S$ being *well-founded*, which is central to the termination argument. The theory of well-foundedness is based on the Main Repetition Theorem first formulated by Floyd and more recently by Dijkstra and Scholten [30].

Theorem 1 *$(Vb.S, <)$ forms a partially ordered set.*

Proof

$$\begin{aligned} Irreflexivity &: (\forall e\colon e \in Vb.S\colon e \not< e) \\ Asymmetry &: (\forall e, f\colon e, f \in Vb.S\colon e < f \Rightarrow f \not< e) \\ Transitivity &: (\forall e, f, g\colon e, f, g \in Vb.S\colon e < f \wedge f < g \Rightarrow e < g) \end{aligned}$$

By inspection, all three properties hold. *End of Proof.*

Theorem 2 *The decreasing chains in $Vb.S$ are finite.*

Proof Two circles corresponding to siteless equisets e and f with $e < f$ intersect the convex hull in a common segment. Consider the intersection of such a segment with one of the circles and let the length of the segment be denoted by the function *cutout* applied to an equiset. Then, the relation defined by comparing the values provided by *cutout* is superimposed onto the less-than relation (i.e., $e < f$ implies $cutout.e < cutout.f$ and function *cutout* is monotonic). Since a segment length is always positive, there is a minimum length and a minimum node. Hence, the decreasing chains are finite. *End of Proof.*

From the theory of well-foundedness, we conclude that repetition over the decreasing chains terminates. This is important for the update step in the sweephull algorithm.

Remark The definition of "less than" can be extended to apply to any Delaunay polygon without invalidating the well-foundedness of $Vb.S$. Simply, replace "$\mathcal{R}^2 - convexPoints.S$" in the definition (3.35) of less than by "$convexPoints.e$" for some $e \in Vb.S$. This is the termination argument for incremental algorithms.

3.3.3.4 Time complexity analysis

For the time complexity analysis, we distinguish three parts of the algorithm: establishing hull order, accessing the decreasing chains, and performing the Voronoi update step. Hull order can be established by a sort in either Cartesian or polar coordinates in time $O(N \log N)$. For accessing the decreasing chains, the convex hull algorithm as described in [15] can be used. Preparata and Shamos prove the following theorem.

Theorem 3 *The convex hull of a set of N points in the plane can be found online in $\Theta(N \log N)$ time with $\Theta(\log N)$ update time, that is, in real-time.*

Thus, updating the convex hull and therefore accessing the decreasing chains within the incremental loop can be performed in $\Theta(\log N)$ time.[2]

Remark Accessing the decreasing chains corresponds to the *locate* procedure used in Guibas and Stolfi's incremental algorithm which takes $O(N)$ time [16]. They refer to point location methods that are asymptotically faster (in the worst case), but in

[2]On average, there are $N^{1/3}$ sites on the convex hull of a uniformly distributed set of sites [7].

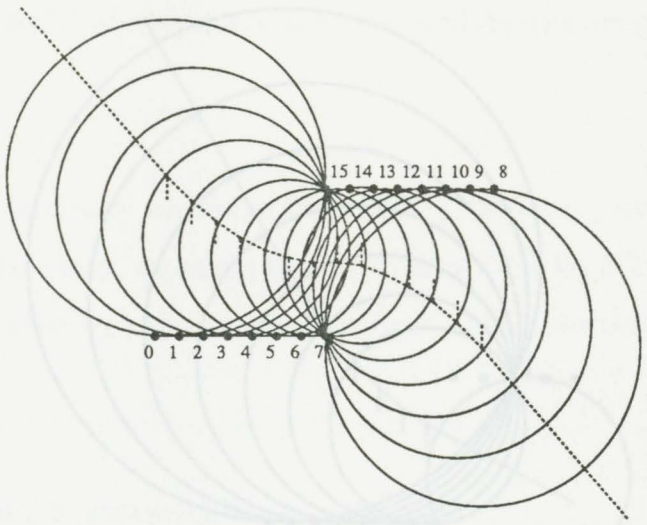

Figure 3.7: Example for $O(N^2)$ complexity of incremental algorithms

their opinion, these methods are probably too complex to be of practical use.

The amount of work performed by the Voronoi update step is measured by estimating the number of equisets generated. The equisets are in one-to-one correspondence to the Voronoi vertices, which are by Euler's formula[3] of order $O(N)$. If all equisets are generated in one step, the update step is of $O(N)$ worst case.

We show the instantiation of the worst case using an example by Guibas, Knuth, and Sharir [23]. Let two halves of n sites ($2n = N$) be located as depicted in Figure 3.7. One half (the sites with indices $0, \ldots, n-1$) is located on a lower horizontal line and the other half (the sites with indices $n, \ldots, N-1$) on a higher line. The lower sites are to the left and the higher sites are to the right of a dividing vertical line. Figure 3.8 illustrates how the update steps for the sites $0, \ldots, n-1$ on the

[3]Euler's formula: $v - e + f = 2$, where v, e, and f are the number of vertices, edges, and faces, respectively.

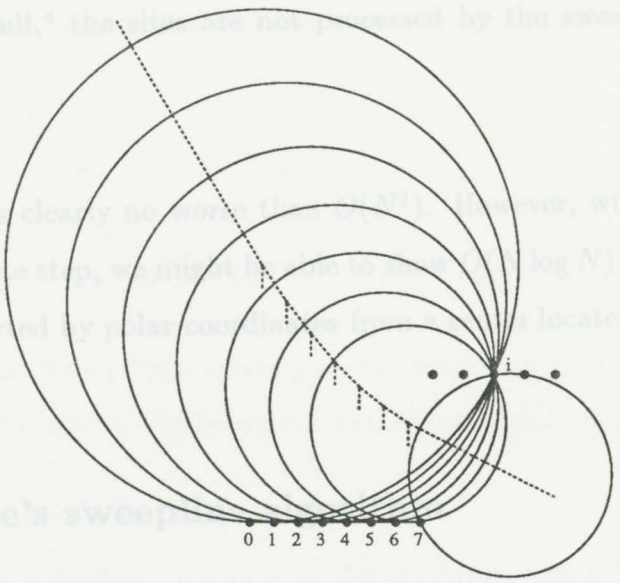

Figure 3.8: Update step of site i $(n < i < N)$ of the example

lower line are performed in constant time. However, the update step for the site i $(n+1 \leq i < N)$ requires the generation of $n-1$ new equisets with pairs of subsequent sites of $0, \ldots, n-1$. Additionally, a new equiset $\{i-1, i, n-1\}$ is generated for a site i $(n+2 \leq i < N)$. The circles corresponding to the generated equisets for a site i $(n+2 \leq i < N)$ are shown in Figure 3.8. Hence, $n^2 - 1$ equisets are generated for all updates in this example.

Remark Observe that the sites of this example are in convex hull order. However, if we chose to establish hull order by sorting the sites by a polar sort with the origin inside the convex hull of the given sites, the worst case does not occur, because the sites i $(n \leq i < N)$ are not processed in required sequence from right to left after processing the sites j $(0 \leq j < n)$. That is, if the center of the polar sort is located

inside the convex hull,[4] the sites are not processed by the sweephull algorithm in worst-case order.

Our algorithm is clearly no worse than $O(N^2)$. However, with a more detailed analysis of the update step, we might be able to show $O(N \log N)$ performance, if the sites are initially sorted by polar coordinates from a center located inside the convex hull of the sites.

3.3.4 Fortune's sweepline algorithm

Fortune partitions the program state of the sweepline algorithm into two entities: priority queue Q and sweepline status L. Q stores the event points in the plane (either sites or Voronoi vertices), ordered lexicographically, and L denotes the mapped Voronoi regions and edges that intersect the sweepline. A third entity of the state is a list of bisectors B where each bisector may be marked with an endpoint.

We split the update step of the sweepline algorithm into two steps. First, the algorithm locates the newly encountered site in one of the regions of L and updates L accordingly. Second, it advances the sweepline to the next site, or finally, to infinity.

According to our definitions, Q corresponds to the sites in S, L to the non-common subsets in *hull* and B to the siteless equisets in *base*. The invariant of the sweephull algorithm is strengthened. The prime added to the function names Vor' and Vs' indicates that a refined function *siteless'* replaces *siteless* in their definition. The

[4] A convex combination of some sites yields such a center point.

strengthened definitions of Vs' and *siteless'* are presented after the invariant.

$$\textbf{Invariant} \quad Vor'.(2,\{0,\ldots,i-1\},base) \wedge hull = (t\colon t \in Vs'.\{0,\ldots,i-1\}\colon (\nexists e,f\colon e,f \in Vs'.\{0,\ldots,i-1\}\colon t \subset e \wedge t \subset f)) \tag{3.37}$$

$$Vs'.\{0,\ldots,i-1\} = (e\colon e \subseteq \{0,\ldots,i-1\}\colon |e| \leq 3 \wedge siteless'.e)$$

The function *siteless'* differs from *siteless* by not returning true for siteless equisets whose corresponding circles all intersect the half-plane ahead of the sweepline.

$$\begin{aligned} siteless'.e \equiv\ & (\exists p\colon p \in \mathcal{R}^D\colon equidistant.(e,p) \wedge \\ & (\forall s\colon s \in \{0,\ldots,i-1\} - e\colon dst.(p,P.(anySite.e)) \leq dst.(p,P.s)) \wedge \\ & p.y - dst.(p,P.(anySite.e)) \geq P.(i-1).y) \end{aligned}$$

In the second conjunct of function *siteless'*, the less-than relation is not strict, as it is in *siteless*, because Fortune's algorithm computes a Voronoi vertex consisting of more than three sites multiple times.

Figure 3.9 illustrates the invariant for the sweepline algorithm. The dotted circles are the extreme empty circles which represent the siteless′ equisets represented by the sweepline status L. The state of L in that particular situation is represented by the following equisets.

$$\{05\},\{56\},\{63\},\{34\},\{47\},\{71\},\{10\}$$

Unlike the variable *hull* of the sweephull algorithm, L does not represent a convex hull. However, L represents all non-common subsets of the Voronoi base, or by a geometric interpretation using function *convexPoints*, a simple polygon (i.e., no pair of non-consecutive segments in L are sharing a point) where all sites processed so far are found inside. Therefore, L additionally contains sites not on the convex hull

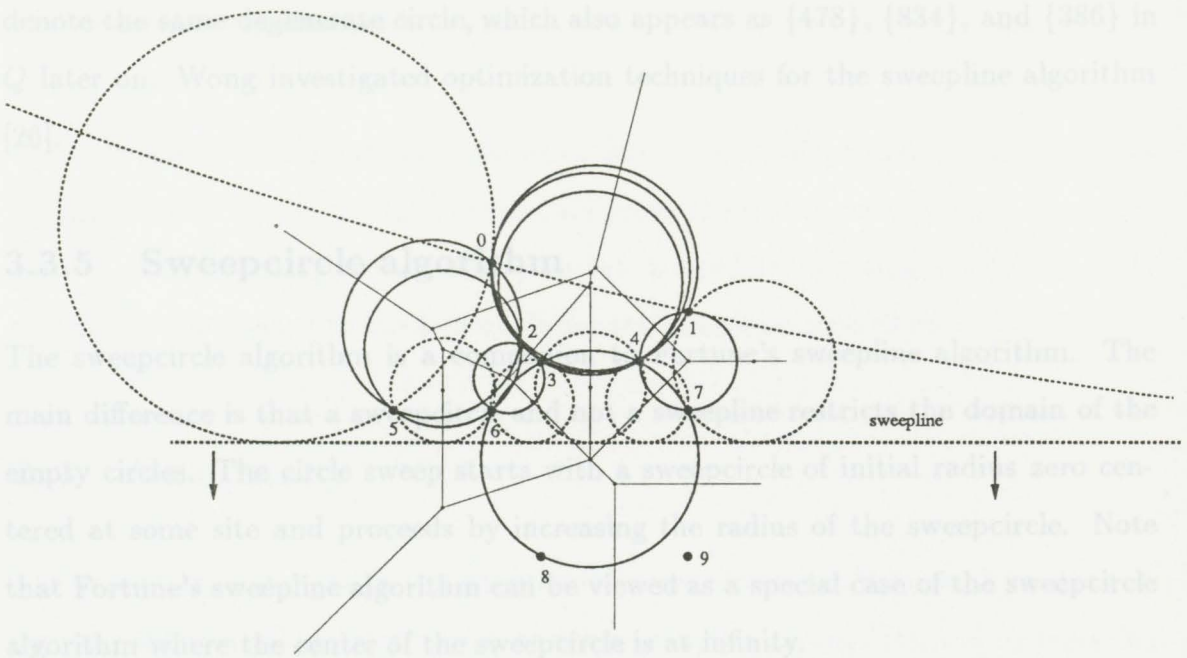

Figure 3.9: Invariant of Fortune's sweepline algorithm

(besides the sites of the convex hull).

Like the sweephull algorithm, L is used for accessing the decreasing chains previously defined, this time however, from the opposite end. The new access procedure is more complicated and needs auxiliary information stored in the priority queue Q: besides the sites yet to be processed, Q maintains all the smallest nodes whose corresponding siteless circles intersect the sweepline. When the sweepline advances, these circles may no longer intersect the sweepline and are removed from Q. That is when the next greater nodes in the decreasing chains are accessed, or more appropriately, generated from the two common subsets.

In the current example, the additional state in Q is $\{\{634\}, \{347\}\}$. Both equisets

denote the same degenerate circle, which also appears as {478}, {834}, and {386} in Q later on. Wong investigated optimization techniques for the sweepline algorithm [26].

3.3.5 Sweepcircle algorithm

The sweepcircle algorithm is a companion to Fortune's sweepline algorithm. The main difference is that a sweepcircle and not a sweepline restricts the domain of the empty circles. The circle sweep starts with a sweepcircle of initial radius zero centered at some site and proceeds by increasing the radius of the sweepcircle. Note that Fortune's sweepline algorithm can be viewed as a special case of the sweepcircle algorithm where the center of the sweepcircle is at infinity.

The invariant for the sweepcircle algorithm is similar to the one for the sweepline algorithm. The sweepcircle invariant is (3.37), except that Vs' is replaced by a restricted Voronoi set Vs'', which is defined in terms of *siteless''* instead of *siteless'*.

$$\textbf{Invariant} \quad Vor''.(2, \{0,\ldots,i-1\}, base) \wedge hull = (t\colon t \in Vs''.\{0,\ldots,i-1\}\colon (\nexists e, f\colon e, f \in Vs''.\{0,\ldots,i-1\}\colon t \subset e \wedge t \subset f)) \qquad (3.38)$$

$$Vs''.\{0,\ldots,i-1\} = (e\colon e \subseteq \{0,\ldots,i-1\}\colon |e| \leq 3 \wedge siteless''.e)$$

The boolean function *siteless''* applied to an equiset e returns true, if there exists an empty circle corresponding to e which lies completely within the sweepcircle.

$$\begin{aligned} siteless''.e \;\equiv\; & (\exists p\colon p \in \mathcal{R}^D\colon equidistant.(e,p) \wedge \\ & (\forall s\colon s \in \{0,\ldots,i-1\} - e\colon dst.(p, P.(anySite.e)) \leq dst.(p, P.s)) \wedge \\ & dst.(o,p) + dst.(p, P.(anySite.e)) \leq dst.(o, P.(i-1)) \end{aligned}$$

The rest of the argumentation is similar.

In Appendix A, a detailed derivation of the site location procedure for the sweepcircle algorithm is given. Like Fortune, we prove the one-to-one property of a mapping $*$. An interesting curve called *polarbola* is used for locating a new site. The polarbola is analogous to the hyperbola for the sweepline algorithm.

3.4 Summary

The derivations presented in this chapter evolve from our specification. By substituting the definition of the function *dst*, by taking a choice for *anySite*, and by replacing quantification with repetition, a brute-force algorithm was derived. The brute-force algorithm is easily generalized to higher dimensions thanks to our derivation of the function "sitelessTest," a refinement of Guibas and Stolfi's "Incircle" test.

The derivation of a brute-force program demonstrated Dijkstra's methodology of strengthening conditions on the program state. The methodology has been successfully applied in the sweephull algorithm derivation as well as in the sweepline and sweepcircle derivations. In this way, we were able to explain the relationship between seemingly unrelated algorithms (i.e., the relationship between the established incremental and sweepline algorithms).

Furthermore, while discovering the sweephull algorithm, Dijkstra's methodology was used as a heuristic for finding the new algorithm. Knowing the precondition of the sweepline and its companion, the sweepcircle algorithm, we tried to find the

weaker precondition of both of these algorithms. Thus, the use of the methodology in the reverse direction revealed the sweephull algorithm. The sweephull algorithm is an improved incremental algorithm. In the simplified site location procedure, we exploited the hull order of the processed site sequence.

The Voronoi graph, built by the common subset relation, is an appropriate model for explaining the sweephull algorithm. The Voronoi graph is not only the backbone of the presented proof but also generalizes to higher dimensions. The Voronoi base is a poset by the "less-than" relation and demonstrates well-foundedness, a fundamental concept for proving termination of an algorithm.

The worst-case time complexity of the sweephull algorithm is clearly $O(N^2)$. However, a more careful analysis might reveal a better upper bound for a strengthened precondition (e.g., the sequence of sites is sorted in polar order around a center inside their convex hull).

Chapter 4

Program implementation

This chapter presents our implementation of the sweephull algorithm in Oberon. It contains a brief introduction to *type extension*, an explanation of the type structure used, and a detailed description of the sweephull update step.

While introducing type extension, Oberon's principal new feature, we present the general setting of our implementation and demonstrate how to establish the precondition *hull order*. The representation of the Voronoi graph, the main data structure of our implementation, is based on circular lists corresponding to equisets. The fact that the elements in a list are alternating in type further strengthens the sweephull invariant for our particular implementation. The core of the sweephull algorithm is the update of the Voronoi graph. The metaphor of pulling a "thread" through the equisets aptly describes the sweephull update step. The correctness argument depends on the traversed order of the finite decreasing chains in the Voronoi graph. The implementation is kept intentionally simple and is not tuned for speed, leaving room for future optimizations.

4.1 Type extension and how hull order is established

Oberon is a single-process multitasking system for personal workstations relying on automatic storage retrieval. The system is implemented in a language, also called Oberon, which offers type extension. Type extension allows for polymorphic operations and provides type guards to ensure type safety which is crucial for the reliability of such operations.

The reason for selecting the language Oberon for implementing the sweephull algorithm is Oberon's integrated support for controlling dynamic data structures. We assume that the reader is familiar with the module concept and the type declarations of Modula-2 (e.g., record, pointer, and procedure type). In this section, we briefly introduce type extension by showing how the module *BinTree* is used to implement the sweephull algorithm. First, we describe the interface to the module *BinTree* (see Figure 4.1) and then its use in the implementation of module *SweepHull* (see Figure 4.2).

Module *BinTree* provides the abstract data type *Tree*, a binary tree. The definition of module *BinTree*, generated from the implementation module, is shown in Figure 4.1. Three procedures, *New*, *Insert*, and *Traverse*, access the parameter *T* of type *Tree*. When a tree is created by *New*, two procedures are passed: *equal* and *less*. The procedure *equal* defines the equality of nodes to ensure uniqueness and *less* imposes a traversal order on the nodes. As the names indicate, procedure *Insert* inserts a new node and procedure *Traverse* traverses the tree. A node is only inserted, if it is not already present. *Traverse* applies a procedure *proc* to each node

```
DEFINITION BinTree;

 TYPE
   ApplyProcedure = PROCEDURE (x: Node);
   CompareFunction = PROCEDURE (a, b: Node): BOOLEAN;
   Node = POINTER TO NodeDesc;
   NodeDesc = RECORD END ;
   Tree = POINTER TO TreeDesc;

 PROCEDURE Insert (T: Tree; x: Node);
 PROCEDURE New (VAR T: Tree; equal, less: CompareFunction);
 PROCEDURE Traverse (T: Tree; proc: ApplyProcedure);

END BinTree.
```

Figure 4.1: Definition of module *BinTree*

by traversing the tree in the order of the imposed relation.

Within the module *SweepHull*, the abstract data type *Tree* represents the sequence of distinct sites S. Thus, we extend the type *NodeDesc* of module BinTree with the record fields $r2$, x, y, and nr in the declaration of type *SiteDesc* in module SweepHull (see Figure 4.2). Syntactically, the record name of the extended type (i.e., *BinTree.NodeDesc*) is declared in parentheses as an additional field. Field $r2$ represents the squared distance between the origin and the site, x and y are the site's coordinates, and nr is the index number of the site used solely for output purposes.

The first statement of procedure *Load* initializes S as a sequence of distinct sites (i.e., procedure *SiteEqual*) in polar order (i.e., procedure *SiteLess*). Then, the sequence is filled with sites from some source by inserting them into S. Procedure *Load* establishes the precondition *hull order*. The *WITH*-statement in procedure *Update* is a type guard. The type guard asserts that the parameter s is of type *Site* (i.e., a pointer to *SiteDesc*) over the entire update statement, otherwise the program aborts. Procedure *Update* is discussed in Section 4.6. The procedure *Sweep* invokes the sweephull algorithm by calling the procedure *Traverse* which applies procedure

```
MODULE SweepHull;
 IMPORT
  BinTree;
 TYPE
  Site = POINTER TO SiteDesc;
  SiteDesc = RECORD (BinTree.NodeDesc) r2: LONGINT; x,y,nr: INTEGER; END;
 VAR
  S: BinTree.Tree; (* sequence of sites *)
  N: INTEGER; (* number of sites *)

 PROCEDURE SiteEqual(a,b: BinTree.Node): BOOLEAN;
  BEGIN RETURN (a(Site)^.x = b(Site)^.x) & (a(Site)^.y = b(Site)^.y);
  END SiteEqual;

 PROCEDURE SiteLess(a,b: BinTree.Node): BOOLEAN;
  BEGIN RETURN a(Site)^.r2 < b(Site)^.r2;
  END SiteLess;

 PROCEDURE Load*;
  VAR s: Site; x,y: INTEGER;
  BEGIN BinTree.New(S,SiteEqual,SiteLess);
   N := 0;
   WHILE (* not finished reading *) DO
    (* read x and y *)
    NEW(s); s^.x := x; s^.y := y; s^.r2 := x*x + y*y; s^.nr := N;
    BinTree.Insert(S,s);
    N := N + 1;
   END
  END Load;

 PROCEDURE Update*(s: BinTree.Node);
  BEGIN WITH s: Site DO
   (* Update statement *)
   END
  END Update;

 PROCEDURE Sweep*;
  BEGIN BinTree.Traverse(S,Update);
  END Sweep;

BEGIN
 S := NIL;
END SweepHull.
```

Figure 4.2: Implementation of module *SweepHull*

Update to each node ordered by distance from the origin and, thus, in hull order.

Note that the three procedures, *Load*, *Update*, and *Sweep*, are marked with an asterisk. This export mark indicates that the procedures are part of the module interface. The parameterless procedures *Load* and *Sweep* can be invoked directly from the command interpreter.

4.2 Representation of the Voronoi graph

An equiset is represented by a doubly-connected circular list. The necessary type and variable declarations are shown in Figure 4.3. The elements of the list are called *OneEquiSet* and *TwoEquiSet*, where *One* and *Two* refer to the equiset's cardinality. Both One- and TwoEquiSet extend the type *SmallEquiSet* which connects instances of them into a circular list so that the element types alternate. The pointer to the next element of the list is found in the record *SmallEquiSetDesc* describing the pointer type SmallEquiSet. The record fields *clockwise* and *counterclockwise* point to their neighbors of the circular list. In Figure 4.4, OneEquiSets are denoted by a circled site index. TwoEquiSets appear as small black dots with three incident edges.

The record field *site* in the descriptor of a OneEquiSet refers to a single site. Two or more sites are associated with a TwoEquiSet. In the case of two sites, the equiset consists of the TwoEquiSet's clockwise and counterclockwise neighbors, both of which are OneEquiSets. In the case of several sites, we consider all sites (at least three) stored in the whole circular list.

```
MODULE SweepHull;
 IMPORT
  BinTree, Texts, Oberon, Files, TextFrames, MenuViewers;
 TYPE
  Site = POINTER TO SiteDesc;
  SiteDesc = RECORD (BinTree.NodeDesc) r2: LONGINT; x,y,nr: INTEGER; END;
  SmallEquiSet = POINTER TO SmallEquiSetDesc;
  SmallEquiSetDesc = RECORD clockwise, counterwise : SmallEquiSet END;
  OneEquiSet = POINTER TO OneEquiSetDesc;
  OneEquiSetDesc = RECORD (SmallEquiSetDesc) site : Site END;
  TwoEquiSet = POINTER TO TwoEquiSetDesc;
  TwoEquiSetDesc = RECORD (SmallEquiSetDesc) hull: BOOLEAN; link : TwoEquiSet END;

 VAR
  S: BinTree.Tree; (* sequence of sites *)
  N: INTEGER; (* number of sites *)
  CH: TwoEquiSet; (* access to Voronoi diagram by its convex hull *)
  MARK: BOOLEAN; (* current boolean value to designate a hull TwoEquiSet *)
 (* program code *)
END SweepHull.
```

Figure 4.3: Type and variable declarations of module *SweepHull*

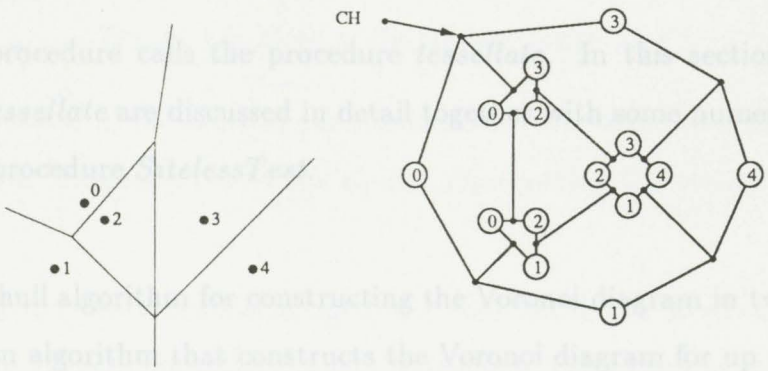

Figure 4.4: Voronoi diagram and pointer structure (two-dimensional)

Figure 4.4 depicts the described pointer structure and its corresponding Voronoi diagram. The convex hull is accessed through the pointer variable CH which points to a TwoEquiSet. Unlike the other equisets, the TwoEquiSets of the convex hull equiset are interpreted as hull segments (because they are specially marked with the inversed boolean value in the *hull* field of the TwoEquiSet).

The conditions added to the invariant $Vor.(2, S, CH)$ are manifested in the depicted data structure. Informally, CH is the data structure accessing and containing the Voronoi base. The links among TwoEquiSets represent all common subsets of cardinality two and the corresponding OneEquiSets must match. These rules also apply to the circular list representing the convex hull. The update step, described in the next section, is solely motivated by the need for re-establishing the invariant on the pointer structure when considering a new site s.

4.3 Core of the update procedure

The update procedure calls the procedure *tessellate*. In this section, procedures *Update* and *tessellate* are discussed in detail together with some numerical considerations of the procedure *SitelessTest*.

The sweephull algorithm for constructing the Voronoi diagram in two dimensions incorporates an algorithm that constructs the Voronoi diagram for up to two dimensions (i.e., 0, 1, and 2). This is because all sites might be located on a line or only a single point might be given. Therefore, the pointer structure must be able to represent zero-, one-, and two-dimensional input. Figure 4.5 shows a Voronoi diagram of collinear sites and its corresponding pointer structure. The structure is computed by

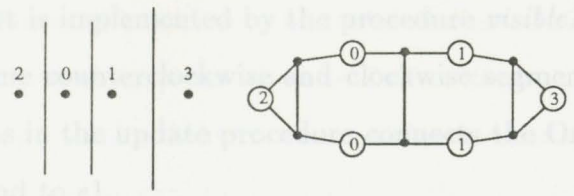

Figure 4.5: Voronoi diagram and pointer structure (collinear sites)

```
PROCEDURE Update*(s: BinTree.Node);
 VAR
  te1,te2: TwoEquiSet; e,e1,e2: TwoEquiSet; (* e between e1,e2 *)
  minoe,oe: OneEquiSet;

  PROCEDURE visibleHull(VAR e1,e2: TwoEquiSet; VAR minoe: OneEquiSet);
   BEGIN .... END visibleHull;

  PROCEDURE tessellate(e: TwoEquiSet);
   BEGIN .... END tessellate;

 BEGIN WITH s: Site DO
   visibleHull(e1,e2,...);
   (* connect hull clockwise e2 *)
    e := e2;
    tessellate(e^.link);
    WHILE e # e1 DO
     e := e^.counterwise^.counterwise(TwoEquiSet);
     tessellate(e^.link)
    END;
   (* connect hull counterclockwise e1 *)
  END (* WITH *)
 END Update;
```

Figure 4.6: Core of the update procedure

incremental updates in the order of the site indices. Note that the lower-dimensional Voronoi diagram is represented by the equiset representing the convex hull.

The core of the two-dimensional update is sketched in the program fragment depicted in Figure 4.6 with unimportant details omitted. The complete program is listed in Appendix B. The update step, which incorporates a new site s into the pointer structure, is embedded in an incremental convex hull algorithm. The main part of the convex hull algorithm involves finding the extreme segments of the convex hull visible

from site s. This part is implemented by the procedure *visibleHull*, which returns in $e1$ and $e2$ the extreme counterclockwise and clockwise segments, respectively. The first of two comments in the update procedure connects the OneEquiSet storing site s to $e2$ and the second to $e1$.

The two comments bracket the embedded Voronoi update step, a WHILE-loop over the visible segments (i.e., TwoEquiSets) of the convex hull between $e2$ and $e1$. The loop invokes the local procedure *tessellate* in clockwise order. Procedure *tessellate* recursively traces the chains whose starting node is accessed by the link field of the TwoEquiSets, as long as the new site s is inside the encountered equiset e (i.e., $SitelessTest(e,s) < 0$ is equivalent to $sitelessTest.(e,s) =$ "s inside").

The trace of the complete chains is intercepted if either $SitelessTest(e,s) = 0$ (i.e., $sitelessTest.(e,s) =$"s on") or $SitelessTest(e,s) > 0$ (i.e., $sitelessTest.(e,s) =$"s outside"), but different actions are taken. In the "on"-case, the equiset e is simply expanded by a OneEquiSet storing s. In the "outside"-case, e is also expanded, but as a TwoEquiSet and not as the whole circular list. The corresponding procedures *Expand* and *TwoThree* are given in Appendix B. Similarly, the trace is intercepted when a convex hull segment is encountered.

Simultaneously, the new equisets are "threaded" together (by the link field of the TwoEquiSets) to $BOUNDARY.sitelessPoints.\{s\}$, the new boundary of the Voronoi region of s. To understand the term "threaded," let us examine more closely the trace starting at one of the convex hull segments. The starting node (i.e., equiset) of all the chains is indicated by the TwoEquiSet on the hull. The trace is branched clockwise into the neighboring equisets, except the one where it came from (i.e., entry-link).

Thus, the procedure *tessellate* traverses a tree of chains in a counterclockwise-first order with respect to the entry-link. Therefore, the interception and hence the expansion of the existing equisets occurs in the clockwise order of the Voronoi region of s. Hence, the new equisets can be threaded together in order to re-establish the invariant imposed on the pointer structure.

In the sweephull implementation, the procedure *SitelessTest* uses the function *anySite*, which returns a random site, by assigning arbitrary sites to a, b and c. The sites are fixed with respect to the entry-link. Despite the possibility of performing a test twice with an identical equiset and site, but with a different entry-link, the outcome of the test is the same. The reason is that the derivation of *sitelessTest* does not make any assumption on the outcome of the random function *anySite*. However, this argument is only valid, if the arithmetic is exact. This however, can be guaranteed, if all calculations are performed in integer arithmetic, thereby avoiding the inaccuracies of floating-point arithmetic. The problem which arises is integer overflow. Unfortunately, the current Oberon implementation does not indicate integer overflows. Furthermore, our estimation of the domain of the procedure *SitelessTest* (i.e., where the implementation always achieves correct results) yields a square which is 137 grid-points wide, assuming 32-bit integers. This very restricted domain is not sufficient in practice and we therefore use floating-point arithmetic. Thus, absolute correctness is not guaranteed in our implementation due to round-off errors.

To establish precondition (3.33), the sites are sorted by their squared distance $r2$ to the origin. By storing $r2$ along with the coordinates, the procedure *sitelessTest* does not need to recompute the squared distance. In comparison with the "Incircle Test" used by Stolfi and Guibas, we save one multiplication while maintaining the

same number of additions. Similarly, using an appropriate data representation, the result of $det.(a, b, c)$ and other terms of the calculation which remain constant for the life-time of an equiset, can be stored within each equiset. Thus, more operations can be saved. More importantly, the coordinates of the same site can be used for a repeated test of *SitelessTest*, yielding a stable evaluation of the test[1] under floating-point arithmetic. A challenge is to replace the incremental convex hull algorithm by a more sophisticated algorithm which uses a tree structure for finding the extreme edges [15].

4.4 Summary

The sweephull algorithm does not compute the locations of the Voronoi vertices. Therefore, all calculations can be performed solely in integer arithmetic, thereby avoiding the inaccuracies of floating-point arithmetic. The only problem to overcome is integer overflow. However, our sweephull algorithm is better off than the sweepline or sweepcircle algorithms, because these algorithms rely on calculating the tangent point of an empty circle with the sweepline or sweepcircle, thus requiring floating-point arithmetic. In the sweephull algorithm, a change from floating-point numbers to integers is straightforward but restrictive.

Furthermore, having equisets of any number of sites, saves comparisons (i.e., calls of "*SitelessTest*"). In implementations where a static representation is preferred, co-circular points are normally represented by multiple "equisets," thereby introducing zero-length edges (i.e., Fortune's sweepline algorithm and Guibas and Stolfi's divide-and-conquer). This representation not only results in additional comparisons in the

[1]Two tests involving same site and equiset return the identical values.

case of cocircularity, but it also introduces additional numerical problems which are difficult to overcome.

Oberon increased the expressive power and flexibility of its predecessors while guaranteeing the type safety of the language. Type safety effectively supported our effort in controlling the dynamic data structure. The use of the new programming environment was convenient and led to an efficient implementation.

Chapter 5

Conclusions

5.1 Summary of results

In the past, degeneracies caused great difficulties when implementing Voronoi algorithms. The "big" assumption stated by Preparata and Shamos that "no four sites are cocircular" does not help to overcome the difficulties in suppressing a supposedly inconsequential and unpleasant lengthy detail. Thus, the assumption is an inadequate simplification of the problem.

In our specification of the Voronoi diagram, which is independent of the dimension, we circumvented these difficulties by paying careful attention to degeneracies. The cornerstone of our specification are the formal definitions of the boolean functions *equi* and *siteless* defining the finite Voronoi set of all siteless equisets. An empty hypersphere is uniquely represented by a siteless equiset even in the presence of degeneracies.

We avoided the term "straight-line dual" to describe the Delaunay diagram and replaced the notion of duality by the two functions *sitelessPoints* and *convexPoints*.

The functions simply provide two different interpretations of the Voronoi set: the Voronoi diagram and the Delaunay diagram. Both diagrams have partitioning properties such that each point is associated with a unique region. Hence, we avoid another irregularity in the theory of Voronoi diagrams.

We are not alone with our criticisms and propositions. We found support in a recent paper by Elbaz and Spehner with similar conclusions [25]. They give a variant of the Guibas and Stolfi algorithm which determines the Delaunay diagram and not a triangulation. They claim that constructing the dual Voronoi diagram leads to needless calculations and emphasize a favorable time behavior of their divide-and-conquer implementation in the presence of degeneracies. Furthermore, they state and prove a theorem for the partitioning property of the two-dimensional Delaunay diagram.

A matrix is a mathematical object accessible to formal manipulation. The reason why Guibas and Stolfi based their proof of the "Incircle test" on a spatial interpretation of the "Incircle matrix" is not obvious. Furthermore, they do not explain why interchanging the parameters inverts the outcome of the test. Interpreting the test concerning a circle and a point of the plane in three-dimensional space prevents a generalization of the test to higher dimensions, since higher dimensional spaces are not intuitive.

We formulated "sitelessTest" as a sign evaluation of a variable Δ in an equation containing two matrices. The test is motivated by our specification and is invulnerable to changes of the parameter order. With our formal approach, we are able to speculate about a similar test in higher dimensions.

Many presentations of algorithms in computational geometry emphasize the analysis of time complexity. Our emphasis, however, is on constructive program design.

In a first attempt, we demonstrated our methodology with derivations of loop-less procedures (e.g., *sitelessTest*) and combinatorial procedures (e.g., *sitelessExpand*). In a second attempt, our methodology or derivation strategy unified the discussion of seemingly unrelated algorithms such as Fortune's sweepline algorithm and incremental algorithms.

The concept of strengthening conditions on a program state resulted not only in an elegant presentation but also in a method for finding new algorithms. The opposite way of weakening the precondition of the sweepline and sweepcircle algorithm revealed the sweephull algorithm. Unfortunately, we were not first in finding the sweephull algorithm. Very recently, we found an article by Tipper presenting similar ideas [24].

In addition to finding the sweephull algorithm, we provided a sketch for its correctness proof, which can be extended to higher dimensions or refined to prove the correctness of either the sweepline or the sweepcircle algorithm. We did the latter and were able to explain the sweepline algorithm in a straightforward manner without reference to Fortune's plane-distorting mapping. We further unified the discussion of the Voronoi diagram with two conjectures, thereby relating the Voronoi diagram to the theory of partially-ordered sets and providing an example of well-foundedness.

Finally, we presented our sweephull implementation in Oberon, derived the mathematical formulae necessary for the sweepcircle algorithm, and verified them by implementation.

5.2 Future research

We proposed three fundamental conjectures which unified several existing theorems. However, we were not able to prove them formally in the style of Dijkstra's mono-

graph. The major handicaps were the underlying set-theoretic and spatial interpretations of our formulae, which we tried to avoid initially. One approach to proving these conjectures would be to abstract the formulae from any interpretation and enrich our repertoire of general formulae.

After the intermediate step of implementing the brute-force program for three dimensions, which would be a good test of our speculation about the function *sitelessTest*, proceeding and adapting the sweephull algorithm to three dimensions should be feasible using a three-dimensional convex hull algorithm.

A further investigation of the sweephull's average time complexity seems to be another worth-while avenue of future research.

Bibliography

[1] P.G.L. Dirichlet, "Über die Reduktion der positiven quadratischen Formen mit drei unbestimmten ganzen Zahlen," *J. Reine Angew. Math*, Vol 40, pp. 209–227, 1850.

[2] G. Voronoi, "Nouvelles applications des paramètres continus à la théorie des formes quadratiques. Première Mémoire: Sur quelques propriétés des formes quadratiques positives parfaites," *J. Reine Angew. Math*, Vol 133, pp. 97–178, 1907.

[3] G. Voronoi, "Deuxième Memoire: Recherches sur les parallélloèdres primitifs," *J. Reine Angew. Math*, Vol 134, pp. 198–287, 1908.

[4] B. Delauney, "Sur la sphère vide," *Bull. Acad. Sci USSR(VII), Classe Sci. Mat. Nat.*, pp. 793–800, 1934.

[5] M. Shamos, "Geometric Complexity," *Seventh ACM Annual Symp. on Theory of Comput.*, pp. 224–233, May 1975.

[6] M. Shamos and D. Hoey, "Closest-Point Problems," *Proc. 16th Ann. IEEE Symposium on Automata and Computability Theory*, pp. 151–162, 1975.

[7] C. Lawson, "Software for C Surface Interpolation," *Mathematical Software III*, Proceedings of a Symposium by the Mathematics Research Center, pp. 161–194, March 1977.

[8] P. Green and R. Sibson, "Computing Dirichlet tessellations in the plane," *The Computer Journal*, Vol 21 No 2, pp. 168–173, 1978.

[9] K. Brown, "Voronoi Diagrams from Convex Hulls," *Information Processing Letters*, Vol 9, No 5, pp. 223–228, December 1979.

[10] D. T. Lee and J. Schachter, "Two Algorithms for Constructing a Delaunay Triangulation," *International Journal of Computer and Information Sciences*, Vol 9, No 3, pp. 219–242, 1980.

[11] A. Bowyer, "Computing Dirichlet tessellations," *The Computer Journal*, Vol 24, No 2, pp. 162–166, 1981.

[12] D. Watson, "Computing the n-dimensional Delaunay Tessellation with Application to Voronoi Polytopes," *The Computer Journal*, Vol 24, No 2, pp. 167–172, 1981.

[13] T. Ohya, M. Iri, and K. Murota "Improvements of the Incremental Method for the Voronoi Diagram with Computational Comparison of Various Algorithms," *Journal of the Operations Research Society of Japan*, Vol 27, No 4, pp. 306–336, December 1984.

[14] D. T. Lee and F. Preparata, "Computational Geometry — A Survey," *IEEE Transaction on Computers*, Vol c-33, No 12, pp. 1072–1101, December 1984.

[15] F. Preparata and M. Shamos, *Computational Geometry*, Springer-Verlag, 1985.

[16] L. Guibas and J. Stolfi, "Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams," *ACM Transactions on Graphics*, Vol 4, No 2, pp. 74–123, April 1985.

[17] H. Edelsbrunner and R. Seidel, "Voronoi diagrams and arrangements," *Discrete Comput. Geom.*, Vol 1, pp. 25–44, 1986.

[18] S. Fortune, "A Sweepline Algorithm for Voronoi Diagrams," *Proc. 2nd Ann. Symp. on Computational Geometry, ACM*, pp. 313–322, 1986.

[19] S. Fortune, "A Sweepline Algorithm for Voronoi Diagrams," *Algorithmica 2*, pp. 175–193, Springer-Verlag, 1987.

[20] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, Springer-Verlag, 1987.

[21] J. O'Rourke, "Computational Geometry," *Annual Reviews Computer Science*, Vol 3, pp. 389–411, 1988.

[22] G. Toussaint (editor), *Computational Morphology*, North-Holland, 1988.

[23] L. Guibas, D. Knuth, and M. Sharir, "Randomized Incremental Construction of Delaunay and Voronoi Diagrams," Technical Report, Department of Computer Science, Stanford University, January 1990.

[24] J. Tipper, "A Straightforward Iterative Algorithm for the Planar Voronoi diagram," *Information Processing Letters*, Vol 34, pp. 155–160, April 1990.

[25] M. Elbaz and J.-C. Spehner, "Construction of Voronoi Diagrams in the Plane by Using Maps," *Theoretical Computer Science*, No 77, pp. 331–343, December 1990.

[26] K. Wong, "Technique for Optimizing Fortune's Plane-Sweep Algorithm for Voronoi Diagrams," Master Thesis, University of Victoria, April 1991.

[27] Z. Manna, *Mathematical Theory of Computation*, McGraw-Hill, 1974.

[28] E. Dijkstra, *A Discipline of Programming*, Prentice-Hall, 1976.

[29] K. Chandy and J. Misra, *Parallel Program Design*, Addison-Wesley, 1988.

[30] E. Dijkstra and C. Scholten, *Predicate Calculus and Program Semantics*, Springer-Verlag, 1989.

[31] N. Wirth and J. Gutknecht, "The Oberon System," *Software Practice & Experience*, Vol 19, No 9, pp. 857–894, July 1988.

Appendix A

Arithmetic of the sweepcircle algorithm

Our sweepcircle algorithm generalizes Fortune's sweepline algorithm [19] by replacing the sweepline with a sweepcircle. The circle sweep starts at a site, the center of a sweepcircle of initial radius zero, and proceeds by increasing the radius of the sweepcircle. Without loss of generality, we assume that the center coincides with the origin. The sweepcircle is partitioned into arcs associated with a site. An arc is associated to a site by being a compound subset of the intersection of the sites' mapped regions with the sweepcircle. The mapping is similar to the one used in Fortune's paper. Instead of mapping a point in a particular direction, a point is mapped along its ray from the center. The partitioning of the circle is referred to as the sweepcircle status.

We restrict the sites to be points in the plane. In addition to the notation used by Fortune, we define (p_δ, p_φ) to be the polar coordinates of a point p in $\mathcal{R}^2$, where p_δ is the distance from the origin and p_φ is the angle. The origin is denoted by o. The

points $p, q \in \mathcal{R}^2$ are lexicographically ordered, $p < q$, iff $p_\delta < q_\delta \vee p_\delta = q_\delta \wedge p_\varphi < q_\varphi$. $C_{c,r}$ denotes a circle centered at point c with radius r and $E_{f1,f2,d}$ denotes an ellipse with foci $f1$ and $f2$ and major diameter d.

A.1 Mapping $*$

The mapping $*$: $\mathcal{R}^2 \rightarrow \mathcal{R}^2$ is defined by $*(z) = (z_\delta + d(z), z_\varphi)$.

Lemma 1 *The mapping $*$ is one-to-one on the Voronoi diagram V.*

Proof We prove the lemma by contradiction. Suppose the mapping is not one-to-one. That is,

$$z = *(a) = *(b) \wedge a \neq b, \text{ where } a \in V \wedge b \in V$$

By the definition of the Voronoi diagram and the mapping $*$:

$$a \in V \Rightarrow \exists p, q : d_p(a) = d_q(a) = d_z(a) = d(a)$$

$$b \in V \Rightarrow \exists m, n : d_m(b) = d_n(b) = d_z(b) = d(b)$$

where p, q are the closest sites to a and similarly for m, n and b. By the definition of the mapping $*$ and the supposition, $z = *(a) = *(b)$, the points a, b, and z are collinear and z is either to the left or to the right of a and b. Consider two circles centered at a and b with radii $d(a)$ and $d(b)$ (i.e., the distance to their closest sites). Both circles touch at point z and one is contained entirely within the other. A contradiction arises because the sites lying on the inner circle are closer to the center of the outer circle than the sites lying on the outer circle. *End of Proof.*

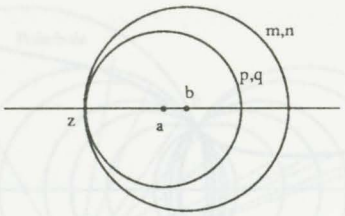

Figure A.1: Contradiction

In the example depicted in Figure A.1, there exist the points p and q such that $d_b(p) < d_b(m) \vee d_b(q) < d_b(m)$, which is a contradiction to m being the closest site to b.

Lemma 2 *In region $*(R_p)$, the site p is the origin's closest point:*

$$\{x \in \mathcal{R}^2 \mid d_o(x) \leq d_o(p)\} \cap *(R_p) = \{p\}.$$

Proof Consider $q \in R_p$. By the definition of the mapping $*$, $d_o(*(q)) = d_o(q) + d_q(p)$, and by the Triangle Inequality $d_o(q) + d_q(p) \geq d_o(p)$, where equality holds only if o, q, and p are collinear. That is, $*(q) = p$. *End of Proof.*

Informally, the mapped region of site p lies outside the sweepcircle, when encountering p. Moreover, p is the lexicographically lowest point of its mapped region. To analyze the mapping $*$ further, we consider the auxiliary function $*_p : \mathcal{R}^2 \rightarrow \mathcal{R}^2$, defined by $*_p(z) = (z_\delta + d_p(z), z_\varphi)$. We apply the mapping $*_p$ to lines and call the resulting curves *polarbolas* (see Figure A.2). Polarbolas differ from hyperbolas by using polar rather than Cartesian coordinates for their construction. Without loss of generality, lines are given as perpendicular bisectors between two points p and q, $B_{p,q}$, and we assume p to be further away from the origin than q.

Lemma 3 *Suppose o, q, and p are collinear and let A be the line resulting from scaling $B_{q,p}$ by a factor of two away from the origin. Then A is an asymptote of*

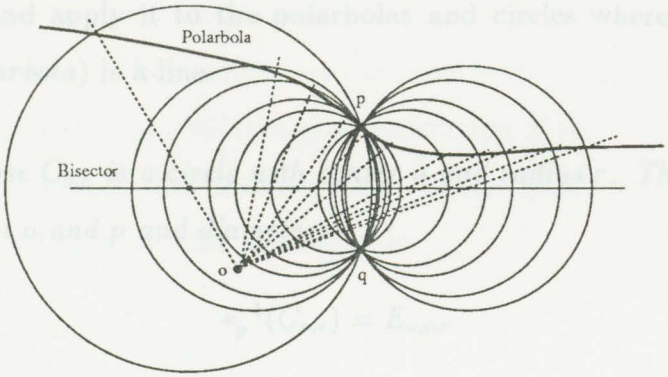

Figure A.2: Polarbola

$*_p(B_{p,q})$. *Furthermore, if* $2d_o(q) = d_o(p)$, *then* $A = *_p(B_{p,q})$.

Proof Without loss of generality, assume that $B_{p,q}$ is horizontal. Let m be the slope of a ray through o and let $a = (d_o(p) + d_o(q))/2$ and $b = (d_o(p) - d_o(q))/2$. The y-coordinate of $*(t)$, where point t is the intersection of a ray and $B_{p,q}$, is given by the following.

$$y = a + \frac{\sqrt{(m^2+1)(a^2+m^2b^2)}}{m^2+1}$$

Decreasing m to zero

$$\lim_{m \to 0} y = 2a$$

shows that A is an asymptote. Furthermore if $a = b$, then

$$y = 2a$$

which is independent of m. *End of Proof.*

A.2 Inverse mapping $*^{-1}$

We are most interested in finding the intersection of the polarbola $*_p(B_{q,p})$ and the sweepcircle. Thus, we define the inverse mapping $*_p^{-1}$: $\mathcal{R}^2 \rightarrow \mathcal{R}^2$ such that

$*_p^{-1}(*_p(z)) = z$ and apply it to the polarbolas and circles where $*_p$ is one-to-one. Trivially, $*_p^{-1}(polarbola)$ is a line.

Lemma 4 *Suppose $C_{o,r}$ is a circle with center o and radius r. Then $*_p^{-1}(C_{o,r})$ is an ellipse with foci at o and p and diameter r:*

$$*_p^{-1}(C_{o,r}) = E_{o,p,r}$$

Proof Consider a point e such that $*_p(e) = (r, \varphi)$ for constant r. Applying the definition of $*_p$ and making use of the one-to-one property yields

$$*_p(e) = (e_\delta + d_p(e), e_\varphi) = (r, \varphi)$$

Then, by replacing e_δ with $d_o(e)$ we obtain

$$d_o(e) + d_p(e) = r \wedge e_\varphi = \varphi, \text{ for all } \varphi$$

which satisfies the definition of an ellipse with foci o,p and diameter r. Thus, e is a point on the ellipse $E_{o,p,r}$. *End of Proof.*

Lemma 5 *Suppose $C_{o,r}$ is a circle with center o and radius r. Then the circles inversely mapped by either the point p or q intersect on the bisector between p and q:*

$$*_p^{-1}(C_{o,r}) \cap *_q^{-1}(C_{o,r}) \subset B_{p,q}$$

Proof For all points e:

$$e \in *_p^{-1}(C_{o,r}) \Rightarrow d_o(e) + d_p(e) = r \wedge e \in *_q^{-1}(C_{o,r}) \Rightarrow d_o(e) + d_q(e) = r$$

Therefore, $d_p(e) = d_q(e)$ and e lies on $B_{p,q}$. *End of Proof.*

Now, we are able to calculate the angles of the intersections between a polarbola $*_p(B_{p,q})$ and a circle $C_{o,r}$. These angles are the same as the angles of the intersections of two ellipses $E_{o,p,r}$ and $E_{o,q,r}$. We first give the formula of the ellipse $E_{o,p,r}$ and $E_{o,q,r}$ and then find the angles of their intersections. A point (δ, φ) on an ellipse with one focus point at the origin o satisfies the equation

$$\delta = \frac{ef}{1 - e\cos\varphi}$$

where e is the eccentricity ($0 < e < 1$ for an ellipse) and f is the distance between the focus o and the directrix perpendicular to the major diameter. The eccentricity e is defined to be the ratio of the distance c between the ellipse center and their foci and the length of the major axis a.

$$e = \frac{c}{a}$$

Trivially for $E_{o,p,r}$: $c = p_\delta/2$ and $a = r/2$. Thus,

$$e = \frac{p_\delta}{r}$$

By the alternative definition of an ellipse, where the ratio of the distance of a point on an ellipse to the focus and its distance from the directrix is constant e, we calculate the constant ratio e for the point s, which is nearest to point l of the directrix (see Figure A.3).

$$\frac{d_s(o)}{d_s(l)} = e = \frac{p_\delta}{r}$$

Because $d_s(o) = (r - p_\delta)/2$ and $d_s(l) = f - (r - p_\delta)/2$

$$f = \frac{r^2 - p_\delta^2}{2p_\delta}$$

Replacing e and f, and rotating the ellipse by p_φ gives

$$\delta = \frac{r^2 - p_\delta^2}{2(r - p_\delta\cos(\varphi - p_\varphi))}$$

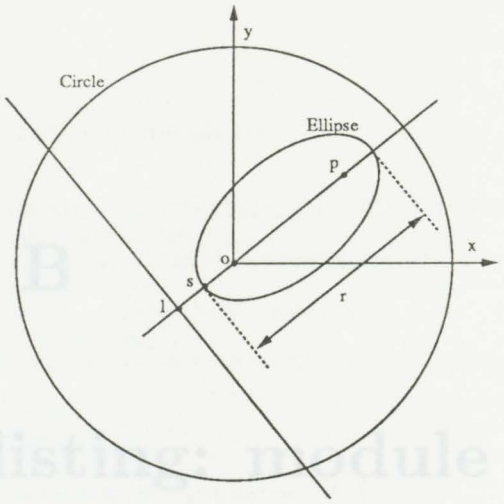

Figure A.3: Inverse mapping of a circle

Similarly, we can derive the equation for $E_{o,q,r}$

$$\delta = \frac{r^2 - q_\delta^2}{2(r - q_\delta \cos(\varphi - q_\varphi))}$$

and finally obtain the implicit equation for the angles of the two intersections:

$$A \cos\varphi + B \sin\varphi + C = 0$$

where

$$\begin{aligned} A &= q_x(p_\delta^2 - r^2) - p_x(q_\delta^2 - r^2) \\ B &= q_y(p_\delta^2 - r^2) - p_y(q_\delta^2 - r^2) \\ C &= r(q_\delta^2 - p_\delta^2) \end{aligned}$$

The explicit solution of the angles is

$$\varphi_{1,2} = 2 \arctan \frac{-B \pm \sqrt{A^2 + B^2 - C^2}}{C - A}$$

Note that if we only compare angles with each other, we do not have to compute the arcus tangent and swapping p and q means swapping φ_1 with φ_2.

Appendix B

Program listing: module SweepHull

```
MODULE SweepHull;
 IMPORT
  BinTree, Texts, Oberon, Files, TextFrames, MenuViewers;
 TYPE
  Site = POINTER TO SiteDesc;
  SiteDesc = RECORD (BinTree.NodeDesc) r2: LONGINT; x,y,nr: INTEGER; END;
  SmallEquiSet = POINTER TO SmallEquiSetDesc;
  SmallEquiSetDesc = RECORD clockwise, counterwise : SmallEquiSet END;
  OneEquiSet = POINTER TO OneEquiSetDesc;
  OneEquiSetDesc = RECORD (SmallEquiSetDesc) site : Site END;
  TwoEquiSet = POINTER TO TwoEquiSetDesc;
  TwoEquiSetDesc = RECORD (SmallEquiSetDesc) hull: BOOLEAN; link : TwoEquiSet END;

 VAR
  S: BinTree.Tree; (* sequence of sites *)
  N: INTEGER; (* number of sites *)
  CH: TwoEquiSet; (* access to Voronoi diagram by its convex hull *)
  MARK: BOOLEAN; (* current boolean value to designate a hull TwoEquiSet *)
  W: Texts.Writer;

(* ---------- I N P U T ---------- *)

 PROCEDURE SiteEqual(a,b: BinTree.Node): BOOLEAN;
  BEGIN RETURN (a(Site)^.x = b(Site)^.x) & (a(Site)^.y = b(Site)^.y);
  END SiteEqual;

 PROCEDURE SiteLess(a,b: BinTree.Node): BOOLEAN;
  BEGIN RETURN a(Site)^.r2 < b(Site)^.r2;
  END SiteLess;

 PROCEDURE Load*;
  VAR
   Sit: Files.File; SitR: Files.Rider;
   ch: CHAR; T: Texts.Text; Scan: Texts.Scanner;
   x: LONGINT; s: Site;
  BEGIN BinTree.New(S,SiteEqual,SiteLess);
   Sit := Files.Old("Voronoi.Sit");
   Files.Set(SitR,Sit,0); Files.Read(SitR,ch);
   T := TextFrames.Text(""); Texts.OpenWriter(W);
   WHILE ~SitR.eof DO
    Texts.Write(W,ch); Files.Read(SitR,ch);
   END;
   Files.Close(Sit); Texts.Append(T,W.buf);
   Texts.OpenScanner(Scan,T,0);
   Texts.Scan(Scan); N := 0;
   WHILE Scan.class = Texts.Int DO
    x := Scan.i; Texts.Scan(Scan);
    IF Scan.class = Texts.Int THEN
     NEW(s); s^.x := SHORT(x); s^.y := SHORT(Scan.i);
     s^.r2 := s^.x*s^.x + s^.y*s^.y;
     s^.nr := N; BinTree.Insert(S,s);
    END;
    Texts.Scan(Scan); N := N + 1;
   END;
   Texts.WriteInt(W,N,1); Texts.WriteString(W," sites loaded");
   Texts.WriteLn(W); Texts.Append(Oberon.Log,W.buf);
  END Load;
```

```
(* ---------- O U T P U T ---------- *)

PROCEDURE Write(VAR W: Texts.Writer; s: Site);
 BEGIN Texts.WriteInt(W,s^.nr,1);
 END Write;

PROCEDURE WriteSet(VAR W: Texts.Writer; e: TwoEquiSet);
 VAR e0: TwoEquiSet;
 BEGIN
  Texts.Write(W,"{");
  IF (e # NIL) THEN
   e0 := e;
   REPEAT
    Write(W,e^.clockwise(OneEquiSet)^.site);
    e := e^.clockwise^.clockwise(TwoEquiSet);
    IF e^.hull = MARK THEN Texts.Write(W,".") END;
    IF N > 10 THEN Texts.Write(W,",") END;
   UNTIL e = e0;
  ELSE
   Texts.WriteString(W,"NIL")
  END;
  Texts.Write(W,"}");
 END WriteSet;

PROCEDURE Mark(e: TwoEquiSet; mark: BOOLEAN);
 VAR e0: TwoEquiSet;
 BEGIN
  e0 := e;
  REPEAT
   e^.hull := mark;
   e := e^.clockwise^.clockwise(TwoEquiSet);
  UNTIL e = e0;
 END Mark;

PROCEDURE VoronoiBase*;
 VAR
  V: MenuViewers.Viewer;
  T: Texts.Text;
  X,Y: INTEGER;
  e,e0: TwoEquiSet;

 PROCEDURE markWrite(e: TwoEquiSet);
  VAR e0: TwoEquiSet;
   BEGIN
    IF (e # NIL) & (e^.hull # MARK) THEN
     WriteSet(W,e);
     Mark(e,MARK);
     e0 := e;
     e := e^.clockwise^.clockwise(TwoEquiSet);
     WHILE e#e0 DO
      markWrite(e^.link);
      e := e^.clockwise^.clockwise(TwoEquiSet)
     END
    END
   END markWrite;
```

```
  BEGIN
   IF CH # NIL THEN
    T := TextFrames.Text("");
    Oberon.AllocateUserViewer(0, X, Y);
    V := MenuViewers.New(
     TextFrames.NewMenu("VoronoiBase.Eqs", "System.Close System.Copy System.Grow Edit.Search Edit.Store"),
     TextFrames.NewText(T, 0),
     TextFrames.menuH,
     X, Y);
    Texts.WriteString(W,"B = set of all large siteless equisets { ");
    markWrite(CH^.link);
    Texts.WriteString(W," }"); Texts.WriteLn(W);
    Texts.WriteString(W,"CH = convex hull in clockwise order ");
    WriteSet(W,CH); Texts.Append(T,W.buf);
    Mark(CH,~MARK);
    MARK := ~MARK
   END
  END VoronoiBase;

(* ---------- S W E E P H U L L   A L G O R I T H M ---------- *)
  PROCEDURE Distance*(a,b: Site): LONGINT;
   BEGIN RETURN (a^.x - b^.x) * (a^.x - b^.x) + (a^.y - b^.y) * (a^.y - b^.y);
   END Distance;

  PROCEDURE Det3(a,b,c : Site): LONGINT;
   (*
   Returns a positive INTEGER, if a,b,c are in clockwise order (point c is right of line a,b)
   Returns Zero, if a,b,c are collinear
   Returns a negative INTEGER, if a,b,c are in counterclockwise order (point c is left of line a,b)
   *)
   VAR A,B,C,D : INTEGER; N: LONGINT;
   BEGIN
    A := b^.x - a^.x;
    B := b^.y - a^.y;
    C := c^.x - a^.x;
    D := c^.y - a^.y;
    N := C * B - A * D;
    RETURN N;
   END Det3;

  PROCEDURE EquiTest(e: TwoEquiSet; s: Site) : LONGINT;
   (*
   Returns Zero, if s,e are collinear
   Returns not Zero, if s,e are cocircular (positive if clockwise, negative if counterwise)
   *)
   VAR a,b: Site;
   BEGIN
    a := e^.counterwise(OneEquiSet)^.site;
    b := e^.clockwise(OneEquiSet)^.site;
    RETURN Det3(s,a,b);
   END EquiTest;
```

```
PROCEDURE Det4(a,b,c,d : Site) : LONGINT;
 (*
 Returns a positive INTEGER, if the circle a,b,c includes d
 Returns Zero, if a,b,c,d are cocircular
 Returns a negative INTEGER, if d is outside the circle a,b,c
 *)
 VAR t: LONGINT; d1,d2,d3,v,v1,v2,v3 : LONGREAL;
 BEGIN
  t := Det3(a,b,c);
  IF t <= 0 THEN HALT(30) END;
  d1 := (d^.r2 - a^.r2); v1 := t; v1 := v1 * d1;
  d2 := (a^.r2 - c^.r2); v2 := Det3(a,b,d); v2 := v2 * d2;
  d3 := (a^.r2 - b^.r2); v3 := Det3(a,d,c); v3 := v3 * d3;
  v := v1 + v2 + v3;
  IF v < 0 THEN RETURN 1 ELSIF v > 0 THEN RETURN -1 ELSE RETURN 0 END;
 END Det4;

PROCEDURE SitelessTest(e: TwoEquiSet; s: Site): LONGINT;
 (*
 S = {s, sites in e}
 Returns a positive INTEGER, if siteless.e
 Returns Zero, if s degenerates e (not siteless.e)
 Returns a negative INTEGER, if not siteless.e
 *)
 VAR a,b,c: Site;
 BEGIN
  a := e^.counterwise(OneEquiSet)^.site;
  b := e^.clockwise(OneEquiSet)^.site;
  c := e^.clockwise^.clockwise^.clockwise(OneEquiSet)^.site;
  RETURN - Det4(a,b,c,s);
 END SitelessTest;

 PROCEDURE Connect(se1,se2,se3: SmallEquiSet);
 BEGIN
  se1^.clockwise := se2; se2^.counterwise := se1;
  se2^.clockwise := se3; se3^.counterwise := se2;
 END Connect;

PROCEDURE Expand(e: TwoEquiSet; s: Site; VAR thread: TwoEquiSet);
 VAR
  oe: OneEquiSet;
  te: TwoEquiSet;
 BEGIN
  NEW(oe); oe^.site := s;
  NEW(te); te^.hull := ~MARK;
  te^.link := thread; thread^.link := te;
  (* c and d *) Connect(oe,te,e^.clockwise);
  NEW(thread); thread^.hull := ~MARK;
  (* a and b *) Connect(e^.counterwise,thread,oe);
 END Expand;
```

```
PROCEDURE TwoThree(e: TwoEquiSet; s: Site; VAR thread: TwoEquiSet);
 VAR
  oe,oe1,oe2: OneEquiSet;
  te: TwoEquiSet;
 BEGIN
  IF e^.hull # MARK THEN (* usually *)
   NEW(oe1); oe1^.site := e^.counterwise(OneEquiSet)^.site;
   NEW(oe2); oe2^.site := e^.clockwise(OneEquiSet)^.site;
   NEW(oe); oe^.site := s;
   NEW(te); te^.hull := ~MARK;
   (* a and b *) Connect(oe1,te,oe2); te^.link := e^.link; e^.link^.link := te;
   NEW(te); te^.hull := ~MARK;
   (* c and d *) Connect(oe,te,oe1); te^.link := thread; thread^.link := te;
   NEW(thread); thread^.hull := ~MARK;
   (* e and f *) Connect(oe2,thread,oe);
  ELSE (* no equiset affected *)
   NEW(oe1); oe1^.site := e^.counterwise(OneEquiSet)^.site;
   NEW(oe2); oe2^.site := e^.clockwise(OneEquiSet)^.site;
   NEW(oe); oe^.site := s;
   NEW(te); te^.hull := ~MARK;
   (* a and b *) Connect(oe2,te,oe1); te^.link := e^.link; e^.link^.link := te;
   NEW(te); te^.hull := ~MARK;
   (* e and f *) Connect(oe,te,oe2); te^.link := thread; thread^.link := te;
   NEW(thread); thread^.hull := ~MARK;
   (* c and d *) Connect(oe1,thread,oe);
  END
 END TwoThree;

PROCEDURE Update*(s: BinTree.Node);
 VAR
  te1,te2: TwoEquiSet; e,e1,e2: TwoEquiSet; (* e between e1,e2 *)
  minoe,oe: OneEquiSet;

  PROCEDURE visibleHull(VAR e1,e2: TwoEquiSet; VAR minoe: OneEquiSet);
   (*
   from view out of s it returns
   in e1 the most counterwise (rightmost) and
   in e2 the most clockwise (leftmost) visible edge
   if the hull is still degenerate it returns the visible OneEquiSet site
   *)
   VAR e: TwoEquiSet;
   BEGIN WITH s: Site DO
     (* find non-degenerate edge *)
     minoe := CH^.clockwise(OneEquiSet);
     e := CH^.clockwise^.clockwise(TwoEquiSet);
     WHILE (EquiTest(e,s) = 0) & (e # CH) DO
      IF Distance(s,e^.clockwise(OneEquiSet)^.site) < Distance(s,minoe^.site) THEN
       minoe := e^.clockwise(OneEquiSet)
      END;
      e := e^.clockwise^.clockwise(TwoEquiSet);
     END;
     IF e = CH THEN (* degenerate *)
      e1 := NIL; e2 := NIL;
     ELSE (* find visible part *)
      minoe := NIL;
      e1 := e; e2 := e;
      IF EquiTest(e,s) < 0 THEN (* e visible from s *)
```

```
        WHILE EquiTest(e2^.clockwise^.clockwise(TwoEquiSet),s) < 0 DO
          e2 := e2^.clockwise^.clockwise(TwoEquiSet)
        END;
        WHILE EquiTest(e1^.counterwise^.counterwise(TwoEquiSet),s) < 0 DO
          e1 := e1^.counterwise^.counterwise(TwoEquiSet);
        END;
      ELSE (* e invisible from s *)
        WHILE EquiTest(e1^.clockwise^.clockwise(TwoEquiSet),s) >= 0 DO
          e1 := e1^.clockwise^.clockwise(TwoEquiSet)
        END;
        e1 := e1^.clockwise^.clockwise(TwoEquiSet);
        WHILE EquiTest(e2^.counterwise^.counterwise(TwoEquiSet),s) >= 0 DO
          e2 := e2^.counterwise^.counterwise(TwoEquiSet)
        END;
        e2 := e2^.counterwise^.counterwise(TwoEquiSet);
      END;
    END
  END (* WITH *)
END visibleHull;

PROCEDURE tessellate(e: TwoEquiSet);
  VAR e1,e2: TwoEquiSet; t: LONGINT;
  BEGIN WITH s: Site DO
    IF e^.hull = MARK THEN
      TwoThree(e^.link,s,te2);
    ELSE
      t := SitelessTest(e,s);
      IF t < 0 THEN (* not siteless.e *)
        e1 := e;
        e := e^.clockwise^.clockwise(TwoEquiSet);
        WHILE e # e1 DO
          tessellate(e^.link);
          e := e^.clockwise^.clockwise(TwoEquiSet);
        END
      ELSIF t > 0 THEN (* siteless.e *)
        TwoThree(e^.link,s,te2);
      ELSE (* cocircular *)
        Expand(e,s,te2);
      END
    END;
  END (* WITH *)
END tessellate;

BEGIN WITH s: Site DO
  IF CH # NIL THEN
    visibleHull(e1,e2,minoe);
    IF minoe = NIL THEN (* 2 dimensional *)
      NEW(oe); oe^.site := s;
      NEW(te2); te2^.hull := MARK;
      (* c and d *) Connect(oe,te2,e2^.clockwise);
      e := e2;
      tessellate(e^.link);
      WHILE e # e1 DO
        e := e^.counterwise^.counterwise(TwoEquiSet);
        tessellate(e^.link)
      END;
```

```
      NEW(te1); te1^.hull := MARK;
      (* a and b *) Connect(e1^.counterwise,te1,oe);
      te1^.link := te2; te2^.link := te1;
      CH := te1;
     ELSE (* 1 dimensional *)
      IF CH^.clockwise = CH^.counterwise THEN (* two sites *)
       NEW(oe); oe^.site := s;
       NEW(te1); te1^.hull := MARK;
       Connect(CH,oe,te1);
       Connect(te1,minoe,CH);
       CH^.link := te1; te1^.link := CH;
      ELSE (* at least two collinear sites *)
       NEW(oe); oe^.site := minoe^.site;
       NEW(te1); te1^.hull := MARK;
       NEW(te2); te2^.hull := MARK;
       Connect(te1,oe,minoe^.clockwise);
       NEW(oe); oe^.site := s;
       Connect(minoe,te2,oe);
       oe^.clockwise := te1; te1^.counterwise := oe;
       te1^.link := te2; te2^.link := te1;
      END
     END
    ELSE (* 0 dimensional *)
     NEW(oe); oe^.site := s;
     NEW(te2); te2^.hull := MARK;
     Connect(oe,te2,oe);
     te2^.link := NIL;
     CH := te2;
    END;
   END (* WITH *)
  END Update;

 PROCEDURE Sweep*;
  BEGIN CH := NIL;
   IF S # NIL THEN
    Texts.WriteString(W,"start sweep: "); Texts.WriteInt(W,Oberon.Time(),1); Texts.WriteLn(W);
    Texts.Append(Oberon.Log,W.buf);
    BinTree.Traverse(S,Update);
    Texts.WriteString(W,"stop sweep: "); Texts.WriteInt(W,Oberon.Time(),1); Texts.WriteLn(W);
    Texts.Append(Oberon.Log,W.buf)
   END
  END Sweep;

BEGIN
 S := NIL;
 Texts.OpenWriter(W);
 MARK := TRUE;
END SweepHull.
```

VITA

Surname: **Heuberger** Given Names: **Philipp André**
Place of Birth: **Zürich, Switzerland** Date of Birth: **July, 2nd 1962**

Educational Institutions Attended:

Swiss Federal Institute of Technology Zürich	1982 to 1988
University of Victoria	1989 to 1991

Degrees Awarded:

Diploma	1988	Swiss Federal Institute of Technology (ETH), Zürich, Switzerland

Partial Copyright License

Title of Thesis:

A Constructive Approach to Sweep Algorithms for Voronoi Diagrams

Author: [redacted]
Philipp A. Heuberger
May 20, 1991