Degeneracy Proof Predicates for the Additively Weighted Voronoi Diagram

by

David L. Millman

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science
Department of Computer Science
Courant Institute of Mathematical Sciences
New York University
May 2007

Prof. Chee K. Yap

© David L. Millman
All Rights Reserved, 2007

To my mom, dad and sister.

Abstract

This thesis focuses on the Additively Weighted Voronoi diagram. It begins by presenting the history of the diagram and some of the early algorithms used for its generation [OBSC00, Aur91]. The paper then addresses the more recent incremental approach to calculating the diagram, as seen in the 2D Apollonius Graphs (Delaunay Graphs of Disks) package of CGAL [KY06]. Next, the algorithm of Boissonnat et al. [BD05] for calculating Convex Hulls is presented. We then apply the predicates presented by Bossonnat to the CGAL implementation of the AW-Voronoi diagram, and the results are discussed. The main contribution of this paper results in predicates of the AW-Voronoi diagram, with a lower algebraic degree which also handle degeneracies in such a way as to produce a conical result.

Acknowledgments

I would like to thank Professor Chee Yap for his encouragement, knowledge and advice as an adviser and mentor. I would also like to thank Professor Ernest Davis for being my thesis second reader. Many thanks to Christophe Delage for his time spent speaking with me at INRIA, Lee Parnell Thompson and Sudhama Bhatia for bouncing ideas around with me in the lab, and Doug McNamara for his grammatical feedback. I especially would like to thank my Mom and Dad, my sister Lisa and my Grandma for all their love as well as their amazing support over the past two years.

Table of Contents

Abstract **iv**

Acknowledgments **v**

List of Figures **viii**

Introduction **1**

1 Preliminary **4**

1.1 Basic definitions 4

1.2 Additively-Weighted Voronoi Diagram 4

1.3 Power Diagram 6

2 Additively Weighted Voronoi Diagram in 2D **8**

2.1 Insertion 8

2.2 Deletion 10

2.3 Predicates 11

3 Convex Hull and Voronoi Diagram of Additively Weighted Points **13**

3.1 Correspondence between the Convex Hull and the Power Diagram 13

4 Degeneracy Proof Predicates for the Additively Weighted Voronoi Diagram in 2D **19**

4.1 VertexConflict Introduction 20

4.2 Orientation predicate 22

4.3 RadicalIntersection predicate 23

4.4 RadicalSide predicate 24

4.5 VertexConflict Non-Degenerate 25

4.6 Degeneracies 26

4.7 OrderOnLine 27

4.8 NDPowerTest 28
4.9 NDRadicalSide and NDRadicalIntersection 30
4.10 NDVertexConflict 32
4.11 NDEdgeConflict 32
4.12 Algebraic Degree Analysis 36
4.13 Experimental Results 38
4.14 Further Work 39

Appendix **41**

A.1 Increasing the Weights of the Generator Sites in the AW-Voronoi Diagram does not change the Diagram 41
A.2 The Bisectors of a AW-Voronoi Diagram in 2D are Hyperbolic Arcs 41
A.3 The Bisectors of a Power Diagram in 2D are Lines 42
A.4 Parabolas of a beach line intersect at two points 44

Bibliography **46**

List of Figures

1 A portion of John Snow's map showing the area surrounding the broad street water pump. Each bar represents a death at that address, and the solid gray line (which replaced Snow's original dotted line) is equidistant from the Broad Street pump and the nearest alternative pump. 2

1.1 An example of the AW-Voronoi diagram in $\mathbb{R}^2$ 5

1.2 An example of the Power diagram in $\mathbb{R}^2$ 6

2.1 A set of generator sites (red) and their induced AW-Voronoi diagram (blue), as well as the dual of the AW-Voronoi diagram, the Apollonius graph(green) 9

3.1 A convex hull in $\mathbb{R}^2$ of two sites demonstrating facets of circularity $k, k = 0, 1$ . . 14

3.2 A face, f and one of its sub-faces, f', demonstrating the half space of f and f' where f is a $2-$face 15

3.3 case 1: aff(f) is outside of $\mathbb{S}$ 15

3.4 case 2: f is inside of $\mathbb{S}$ 16

3.5 case 3: f intersects $\mathbb{S}$ 16

3.6 case 4: $tag[f'] = \oslash$ and aff(f) $\cap\, \mathbb{S} \subseteq H(f, f')$ 16

3.7 case 5: f does not intersect $\mathbb{S}$ but aff(f) does 17

4.1 Four sites which are tangent to the same Voronoi circle, inserted in a different order resulting in two different. 19

4.2 This figure demonstrates the projection of $\partial V(s_0)$ (black) formed by sites, $s_i, i = 0, 1, 2$ (also black) onto $\mathbb{S}$ (cyan) and its correspondence with the power diagram (red) of the inverted weighted points $\sigma_i, i = 0, 1, 2$ (also red) which are transformed according to transformation 4.1, where s_0 is the inversion pole. Note, to clearly illustrate the concept, this picture is not drawn to scale. 21

4.3 The six cases for insertion of q when all points are in general position. The unit sphere $\mathbb{S}$ is shown in cyan, the power diagram of $\sigma_i, i = 1, 2, q$ in red, where power cells, $P(\sigma_i), i = 1, 2, q$ are labeled accordingly. 22

4.4 The three cases of the Orientation Predicate. The unit sphere $\mathbb{S}$ shown in cyan, the power diagram of $\sigma_i, i = 1, 2, 3$ in red, where power cells, $P(\sigma_i), i = 1, 2, 3$ are labeled accordingly. 23

4.5 The three cases of the RadicalIntersection Predicate. The unit sphere $\mathbb{S}$ shown in cyan, the power diagram of $\sigma_i, i = 1, 2, 3$ in red where power cells, $P(\sigma_i), i = 1, 2, 3$ are labeled accordingly. 24

4.6 The three cases of the RadicalSide Predicate. The unit sphere $\mathbb{S}$ shown in cyan, the power diagram of $\sigma_i, i = 1, 2, 3$ in red where power cells, $P(\sigma_i), i = 1, 2, 3$ are labeled accordingly. 25

4.7 An example where RadicalIntersection = 0, but VertexConflict could be determined without perturbation. The unit sphere $\mathbb{S}$ shown in cyan, the power diagram of $\sigma_i, i = 1, 2, q$ in red where power cells, $P(\sigma_i), i = 1, 2, q$ are labeled accordingly. 27

4.8 OrderOnLine Predicate, with inverted weighted points $\sigma_i, i = 1, 2, 3$ in red. . . . 28

4.9 A graphical representation of the PowerTest Predicate in $\mathbb{R}^2$, inverted weighted points $\sigma_i, i = 1, 2, 3$ in black, and their corresponding power circle in blue, inverted weighted point, the query site σ_4 in yellow, and the unit circle, $\mathbb{S}$ in cyan 29

4.10 A graphical representation of the perturbation of σ_3 in the case of RadicalIntersection = 0, in $\mathbb{R}^2$. The first frame shows inverted weighted points $\sigma_i, i = 1, 2, 3$ in black, and their corresponding power circle in blue, the unit circle $\mathbb{S}$ in cyan, and the power diagram in red. The second frame is an overlay of the first, which shows the result of perturbing the weight of σ_3. The perturbed σ_3 with weight $w_3 + \epsilon$ is shown in magenta, the resultant perturbed power circle is shown in green, and perturbed power diagram dark red. 31

4.11 Graphical representation of α_{01} the AW-Voronoi edge between s_0 and s_1, and its correspondence to a_{01}, a section of the unit circle shown in green. The sites $s_i, i = 0, 1, 2, 3$ as well as the corresponding AW-Voronoi diagram are shown in black. The sites transformed by Equation 4.1 into the inverted weighted points $\sigma_i, i = 0, 1, 2, 3$ and the corresponding Power diagram are shown in red. The unit circle $\mathbb{S}$, as well as the correspondence between the intersection of the power diagram with $\mathbb{S}$ and the projection of the $\partial V(s_0)$ onto $\mathbb{S}$ are shown in cyan. Note, this picture is not drawn to scale. 35

4.12 perturb constant vs. number of filter failures for nearly degenerate case of the original predicates (red) and NCVertexConflict (green). 40

A.1 A picture of two sites with a line segment connecting their centers 42

A.2 Intersecting parabolas at points p, Sweep line l in green, parabola P_0 and associated focus s_0 in blue, parabola P_1 and associated focus s_1 in red 45

Introduction

One of the fundamental structures of Computational Geometry is the Voronoi diagram of a set of points. Intuitively, a Voronoi diagram is created by taking a given set of points in space, the representative points, and assigning all locations to the nearest representative point. This assignment partitions the space into a set of cells, where locations which are equidistant to multiple representative points form the cell boundaries.

Structures resembling the Voronoi digram can be found as early as 1644. Descartes used a Voronoi like diagram to show the physical inclination of matter in the solar system [Des28]. It was not until the mid 19th century that the first comprehensive presentation of the diagram appeared. Peter Gustave Lejeune Dirichlet [Dir50] and Georgy Fedoseevich Voronoy (Georges Voronoï) [Vor07], first studied a special form of the Voronoi diagram of regularly placed points while investigating positive definite quadratic forms. Dirichlet considered the Voronoi diagram in 2D and 3D, and Voronoï considered the diagram in d dimension.

The diagram also seems to have been discovered independently many times and in many fields. One of the more interesting examples is in from 1911 in which Thiessen [Thi11] created the diagram for meteorology to aid in estimating average regional rain fall. Shannon used a variation of the diagram for coding theory, where distance is based on edit distance [Sha48a, Sha48b], and even as recently as 1985, Hoofd et al. [HTK$^+$85] independently discovered the Voronoi diagram when studying anatomy, where Voronoi regions represent the centers of capillaries in tissue sections.

One of the early and most famous examples of the use of a Voronoi like diagram is John Snow's, Report on the Cholera Outbreak in the parish of St. James [Sno54]. Prior to this paper, the Anesthesiologist, published a report hypothesizing that Cholera was spread through contaminated water. In the Fall of 1854 there was a massive Cholera outbreak in which over 500 people died. In his report Snow showed a map with the locations of the deaths caused from the disease as well as a line labeling a boundary equidistant from the Broad Street pump and other pumps (see Figure 1). This generated enough support for his hypothesis that he convinced the authorities to remove the handle from the pump on Broad Street. Once the handle was removed

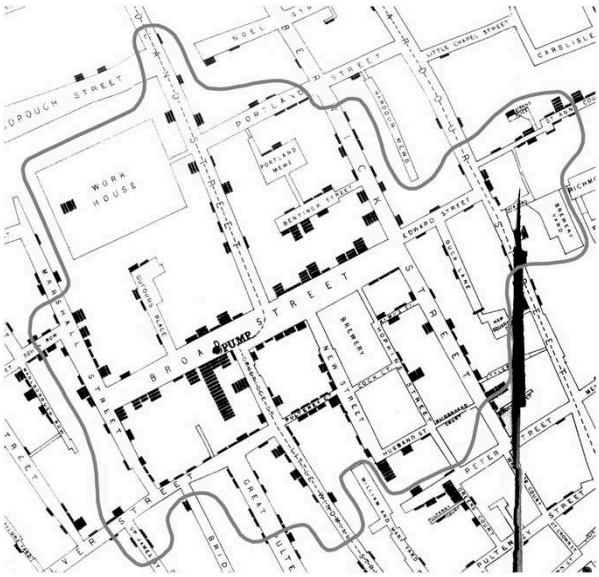

Figure 1: A portion of John Snow's map showing the area surrounding the broad street water pump. Each bar represents a death at that address, and the solid gray line (which replaced Snow's original dotted line) is equidistant from the Broad Street pump and the nearest alternative pump.

the outbreak ended quickly. What is of interest about the map is it represents the Voronoi cell of the Broad street pump, where distance is measured by the actual walking distance.

Since the 1970's there have been many extensions and generalizations of the Voronoi diagram in many fields. Some of the more well known generalizations are the additively/multiplicatively weighted Voronoi diagram, power diagram, Voronoi diagrams of lines, sets of points, polygons, as well as visibility-shortest-path diagrams with barriers. This paper will focus on the additively weighted Voronoi diagram, as well as touch on the power diagram.

The additively weighted Voronoi diagram, alternatively the Apollonius diagram, has many interesting uses. Some of these are marketing-area analysis, modeling crystal growth and cell structure. One of the most interesting uses of the additively weighted Voronoi diagram is its application in animation. For instance, Tao and Huang [TH96] applied the additively weighted Voronoi diagram for diagram warping, image correction of distorted images, animation effects and human-expression synthesis.

Algorithms for calculating the additively weighted Voronoi diagram can be found as early

as the 1970s. Drysdale and Lee [DL78] presented the first known algorithm for computing the additively weighted Voronoi diagram in two dimensional space. This algorithm has a running time of $O(nc^{\sqrt{\log n}})$ where c is a constant, but only works when the sites are disjoint. Later Lee and Drysdale [LD81] presented a divide and conquer algorithm which still only works on non-intersecting sites, but has a running time of $O(n \log^2 n)$. Sharir [Sha85] presented a divide-and-conqueror algorithm which also runs in $O(n \log^2 n)$; the novelty of this algorithm is it could handle intersecting sites. Initially Kirkpatrick [Kir79] was thought to have presented the first $O(n \log n)$ algorithm for calculating the diagram, but the algorithms correctness was an issue, and Yap [Yap85, Yap87] was the first to discover such an algorithm in fall of 1984. A final early algorithm to note is Fortune's sweepline algorithm [For86] which run in $O(n \log n)$. In this algorithm Fortune applies a transformation to change the sites into points and then uses the standard Fortune sweepline algorithm on the transformed sites. More recently, incremental approaches to calculating the additively weighted voronoi diagram have started to emerge. Section 2 will look at one such approach which is used by the 2D Apollonius Graphs (Delaunay Graphs of Disks) package of CGAL [KY06].

1
Preliminary

We begin by defining some basic terms used throughout, as well as the Additively Weighted Voronoi diagram, and the Power Diagram. Also, one should note, most of the diagrams of this paper are in color to aid visualization.

1.1 Basic definitions

We will begin by stating Frey and George's [FG00] definition of a *predicate* as the sign, $\{-, 0, +\}$, of a homogeneous polynomial over some input values and the *algebraic degree*, or degree for short, of the predicate as the largest degree of the irreducible factors. Moreover, the degree of an algorithm is said to be the maximum degree of its predicates. We will return to this idea repeatedly though out the paper, so an example will be helpful. The standard SideOfLine test determines given three points, $p_i = (x_i, y_i), i = 1, 2, 3$, of $\mathbb{R}^2$ which side of the line formed by p_1, and p_2 does p_3 lay. Mathematically SideOfLine can be can be evaluated by calculating,

$$\text{SideOfLine}(\text{p}_1, \text{p}_2, \text{p}_3) = \text{sign}\left(\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}\right)$$

which means SideOfLine has an algebraic degree of 2. The algorithm that randomly generates three points, a, b, c of $\mathbb{R}^2$ and calculates SideOfLine, would also have a degree of 2, provided the point generation has a degree less then or equal to 2. Also note in the literature the term predicate is sometimes used when referring to an algorithm which returns $\{-, 0, +\}$ or $\{true, false\}$, but the definition of degree remains the same with respect to an algorithm.

1.2 Additively-Weighted Voronoi Diagram

We define a *weighted point*, or *site* for short, of $\mathbb{R}^d$ is the pair (p, w) where p is a point in $\mathbb{R}^d$ and w, the weight, is a real number. When w is positive a site can be interpreted as a hypersphere,

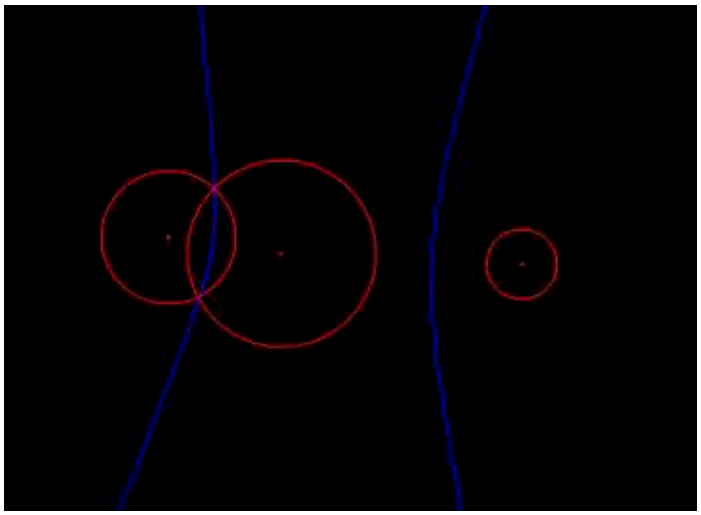

Figure 1.1: An example of the AW-Voronoi diagram in $\mathbb{R}^2$

for example, a circle in $\mathbb{R}^2$ and a sphere in $\mathbb{R}^3$.

Given a set of n sites $\mathcal{S} = \{s_1, \ldots, s_n\}$ of $\mathbb{R}^d$, where $s_i = (p_i, w_i)$, we define the *additively weighted distance* from a point $x \in \mathbb{R}^d$ to s_i as,

$$d_+(s_i, x) = \|p_i - x\| - w_i$$

If w_i is positive then all points outside of s_i have a positive distance; likewise, all points inside of s_i have a negative distance.

We can now define the *additively weighted Voronoi diagram*, or AW-Voronoi for short, as the subdivision of $\mathbb{R}^d$ into n cells $\mathcal{V}(\mathcal{S}) = \{V(s_1), \ldots, V(s_n)\}$ such that,

$$V(s_i) = \{x \in \mathbb{R}^d : d_+(s_i, x) \leq d_+(s_j, x), \forall_j : j = 1, \ldots, n\}.$$

We say the *bisector* of two sites, s_i, s_j, is the locus of points which are equidistant from s_i and s_j (see Figure 1.1), and note that in the AW-Voronoi diagram bisectors in $\mathbb{R}^2$ are hyperbolic curves with focii p_i and p_j (see A.2 for proof in $\mathbb{R}^2$).

Unlike the Voronoi diagram, the AW-Voronoi diagram can contain empty cells, i.e. $V(s_i) = \varnothing$. An empty cell is caused by a site, s' being completely contained inside another site, s. In this case s' is a *trivial* site, and say we that s' is *hidden* by s (conversely s *hides* s').

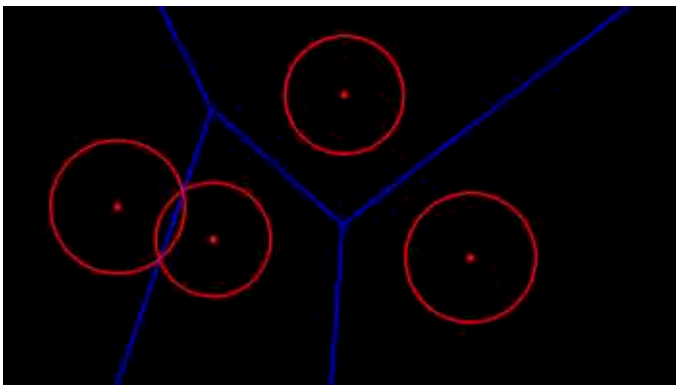

Figure 1.2: An example of the Power diagram in $\mathbb{R}^2$

One final preliminary point about the AW-Voronoi diagram concerns negatively weighted sites. Since adding a constant to all the weights of the sites does not change the AW-Voronoi diagram (see A.1 for proof in $\mathbb{R}^d$) we can assume, when dealing with the AW-Voronoi diagram all sites have positive weights. We next move to the topic of the power diagram.

1.3 Power Diagram

Once again we are given a set of n sites $\mathcal{S} = \{s_1, \ldots, s_n\}$ of $\mathbb{R}^d$ and we define the *power* of a point $x \in \mathbb{R}^d$ to s_i as,

$$d_p(s_i, x) = \|p_i - x\|^2 - w_i^2.$$

We will also define the power cell of s_i as

$$P(s_i) = \{x \in \mathbb{R}^d : d_p(s_i, x) \leq d_p(s_j, x), i \neq j, j = 1, \ldots, n\}.$$

and the *power diagram*, as the subdivision of $\mathbb{R}^d$ into non-empty cells $\mathcal{P}(\mathcal{S}) = \{P(s_i) : i = 1, \ldots, n, P(s_i) \neq \varnothing\}$. Similar to the AW-Voronoi diagram a *bisector* of two sites, s_i, s_j, is the locus of points whose power to s_i and s_j are equal (see Figure 1.2). However, unlike the AW-Voronoi diagram, the bisectors of the power diagram, are hyperplanes (see A.3 for proof in $\mathbb{R}^2$). Also note, we write $P(s_1, \ldots, s_k) = P(s_1) \cap \ldots \cap P(s_k)$ when $P(s_1, \ldots, s_k)$ is a $d-k+1$ dimensional face of $\mathcal{P}(\mathcal{S})$, if the sites are in general position.

We finish the preliminary on the power diagram by considering negatively weighted sites. Since the weight of the site is squared when calculating the power of a point, we can replace any site $s = (p, w)$ where $w < 0$ with an equivalent site $s' = (p, w')$ where $w' = |w|$, leaving the power diagram unchanged.

2
Additively Weighted Voronoi Diagram in 2D

In this section we review the work of Karavelas et al. [KE02, KY02a, KY02b], which presents a 2D incremental algorithm for the insertion and deletion of sites in the Apollonius diagram given a set of sites on a 2D plane.

Karavelas et al.'s papers cover the following material: [KE02] contains the details of the predicates used for evaluating the algorithm, [KY02a] explains the general algorithm and restates the important lemmas and theorems of [KE02, KY02b], and [KY02b] covers the details and proofs of the algorithm presented in [KY02a]. Their presented algorithm has a running time of $\mathcal{O}(nT(h)+h\log h)$ where $T(k)$ is the time to locate the nearest neighbor of a query site to a given set of k sites, and h is the number of sites with non-empty cells. The novelty of their approach is that the dual of the Apollonius diagram, the Apollonius graph, also know as the AW-Delaunay graph, is used to represent the diagram, making insertion and deletion easier. In the Apollonius graph, vertices are the generator sites, and edges correspond to any sites whose voronoi cell share a face (see Figure 2.1).

2.1 Insertion

This section presents Karavelas et al.'s incremental algorithm for constructing the AW-Voronoi diagram. Let $\mathcal{S}$ be a set of n sites, and assume that the diagram for the subset $\mathcal{S}_m$ of $\mathcal{S}$ has already been constructed, such that m is the number of sites in $\mathcal{S}_m$. Our goal is to insert the site $s \notin \mathcal{S}_m$ into $\mathcal{S}_m$. This is done in three steps: (i) locate $NN(s)$, the nearest neighbor of s in $\mathcal{S}_m$; (ii) test if s is trivial; and if it is not (iii) find and repair the conflict region of s in $\mathcal{A}(\mathcal{S}_m)$, the AW-Delaunay graph of the partially formed AW-Voronoi diagram.

The first step of site insertion is (i) to locate the nearest neighbor. Finding $NN(s)$ reduces to simply finding c, the center of s, in $\mathcal{V}(\mathcal{S}_m)$, which can be done by randomly selecting a site $s' \in \mathcal{S}_m$, and looking at all the neighbors of s' in the diagram. If there exists a site s'' such

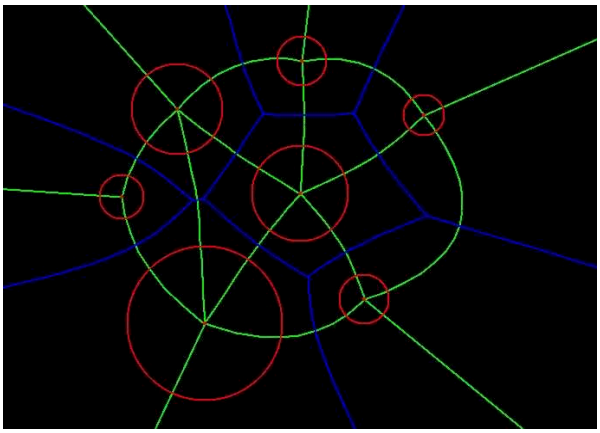

Figure 2.1: A set of generator sites (red) and their induced AW-Voronoi diagram (blue), as well as the dual of the AW-Voronoi diagram, the Apollonius graph(green)

that $d_+(s, s'') < d_+(s, s')$, then we repeat the process with s''. If there does not, $NN(s) = s'$. Location is done in $O(h)$ where h is the number of non-trivial sites in S. Their general paper, [KY02a] mentions a method for speeding up the location phase, which uses a Delaunay hierarchy. The analysis of the method does not generalize to the AW-Delaunay hierarchy, but in practice the method results in a nearest neighbor location time of $O(\log h)$.

Once $NN(s)$ is known we (ii) test if s is trivial. Doing this is simply a case of checking to see if $s \subset NN(s)$ [KY02b, Lemma 1].

Finally (iii) we discover the conflict region caused from the insertion of s, and repair the region to make the diagram valid. Let $R_m(s)$ be the conflict region of $\mathcal{S}_m$ with respect to s, and let $\partial R_m(s)$ be the boundary of $R_m(s)$. The previous statement means that $R_m(s)$ is a subset of $\mathcal{V}_1(\mathcal{S}_m)$, the Voronoi skeleton of $\mathcal{S}_m$, and ∂R_m is a set of points on the edges of $\mathcal{V}_1(\mathcal{S}_m)$. Also note, the points in $\partial R_m(s)$ are the vertices of the Voronoi cell, V_s in $\mathcal{V}_1(\mathcal{S}_{m+1})$ where $\mathcal{S}_{m+1} = \mathcal{S}_m \bigcup \{s\}$. It can be shown that $R_m(s)$ is connected [KMM93, Lemma 1], so all that needs to be done is to find the boundary $\partial R_m(s)$ of $R_m(s)$ and then repair the AW-Voronoi diagram as described by Klein et al. [KMM93].

Finding $\partial R_m(s)$ of $R_m(s)$ can be found by performing a depth first search on $\mathcal{V}_1(\mathcal{S}_m)$, starting from a point on the skeleton which is known to be in conflict. In the comprehensive explanation of the algorithm [KY02b], Karavelas and Yvinec show that if s is non-trivial, then s must be in

conflict with at least one edge of $V_{NN(s)}$, the Voronoi cell of $NN(s)$, in $\mathcal{V}(\mathcal{S}_m)$[KY02b, Lemma 2]. So, to find this edge we can simply walk along the boundary of $V_{NN(s)}$ until we find an edge in conflict.

Next, we move to the topic of managing trivial sites. Recall that a site, s, is trivial if and only if it is contained entirely within another site, s'. This means that when inserting, there are only two ways in which a site can become trivial: (i) either a new site is inserted within another site, or (ii) the new site contains existing sites, these existing sites will become trivial once the new site is inserted. To maintain trivial sites we let each non-trivial site, s' maintain a list of trivial sites which it contains, $L_{tr}(s')$. Also, we must consider the issue of a newly inserted site s'' such that, $s' \subset s''$. In this case we simply add s' to $L_{tr}(s'')$, and move all sites from $L_{tr}(s')$ to $L_{tr}(s'')$.

Karavelas handles the issue of degeneracies with a lazy evaluation[KY02a], such that, any new site found to be tangent to a tritangent Voronoi circle is considered to be not in conflict with the corresponding Voronoi vertex. This scheme causes the diagram to be non-canonical, and dependent on the insertion order of the sites.

The insertion section ends with an analysis of the incremental insertion algorithm [KY02a, Theorem 1].

Theorem 1. Let $\mathcal{S}$ be a set of n sites among which h are non-trivial. We can construct the AW-Voronoi diagram incrementally in $O(nT(h) + h \log h)$ expected time, where $T(k)$ it the time to locate the nearest neighbor of a query site within a set of k sites.

2.2 Deletion

This section describes the process of deleting a site from a preexisting AW-Voronoi diagram. Suppose we have already constructed $\mathcal{V}(\mathcal{S})$, the AW-Voronoi diagram for the set of sites $\mathcal{S}$, and let $s \in \mathcal{S}$ be a site which we want to delete from $\mathcal{V}(\mathcal{S})$. The deletion can split into two cases: (i) s is trivial, (ii) s is non-trivial.

First, we will look at (i)s is non-trivial. Let $\mathcal{S}_\gamma$ be the set of neighbors of V_s in $\mathcal{A}(\mathcal{S})$, the Apollonius graph of $\mathcal{S}$. The AW-Voronoi diagram after the deletion of s can be found by constructing the AW-Voronoi diagram of $\mathcal{S}_\gamma \bigcup L_{tr}(s)$.

Next, we will look at (ii) s is trivial. To delete s we must find the non-trivial site s' such that $s \in L_{tr}(s')$ and then delete s from $L_{tr}(s')$. Since $s \subset NN(s)$, s must be in the list of some s' which is in the same connected component of the union of sites as $NN(s)$. It has been shown that the subgraph $\mathcal{K}(\mathcal{S})$ of $\mathcal{A}(\mathcal{S})$ that consists of all edges of $\mathcal{A}(\mathcal{S})$ connecting intersecting sites, is a spanning subgraph of the connectivity graph of the set of sites [Kar01, Chapter 5]. So, the deletion can be done in three stages: (i) find the nearest neighbor $NN(s)$ of s; (ii) walk on the connected component $\mathcal{C}$ of $NN(s)$ in the graph $\mathcal{K}(\mathcal{S})$ and for every site $s' \in \mathcal{C}$ that contains s, test if $s \in L_{tr}(s')$; (iii) once the site s' such that $s \in L_{tr}(s')$ is found, delete s from $L_{tr}(s')$.

The deletion section ends with an analysis of the incremental deletion algorithm,

Theorem 2. Let $\mathcal{S}$ be a set of n sites, among which h are non-trivial. Let $s \in \mathcal{S}$, and let $L_{tr}(s)$ be the list of trivial sites whose parent is s. If s is non-trivial, it can be deleted from $\mathcal{V}(\mathcal{S})$ in expected time $O((d+t)T(d+t') + (d+t')\log(d+t'))$, where d is the degree of s in $\mathcal{A}(\mathcal{S})$, t is the cardinality of $L_{tr}(s)$ and t' is the number of sites in $L_{tr}(s)$ that become non-trivial after the deletion of s. If s is trivial, it can be deleted from $\mathcal{L}_{tr}(\mathcal{S})$ in worst case time of $O(n)$.

2.3 Predicates

This section discusses the predicates used to calculate the AW-Voronoi diagram.

1. SideOfBisector: Given two sites s_1 and s_2 as well as a query site q, determine if q is closer to s_1 or s_2. The algebraic degree of this predicate is 4.

2. IsTrivial: Given a site s_1 and a query site q determine if $q \subset s_1$. This is used to determine whether the query site is trivial. The algebraic degree of this predicate is 2.

3. Orientation: Given two sites s_1 and s_2 as well as the tritangent Voronoi circle C_{345}, with centers c_1, c_2, c_{345}, respectively determine the result of the orientation test $CCW(c_1, c_2, c_{345})$. This is used to find the first conflict of a new site s given its nearest neighbor, $NN(s)$ [KE02]. The algebraic degree of this predicate is 14.

4. EdgeConflictType: Given a Voronoi edge α and a query site q, determine the type of the conflict region of q with α. This predicate is used to discover the conflict region of q

with respect to the pre-existing AW-Voronoi diagram [KE02]. The algebraic degree of this predicate is 16.

Before moving on, it is important to note that the Orientation and EdgeConflictType are evaluated using a method introduced by Karavales [KE02, Section 5] know as the *inversion approach*. In general this method is used to compute the Voronoi circle C_{123}, corresponding to sites s_1, s_2, s_3 ordered counter-clockwise around the boundary of C_{123}, via the inversion,

$$\sigma_i = (c_i, r_i), \qquad c_i = \tfrac{q_i}{\alpha}, \qquad r_i = \frac{\omega_i}{\alpha}$$
$$q_i = p_i - p_1, \quad \omega_i = w_i - w_1, \quad \alpha = q_i^2 - w_i^2 \text{ for i} = 2, 3.$$

The reason for this inversion is it maps C_{123}, the circle tangent to s_1, s_2, s_3, to l, a line co-tangent to σ_2, σ_3. We will return to these predicates in Section 4.

3
Convex Hull and Voronoi Diagram of Additively Weighted Points

In this section we review the work of Boissonnat and Delage [BD05], who present a method for constructing the convex hull of additively weighted points in $\mathbb{R}^d$, and then apply this method to constructing AW-Voronoi diagrams via a single cell construction method [BK03].

They begin by first showing a correspondence between the convex hull and the power diagram in $\mathbb{R}^d$. The correspondence is used to show a new approach to constructing the convex hull in $\mathbb{R}^d$. Finally, they apply the result to the AW-Voronoi diagram in $\mathbb{R}^d$.

3.1 Correspondence between the Convex Hull and the Power Diagram

This section presents the method Boissonnat and Delage [BD05] use to show a correspondence between the power diagram in $\mathbb{R}^d$ and the convex hull of additively weighted points in $\mathbb{R}^d$. Let $\mathcal{S}$ be a set of n weighted points in $\mathbb{R}^d$ such that all n sites are hyperspheres, $\mathcal{S} = \{s_1, \ldots, s_n\}$. The *convex hull* of $\mathcal{S}$, $\mathrm{CH}(\mathcal{S})$, is the smallest closed convex subset of $\mathbb{R}^d$ containing all hyperspheres of $\mathcal{S}$. We define a *supporting hyperplane* H of $\mathcal{S}$ to be a hyperplane tangent to at least one of the hyperspheres in $\mathcal{S}$ such that all other hyperspheres in $\mathcal{S}$ are in the same halfspace partitioned by H. Also, we define a *facet of* $\mathrm{CH}(\mathcal{S})$ *of circularity* k, $0 \leq k < d$ to be the portion of $\partial\mathrm{CH}(\mathcal{S})$ that consist of the points whose supporting hyperplanes are tangent to the same subset of $d - k$ hyperspheres. For example if $d = 3$, faces of circularity 0 are planar faces tangent to 3 hypersphere, faces of circularity 1 are conical patches (faces which are the same shape as a section of a cone) which are tangent to 2 hypersphere and faces of circularity 2 are spherical patches (faces which are the same shape as a section of a sphere) contained in some site s_i. Another example of is when $d = 2$, the faces of circularity 0 are lines tangent to 2 circles, and faces of circularity 1 are circular arcs (faces which are part of a circle) which are contained in some site, s_i (see Figure 3.1).

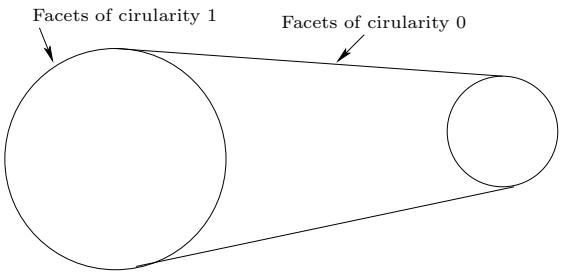

Figure 3.1: A convex hull in $\mathbb{R}^2$ of two sites demonstrating facets of circularity $k, k = 0, 1$

Next, Boissonnat and Delage show that if we let $\mathbb{S}$ be the unit hypersphere and convert the set of n hypersphere $\mathcal{S} = \{s_1, \ldots, s_n\}$ to $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$ where, $s_i = (p_i, w_i)$, $\sigma_i = (p_i, r_i)$ where r_i is defined by the transformation $r_i^2 = p_i^2 + 2w_i$ then, for each $k = 0, \ldots, d-1$, the k-faces of the power digram of Σ intersected with $\mathbb{S}$, $\mathcal{P}(\Sigma) \cap \mathbb{S}$ are in 1-1 correspondence with the facets of circularity k of $\partial\text{CH}(\mathcal{S})$ [BD05, lemma 1].

This lemma tells us that computing the convex hull of $\mathcal{S}$ reduces to transforming $\mathcal{S}$ to Σ and computing the intersection of the power diagram of Σ, $\mathcal{P}(\Sigma)$, with the unit hypersphere, $\mathbb{S}$. Before presenting a static algorithm we need to first define a few terms. A k-face, f', is a *sub-face* of a $(k+1)$-face, f, when $f' \subseteq f$. Conversely, a $(k+1)$-face, f, is *super-face* of a k-face f' when $f' \subseteq f$. Also, we notate the affine hull of a face f as aff(f). Finally, we define the *halfspace* of a face f and its sub-face f', $H(f, f')$, to be the halfspace of aff(f) bounded by aff(f') which contains f. For example if f is a 1-face (a line segment), f' is one of its two endpoints, and $H(f, f')$ is the ray issued from f' which contains f. A second example is when f is a 2-face (a polygon); then f' is a line segment and $H(f, f')$ is the half plane of the plane though f bounded by the line though f' which contains f (see Figure 3.2)

In the static algorithm we first construct the power diagram $\mathcal{P}(\Sigma)$ and determine for each face, f of $\mathcal{P}(\Sigma)$, if f intersects $\mathbb{S}$. The result is stored in $tag[f]$:

- $tag[f] = \oslash$ if and only if aff(f) is outside $\mathbb{S}$
- $tag[f] = \ominus$ if and only if f is outside of $\mathbb{S}$ but aff(f) intersects $\mathbb{S}$
- $tag[f] = \oplus$ if and only if f intersects $\mathbb{S}$

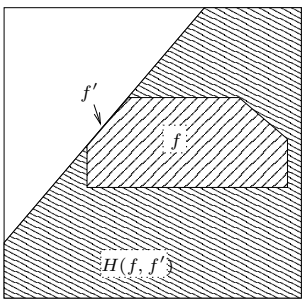

Figure 3.2: A face, f and one of its sub-faces, f', demonstrating the half space of f and f' where f is a $2-$face

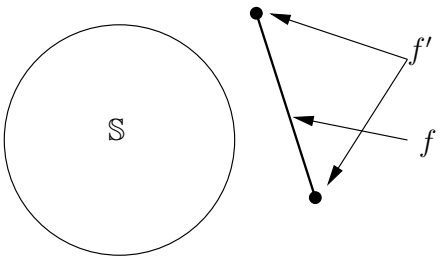

Figure 3.3: case 1: aff(f) is outside of $\mathbb{S}$

- $tag[f] = \odot$ if and only if f is inside of $\mathbb{S}$

Assuming we know $tag[f']$ for each subface f' of f, we can computer $tag[f]$ by:

a: if aff(f) does not intersect $\mathbb{S}$, $tag[f] = \oslash$ (see Figure 3.3)

b: else, if for each sub-face f' of f, $tag[f'] = \odot$ then $tag[f] = \odot$ (see Figure 3.4)

c: else, if there is a sub-face f' of f, such that $tag[f'] = \odot$ then $tag[f] = \oplus$ (see Figure 3.5)

d: else, if for each sub-face f' of f, $tag[f'] = \oslash$ and aff(f) $\cap\, \mathbb{S} \subseteq H(f, f')$, then $tag[f] = \oplus$ (see Figure 3.6)

e: else, f does not intersect $\mathbb{S}$ but aff(f) does, so $tag[f] = \ominus$ (see Figure 3.7)

Without loss of generality, let k-face, $f = P(\sigma_1, \ldots, \sigma_k)$ and $(k-1)$-face $f' = P(\sigma_1, \ldots, \sigma_{k+1})$. To differentiate between the two cases in the algorithm, we need the two predicates:

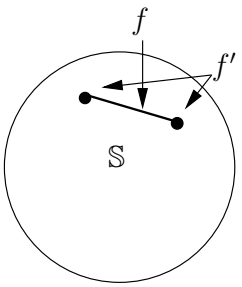

Figure 3.4: case 2: f is inside of $\mathbb{S}$

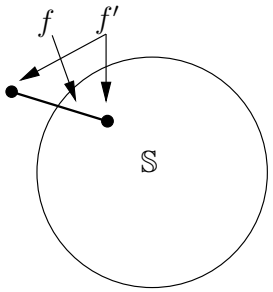

Figure 3.5: case 3: f intersects $\mathbb{S}$

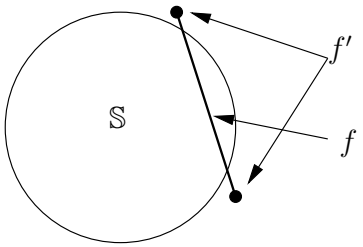

Figure 3.6: case 4: $tag[f'] = \oslash$ and $\text{aff}(f) \cap \mathbb{S} \subseteq H(f, f')$

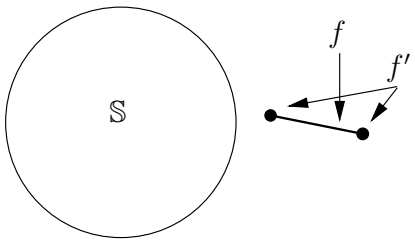

Figure 3.7: case 5: f does not intersect $\mathbb{S}$ but aff(f) does

- k-RadicalIntersection(f), which determines if aff(f) is inside $\mathbb{S}$
- k-RadicalSide(f, f'), which determines if aff(f) $\cap\, \mathbb{S} \subseteq H(f, f')$, assuming aff($f$) intersects $\mathbb{S}$ and f is outside of $\mathbb{S}$.

We will return to the actual calculation of these predicates after covering the rest of the algorithm. To compute the tag for the $(d-1)$-face f we begin by first calculating the tag for each 0-face of f, we then use this information to calculate the tag for each 1-face of f, etc. until we have calculated the the tag for f. It then follows that the time complexity of this algorithm is bounded by the time complexity of the power diagram algorithm used, which in this case is $O(n \log n + n^{\lceil \frac{d}{2} \rceil})$.

Now we will briefly return to the predicates necessary for calculating this algorithm. The k-RadicalIntersection predicate is used to determine if a face of the power diagram f is inside of $\mathbb{S}$. This can be accomplished by seeing where the origin projects onto aff(f), and comparing the norm of the projection, π, to 1. The details of computing π can be found in Boissonnat et al. [BD05, index A.3]. The k-RadicalSide predicate determines if for a face of the power diagram f and one of its sub-faces f', if aff(f) $\cap\, \mathbb{S} \subseteq H(f, f')$, assuming aff($f$) intersects $\mathbb{S}$ and f is outside of $\mathbb{S}$. Assume without loss of generality that f is defined by sites $\sigma_0, \ldots, \sigma_{k-1}$ and f' is defined by sites $\sigma_0, \ldots, \sigma_k$, and recall $P(\sigma_i)$ is the power cell of σ_i, and $P(\sigma_0, \ldots, \sigma_k)$ is the $d-k+1$ dimensional face, which is the intersection of the power cells of $\sigma_0, \ldots, \sigma_1$, ie, $P(\sigma_0, \ldots, \sigma_k) = P(\sigma_0) \cap \ldots \cap P(\sigma_k)$ (see section 1.3). The k-RadicalSide predicate can be determined by calculating which side of $P(\sigma_0, \ldots, \sigma_k)$ does the orthogonal projection onto aff(f) of the origin reside.

Boissonnat and Delage [BD05] also present a dynamic algorithm for calculating the convex hull of a set of n points using the same result, along with an algorithm which uses the single cell construction technique similar to the algorithm presented in Karavelas [KY02a]. We have now covered the major concepts necessary for the rest of this paper.

4
Degeneracy Proof Predicates for the Additively Weighted Voronoi Diagram in 2D

The algorithm presented by Karavelas et al. [KY02a] leads to non-canonical AW-Voronoi diagrams in the case of degenerate inputs. Their algorithm uses the VertexConflict and EdgeConflict predicates to determine the conflict region of a newly inserted site. When a degeneracy is encountered VertexConflict uses a lazy evaluation where any site found to be tangent to a tritangent voronoi circle is considered to be not in conflict. This in turn causes EdgeConflict to have a different result, as EdgeConflict is only necessary when both vertices in conflict or not in conflict. Hence, the lazy evaluation of VertexConflict causes the resultant diagram to be non-canonical and dependent on the insertion order (see Figure 4.1). To deal with this problem we create degeneracy-proof predicates for the AW-Voronoi diagram in $\mathbb{R}^2$ by expressing the VertexConflict predicate in terms of the k-RadicalIntersection and k-RadicalSide predicates of [BD05], and then explicitly handle degeneracies. Not only does this yield a consistent result but also predicates of lower degree along with a $39-66$ percent increase in speed. This same method was then applied to EdgeConflict.

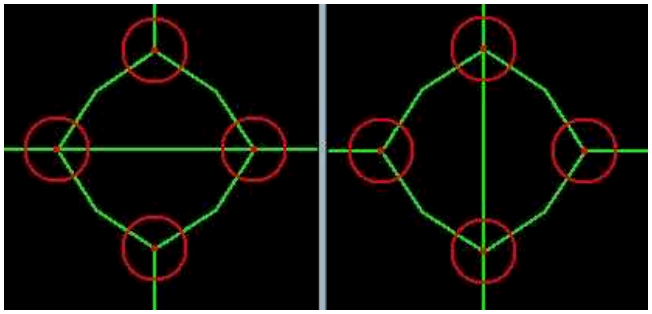

Figure 4.1: Four sites which are tangent to the same Voronoi circle, inserted in a different order resulting in two different.

4.1 VertexConflict Introduction

The VertexConflict predicate takes four sites, s_0, s_1, s_2, q where sites $s_i, i = 0, 1, 2$ ordered counter clockwise, form the tritangent voronoi circle C_{012}, and returns the sign of the additively weighted distance of q from the center of C_{012}. VertexConflict is evaluated by calling the InCircle predicate which determines the sign of the distance $\delta(C_{012}, q)$ of q from C_{012}. The sign of the $\delta(C_{012}, q)$ is calculated by using the *inversion approach* (see Section 2.3), to transform C_{012} to l, then obtaining q' by applying the same transformation to q. Now the problem reduces to finding the sign of the distance from q' to l.

We will now present another method of determining the VertexConflict predicate. Boissonnat and Karavelas [BK03] have show that the projection of a cell of the AW-Voronoi onto a sphere coincides with the intersection of the Power Diagram and the sphere (see Figure 4.2). This means, given a set of n sites $\mathcal{S} = \{s_1, \ldots, s_n\}$ of $\mathbb{R}^d$, the projection of the partial AW-Voronoi cell of s_i, $\partial V(s_i)$ onto a unit sphere centered at p_i corresponds to the intersection between the power diagram diagram of $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$ and the unit hyper sphere centered at the origin, where:

$$\begin{aligned} \sigma_j = (c_j, r_j, \alpha_j), \qquad c_j = \tfrac{q_j}{\alpha_j}, \qquad r_j = \frac{\omega_j^*}{\alpha_j} \\ q_j = p_j - p_i, \quad \omega_j^* = w_j - w_i, \quad \alpha_j = \begin{cases} q_i^2 - \omega_i^{*2}, & i \neq j \\ 1, & i = j \end{cases} \end{aligned} \tag{4.1}$$

Note that we refer to sites which have undergone this transformation as inverted weighted points, and when referring to them throughout the predicates section we will refer to the components of the inverted weighted point σ_j as x_j, the x coordinate of c_j, y_j, the y coordinate of c_j, r_j, the weight of σ_j and α_j, the α value of σ_j. Also, we introduce the function $invert(s_j, s_i)$ which performs the inversion presented above, where s_i is the inversion pole.

Since we are now interested in calculating the attributes of the intersection of the Power diagram and the unit sphere, we can use the predicates presented in [BD05], as they are less costly then Karavelas' original predicates. To achieve this we must differentiate when the insertion site

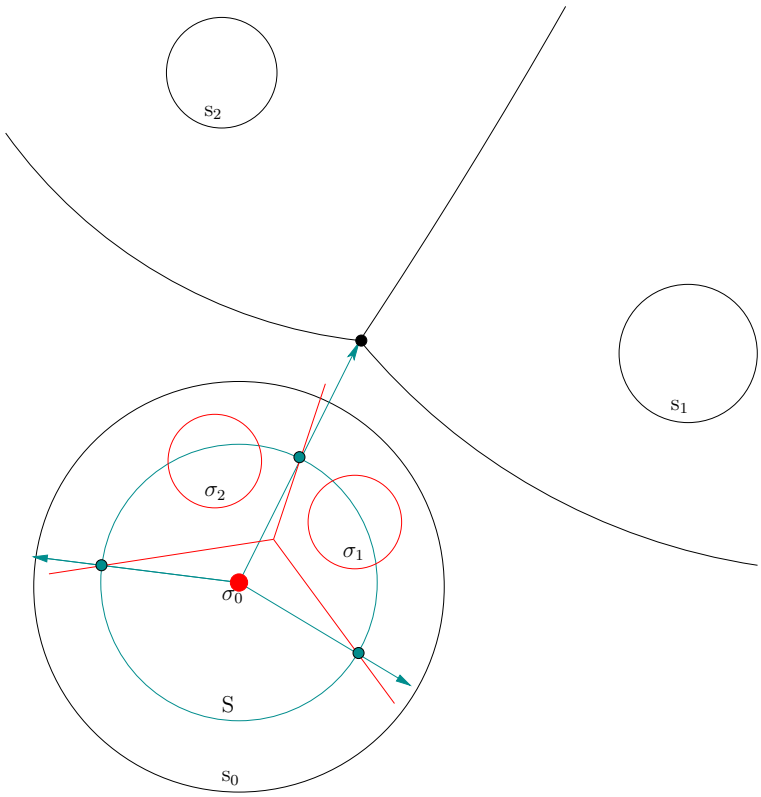

Figure 4.2: This figure demonstrates the projection of $\partial V(s_0)$ (black) formed by sites, $s_i, i = 0, 1, 2$ (also black) onto $\mathbb{S}$ (cyan) and its correspondence with the power diagram (red) of the inverted weighted points $\sigma_i, i = 0, 1, 2$ (also red) which are transformed according to transformation 4.1, where s_0 is the inversion pole. Note, to clearly illustrate the concept, this picture is not drawn to scale.

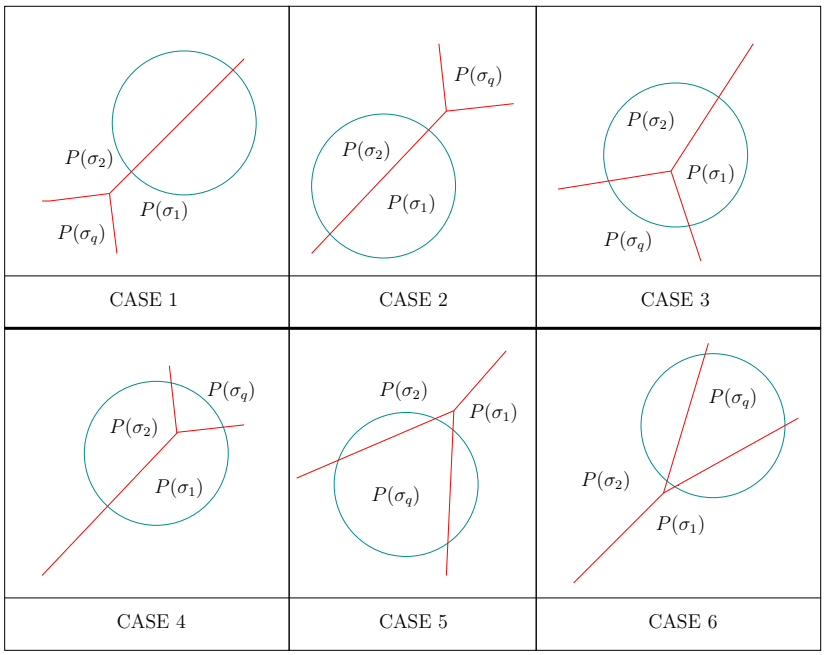

Figure 4.3: The six cases for insertion of q when all points are in general position. The unit sphere $\mathbb{S}$ is shown in cyan, the power diagram of $\sigma_i, i = 1, 2, q$ in red, where power cells, $P(\sigma_i), i = 1, 2, q$ are labeled accordingly.

q causes the Voronio circle, C_{012} to be invalid. Figure 4.3 shows the six non-degenerate cases which can occur (note, we will define a non-degenerate case after presenting our predicates). Cases $1, 2, 3$ correspond to no vertex-conflict, cases $4, 5, 6$ correspond to vertex conflict.

Before describing how the six non-degenerate cases can be differentiated, it would be helpful to present the necessary predicates for calculating our version of the vertex conflict predicate. Graphical examples in $\mathbb{R}^2$, as well as the formula which must be evaluated for these predicates will also be shown. We will then present a method for determining when q causes a vertex conflict.

4.2 Orientation predicate

The first predicate which we will cover is the orientation predicate. This predicate evaluates if three inverted weighted points, $\sigma_i, i = 1, 2, 3$, are a right or left turn, when traveling from σ_1 to

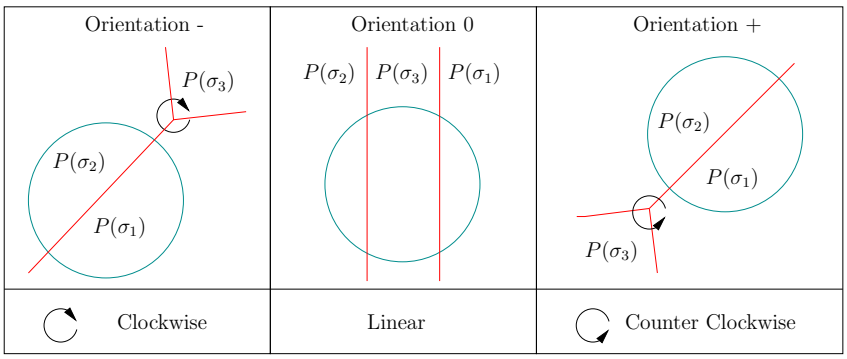

Figure 4.4: The three cases of the Orientation Predicate. The unit sphere $\mathbb{S}$ shown in cyan, the power diagram of $\sigma_i, i = 1, 2, 3$ in red, where power cells, $P(\sigma_i), i = 1, 2, 3$ are labeled accordingly.

σ_2 to σ_3. In other words, in $\mathbb{R}^2$ we determine if the orientation of $\sigma_1, \sigma_2, \sigma_3$ is clockwise, linear or counter clockwise (see Figure 4.4). Mathematically this can be expressed as,

$$\text{Orientation}(\sigma_1, \sigma_2, \sigma_3) = \text{sign}\left(\begin{vmatrix} \alpha_1 & \alpha_2 & \alpha_3 \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{vmatrix}\right)$$

where Orientation is negative, zero or positive when the sites $\sigma_1, \sigma_2, \sigma_3$ are oriented clockwise, linearly or counter clockwise respectively.

4.3 RadicalIntersection predicate

As stated earlier, the k-RadicalIntersection predicate, referred to as the RadicalIntersection predicate throughout, determines if a face, f of the power diagram is inside $\mathbb{S}$. This can be accomplished by seeing where the origin projects onto aff(f), and comparing the norm of the projection, π, to 1. Since we are only interested in the location of f_1, the 1-face of the power diagram of the inverted weighted points $\sigma_i, i = 1, 2, 3$ with respect to $\mathbb{S}$, our predicate will determine, given three inverted weighted points, $\sigma_i, i = 1, 2, 3$, is f_1 inside, tangent or outside of $\mathbb{S}$ (see Figure

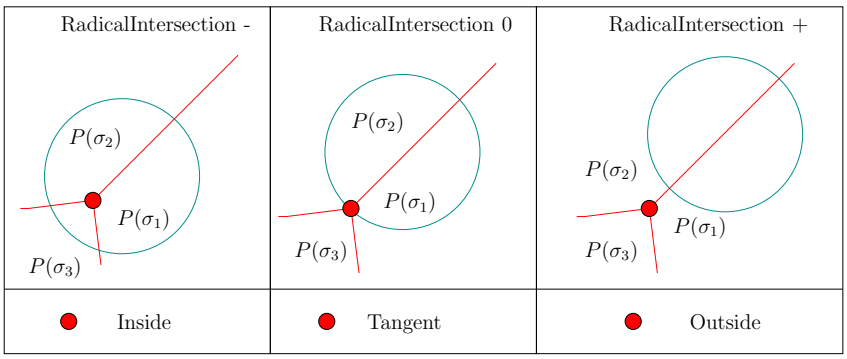

Figure 4.5: The three cases of the RadicalIntersection Predicate. The unit sphere $\mathbb{S}$ shown in cyan, the power diagram of $\sigma_i, i = 1, 2, 3$ in red where power cells, $P(\sigma_i), i = 1, 2, 3$ are labeled accordingly.

4.5). Mathematically this can be expressed as,

$$\text{RadicalIntersection}(\sigma_1, \sigma_2, \sigma_3) = \text{sign}\left(\begin{vmatrix} \alpha_1 & r_1 & y_1 \\ \alpha_2 & r_2 & y_2 \\ \alpha_3 & r_3 & y_3 \end{vmatrix}^2 + \begin{vmatrix} \alpha_1 & x_1 & r_1 \\ \alpha_2 & x_2 & r_2 \\ \alpha_3 & x_3 & r_3 \end{vmatrix}^2 - \begin{vmatrix} \alpha_1 & x_1 & y_1 \\ \alpha_2 & x_2 & y_2 \\ \alpha_3 & x_3 & y_3 \end{vmatrix}^2\right)$$

where RadicalIntersection is negative, zero or positive when f_1 is inside, tangent or outside of $\mathbb{S}$ respectively.

4.4 RadicalSide predicate

The next predicate which we will cover is the k-RadicalSide predicate, in our case $k = 1$, and we will refer to the k-RadicalSide predicate where $k = 1$ as the RadicalSide predicate throughout. Recall, the RadicalSide predicate determines for a face of the power diagram f and one of its sub-faces f', if $\text{aff}(f) \cap \mathbb{S} \subseteq H(f, f')$, assuming $\text{aff}(f)$ intersects $\mathbb{S}$ and f is outside of $\mathbb{S}$. Without loss of generality, let f, defined by sites $\sigma_i, i = 1, 2$ and f' defined by sites $\sigma_i, i = 1, 2, 3$. The RadicalSide predicate can be determined by calculating which side of f' the orthogonal projection

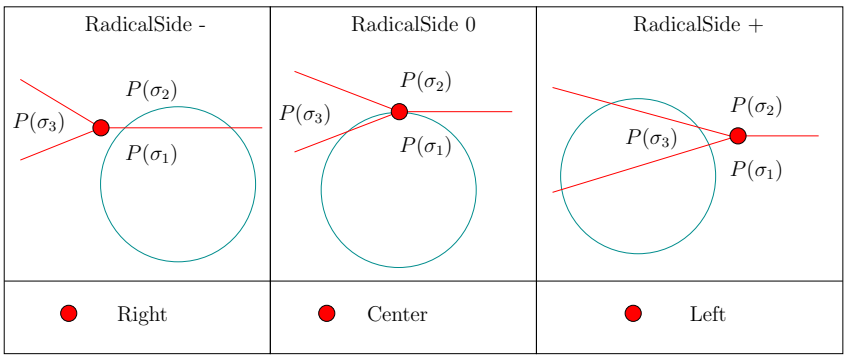

Figure 4.6: The three cases of the RadicalSide Predicate. The unit sphere $\mathbb{S}$ shown in cyan, the power diagram of $\sigma_i, i = 1, 2, 3$ in red where power cells, $P(\sigma_i), i = 1, 2, 3$ are labeled accordingly.

onto aff(f) of the origin resides (see Figure 4.6). Mathematically this can be expressed as,

$$\text{RadicalSide}(\sigma_1, \sigma_2, \sigma_3) =$$
$$\text{sign}\left(-\begin{vmatrix}\alpha_1 & x_1\\ \alpha_2 & x_2\end{vmatrix} * \begin{vmatrix}\alpha_1 & r_1 & x_1\\ \alpha_2 & r_2 & x_2\\ \alpha_3 & r_3 & x_3\end{vmatrix} - \begin{vmatrix}\alpha_1 & y_1\\ \alpha_2 & y_2\end{vmatrix} * \begin{vmatrix}\alpha_1 & r_1 & y_1\\ \alpha_2 & r_2 & y_2\\ \alpha_3 & r_3 & y_3\end{vmatrix}\right)$$

where RadicalSide is negative, zero or positive when the orthogonal projection of the origin on to aff(f) right, center or left of f', respectively. Also note, RadicalSide $= 0$ can only happen when RadicalIntersection $= 0$, else the predicates assumptions are invalid. We will return to this issue when we investigate the issue of degeneracies.

4.5 VertexConflict Non-Degenerate

We begin by defining a non-degenerate case as one in which Orientation $\neq 0$, RadicalSide $\neq 0$ and RadicalIntersection $\neq 0$. We can now describe a new algorithm for calculating VertexConflict in the non-degenerate case which can differentiate the cases $1, 2, 3$ from $4, 5, 6$. We first take our three sites $s_i, i = 0, 1, 2$ and our query sites q. We invert s_1, s_2, q with respect to s_0 to get the inverted weighted points $\sigma_1, \sigma_2, \sigma_q$. We can now determine that there is VertexConflict when RadicalSide$(\sigma_1, \sigma_2, \sigma_{\text{q}})$ and RadicalIntersection$(\sigma_1, \sigma_2, \sigma_{\text{q}})$ are *positive* or when

RadicalIntersection($\sigma_1, \sigma_2, \sigma_q$) is *negative* and Orientation($\sigma_1, \sigma_2, \sigma_3$) is *positive*, provided none of this predicates are *zero*. If any of them are *zero*, we simply return Degenerate (See Algorithm (1)).

Algorithm 1 VertexConflict Non-Degenerate

```
//Let s_i, i = 0, 1, 2 be three site ordered counter clockwise which form
//   the tritangent voronoi circle C_012, and q be the query site.
//Return Conflict if the sign of the additively weighted distance
//   of q from C_0,1,2 is negative, NoConflict if it is positive
//   and Degenerate if the input is degenerate
σ_1 ← invert(s1, s_0)
σ_2 ← invert(s2, s_0)
σ_q ← invert(q, s_0)
orient ← Orientation(σ_1, σ_2, σ_q)
radInt ← RadicalIntersection(σ_1, σ_2, σ_q)
radSide ← RadicalSide(σ_1, σ_2, σ_q)
if orient = 0 || radInt = 0 || radSide = 0 then
  return Degenerate
end if
if radInt > 0 then
  if radSide > 0 then
    return Conflict
  else
    return NoConflict
  end if
else
  if orient < 0 then
    return NoConflict
  else
    return Conflict
  end if
end if
```

4.6 Degeneracies

Now that we have described a method which can determine VertexConflict, we will next present a method to handle degeneracies. Degeneracies are handled via the symbolic perturbation method presented in by Boissonnat and Delage[BD05, section 5.2]. Generally, in the case of a predicate evaluating to zero, we grow the site with the largest radius by $\epsilon > 0$. In the case of the largest radii being equal we use the lexicographical ordering of the centers to determine the perturbation site.

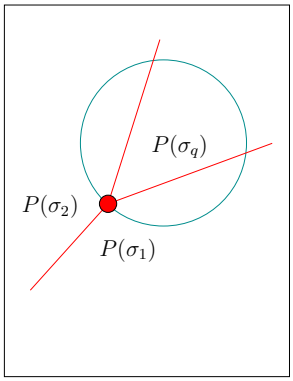

Figure 4.7: An example where RadicalIntersection = 0, but VertexConflict could be determined without perturbation. The unit sphere $\mathbb{S}$ shown in cyan, the power diagram of $\sigma_i, i = 1, 2, q$ in red where power cells, $P(\sigma_i), i = 1, 2, q$ are labeled accordingly.

This method is consistent with the original implementation, and preserves the combinatorial structure of the diagram, provided ϵ is sufficiently small. This method does not increase the algebraic degree of the predicates, but does cause unnecessary perturbations as some predicates return *zero* in non-degenerate cases, for example, when Orientation < 0, RadicalSide > 0 and RadicalIntersection $= 0$ (see Figure 4.7). Introducing this symbolic perturbation scheme requires a modification of the the previous predicates along with the introduction of two more predicates.

The degeneracies section will proceed as follows. First, we introduction two new predicates, OrderOnLine predicate followed by NDPowerTest, a non-degenerate version of PowerTest predicate presented in [BD05]. Second, we use these predicates to create the NDRadicalSide, non-degenerate RadicalSide and NDRadicalIntersection, non-degenerate RadicalIntersection. Note, we do not present a non-degenerate Orientation predicate, as Orientation $= 0$ will be used for determining EdgeConflict. Third, once we have introduced our new non-degenerate predicates we present the full algorithm for NDVertexConflict (non-degenerate VertexConflict).

4.7 OrderOnLine

OrderOnLine is a predicate which given three co-linear inverted weighted points $\sigma_i, i = 1, 2, 3$, determines if they are ordered on a line (see Figure 4.8). OrderOnLine will depend on a general-

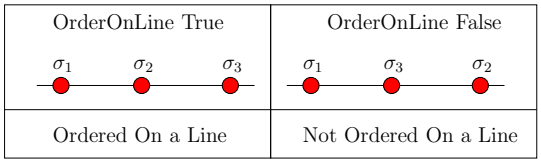

Figure 4.8: OrderOnLine Predicate, with inverted weighted points $\sigma_i, i = 1, 2, 3$ in red.

ization of the standard two site order on a line test (oolTest), which determines the orientation of two inverted weighted points. Mathematically, oolTest can be expressed as,

$$\text{oolTest}(\sigma_1, \sigma_2, \sigma_3) \quad = \quad \text{sign} \left(\begin{cases} \begin{vmatrix} \alpha_1 & x_1 \\ \alpha_2 & x_2 \end{vmatrix}, \begin{vmatrix} \alpha_1 & x_1 \\ \alpha_2 & x_2 \end{vmatrix} \neq 0 \\ \\ \begin{vmatrix} \alpha_1 & y_1 \\ \alpha_2 & y_2 \end{vmatrix}, \begin{vmatrix} \alpha_1 & x_1 \\ \alpha_2 & x_2 \end{vmatrix} = 0 \end{cases} \right)$$

Now using our previously defined oolTest, we can define the OrderOnLine predicate by first testing if oolTest(σ_1, σ_2) is positive, if it is, return true if oolTest(σ_2, σ_3) is positive false otherwise. If oolTest(σ_1, σ_2) is non-positive return true if oolTest(σ_3, σ_2) is positive and false otherwise (see Algorithm 2).

Algorithm 2 OrderOnLine Predicate

```
//Let σ_i, i = 1,2,3 be three co-linear sites
//Return true if are σ_i, 1,2,3 are ordered on the line false otherwise
if oolTest(σ_1, σ_2) > 0 then
  return oolTest(σ_2, σ_3) > 0
else
  return oolTest(σ_3, σ_2) > 0
end if
```

4.8 NDPowerTest

Now we will turn to the NDPowerTest predicate. Before discussing this predicate, we must first introduce the notion of the power product of two weighted points, s_i, s_j, as $d_p(s_i, s_j) =$

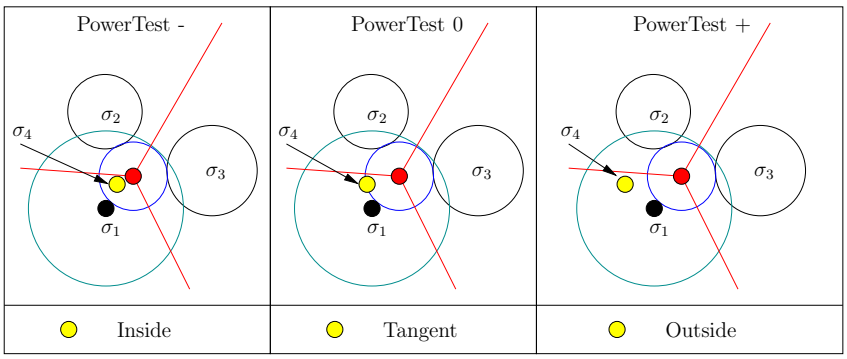

Figure 4.9: A graphical representation of the PowerTest Predicate in $\mathbb{R}^2$, inverted weighted points $\sigma_i, i = 1, 2, 3$ in black, and their corresponding power circle in blue, inverted weighted point, the query site σ_4 in yellow, and the unit circle, $\mathbb{S}$ in cyan

$(p_i p_j)^2 - w_i - w_j$. We say that the two weighted points, s_i, s_j are orthogonal if, $d_p(s_i, s_j) = 0$, and we call the power circle of three weighted points, s_i, s_j, s_k the unique circle orthogonal to s_i, s_j, s_j. Essentially in the power diagram, a power circle is equivalent to the voronoi circle of the voronoi diagram. In general, the power test computes the sign of the power product of a query site from a power circle of a power diagram. Once again in terms of the standard Voronoi diagram, the PowerTest is conceptually equivalent to the InCircle predicate (see Figure 4.9). Mathematically, the PowerTest can be expressed as,

$$\text{PowerTest}(\sigma_1, \sigma_2, \sigma_3, \sigma_4) \quad = \quad \text{sign} \left(- \begin{vmatrix} \alpha_1 & \alpha_2 & \alpha_3 & \alpha_4 \\ x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ r_1 & r_2 & r_3 & r_4 \end{vmatrix} \right)$$

We will create NDPowerTest by explicitly handling the degeneracies which can occur in the case of the PowerTest, by making the observation that we are only using the PowerTest when the orientation of our points is *zero*. Our power test predicate will take three inverted weighted points $\sigma_i, i = 1, 2, 3$ and a label which specifies the perturbed site in the case of degeneracies. The sites σ_1 and σ_2, and the inverted weighted point $\sigma_0, p_0 = (0, 0), r_0 = 0, \alpha_0 = 1$ define a

power circle, C_{012}, and σ_3 is the query site. The NDPowerTest predicate will first calculate PowerTest to determine if s_3 is a positive or negative distance from C_{012}. If PowerTest $= 0$ and turns out to be degenerate, we only need to decide if the perturbation will force s_3 inside or outside of the power circle, which is the same as deciding which way the perturbation scheme will cause the power circle to shift, as we already know of σ_1, σ_2, q is collinear we can simply use the OrderOnLine predicate (see Algorithm 3).

Algorithm 3 NDPowerTest Predicate

```
//Let σ_i, i = 1,2,3 be three co-linear sites and perturb
//   be the label of the site to perturb if necessary
//   and let σ_0 be a inverted weighted site at the origin
//   such that r_0 = 0 and α_0 = 1.
//Return the sign of the distance of σ_3 from the
//   power circle formed by σ_1 and σ_2 and the origin
powTest ← PowerTest(σ_0, σ_1, σ_2, σ_3)
if powTest ≠ 0 then
  return powTest
end if
if perturb = 1 then
  if OrderOnLine(σ_1, σ_2, σ_3) then
    return NEGATIVE
  else
    return POSITIVE
  end if
else if perturb = 2 then
  if OrderOnLine(σ_2, σ_1, σ_3) then
    return NEGATIVE
  else
    return POSITIVE
  end if
else
  return NEGATIVE
end if
```

4.9 NDRadicalSide and NDRadicalIntersection

Next we will modify RadicalSide and RadicalIntersection to handle degeneracies. Recall that RadicalSide $= 0$ can only occur when RadicalIntersection $= 0$. Because of this property, we can use a lazy evaluation in NDRadicalSide, where it returns POSITIVE in the case of degenera-

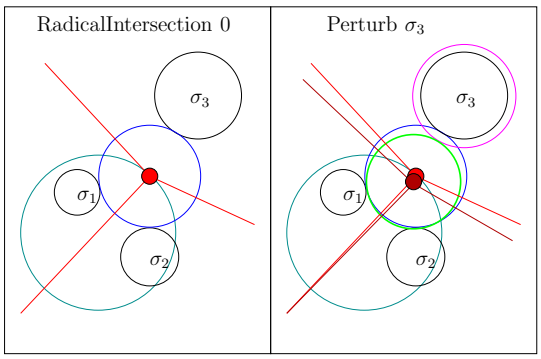

Figure 4.10: A graphical representation of the perturbation of σ_3 in the case of RadicalIntersection $= 0$, in $\mathbb{R}^2$. The first frame shows inverted weighted points $\sigma_i, i = 1, 2, 3$ in black, and their corresponding power circle in blue, the unit circle $\mathbb{S}$ in cyan, and the power diagram in red. The second frame is an overlay of the first, which shows the result of perturbing the weight of σ_3. The perturbed σ_3 with weight $w_3 + \epsilon$ is shown in magenta, the resultant perturbed power circle is shown in green, and perturbed power diagram dark red.

cies, and simply handle the issue in NDRadicalIntersection, thus, avoiding extra computation. Now, recall that RadicalIntersection $= 0$ means the face of the power diagram of which we are attempting to calculate the position with respect to $\mathbb{S}$ is on $\mathbb{S}$. Explicitly this means the power circle of $\sigma_i, i = 1, 2, 3$, C_{123} lies on $\mathbb{S}$, so, we need to determine how our perturbation scheme will affect the perturbed center of C_{123}.

Without loss of generality assume σ_3 is the point which is going to be perturbed. This perturbation scheme will cause the center of C_{123} to move away from the perturbed point in the direction of the bisector between σ_1 and σ_2 (see Figure 4.10), hence we can determine how C_{123} will be perturbed using NDRadicalSide predicate.

Now, NDRadicalIntersection can be evaluated by calculating RadicalIntersection on $\sigma_i, i = 1, 2, 3$, if the result is 0, evaluate NDRadicalSide$(\sigma_3, \sigma_2, \sigma_1)$ if σ_1 is the perturbed site, NDRadicalSide$(\sigma_1, \sigma_3, \sigma_2)$ if σ_2 is the perturbed site or NDRadicalSide$(\sigma_1, \sigma_2, \sigma_3)$ if σ_3 is the perturbed site (see Algorithm 4).

Algorithm 4 NDRadicalIntersection Predicate

```
//Let σ_i, i = 1,2,3 be three sites and perturb
//   be the label of the site to perturb if necessary
//Return NEGATIVE if center of the power circle C_123
//   is inside 𝕊.
radInt ← RadicalIntersection(σ_1, σ_2, σ_3)
if radInt ≠ 0 then
   return radInt
end if
if perturb = σ_1 then
   return NDRadicalSide(σ_3, σ_2, σ_1)
else if perturb = σ_2 then
   return NDRadicalSide(σ_1, σ_3, σ_2)
else
   return NDRadicalSide(σ_1, σ_2, σ_3)
end if
```

4.10 NDVertexConflict

Now that we have modified RadicalIntersection and RadicalSide to handle degeneracies as well as presented NDPowerTest, we can create a NDVertexConflict, VertexConflict which handles degeneracies. We will begin by first simplify to predicate to simply return *true* if the vertex q is in conflict with C_{012}, and $false$ otherwise. Creating NDVertexConflict is simply an issue of substituting NDRadicalSide for RadicalSide, NDRadicalIntersection for RadicalIntersection, and handling the unmodified Orientation predicate. The first two are trivial, and the final can easily be achieved by evaluating NDPowerTest in the case of Orientation = 0 (see Algorithm 5). Before continuing we should also note that instead of handling the special case where the inversion site is also the the perturbed site, we simply return the predicate NDVertexConflict$(\sigma_1, \sigma_2, \sigma_0, \mathrm{q})$, because it has the same result since the perturbation scheme is consistent, and we maintain the counter-clockwise ordering of $\sigma_i, i = 0, 1, 2$.

4.11 NDEdgeConflict

Next we discuss the EdgeConflict Predicate. This predicate takes four sites $s_i, i = 0, 1, 2, 3$ which defines α_{01}, the bisector between s_0 and s_1 with endpoints defined by s_0, s_1, s_2 and s_0, s_3, s_2

Algorithm 5 NDVertexConflict Predicate

```
//Let s_i, i = 0, 1, 2 be three sites ordered counter clockwise which form
//   the tritangent voronoi circle C_012, and q be our query site.
//Return true if the sign of the additively weighted distance
//   of q from C_012 is negative and false otherwise,
//   using the perturbation method presented in 4.6
perturbSite ← GetSiteToPerturb(s_0, s_1, s_2, q)
if perturbSite = s_0 then
   return NDVertexConflict(s_1, s_2, s_0, s_q)
end if
σ_1 ← invert(s1, s_0)
σ_2 ← invert(s2, s_0)
σ_q ← invert(q, s_0)
orient ← Orientation(σ_1, σ_2, σ_q)
radInt ← NDRadicalIntersection(σ_1, σ_2, σ_q, perturbSite)
radSide ← NDRadicalSide(σ_1, σ_2, σ_q, perturbSite)
powTest ← NDPowerTest(σ_1, σ_2, σ_q, perturbSite)
if orient = 0 then
   return (powTest < 0)
else if radInt > 0 then
   return (radSide > 0)
else
   return !(orient < 0)
end if
```

oriented counter clockwise, as well as a query site q and a flag specifying if both vertices are in conflict or not, and determines whether q is in conflict with α_{01}. It is important to note that if one vertex of an edge is in conflict and the other is not, this predicate is unnecessary, as we already know that the edge is in conflict. In the original implementation EdgeConflict was achieved by determining if q is in conflict with the bitangent Voronoi circle C_{01} defined by s_0, s_1. This in turn was calculated in a similar manner to the original implementation of VertexConflict, via the inversion method.

We now present our method for evaluating the NDEdgeConflict. Recall that it has been shown by Boissonnat and Karavelas [BK03], that the projection of a cell of the AW-Voronoi diagram onto a sphere coincides with the intersection of the Power Diagram and the sphere. For the VertexConflict predicate we use this result to help us determine if the query site is in conflict with the vertex in question. Once again we will use the same result to help us with EdgeConflict, as this result also tells us that in $\mathbb{R}^2$ while the vertices of the AW-Voronoi cell coincide with the intersection of the Power Diagram and the unit circle, the faces of the AW-Voronoi cell corresponds to the sections on the face of of the unit circle between the intersections (see Figure 4.11). Let a_{ij} be the section of the unit sphere which corresponds to α_{ij}. To determine when q causes an edge conflict with with edge α_{01}, we simply need to determine if the insertion of q causes a_{01} to become disconnected, or non-existent.

We will first describe the case when both vertices are in conflict, and begin by dealing with the simple case where our query site q causes our inversion pole s_0 to become trivial. This can be determined by simply checking if α_q is non-positive. Next we will handle the degenerate case where, $\sigma_i, i = 1, 2, 3, q$ are collinear. This case is also very simple, since the four inverted weighted points are collinear, we only need to determine if the query site is in between σ_1 and σ_2, or if it is in between σ_3 and σ_2. If this occurs, then we know q will cause a_{01} to break into two parts meaning edge conflict. Next we will look at the non trivial and non degenerate cases. We can simplify this problem by simply realizing that we already know both vertices are in conflict greatly limiting the number of configurations we must consider resulting in the $isConflict$ case of Algorithm 6.

Next we will turn to the case when neither of the vertices are in conflict. We begin by realizing that, we do not need to check for α_q being non-positive as we already know that the vertices

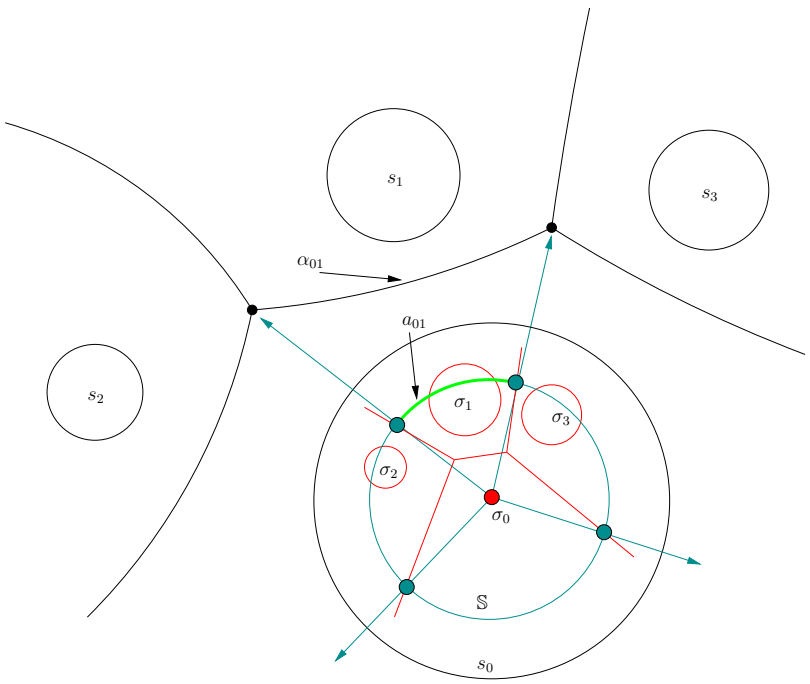

Figure 4.11: Graphical representation of α_{01} the AW-Voronoi edge between s_0 and s_1, and its correspondence to a_{01}, a section of the unit circle shown in green. The sites $s_i, i = 0, 1, 2, 3$ as well as the corresponding AW-Voronoi diagram are shown in black. The sites transformed by Equation 4.1 into the inverted weighted points $\sigma_i, i = 0, 1, 2, 3$ and the corresponding Power diagram are shown in red. The unit circle $\mathbb{S}$, as well as the correspondence between the intersection of the power diagram with $\mathbb{S}$ and the projection of the $\partial V(s_0)$ onto $\mathbb{S}$ are shown in cyan. Note, this picture is not drawn to scale.

are both not in conflict, hence this could not happen. So, the first case which we will want to handle is the degenerate case where $\sigma_i, i = 1, 2, 3, q$ are collinear. As the vertices are both not in conflict, the only way that this situation could result in an edge conflict, is for σ_q, σ_1 and σ_2 to be on the line in order and σ_q, σ_1 and σ_3 to be in a line in order. Now we simply need to do the same task as before and consider the possible configuration under the presupposition that both vertices are not in conflict resulting in the *else* section of Algorithm 6.

Finally, it should be noted that RadicalSide and RadicalIntersection are being used instead of, their non-degenerate counterparts. Since we already know the result of the perturbation from the fact that the vertices are in conflict or not, we can use the simpler and faster predicates as we already know the results of the perturbation. Algorithm 6 describes the full NDEdgeConflict in pseudo code.

4.12 Algebraic Degree Analysis

We will now analyze the algebraic complexity of the presented non-degenerate predicates, and compare them to the original predicates. In this section we will continue to refer to the non-degenerate predicates presented in this paper as ND$\langle PredicateName \rangle$ and the original predicates as simply $\langle PredicateName \rangle$. We begin by first recalling that Boissonnat et al. [BD05] showed the degree of the presented versions of Orientation, RadicalSide, RadicalIntersection and the PowerTest, and Karavelas et al. [KE02] showed the degree of VertexConflict and EdgeConflict as,

Predicate	degree
Orientation	4
RadicalSide	3
RadicalIntersection	6
PowerTest	5
VertexConflict	14
EdgeConflict	14

To evaluate the non degenerate version of these predicates we must first determine the degree of the OrderOnLine predicate. As OrderOnLine only calls oolTest, we only need to determine

Algorithm 6 NDEdgeConflict Predicate

//Let $s_i, i = 0, 1, 2, 3$ be four sites which define the bisector of s_0 and s_1,
// which has endpoints s_0, s_1, s_2 and s_0, s_3, s_1 in counter clockwise
// ordering, and let q be the query site, $isConflict$ denotes if the two
// endpoints are, or are not in in VertexConflict with q.
//Return true if q is in conflict with the edge false if it is not.
$\sigma_1 \leftarrow invert(s1, s_0)$
$\sigma_2 \leftarrow invert(s2, s_0)$
$\sigma_3 \leftarrow invert(s3, s_0)$
$\sigma_q \leftarrow invert(q, s_0)$
$orient12Q \leftarrow \text{Orientation}(\sigma_1, \sigma_2, \sigma_\text{q})$
$orient31Q \leftarrow \text{Orientation}(\sigma_3, \sigma_1, \sigma_\text{q})$
$orient123 \leftarrow \text{Orientation}(\sigma_1, \sigma_2, \sigma_3)$
$ri12Q \leftarrow \text{RadicalIntersection}(\sigma_1, \sigma_2, \sigma_\text{q})$
$ri13Q \leftarrow \text{RadicalIntersection}(\sigma_1, \sigma_3, \sigma_\text{q})$
$rs1Q2 \leftarrow \text{RadicalSide}(\sigma_1, \sigma_\text{q}, \sigma_2)$
$rs1Q3 \leftarrow \text{RadicalSide}(\sigma_1, \sigma_1, \sigma_3)$
$oolQ13 \leftarrow \text{OrderOnLine}(\sigma_\text{q}, \sigma_1, \sigma_3)$
$oolQ12 \leftarrow \text{OrderOnLine}(\sigma_\text{q}, \sigma_1, \sigma_2)$
if $isConflict$ **then**
  if $\alpha_q \leq 0$ **then**
    return true
  else if $orient123 = 0$ && $orient12Q = 0$ && $orient31Q = 0$ **then**
    return ($oolQ12$ || $oolQ13$)
  else if !(($ri12Q \geq 0$ && $rs1Q2 < 0$)&&($ri13Q \geq 0$ && $rs1Q3 < 0$)) **then**
    return true
  else if $orient123 \geq 0$ **then**
    return ($orient12Q \leq 0$ && $orient31Q \leq 0$)
  else
    return ($orient12Q \leq 0$ || $orient31Q \leq 0$)
  end if
else
  if $orient123 = 0$ && $orient12Q = 0$ && $orient31Q = 0$ **then**
    return ($oolQ12$ && $oolQ13$)
  else if !(($ri12Q \geq 0$ && $rs1Q2 < 0$)&&($ri13Q \geq 0$ && $rs1Q3 < 0$)) **then**
    return false
  else if $orient123 \geq 0$ **then**
    return ($orient12Q \leq 0$ || $orient31Q \leq 0$)
  else
    return ($orient12Q \leq 0$ && $orient31Q \leq 0$)
  end if
end if

the degree of oolTest. Recall that our transformation computes a value α_i with respect to our pole s_j, for each site s_i with center p_i and then defines their center by $\frac{p_i}{\alpha_i}$. Also note that if s_i and s_j do not hide each other then $\alpha_i > 0$ [BK03], and when $\alpha_i \leq 0$ these predicates are not needed.

Without loss of generality, recall the standard oolTest can be defined on two points $p_i = (x_i^{'}, y_i^{'}), i = 1, 2$ in $\mathbb{R}^2$ as

$$\text{sign}\left(\begin{vmatrix} 1 & x_1^{'} \\ 1 & x_2^{'} \end{vmatrix}\right) = \text{sign}\left(\begin{vmatrix} 1 & \frac{x_1^{'}}{\alpha_1} \\ 1 & \frac{x_2^{'}}{\alpha_2} \end{vmatrix}\right) = \text{sign}\left(\begin{vmatrix} \alpha_1 & x_1 \\ \alpha_2 & x_2 \end{vmatrix}\right)$$

which has degree 3, which follows, OrderOnLine has degree of 3. As OrderOnLine does not raise the algebraic degree of any of the non degenerate predicates, NDVertexConflict and NDEdgeConflict both have an algebraic degree of 6 which is smaller then VertexConflict and EdgeConflict's original degree of 14.

New Predicate	degree	Old Predicate	degree
Orientation	4	-	-
NDRadicalSide	3	RadicalSide	3
NDRadicalIntersection	6	RadicalIntersection	6
NDPowerTest	5	PowerTest	5
NDVertexConflict	6	VertexConflict	14
NDEdgeConflict	6	EdgeConflict	14

4.13 Experimental Results

The predicate NDVertexConflict, has been optimized to reduce redundant computation between predicates, and is implemented in CGAL [CGA]. Also exact predicates are implemented via a filtered traits class which supports dynamic filtering in CGAL through the Filtered_exact mechanism [HB01], and the CORE [gNYU] library. The Filtered_exact mechanism allows for the slower exact predicates to be evaluated only when necessary. Timings were calculated on a Dell 600m with a Pentium M 1.6GHz processor with 512MB RAM, running Linux.

Experimental results for time comparisons were calculated by randomly generating 1024 sites and for each set of four sites $s_i, i = 1, \ldots, 4$ an AW-Voronoi vertex is created from s_1, s_2, s_3. The site s_4 is then used to evaluate the vertex conflict predicate $50,000$ times to minimize any system overhead, as well as the start up cost of creating the AW-Voronoi vertex. This process was then again done for the AW-Voronoi vertex of $s2, s_3, s_4$ with s_1 as the query site, s_3, s_4, s_1 as an AW-Voronoi vertex with s_2 as the query site and s_4, s_1, s_2 as the AW-Voronoi vertex with s_3 as the query site. This process was timed on both predicate, across ten datasets to ensure consistency. The NDVertexConflict predicate performed 39% better then the origional predicate in the the case where the AW-Voronoi vertex was not an infinite vertex, and performed 66% better when the AW-Voronoi vertex was an infinite vertex.

Next, experimental results with respect to filter failures were calculated. Filter failure testing was accomplished by generating a set of highly degenerate sites as follows. Create an AW-Voronoi diagram of 1000 ± 100 voronoi vertices, then let the set of tritangent voronoi circles which correspond to the voronoi vertices of the input set be our new test set. The new test set would now be a highly degenerate set of 1000 ± 100 sites. We then repeatedly calculate the AW-Voronoi diagram of this input set but perturbed the input sites centers and weights by ϵ, such that $0 \leq \epsilon < \mathrm{E}$ where at every iteration E was incremented by $1x10^{-16}$ for $1x10^{-16} \leq \mathrm{E} < 5 * 10^{-14}$. NDVertexConflict experimentally had $10 - 20\%$ fewer filter failures then the origonal vertex conflict predicate. This results in machine precision being sufficent for NDVertexConflict where the origional predicate needed to result to the slower exact computation version of its predicate (Figure 4.12).

4.14 Further Work

While NDEdgeConflict has been implemented in CGAL, this work could be extended by first optimizing the NDEdgeConflict predicate by reducing redundant determinant computations, and running experimental results. This could further be extended by sharing the substantial number of calculations which are first done in NDVertexConflict then again in NDEdgeConflict, but this would require small modifications to the algorithm. Sharing these calculation could further reduce the number of filter failures, and decrease the amount of time required to calculate the

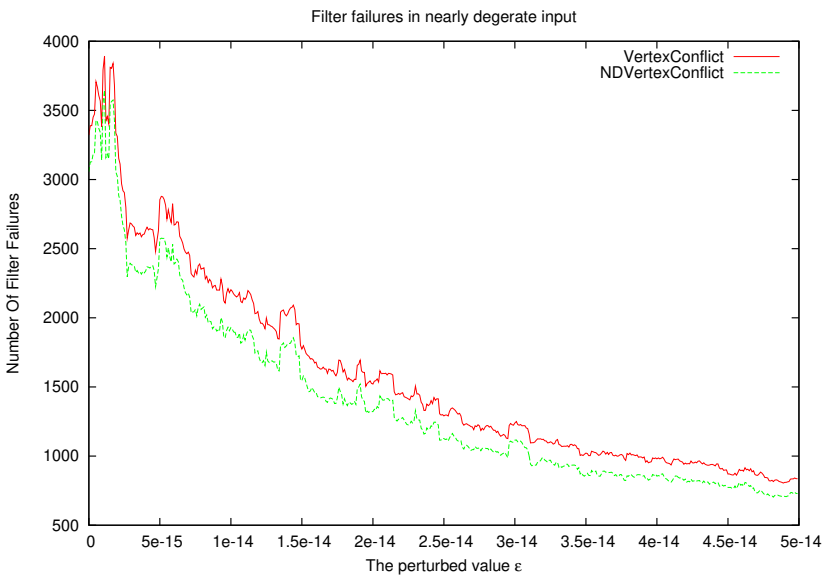

Figure 4.12: perturb constant vs. number of filter failures for nearly degenerate case of the original predicates (red) and NCVertexConflict (green).

conflict region of a query site. Another further extension would be to investigate degeneracies in $\mathbb{R}^3$ and handle them via the method presented above.

Appendix

Proofs

This section contains miscellaneous proofs which are referred to in the paper.

A.1 Increasing the Weights of the Generator Sites in the AW-Voronoi Diagram does not change the Diagram

Our goal is to prove that increasing the weights of the generator sites of the AW-Voronoi diagram of $\mathbb{R}^d$ does not change the diagram. Let s_i, s_j be two sites in $\mathbb{R}^d$, $k \in \mathbb{R}$, and define $s_i^{'} = (p_i, w_i + k), s_j^{'} = (p_j, w_j + k)$ of $\mathbb{R}^d$. Note that if $k < 0$ then we are actually decreasing the weights of s_i, s_j. It is clear that,

$$\begin{aligned} d_+(s_i^{'}, x) &= d_+(s_j^{'}, x) \\ \|p_i - x\| - (w_i + k) &= \|p_j - x\| - (w_j + k) \\ \|p_i - x\| - w_i &= \|p_j - x\| - w_j \\ d_+(s_i, x) &= d_+(s_j, x) \end{aligned}$$

Hence increasing the weights by a constant factor does not modify the diagram.

A.2 The Bisectors of a AW-Voronoi Diagram in 2D are Hyperbolic Arcs

Our goal is to prove that the bisectors of the AW-Voronoi diagram of a set of additively weighted points in $\mathbb{R}^2$ are hyperbolic arcs. It is straightforward to see this as the bisector of the additively

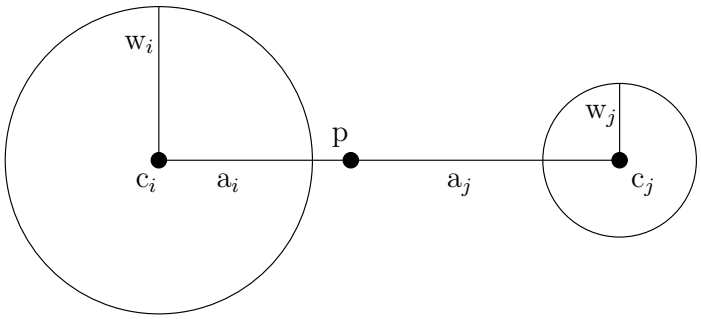

Figure A.1: A picture of two sites with a line segment connecting their centers

weighted sites s_i, s_j is defined as the the locus of points $x \in \mathbb{R}^2$ such that,

$$\begin{aligned} d_+(s_i, x) &= d_+(s_j, x) \\ \|p_i - x\| - w_i &= \|p_j - x\| - w_j \\ \|p_i - x\| - \|p_j - x\| &= w_i - w_j \end{aligned}$$

So the bisector of s_i, s_j consists of the locus of points such that the difference between the distance p_i and p_j is the constant $w_i - w_j$. Which is the same as the hyperbola with focii p_i and p_j. We should also restate that this holds when the weights are negative as well as zero, since increasing the weights by a constant factor will not change the diagram (see proof A.1).

A.3 The Bisectors of a Power Diagram in 2D are Lines

Our goal is to prove that the bisectors of the power diagram of a set of weighted points (positive, negative or zero) in $\mathbb{R}^2$ are lines. We will begin by showing the bisectors are straight for positively weighted points. Let s_i and s_j be two sites in $\mathbb{R}^2$ such that $w_i, w_j > 0$ and let segment $\overline{c_i c_j}$ be the line segment connecting the centers of s_i and s_j. Also, define p, the point on $\overline{c_i c_j}$ such that $d_p(s_i, p) = d_p(s_j, p)$. To show the bisectors of a power diagram are straight, it suffices to show that for any point q, which is on the bisector of s_i and s_j, the projection of q onto $\overline{c_i c_j}$ is p.

It is important to note, since q is on the bisector of s_i and s_j:

$$\begin{aligned} d_p(s_i, q) &= d_p(s_j, q) \\ (q - c_i)^2 - w_i^2 &= (q - c_j)^2 - w_j^2 \\ (q - c_i)^2 - (q - c_j)^2 &= w_i^2 - w_j^2 \end{aligned} \tag{A.1}$$

Finally we will label the point q' the projection of q onto segment $\overline{c_i c_j}$. For simplicity we define $a_i = (q' - c_i)$, $a_j = (q' - c_j)$, $b = (q' - q)$ (see Figure A.1). Using the Pythagorean theorem we get:

$$(q - c_i)^2 = a_i^2 + b^2 \tag{A.2}$$

$$(q - c_j)^2 = a_j^2 + b^2 \tag{A.3}$$

And subtracting (A.2) and (A.3):

$$(q - c_i)^2 - (q - c_j)^2 = a_i^2 - a_j^2 \tag{A.4}$$

Combining (A.1) with (A.4) results in:

$$\begin{aligned} a_i^2 - a_j^2 &= w_i^2 - w_j^2 \\ a_i^2 - w_i^2 &= a_j^2 - w_j^2 \\ (q' - c_i)^2 - w_i^2 &= (q' - c_j)^2 - w_j^2 \\ d_p(s_i, q') &= d_p(s_j, q') \end{aligned}$$

But this means that q' is the point which is equidistant from s_i and s_j which is on $\overline{c_i c_j}$, i.e. $q' = p$. So, the bisectors of a power diagram of positively weighted points are straight lines. Note, this also holds when $w_i = 0$ and/or $w_j = 0$.

Next, we consider negatively weighted sites. Let s and s' be two sites such that $c = c'$, $w' < 0$ and $w = |w'|$. In this case we refer to s as the *positively weighted point equivalent* to s'. Recall that the the power of a point $x \in \mathbb{R}^2$ to a site $s_i \in \mathbb{R}^2$ is defined as:

$$d_p(s_i, x) = (x - c_i)^2 - w_i^2$$

This means that $\{\forall_x | x \in \mathbb{R}^2 : d_p(s, x) = d_p(s', x)\}$, so for any negatively weighted site, s' we could substitute its positively weighted equivalent without changing the cell of s'; also note that the reverse is true. Thus the bisectors in a power diagram containing sites with positive, negative and/or zero weights are straight.

A.4 Parabolas of a beach line intersect at two points

Our goal is to prove that two parabolas along the beach line intersect at two points. We will begin by first deriving an equation for a parabola of the sweep line and then showing our main goal. Let $s_0 = (x_0, y_0)$ be a point, l be the current sweep line, and P be the parabola defined by s_0 and l. Since a parabola geometrically is all the points $p = (x, y)$, equidistant between a point and a line P can be defined algebraically as,

$$\begin{aligned}
(y - l)^2 &= (x - x_0)^2 + (y - y_0)^2 \\
y^2 - 2yl + l^2 &= x^2 - 2xx_0 + x_0^2 + y^2 - 2yy_0 + y_0^2 \\
2yy_0 - 2ly &= x^2 - 2xx_0 + x_0^2 + y_0^2 - l^2 \\
2y(y_0 - l) &= x^2 - 2xx_0 + x_0^2 + y_0^2 - l^2 \\
y &= \frac{x^2 - 2xx_0 + x_0^2 + y_0^2 - l^2}{2(y_0 - l)}
\end{aligned} \tag{A.5}$$

Let, two points $s_i = (x_i, y_i), i = 0, 1$ and sweepline l define parabolas $P_i, i = 0, 1$ respectively. Also, let $p = (x, y)$ be the points equidistant from s_0, s_1 and l, and note that p corresponds to the intersections of P_0 and P_1. Let sweepline l be $y = 0$, assume $x_0 = 0$ and $x_1 \geq x_0$ (see Figure A.2).

Then we have parabolas P_0 and P_1 given by

$$\begin{aligned}
y &= \frac{x^2 + y_0^2}{2y_0} \\
y &= \frac{x^2 - 2xx_1 + x_1^2 + y_1^2}{2y_1}
\end{aligned}$$

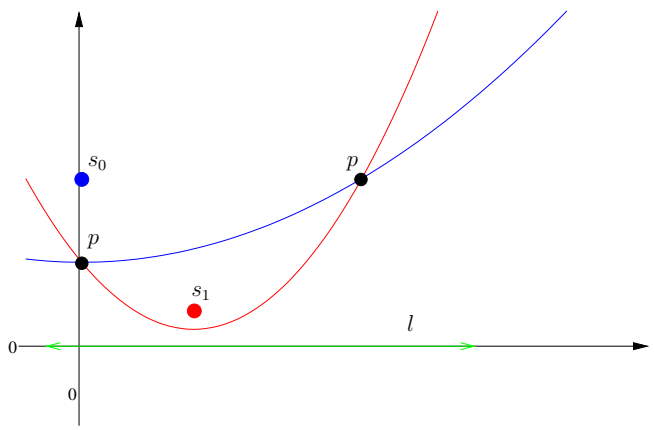

Figure A.2: Intersecting parabolas at points p, Sweep line l in green, parabola P_0 and associated focus s_0 in blue, parabola P_1 and associated focus s_1 in red

respectively. Hence x satisfies the equation,

$$\begin{aligned}
\frac{x^2+y_0^2}{2y_0} &= \frac{x^2-2xx_1+x_1^2+y_1^2}{2y_1} \\
\frac{x^2}{2y_0}+\frac{y_0^2}{2y_0}-\frac{x^2}{2y_1}+\frac{2xx_1}{2y_1}-\frac{x_1^2+y_1^2}{2y_1} &= 0 \\
\left(\frac{1}{2y_0}-\frac{1}{2y_1}\right)x^2+\frac{2x_1}{2y_1}x+\frac{y_0^2}{2y_0}-\frac{x_1^2+y_1^2}{2y_1} &= 0
\end{aligned}$$

and we can solve for the intersection points by solving the quadratic equation. So to show the parabolas of a beach line intersect at 2 points it would suffice to show the discriminant, $\Delta = b^2 - 4ac > 0$ where,

$$\begin{aligned}
a &= \frac{1}{2y_0}-\frac{1}{2y_1}=\frac{y_1-y_0}{2y_0y_1} \\
b &= \frac{2x_1}{2y_1}=\frac{2x_1y_0}{2y_0y_1} \\
c &= \frac{y_0^2}{2y_0}-\frac{x_1^2+y_1^2}{2y_1}=\frac{y_0^2y_1-x_1^2y_0-y_0y_1^2}{2y_0y_1}
\end{aligned}$$

So plugging this into Δ we get,

$$\begin{aligned}
\Delta &= \left(\frac{2x_1y_0}{2y_0y_1}\right)^2 - 4\left(\frac{y_1-y_0}{2y_0y_1}\right)\left(\frac{y_0^2y_1 - x_1^2y_0 - y_0y_1^2}{2y_0y_1}\right) \\
&= \frac{x_1^2y_0^2}{y_0^2y_1^2} - \frac{(y_1-y_0)(y_0^2y_1 - x_1^2y_0 - y_0y_1^2)}{y_0^2y_1^2} \\
&= \frac{x_1^2y_0^2 - y_0^2y_1(y_1-y_0) + x_1^2y_0(y_1-y_0) + y_0y_1^2(y_1-y_0)}{y_0^2y_1^2} \\
&= \frac{x_1^2y_0^2 - y_0^2y_1^2 + y_0^3y_1 + x_1^2y_0y_1 - x_1^2y_0^2 + y_0y_1^3 - y_0^2y_1^2}{y_0^2y_1^2} \\
&= \frac{x_1^2y_0y_1 + y_0^3y_1 - 2y_0^2y_1^2 + y_0y_1^3}{y_0^2y_1^2} \\
&= \frac{y_0y_1(x_1^2 + y_0^2 - 2y_0y_1 + y_1^2)}{y_0^2y_1^2} \\
&= \frac{x_1^2 + (y_0-y_1)^2}{y_0y_1}
\end{aligned}$$

and since $x_1, y_0, y_1 > 0$, and $(y_0 - y_1)^2 \geq 0$, Δ is positive. Hence we have two solutions. One thing to note, when $y_0 = y_1$, hence $a = 0$ and $\frac{-b \pm \sqrt{b^2-4ac}}{2a} = \infty$. This case must be handled with care since p is the midpoint of the segment $\overline{s_0s_1}$, and the second intersection point is at infinity.

Bibliography

[Aur91] Franz Aurenhammer. Voronoi diagrams a survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23(3):345–405, 1991.

[BD05] Jean-Daniel Boissonnat and Christophe Delage. Convex hulls and voronoi diagrams of additively weighted points. In *13th European Symposium on Algorithms*, pages 367–378, 2005.

[BK03] Jean-Daniel Boissonnat and Menelaos I. Karavelas. On the combinatorial complexity of euclkirkpatricknoi cells and convex hulls of d-dimensional spheres. In *SODA '03: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 305–312, Philadelphia, PA, USA, 2003. Society for Industrial and Applied Mathematics.

[CGA] CGAL. Computational geometry algorithms library. http://www.cgal.org/.

[Des28] Rene Descartes. Le monde, ou traitè de la lumiére, 1628.

[Dir50] Gustave Lejeune Dirichlet. *Ü*ber die reduktion der positiven quadratischen formen mit drei unbestimmten ganzen zahlen. *Journal für die Reine und Angewandte Mathematik*, pages 209–227, 1850.

[DL78] Robert L. (Scot) Drysdale and D.T. Lee. Generalized voronoi diagrams in the plane. In *16th Allerton Conf. Commun. Control Comput.*, pages 833–842, 1978.

[FG00] Pascal Jean Frey and Paul-Louis George. *Mesh Generation: Application to Finite Elements*. Hermes Science, 2000.

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

[gNYU] Exact Geometric Computation group New York University. Core. http://www.cs.nyu.edu/exact/.

[HB01] Christoph Burnikel Sylvain Pion Herv Brnnimann. Interval arithmetic yields efficient dynamic filters for computational geometry. *Discrete Applied Mathematics*, pages 25–47, 2001.

[HTK+85] L Hoofd, Z Turek, K Kubat, BEM Ringnalda, and S Kazda. Variability of intercapillary distance estimated on histological sections of rat heart. *Oxygen transport to tissue VII*, 1985.

[Kar01] Menelaos I. Karavelas. *Proximity Structures for Moving Objects in Constrained and Unconstrained Environments*. PhD thesis, Stanford University, 2001.

[KE02] Menelaos I. Karavelas and Ioannis Z. Emiris. Predicates for the planar additively weighted voronoi diagram. Technical Report ECG-TR-122201-01, INRIA Sophia-Antipolis, 2002.

[Kir79] David Kirkpatrick. Efficient computation of continuous skeletons. In *14th IEEE Symposium on Foundations of Computer Science*, pages 18–27, 1979.

[KMM93] Rolf Klein, Kurt Mehlhorn, and Stefan Meiser. Randomized incremental construction of abstract voronoi diagrams. *Computational Geometry: Theory and Application*, 3:157–184, 1993.

[KY02a] Menelaos I. Karavelas and Mariette Yvinec. Dynamic additively weighted voronoi diagrams in 2d. In *10th European Symposium on Algorithms*, pages 586–598, 2002.

[KY02b] Menelaos I. Karavelas and Mariette Yvinec. Dynamic additively weighted voronoi diagrams in 2d. Technical Report ECG-TR-122201-01, INRIA Sophia-Antipolis, 2002.

[KY06] Menelaos Karavelas and Mariette Yvinec. 2d apollonius graphs (delaunay graphs of disks). In CGAL Editorial Board, editor, *CGAL-3.2 User and Reference Manual*. CGAL, 2006.

[LD81] D.T. Lee and Robert L. (Scot) Drysdale. Generalization of voronoi diagrams in the plane. *SIAM J. Comput.*, 10:73–87, 1981.

[OBSC00] Atsuyuki Okabe, Barry Boots, Kokichi Sugihara, and Sung Nok Chiu. *Spatial tessellations: Concepts and applications of Voronoi diagrams*. Probability and Statistics. Wiley, NYC, 2nd edition, 2000.

[Sha48a] Claude Elwood Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, July 1948.

[Sha48b] Claude Elwood Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:623–656, October 1948.

[Sha85] Micha Sharir. Intersection and closest-pair problems for a set of planar discs. *SIAM J. Comput.*, 14:448–468, 1985.

[Sno54] John Snow. Report on the cholera outbreak in the parish of st james, westminster during the autumn of 1854. *Medical Times*, pages 39–54, 1854.

[TH96] Hai Tao and Thomas Huang. Multi-scale image warping using weighted voronoi diagram. In *International Conference on Image Processing*, pages 241–244, 1996.

[Thi11] Alfred H. Thiessen. Precipitation averages for large areas. *Monthly Weather Review*, 39(7):1082–1084, 1911.

[Vor07] Georgy Voronoi. Nouvelles applications des paramètres continus *à* la théorie des formes quadratiques. *Journal für die Reine und Angewandte Mathematik*, pages 97–178, 1907.

[Yap85] Chee-Keng Yap. An o(n log n) algorithm for the voronoi diagram of a set of simple curve segments. Technical Report 43, Courant Institute New York University, 1985.

[Yap87] Chee-Keng Yap. An o(n log n) algorithm for the voronoi diagram of a set of simple curve segments. *Discrete & Computational Geometry*, 2:365–393, 1987.