Constructing the Voronoi diagram using two sweeps.

Sichao. Wang

University of Windsor

Follow this and additional works at: https://scholar.uwindsor.ca/etd

Recommended Citation
https://scholar.uwindsor.ca/etd/1553

This online database contains the full-text of PhD dissertations and Masters' theses of University of Windsor students from 1954 forward. These documents are made available for personal study and research purposes only, in accordance with the Canadian Copyright Act and the Creative Commons license—CC BY-NC-ND (Attribution, Non-Commercial, No Derivative Works). Under this license, works must always be attributed to the copyright holder (original author), cannot be used for any commercial purposes, and may not be altered. Any other use would require the permission of the copyright holder. Students may inquire about withdrawing their dissertation and/or thesis from this database. For additional inquiries, please contact the repository administrator via email (scholarship@uwindsor.ca) or by telephone at 519-253-3000ext. 3208.
NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilming. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopy de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.
CONSTRUCTING THE VORONOI DIAGRAM USING TWO SWEEPS

By Sichao Wang

A THESIS
SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH THROUGH THE SCHOOL OF COMPUTER SCIENCE IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF COMPUTER SCIENCE AT THE UNIVERSITY OF WINDSOR

Windsor, Ontario Canada
April 1991
The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.
Abstract

The currently known sweep-line algorithm for constructing the nearest neighbor Voronoi diagram uses geometric transformation. This thesis presents another sweep-line algorithm without using geometric transformation. The algorithm conceptually consists of two steps. In the first step, two planar graphs called Left-Right Partial Voronoi Graph (LR-PVG) and Right-Left Partial Voronoi Graph (RL-PVG) are constructed in $O(n \log n)$ time and $O(n)$ space using the two plane sweeps, where $n$ is the number of points in the given point set. In the second step, a top-down non-recursive version of the divide-and-conquer method is used to merge LR-PVG and RL-PVG into the final Voronoi diagram in $O(n \log n)$ time and $O(n)$ space.
Acknowledgments

I would like to thank Dr. Yung H. Tsin, my supervisor, for suggesting Computational Geometry as a promising research area and for his advice, encouragement, and support. I would also like to thank Dr. Myron Hlynka and Dr. Henry E. Toews for serving as my co-supervisors, reading the previous version of this thesis and making valuable suggestions. I am also grateful to Dr. Richard Frost for giving me good advice and providing me with inspirations.

I am pleased to acknowledge financial supports from the Faculty of Graduate Studies and Research at the University of Windsor for offering me the Postgraduate Scholarship (1989–1991), the School of Computer Science, the State Education Committee of China and my supervisor — Dr. Yung H. Tsin through his NSERC operating grants.

Finally, my special thanks go to my parents for giving me constant and emotional support.
Contents

Abstract .............................................................. iii
Acknowledgments ...................................................... vii
List of Figures ......................................................... xi

Chapter 1 A Survey in the Algorithms of Computing the Voronoi Diagram ........................................... 1
  Section 1 An Overview ............................................. 1
  Section 2 Generalizations of the Voronoi Diagram .......... 1
  Topic 1 The Voronoi Diagrams Under L_p-Metrics ............ 2
  Topic 2 Voronoi Diagrams of Higher Orders ................. 2
  Topic 3 The Constrained and Weighted Voronoi Diagrams ... 3
  Topic 4 The Voronoi Diagram for General Objects .......... 5
  Topic 5 The Voronoi Diagram in Higher Dimensions ........ 6

Chapter 2 The Nearest Neighbor Voronoi Diagram ............... 7
  Section 1 The Nearest Neighbor Voronoi Diagram ............ 7
  Section 2 Review of Some Existing Algorithms ............... 9
  Topic 1 Divide-and-Conquer Algorithm ....................... 9
  Topic 2 Sweepline Algorithm Combined with Geometric Transformation ........................................... 13
List of Figures

Figure 1  An example of the nearest neighbor Voronoi diagram 8
Figure 2  The transformed bisector of b(p, q) when y(p) ≠ y(q) 15
Figure 3  The transformed bisector of b(p, q) when y(p) = y(q) 15
Figure 4  The original Voronoi diagram 16
Figure 5  The transformed Voronoi diagram 16
Figure 6  An example of DCEL structure 21
Figure 7  The proof for lemma 3.2.1 23
Figure 8  The dividing curve δ is monotone 34
Figure 9  Shall γ₄ be defined as the portion of the ray emanating from v₄ in the direction of the bisector b(p₁, p₂) 38
Figure 10  Four possible escapings 39
Figure 11  (I) The test succeeds; (II) The test fails 40
Figure 12  The escape scan method 42
Figure 13  A jump example 44
Figure 14  Proof for lemma 3.3.8 45
Figure 15  When v is an intersection of γ₁ with a vertical boundary 47
Figure 16  The worst case situation 50
Figure 17  Illustration for lemma 3.3.9 51
Figure 18  The rules of conducting the marking 54
Figure 19  An example of computing the skeleton for the front hole of γ₁ 54
Figure 20  The front holes 56
Figure 21  when the sweepline is at p₁ 76
Figure 22 when the sweepline is at $p_2$. ............................ 77
Figure 23 when the sweepline is at $p_3$. ............................ 77
Figure 24 when the sweepline is at $p_4$. ............................ 78
Figure 25 when the sweepline is at $p_5$. ............................ 78
Figure 26 when the sweepline is at $p_6$. ............................ 79
Figure 27 when the sweepline is at $p_7$. ............................ 79
Figure 28 when the sweepline is at $p_8$. ............................ 80
Figure 29 when the sweepline is at $p_9$. ............................ 81
Figure 30 when the sweepline is at $p_2$. ............................ 82
Figure 31 when the sweepline is at $p_3$. ............................ 82
Figure 32 when the sweepline is at $p_4$. ............................ 83
Figure 33 when the sweepline is at $p_5$. ............................ 83
Figure 34 when the sweepline is at $p_6$. ............................ 84
Figure 35 when the sweepline is at $p_7$. ............................ 84
Figure 36 when the sweepline is at $p_8$. ............................ 85
Figure 37 when the sweepline is at $p_9$. ............................ 85
Figure 38 The final LR-PVG(S) graph. ............................ 86
Chapter 1  A Survey in the Algorithms of Computing the Voronoi Diagram

1 An Overview

The original version of a Voronoi diagram is defined for a set of \( n \) points \( S \) on the plane in Euclidean metric. The first \( O(n \log n) \) time and \( O(n) \) space solution for the construction of the Voronoi diagram of \( n \) points in the plane is a divide-and-conquer algorithm presented by Shamos and Hoey [SH75]. In 1976, D. T. Lee made modifications to the scanning scheme of Shamos and Hoey’s algorithm [Lee76] (see Section 2 of Chapter 2). Fortune [For86] presented another optimal algorithm which uses the plane-sweep method. His geometric transformation transforms the left-most point (if sweeping from the left to the right) of the Voronoi polygon of a site to the point site itself, therefore the Voronoi polygon of the site is considered only when the site is intercepted by the sweep-line. Also Fortune’s sweepline algorithm does not require the general position assumption (see Section 2, Chapter 2). Moreover, many algorithms using incremental technique (also known as the iterative greedy method) have also been proposed [LS80]. With the incremental method the diagram is constructed by considering one site at a time. These incremental algorithms are relatively simple, but have \( O(n^2) \) worst case time complexity. Nevertheless, their average time complexity is shown to be good [BW80], [GS77], [Ohy84].

2 Generalizations of the Voronoi Diagram

A natural question is “Can the Voronoi diagram be generalized to other distance metric and objects?” In fact, there are a lot of works which have been done on the generalizations of the Voronoi diagram.
The Voronoi Diagrams Under $L_p$-Metrics

The Euclidean metric can be generalized to the notion of $L_p$-metric. In the Euclidean space $E^d$ of coordinates $x_1, x_2, \ldots, x_d$, the $L_p$-metric is defined as: for any real $1 < p < \infty$ the $L_p$-distance of two points $q_1$ and $q_2$ is given by the norm

$$d_p(q_1, q_2) = \left( \sum_{j=1}^{d} |x_j(q_1) - x_j(q_2)|^p \right)^{1/p}$$

Hwang [Hwa79] presented an $O(n \log n)$ time algorithm which constructs the Voronoi diagram for a set of points in $E^2$ under $L_1$-metric (also called Manhattan distance metrics). Lee and Wong [LW80] presented an $O(n \log n)$ time algorithm for $n$ points in $E^2$ under $L_1$ and $L_\infty$-metrics and considered its application to 2-dimensional memory-storage systems. Finally, Lee [Lee80] presented an $O(n \log n)$ time algorithm for the Voronoi diagram for the points in $E^2$ under the $L_p$-metric, where $1 < p < \infty$. All these algorithms use the divide-and-conquer method and are proved to be optimal. Chew and Drysdale [CD85] presented a general optimal algorithm for computing Voronoi diagram under any convex distance function (a generalization to the $L_p$-metric distance function). Klein [Kle89] defined the abstract Voronoi diagram in terms of bisecting Jordan curves without using the notion of distance. His result shows that if the Voronoi diagram is the collection of regions (each region is associated with one point site) which partitions the plane and each region is path-connected, then the CW-CCW scanning scheme of Shamos and Hoey [SH75] is applicable. Moreover, if the dividing curve does not self-intersect then the merging step can be done in $O(n)$ time.

Voronoi Diagrams of Higher Orders

Consider the Voronoi diagram whose polygons are associated with a subset of $k$ points, where $1 \leq k < n$. The $k$-th order Voronoi polygon of a subset $T$ (assume $|T| = k$)
is defined as $V(T)$, such that $V(T) = \{ p \in \mathbb{R}^2 \mid d(p, v) < d(p, u) \text{ for all } v \in T \text{ and } u \in S - T \}$ [SH75]. Then the Voronoi diagram of order $k$ of a set of points $S$ is defined as the collection of all $k$-th order Voronoi polygons, i.e., $Vor_k(S) = \bigcup_{T \subseteq S} \text{ and } |T| = k V(T)$. Thus the ordinary Voronoi diagram is $Vor_1(S)$. The $k$-th order Voronoi diagram was first studied by Lee [Lee86], [Lee82], and he presented an $O(k^2 n \log n)$ time algorithm. Edelsbrunner and Seidel [ES86] presented an $O(n^3)$ time algorithm for constructing $Vor_k(S)$, where $1 \leq k < n$. Also the farthest neighbor Voronoi diagram, i.e., $Vor_n(S)$ is solved by Lee with an $O(n \log n)$ time divide-and-conquer algorithm [Lee80]. Aurenhammer [Aur87] considered the $k$-th order Voronoi diagram in $\mathbb{R}^d$ under the Laguerre metric. He presented an $O(n^d + 1)$ time algorithm which simultaneously constructs $Vor_k(S)$, $1 \leq k < n$. His algorithm is optimal for every even $d$.

The Constrained and Weighted Voronoi Diagrams

The weighted Voronoi diagram has been defined in several ways [AE84], [For86], depending on the distance metric used. In this case, each point site in $S$ is a point having non-negative weight associated with it.

In the model in which the metric distance from any point $x$ in the plane to a point site $p$ in $S$ is the sum of the Euclidean distance $d(x, p)$ and the weight of the site $w(p)$, the bisector of two weighted point sites $p_i$ and $p_j$ is defined as: $b(p_i, p_j) = \{ p \in \mathbb{R}^2 \mid d(p, p_i) + w(p_i) = d(p, p_j) + w(p_j); p_i \neq p_j \text{ and } p_i, p_j \in S \}$. Therefore the bisector $b(p_i, p_j)$ is the geometric locus $\{ p \mid d(p, p_i) + w(p_i) = d(p, p_j) + w(p_j) \}$, and $b(p_i, p_j)$ is one of the two branches of the hyperbola with points $p_i$ and $p_j$ as foci and eccentricity $d(p_i, p_j) / |w(p_i) - w(p_j)|$. Specifically, $b(p_i, p_j)$ belongs to the branch closer to $p_i$ if $w(p_i) > w(p_j)$ or to the branch closer to $p_j$ otherwise. Also there are two degenerate cases:

1. $w(p_i) - w(p_j) = 0$. In this case, $b(p_i, p_j)$ is the perpendicular bisector of $p_i$ and $p_j$;
(2) \( lw(p_i) - w(p_j)l = d(p_i, p_j) \). Then the eccentricity is equal to 1, and each branch of hyperbola generates to a half-line.

The weighted Voronoi diagram of a set of weighted points S is still a partition of the plane, and each Voronoi region consists of points closest to a particular site. Therefore the normal Voronoi diagram is a special case of the weighted Voronoi diagram when all point sites have the same weight. The Voronoi diagram resulting from this definition has an appearance similar to the unweighted Voronoi diagram except that the bisectors are sections of hyperbolas.

Another model is Aurenhammer’s definition of a weighted Voronoi diagram. In his model, the weighted distance from an arbitrary point \( x \) to \( p \) is given by \( d_w(x, p) = d_e(x, p) / w(p) \) where \( d_e \) denotes the Euclidean distance function. The weighted Voronoi region of a point, in this model, is defined as \( V(p) = \{ x \in \mathbb{R}^2 \mid d_w(x, p) \leq d_w(x, q), q \in S \} \). Aurenhammer presented an \( O(n^2) \) time algorithm [Aur84]. His algorithm is optimal since the diagram in this model can have at most \( O(n^2) \) faces, edges and vertices. Aurenhammer [Aur86] considered the 1-dimensional weighted Voronoi diagram and presented an optimal algorithm for constructing the diagram. Subsequently, Aurenhammer extended the previous results to \( d \)-dimensional and presented an algorithm which is optimal when \( d \) is even [Aur87].

The Voronoi diagram with the existence of barriers forms another extension to the Voronoi diagram. Lee and Lin [LL86] first introduced the Voronoi diagram of the endpoints of a set of line segments with the open line segments being considered as barriers. They presented an \( O(n^2) \) algorithm. Wang and Schubert [WS87] improved this bound to \( O(n \log n) \). Chew [Che87] also improves the bound to \( O(n \log n) \) independently. Recently, C.A. Wang and Y.H. Tsin [TW90] presented an \( O(m^2n^2 + n^4) \) time algorithm
for computing the constrained (multiplicative) weighted Voronoi diagram of $n$ weighted points in the presence of $m$ line segments barriers, where the distance between two points is measured by the straight line visible distance. Their algorithm is shown to be optimal both in time and space complexity when $m \geq cn$ for any constant $c$.

The Voronoi Diagram for General Objects

The Voronoi diagram can also be generalized to other objects, such as line segments and circular arcs. This problem was first posed and solved in $O(nc\sqrt{\log n})$ time by Drysdale [Dry79], and later on improved to $O(n \log n^2)$ time by Drysdale and D. T. Lee [DL84]. Drysdale and D. T. Lee's algorithm uses a divide-and-conquer method, and is not optimal. In the merging procedure of Drysdale and D. T. Lee's algorithm, a starter set is found in $O(n \log n)$ time such that each piece of the merge curves passes through at least one point of the starter set. Therefore when the starter set is determined, all pieces of the merge curves can be found by simply tracing from a corresponding point in the starter set. Drysdale and Lee's method avoided the "vertical separability" required for partitioning the given set of objects.

Kirkpatrick [Kir79] improves this time bound to $O(n \log n)$ which is optimal. His method is also based on the divide-and-conquer strategy but does not require partitioning the set $S$ vertically. His idea is to subdivide each Voronoi cell (by introducing "spokes") into simpler subcells, and to use the fact that a certain minimum spanning tree of the given set of objects intersects the Voronoi edges and spokes of the two subsets of objects. This allows one to find all the starting points in linear time. But his method is complicated enough that the details of its correctness are still fuzzy [Yap87]. However, Kirkpatrick's idea of "spokes" and "minimum spanning tree" are of independent interest.

Sharir [Sha85] outlined an $O(n \log^2 n)$ divide-and-conquer algorithm for the Voronoi
diagram of \( n \) circles (possibly intersecting). S. Fortune [For86] used a combination of transformation and sweepline techniques to present simpler \( O(n \log n) \) time and \( O(n) \) space algorithms for computing the Voronoi diagrams for point set (unweighted and weighted) and line segments. Rather than computing the Voronoi diagram directly with a sweepline technique, Fortune first computes the geometric transformation of the diagram, and then reconstruct the original Voronoi diagram from the transformed one in linear time. Yap [Yap87] presented an optimal \( O(n \log n) \) time divide-and-conquer algorithm for computing the Voronoi diagram for a set of simple curve segments (including line segments, circular arcs, etc.). Yap's merging procedure is very different from the previous algorithms in that the object set is "brutally" divided into subsets. His idea is based on the so called window-contour merging, which includes vertical merging and horizontal merging procedures. Yap claimed [Yap87] that this optimal algorithm could be extended to more general algebraic curves, provided that each curve can be broken into a number of small simple curves.

The Voronoi Diagram in Higher Dimensions

Another extension of Voronoi diagram is to consider the Voronoi diagram in \( \mathbb{E}^d \) for \( d \geq 1 \). Many results for this problem have been obtained in [AB83], [BDF78], [Kle80]. For point sets and Euclidean metric, Brown [Bro79] showed that by using the inversion transformation, one could reduce the construction of Voronoi diagram in \( \mathbb{E}^d \) to that of convex hull in \( \mathbb{E}^{d+1} \). Hence, by using Seidel's result [Sei81] and Preparata and Hong's result [PH77] on convex hull, an efficient algorithm for Voronoi diagrams in \( \mathbb{E}^d \) can be derived. This algorithm is optimal for \( d = 2, 3 \) and every even \( d \).

Recently, dynamic Voronoi diagrams and the numerical stability issue in computing Voronoi diagrams are beginning to draw attention.
Chapter 2 The Nearest Neighbor Voronoi Diagram

The Voronoi diagram is a well known geometric structure that has been studied for many years [SH75], [LW78], [LP78], [Dry79], [Kir79], [Lee80], [Sha85], [Yap87] in areas such as pattern recognition [Tou80], wire layout [MC79], computer graphics [New79], robotic vision [LS85], geometric optimization [Saa70], contouring problems [Dav75] and biology.

In Section 1 of this Chapter, we define the nearest neighbor Voronoi diagram. In Section 2, we illustrate two typical techniques used in computing the nearest neighbor Voronoi diagram of a set of points in the plane, i.e., the Divide-and-Conquer method by Shamos and Hoey [SH75] with D.T. Lee’s modifications [Lee78] and the Sweepline technique by Fortune [For86].

1 The Nearest Neighbor Voronoi Diagram

Consider the problem of computing the nearest neighbor Voronoi diagram of a set $S$ of $n$ points in the plane. The problem is to compute, for any point $v$ in $S$, the locus of the point which is closer to $v$ than to any other points in $S$.

Precisely, a Voronoi cell (or region) of any point site $p_i$ from $S$ is denoted as $V(p_i)$ where $V(p_i) = \{ r \in \mathbb{R}^2 | d(r, p_i) \leq d(r, p_j) \text{ for all } p_j \neq p_i \in S \}$. 

Chapter 2 — The Nearest Neighbor Voronoi Diagram

Figure 1 An example of the nearest neighbor Voronoi diagram.

Then the nearest neighbor Voronoi diagram of $S$ is the collection of Voronoi regions associated with each point in $S$, i.e., $\text{Vor}(S) = \{ V(p) \mid p \in S \}$ (see Figure 1).

By computing the Voronoi diagram $\text{Vor}(S)$ of a set of points $S$, we shall mean to take $S$ as input and output a description of the diagram as a planar graph embedded in the plane, consisting of the following items:

1. The coordinates of the Voronoi vertices;
2. The set of Voronoi edges (each as a pair of Voronoi vertices) and for each of these edges, the two edges that are its counterclockwise successors at each extreme point (Doubly-Connected-Edge-Link or DCEL, for details see Section 1 of Chapter 3). This simply provides the counterclockwise edge cycle at each vertex and the clockwise edge cycle around each face.
Chapter 2 – The Nearest Neighbor Voronoi Diagram

Before reviewing some existing algorithms, we introduce the following preliminary definitions:

**Definition 2.1.1:** The *convex hull* of a set of points $S$ in the plane is the smallest convex polygon which contains all the points of $S$ inside. The points of the $S$ on the convex polygon are called the *hull vertices*.

**Definition 2.1.2:** A *supporting line* between two disjoint convex polygons $P$ and $Q$ is the line connecting a vertex $p$ of $P$ to a vertex $q$ of $Q$ such that both polygons $P$ and $Q$ lie on the same side of the line $pq$.

## 2 Review of Some Existing Algorithms

**Divide-and-Conquer Algorithm**

Divide-and-Conquer is the most classical problem solving technique and has proven its value for geometric problems as well [Ben78], [PH77]. The central feature of divide-and-conquer is to invoke the principles of balancing, which suggest that a computational problem should be divided into subproblems of nearly equal size, and then solve each of the subproblems recursively. Then a merging algorithm combines those solutions for the subproblems into a single solution for the original problem [AHU74]. Therefore, the problem of computing Voronoi diagrams is eminently suited to attack by divide-and-conquer.

**Main Procedure** **COMPUTING THE VORONOI DIAGRAM**

1. Linearly partition\(^{[1]}\) $S$ into two subsets $S_1$ and $S_2$ of approximately equal sizes;
2. Construct $\text{Vor}(S_1)$ and $\text{Vor}(S_2)$ recursively;
3. Merge $\text{Vor}(S_1)$ and $\text{Vor}(S_2)$ to obtain $\text{Vor}(S)$.

\(^{[1]}\) Find a vertical line such that the line separates $S$ into two parts.
The initial partition of $S$ according to the median of the $x$-coordinates can be done in time $O(n)$ by the standard median finding algorithms\textsuperscript{[2]}. The merging process can be carried out in $O(n)$ time [SH75], [Lee78].

The divide-and-conquer method depends for its success on various structural geometric properties of the diagram that enable us to merge subproblems in linear time [PS85]. We shall illustrate some important properties which the merging procedure relies on.

**Definition 2.2.3**: The straight line dual $D(S)$ of a Voronoi diagram $Vor(S)$ of a set of $n$ points $S$ in the plane is a graph which has the same vertex set as $Vor(S)$. Two points of $S$ are connected by an edge in $D(S)$ if their associated Voronoi regions share a Voronoi edge.

In the following, we review some well-known geometric properties of convex hulls, Voronoi diagrams and their relationships.

**Lemma 2.2.1**: A Voronoi diagram of $n$ points in the plane has at most $(3n-6)$ edges and $(2n-4)$ vertices.

[proof:] See [PS85].

**Lemma 2.2.2**: A point site $p$ is a hull vertex if and only if its associated Voronoi region $V(p)$ is unbounded.

[proof:] See [PS85].

**Lemma 2.2.3**: Given the Nearest Voronoi Diagram on $n$ points in the plane, their convex hull can be obtained in linear time.

[proof:] See [PS85].

**Theorem 2.2.1**: Let $p_l \in S_{\text{left}}$ and $p_r \in S_{\text{right}}$, and $b(p_l, p_r)$ be a section of the dividing curve. Then in the final Voronoi diagram, all points of $V(p_l)$ lie to left of the oriented bisector $b(p_l, p_r)$, and all points of $V(p_r)$ lie to the right of the oriented bisector $b(p_l, p_r)$.

\textsuperscript{[2]} If more than one point belongs to the separating lines, all of these are assigned to the same set of the partition.
Furthermore, scanning \( V(p_l) \) in a CW direction and \( V(p_r) \) in a CCW direction will find the first intersection between \( b(p_r, p_l) \) and \( V(p_l) \) or \( V(p_r) \).

(proof :) See [Dry79]. □

Theorem 2.2.1 shows that all the edges of \( \text{Vor}(S_{\text{left}}) \) (resp. \( \text{Vor}(S_{\text{right}}) \) which lie to the right (resp. to the left) of \( b(p_l, p_r) \) can simply be discarded during the merging procedure. It is guaranteed by Theorem 2.2.1 that no part of a discarded segment will be included in \( \text{Vor}(S) \). The convexity of Voronoi regions guarantees that a discarded segment will never be intersected by an extension of a segment in \( B(S_{\text{left}}, S_{\text{right}}) \) later in the scanning process.

The Merging Procedure This merging algorithm was first presented by Shamos [Sha75], and later its scanning scheme was modified by D. T. Lee [Lee76].

Assume (inductively) that \( \text{Vor}(S_{\text{right}}) \) and \( \text{Vor}(S_{\text{left}}) \) have been recursively obtained. Thus, as a by-product of this assumption, by Lemma 2.2.3, we also obtain the convex hulls \( \text{CH}(S_{\text{left}}) \) and \( \text{CH}(S_{\text{right}}) \) in linear time. After the convex hulls for \( S_{\text{left}} \) and \( S_{\text{right}} \) are both available, their two supporting lines (see Definition 2.1.2) are constructed in at most \( O(n) \) time using either an algorithm invented by Preparata and Hong [PH77] or a linear algorithm by Shamos [Sha78] (Shamos's algorithm computes the convex hull of two convex polygons in linear time, and hence the two supporting lines of the two convex polygons is computed as a by-product). Let us arbitrarily call one of the two supporting lines the upper supporting line and the other the lower supporting line. These two supporting lines are used to determine the two new hull edges in merging \( \text{CH}(S_{\text{left}}) \) and \( \text{CH}(S_{\text{right}}) \). We call the hull edge determined by the upper (resp. lower) supporting line the upper (resp. the lower) hull edge. Let the perpendicular bisector of the upper (lower resp.) hull edge be called the starter (resp. ender) ray. Starting from the starter
ray, we construct the merge curve edge by edge, until the end ray is reached. This merge procedure is the key step of the divide-and-conquer method.

We can imagine forming the merge curve by first moving from infinity inward along the starter ray \( b(p_1, p_r) \), where \( p_1 \) and \( p_r \) are the two endpoints of the upper hull edge. Then we use the following scanning scheme to find the edge in \( \text{Vor}(S_{\text{left}}) \) or \( \text{Vor}(S_{\text{right}}) \) which first intersects \( b(p_1, p_r) \). In the following description, we always assume that \( b(p_1, p_r) \) is the most recent bisector section on the current dividing chain that we are constructing.

We scan the edges of the region \( V(p_1) \) of \( \text{Vor}(S_{\text{left}}) \) in counterclockwise (CCW) and the edges of the region \( V(p_r) \) of \( \text{Vor}(S_{\text{right}}) \) in clockwise (CW). It has been shown in Theorem 2.2.1 that no points of \( V(p_1) \) in \( \text{Vor}(S_{\text{left}}) \) which lies to the right of the oriented bisector \( b(p_1, p_r) \) will be included in the region \( V(p_1) \) in the final Voronoi diagram. Similarly, no points of \( V(p_r) \) in \( \text{Vor}(S_{\text{right}}) \) which lies to the left of the oriented bisector \( b(p_1, p_r) \) will be included in \( V(p_r) \) in the final Voronoi diagram. Therefore, once the bisector \( b(p_r, p_1) \) is formed, we simply discard all those edges of \( V(p_1) \) in \( \text{Vor}(S_{\text{left}}) \) lying to the right of the bisector \( b(p_r, p_1) \) and all those edges of \( V(p_r) \) in \( \text{Vor}(S_{\text{right}}) \) lying to the left of the bisector \( b(p_r, p_1) \).

In general, when we construct the dividing curve, if \( b(p_1, p_r) \) leaves region \( V(p_1) \) first at edge \( b(p_r, p^*) \) for some \( p^* \in S_{\text{right}} \), then assign \( p^* \) to \( p_r \). If \( b(p_1, p_r) \) leaves region \( V(p_1) \) first at edge \( b(p^*, p_1) \) for some \( p^* \in S_{\text{left}} \), then assign \( p^* \) to \( p_1 \).

The Performance Analysis of the Divide-and-Conquer Algorithm The performance analysis of this divide-and-conquer algorithm is very straightforward. Let \( T(n) \) be the time for computing the nearest neighbor Voronoi diagram for \( n \) points in the plane. For convenience, we assume that \( n \) is a power of 2, then we have the following recurrence relation:
Chapter 2 – The Nearest Neighbor Voronoi Diagram

\[ T(1) = \text{constant}; \]
\[ T(n) = 2T(n/2) + M(n/2, n/2), \]

where \( M(s, t) \) is the time for merging the two recursively computed Voronoi diagrams with \( s \) and \( t \) point sites respectively.

As we have shown that \( M(n/2, n/2) = O(n) \). By solving the above recurrence relation, we have \( T(n) = O(n \log n) \). By Lemma 2.2.1, the total number of edges in \( \text{Vor}(S) \) is bounded by \( 3n-6 \), thus only \( O(n) \) space is needed by this algorithm.

Sweepline Algorithm Combined with Geometric Transformation

Fortune’s algorithm [For86] was very different from any other known algorithms at that time. His elegant idea is based on the sweepline method (or plane-sweep) in a transformed space.

Sweepline Technique  A Sweepline scheme is suggested – uniquely and naturally – by the nature of the geometric problems. Generally, the Sweepline algorithm is characterized by two basic structures: (1) the event-point schedule, which is a sequence of positions, ordered from left to right, to be assumed by the sweepline, and (2) the sweepline status, which is the structure that describes the intersections of the sweepline with the geometric structure. There are two kinds of transition points. One kind is the points of \( S \). The other is the intersections of bisectors. The sweepline sweeps the plane from the bottom to the top (or from left to right), stopping at every transition point (event-point). After the processing at every stop, the sweepline moves upwards (resp. left-wards) to the next event-point until all the event-points have been swept. There is no backtracking. All events to be processed are queued in an \( x \)-queue \( Q \). The status of the sweepline is maintained as a balanced binary search tree \( T \). \( T \) is updated at every transition point.
The Geometric Transformation  Computing the Voronoi diagram directly with a sweep-line technique is difficult, because the Voronoi region of a site itself is intersected by the sweepline long before the site itself is intersected by the sweepline. Steven Fortune presented [For86] an optimal algorithm which transforms the computation of a Voronoi diagram into the computation of its transformed diagram, thereby avoiding the above mentioned difficulty.

The geometric transformation that Fortune introduced maps each point in the plane to the topmost point of its Voronoi circle. (A Voronoi circle of any point \( p \) in the plane is the circle centered at \( p \) with the radius equal to the distance from \( p \) to its nearest neighbor point in the point set \( S \)). Therefore, the lowest point of the transformed Voronoi region is the site itself. Hence we only need to consider the Voronoi region of a site when the site has been intersected by the sweepline. This is why the transformation method is used.

The geometrical effects induced by the transformation are described as follows:

1. The effects on bisector \( b(p, q) \):

   - \( y(p) \neq y(q) \): If \( y(q) > y(p) \), then the bisector \( b(p, q) \) will be transformed into an open hyperbola with \( p \) as the bottommost point of the hyperbola; if \( y(p) > y(q) \), then the bisector \( b(p, q) \) will be transformed into an open hyperbola with \( q \) as the bottommost point of the hyperbola. (See Figure 2).
Chapter 2 – The Nearest Neighbor Voronoi Diagram

Figure 2 The transformed bisector of \( b(p, q) \) when \( y(p) \neq y(q) \).

- \( y(p) = y(q) \): Then the transformed \( b(p, q) \) is still a vertical line except that the part of \( b(p, q) \) below the horizontal line \( \overline{pq} \) is missing (see Figure 3).

Figure 3 The transformed bisector of \( b(p, q) \) when \( y(p) = y(q) \).
2. The effect on Voronoi region:

Each original Voronoi region is "raised" up with the associated site as the lowest point except the region associated with the lowest point site in S (See Figure 4 and Figure 5).

Figure 4 The original Voronoi diagram.

Figure 5 The transformed Voronoi diagram.
Chapter 2 – The Nearest Neighbor Voronoi Diagram

It is obvious from Figure 3 that this transformation is not one-to-one, i.e., a Voronoi region does not necessarily have the same incident structure as its transformed version. Fortunately, if we decompose each Voronoi region $V(p_i)$ into edges and vertices (as we expect), the images of edges and vertices of $V^*(p_i)$ would have the same incident structure as $V(p_i)$.

The Sweepline Algorithm for Computing $\text{Vor}(S)$

The Sweepline algorithm conceptually moves a horizontal line from the bottom to the top (or a vertical line from the left to the right) across the plane, maintaining the regions intersected by the horizontal line (respectively, the vertical line). We observe that a Voronoi region is encountered by the sweepline for the first time at its site, and a region disappears at the intersection of two edges. Since initially we know all the given sites, the intersections can be computed as the regions bounded by the edges are encountered.

For simplicity, we denote the transformed bisector $b(p, q)$ by $B_{pq}$, and the transformed Voronoi region $V(p_i)$ of a point $p_i$ by $V^*(p_i)$. Also we classify each bisector $B(p, q)$ as two pieces, $C^-_{pq}$ and $C^+_{pq}$, where the piece $C^+_{pq}$ is to the left of $p$ and containing $p$, $C^-_{pq}$ is the piece to the right of $p$ and containing $p$. It is easy to see that $C^-_{pq}$ is monotonically decreasing and $C^+_{pq}$ is monotonically increasing. Moreover, any vertical line intersects $C^-_{pq}$ or $C^+_{pq}$ at most once.

The following is the sweepline algorithm given by Steven Fortune [For86] for computing the nearest neighbor Voronoi diagram for a set of points in the plane.

**Algorithm** : SWEELINE $(S)$;

**Input** : $S = \{p_1, p_2, ..., p_n\}$;

**Output** : The bisectors and vertices of the transformed $\text{Vor}(S)$;

**Data structures** : $Q$: a priority queue of points in the plane, ordered lexicographic...
graphically. Each point is labelled as a site, or labelled with a pair of boundaries. T: a sequence \((r_1, c_1, r_2, \ldots, n_k)\) of regions and boundaries.

Begin

sort the elements of S according the x-coordinates of the points and initialize Q with these sorted point sites;

\(p \leftarrow \text{extract}_\text{min}(Q); //extract the element of Q with the least x-coordinate\)

\(T \leftarrow\) the list containing \(V^*(p)\);

while \(Q\) is not empty do

begin

\(p \leftarrow \text{extract}_\text{min}(Q); //pop the first element of Q\)

case 1: \(p\) is a site

Find an occurrence of a region \(V^*(q)\) on \(T\) containing \(p^{[3]}\);

Create bisector \(B(p, q)\);

Update list \(T\) so that it contains ...

\(V^*(q), C_{pq}, V^*(p), C_{pq}^+, V^*(q), \ldots\) in the place of \(V^*(q)\);

Insert intersections between \(C_{pq}^-\) and \(C_{pq}^+\) with neighboring boundaries into \(Q\);

case 2: \(p\) is an intersection

Let \(p\) be the intersection of boundaries \(C_{pq}\) and \(C_{qs}\);

Create the bisector \(B(q, s)\);

Update list \(T\) so that it contains \(C_{pq} = C_{qs}^+\) or \(C_{pq}^-\);

Delete from \(Q\) any intersections between \(C_{pq}\) and their neighbors;

Insert any intersections between \(C_{pq}\) and its neighbors into \(Q\);

---

\([3]\) If \(p\) is found on a boundary, the search returns the region on either side of the boundary.
Mark p as a vertex and as an endpoint of B(q, r), B(r, s), and B(q, s);

end

End.

Each insertion and deletion on the search tree T and the queue Q takes \(O(\log n)\) time [AHU74]. It is shown in [HNS89] that the event queue Q has at most \(O(n)\) event-points at any time. Moreover, the transformed Voronoi diagram is a planar graph with \(O(n)\) vertices. Therefore, the above algorithm takes \(O(n \log n)\) time and \(O(n)\) space in the worst case. The transformed Voronoi diagram can be converted to its actual diagram in linear time.
Chapter 3 Constructing the Voronoi Diagram with Two Sweeps

In this Chapter, we first introduce the DCEL (Doubly-Connected-Edge-Link) representation of a planar graph. Then in Section 2, a Partial Voronoi Graph (PVG) for a set of points in the plane is defined, and an $O(n \log n)$ time and $O(n)$ space Sweepline algorithm for constructing the Left-Right Partial Voronoi Graph (similarly Right-Left Partial Voronoi Graph) is described. In Section 3, an optimal $O(n \log n)$ time and $O(n)$ space algorithm is presented, which merges the two Partial Voronoi Graphs, i.e., LR-PVG and RL-PVG, in linear time. The algorithm uses a top-down non-recursive version of the Divide-and-Conquer technique.

1 The DCEL Representation of Planar Graph

The Doubly-Connected-Edge-Link (DCEL) is a well suited data structure for storing planar graphs embedded in the plane. In the DCEL structure of a planar graph, each edge consists of four information fields $V_1$, $V_2$, $F_1$, and $F_2$, and two pointer fields $P_1$, $P_2$. Therefore, the DCEL structure is easily implemented with six arrays with the same names, each consisting of $n$ cells, where $n$ is the number of edges in the corresponding planar graph (Figure 6 appears in [PS85]).

The meaning of these fields are interpreted as follows. The field $V_1$ contains the origin (or starter) of the edge and the field $V_2$ contains its terminus (or ender). Thus each edge gets an orientation. The fields $F_1$ and $F_2$ are the names of the faces which lie respectively on the left and on the right of the edge oriented from $V_1$ to $V_2$. The pointer $P_1$ (or $P_2$) points to the edge node containing the first edge encountered after edge ($V_1$, $V_2$) when one proceeds counterclockwise (reps. clockwise) around $V_1$ (resp. $V_2$).
But it is often the case that a planar graph $G = (V, E)$ is given in an edge-list form. In an edge-list form, if a vertex has more than one edge incident to it, these edges are simply stored in the order in which they appear as one proceeds counterclockwise around the vertex. It can be easily done in linear time to transform an edge-list representation of a planar graph into a DCEL representation.

2 The Partial-Voronoi Graphs (PVG)

**Definition 3.2.1:** A Left-Right Partial Voronoi Graph (or LR-PVG) for a set of point sites $S$ is the collection of LR-PVG regions for all the point sites in $S$. The LR-PVG region $LR(s)$ for a point $s$ in $S$ is defined as $LR(s) = \{ p \in \mathbb{R}^2 \mid d(p, s) \leq d(p, q), \forall q \in S \text{ such that } x(p) \geq x(s) \text{ and } x(p) \geq x(q) \}$. Similarly, a RL-PVG region for $t$ is defined as $RL(t) = \{ p \in \mathbb{R}^2 \mid d(p, t) \leq d(p, q), \forall q \in S \text{ such that } x(p) \leq x(s) \text{ and } x(p) \leq x(q) \}$.

**Definition 3.2.2:** A polygonal chain $C = q_1q_2...q_k$ is said to be monotone with respect to a straight line $l$ if the perpendicular projections of $q_1, q_2, ..., q_k$ onto $l$ are respectively $l(q_1), l(q_2), ..., l(q_k)$ and these projections are in the same order on $l$ as they are on $C$.  

21
Definition 3.2.3: A Voronoi circle at a Voronoi vertex p is the circle centered at p with its radius equal to the distance from p to any of the point sites whose p.

Definition 3.2.4: For point site s of S, the intersection of the sweepline passing through s and LR(s) is a vertical line segment called the vertical boundary of LR(s). The upper endpoint (lower endpoint resp.) of the vertical boundary is called the upper (lower resp.) corner of LR(s). Starting from the upper (lower resp.) corner of LR(s), the longest sequence of adjacent edges on the boundary of LR(s) which forms a monotone chain is called the upper boundary (lower boundary resp.) of the region LR(s). The edge on the upper (lower resp.) boundary of LR(s) containing the upper (lower resp.) corner is called the upper bound (lower bound resp.) of LR(s). The upper and lower boundaries for RL(s) are defined similarly.

Thus, the boundary of LR(s) (RL(s) resp.) consists of three parts: (1) the upper boundary; (2) the lower boundary; and (3) the vertical boundary.

There are three types of vertices in the LR-PVG (RL-PVG resp.) graph, i.e., (1) The upper and lower corners of LR-PVG (RL-PVG resp.) regions; (2) The intersections between bisectors; and (3) The given point sites in S. We shall call the second type as vertex of the LR-PVG (RL-PVG resp.) region and the third type site vertex of the LR-PVG (RL-PVG resp.) region.

Properties of the Partial Voronoi Graphs

In the following, we show some important geometric properties of the Partial Voronoi Graphs (PVG). Because of the similarity of LR-PVG with RL-PVG, we only state the properties of LR-PVG, as the properties of RL-PVG can be derived similarly.

Definition 3.2.5: A simple polygon P is star-shaped if there exists an internal point z of P such that for any point p of P the line segment pz lies entirely inside P, and z is
called the kernel of $P$.

**Lemma 3.2.1:** If there is a point $q$ which lies inside a region $LR(p)$ ($RL(p)$ respectively), then the line $pq$ lies completely inside $LR(p)$ ($RL(p)$ resp.).

![Diagram showing the proof for Lemma 3.2.1.]

Figure 7 The proof for lemma 3.2.1.

[proof:] Suppose there exists a point $s$ on the line $pq$ such that $s$ belongs to $LR(p')$ for some $p' 
eq p$ (see Figure 7). Then by the definition of the LR-PVG, $d(s, p') \leq d(s, p)$. By the triangle inequality, $d(q, p') \leq d(q, s) + d(s, p') \leq d(q, s) + d(s, p) = d(q, p)$, which implies that $q$ does not belong to $LR(p)$. But this contradicts the assumption that $q$ is inside $LR(p)$. □

Therefore, Lemma 3.2.1 shows that the boundary of each region $LR(p)$ is *star-shaped* with the point site $p$ as the kernel of $LR(p)$.

**Lemma 3.2.2:** The straight line dual $D^*(S)$ of a LR-PVG($S$) (or RL-PVG($S$)) graph is a planar graph embedded in the plane.
[proof:] To prove that $D^*(S)$ is a planar graph, we need to show that any two edges of $D^*(S)$ do not intersect each other except at their endpoints. By the definition of the dual $D^*(S)$, we know that each edge of $D^*(S)$ corresponds to an edge in LR-PVG(S) (RL-PVG(S) resp.). Let $C(p)$ be the set of points whose associated LR-PVG regions share a non-vertical edge with the polygon LR(p) for any point $p$ in $S$.

Assume $C(p) = \{q_1, q_2, ..., q_k\}, q_i \in S, (i = 1, 2, ..., k)$, then for each Voronoi edge on the boundary of $V(p)$, there is a unique $q_i \in C(p) (1 \leq i \leq k)$ such that $b(p, q_i)$ is an edge in $D^*(S)$. It suffices to show that no two such edges intersect. By the fact (see Lemma 3.2.1) that LR(p) is star-shaped, for any two points $s$ and $t$ on the boundary of LR(p), the line segment $\overline{sp}$ does not intersect $\overline{tp}$ except at the endpoint $p$. Therefore no two edges of $D^*(S)$ intersect each other except at their endpoints, i.e., the vertices of $D^*(S)$. Thus $D^*(S)$ is a planar graph with $n$ vertices embedded in the plane, where $n$ is the number of points in $S$. □

**Theorem 3.2.1:** A LR-PVG of a set of point sites $S$ has at most $(7n-6)$ edges, where $n$ is the number of points given in the set $S$.

[proof :] We construct the embedded graph $D^*(S)$ of LR-PVG(S) (see the proof of Lemma 3.2.2) in the plane such that each edge of the embedded graph corresponds to an edge in LR-PVG(S).

By Lemma 3.2.2, the dual $D^*(S)$ is a plane graph, thus the edges of $D^*(S)$ do not intersect each other except at their endpoints. Since $D^*(S)$ has $n$ vertices, by applying Euler's formula, $D^*(S)$ has at most $3n-6$ edges. Thus there are at most $3n-6$ non-vertical edges in LR-PVG. For each LR-PVG region, there is exactly one vertical boundary associated with it. Each site cuts its associated vertical boundary into two edges. Moreover, the upper or lower boundary of each LR-PVG region can only intersect
a vertical boundary at most once (hence cutting the vertical boundary into two parts).
Therefore the total number of edges in a LR-PVG graph of a set of $n$ point sites is at
most $(3n - 6) + 2n + 2n = 7n - 6$. \(\square\)

Lemma 3.2.3: Let $p$ be a site in $S$ and $L(p)$ be the sweepline at site $p$. Suppose for $q, q' \in S_{\text{left}}(p)$, bisector $b(q, q')$ intersects $L(p)$ at $v$. Then the line segment $pv$ lies completely within the vertical boundary of $LR(p)$ iff $d(p, v) \leq d(v, q)$.

[proof:] $\Rightarrow$ If $pv$ lies completely within boundary of $LR(p)$, then $v \in LR(p)$, by definition of a LR-PVG region, $d(v, p) \leq d(v, q)$ holds for any $q \in S_{\text{left}}(p)$ and $p \neq q$.

$\Leftarrow$ Since $v$ is the intersection point $\text{ints}(b(q, q'), L(p))$, $v$ has the shortest distance to $q$ (therefore to $q'$) than to any point sites in $S_{\text{left}}(p) \setminus \{q, q'\}$, i.e., $d(v, q) = d(v, q') \leq d(v, t)$ for $t \in S_{\text{left}}(p) \setminus \{q, q'\}$. But $d(v, p) \leq d(v, q)$, thus $d(v, p) \leq d(v, q) \leq d(v, t)$ for $t \in S_{\text{left}}(p) \setminus \{q, q'\}$, or $d(v, p) \leq d(v, t)$ for $t \in S_{\text{left}}(p)$. By definition $v \in LR(p)$, also we know that the sweepline $L(p)$ is at site $p$, therefore the segment $pv$ lies on the vertical boundary of the region $LR(p)$. \(\square\)

Constructing the Partial Voronoi Graphs

Let $S_{\text{left}}$ denote the set of points of $S$ that lie to the left of the current position of the sweep-line. Initially $S_{\text{left}}$ is empty. As the sweep-line sweeps to the right hand-side, $S_{\text{left}}$ grows one point at a time until there is no more event point or transition point to sweep. In a general position of the sweepline, the regions (edges, resp.) intersecting with the sweep-line are called active LR-PVG regions (edges resp.). The point sites whose associated regions are active are called active points.

We shall only describe the construction of LR-PVG, since the construction for RL-PVG is similar. For each bisector $b$, we associate a variable $X(b)$ with it so that $X(b)$ = the $x$-coordinate of its right end. An intersection between bisectors $b$ and $b'$, denoted as
Chapter 3 – Constructing the Voronoi Diagram with Two Sweeps

\text{ints}(b, b') \text{, is valid if } x(\text{ints}(b, b')) \leq \min(X(b), X(b')). \text{ Therefore by simply comparing the } x\text{-coordinates of the intersection } x(\text{ints}(b, b')) \text{ with the } \min(X(b), X(b')) \text{, we are able to tell whether or not an intersection point is valid, and hence would know whether it should be removed from the event queue } Q.\

Primitive Geometric Functions:

\text{SUCC}(q) \text{ returns the active object (bisector or PVG region) immediately above } q;

\text{PREC}(q) \text{ returns the active object (bisector or PVG region) immediately below } q;

\text{ints}(b, b') \text{ returns the intersection of straight line } b \text{ and straight line } b';

\text{upper}(p) \text{ returns the active edge on the upper boundary of the region } LR(p);

\text{lower}(p) \text{ returns the active edge on the lower boundary of the region } LR(p).

All these primitive operations can be easily done in constant time, if the DCEL data structures of LR-PVG (or RL-PVG) are respectively provided.

Initialization : A DCEL structure representing the dynamic structure of the construction of LR-PVG graph, it is an intermediate output of this algorithm. Initially the DCEL of the LR-PVG(S) is empty;

\( Q = \text{the event points sorted by their } x\text{-coordinates (intersections between bisectors may be inserted into the event queue during the sweep). Initially } Q \text{ contains all the point sites of } S. \)

\( T = \text{a balanced binary search tree which contains a set of bisectors in the vertical ordering in which they intersect the sweep line. Initially } T \text{ is an empty binary tree; } \)

Algorithm 1: COMPUTING THE LEFT-RIGHT PARTIAL VORONOI DIAGRAM

Begin

While \( \neg (Q \text{ is empty}) \) do

begin


Chapter 3 – Constructing the Voronoi Diagram with Two Sweeps

\[ p_1 \leftarrow \text{extract-min}(Q); \] //extract the first element from the queue \( Q \);

case 1: event-point : \( p_1 \)

determine the region \( LR(p_{\text{min}}) \) containing the current event point \( p_1 \);

//\( p_{\text{min}} \) is the nearest neighbor of \( p_1 \) among all the points in \( S_{\text{emb}}(p) \);

\[ q_1 \leftarrow q_1 \leftarrow p_{\text{min}}; \]

while \(-\langle b(p_1, q_1) \rangle \) intersects the line segment from \( p_1 \) to \( \text{ints}(\text{lower}(q_1), L(p_1)) \) and \( -\langle q_1 = \Lambda \rangle \) do

//forming the initial bisector of the lower boundary for \( LR(p_1) \);

begin

delete lower\( (q_1) \) from \( T \) and modify \( \text{DCEL} \) accordingly;

\[ q_1 := \text{PREC}(q_1); \]

end; //extending the region \( LR(p_1) \) downward;

If \( (q_1 \neq \Lambda) \) then //\( q_1 \) is the site whose partial Voronoi region contains the lower corner of \( LR(p_1) \)

begin

create bisector \( b(p_1, q_1) \) and insert it into \( T \);
compute the intersection of \( b(p_1, q_1) \) and \( L(p_1) \) (i.e., the lower corner)
and let it be \( w \);
insert \( b(p_1, q_1) \) into the \( \text{DCEL} \) of \( \text{LR-FVG} \);
insert the intersections between \( b(p_1, q_1) \) with its neighbors (if any) into \( Q \) and update the \( X \)-variables of \( b(p_1, q_1) \) and its neighbors

end;

while \(-\langle b(p_1, q_1) \rangle \) intersects the line segment from \( p_1 \) to \( \text{ints}(\text{upper}(q_1), L(p_1)) \) and \( -\langle q_1 = \Lambda \rangle \) do
Chapter 3 – Constructing the Voronoi Diagram with Two Sweeps

//forming the initial bisector of the upper boundary for LR(p₃);

begin
    delete upper(q₃) from T and modify DCEL accordingly;
    q₃ := SUCC(q₃);
end; //extending the region LR(p₃) upward;

if (q₃ ≠ ∧) then //q₃ is the site whose partial Voronoi region contains the upper corner of LR(p₃)

begin
    create bisector b(p₃, q₃) and insert it into T;
    compute the intersection of b(p₃, q₃) and L(p₃) (i.e., the upper corner)
    and let it be ν;
    insert b(p₃, q₃) into the DCEL of LR-PVG;
    insert the intersections between b(p₃, q₃) with its neighbors (if any) into
    Q and update the X-variables of b(p₃, q₃) and its neighbors
end;

insert that section of L(p₃) in between ω and ν into DCEL; //the vertical boundary
of LR(p₃); //Note ω and ν can be ∧;

case 2: intersection : p₃ = int(b(p₇, p₈), b(p₉, p₁₀))[5]

case 2.1: both b(p₇, p₈) and b(p₉, p₁₀) have been deleted:
    discard p₃;
    if PREC(b(p₇, p₈)) intersects SUCC(b(p₉, p₁₀))
    then insert the intersection into Q;

[5] In general, p₃ might be the intersection of more than two bisectors. The modification is rather simple. For detail see lemma 4.3.10.
Chapter 3 – Constructing the Voronoi Diagram with Two Sweeps

case 2.2: only one of \( b(p_j, p_k) \) and \( b(p_k, p_m) \) has been deleted:

discard \( p_i \);

set the \( X \) variable of the surviving bisector to \(+\infty\);

if the surviving bisector has valid intersections with its SUCCE or PRECE

then insert the one with smaller \( x \)-coordinates into \( Q \) and update the \( X \)-variables;


case 2.3: neither \( b(p_j, p_k) \) nor \( b(p_k, p_m) \) is deleted:

delete \( b(p_j, p_k) \) and \( b(p_k, p_m) \) from \( T \);

create bisector \( b(p_j, p_m) \) and insert it into \( T \);

\( X(b(p_j, p_m)) \leftarrow +\infty \) and modify the DCEL structure;

if \( b(p_j, p_m) \) intersects both PRECE \( (b(p_j, p_m)) \) and SUCCE \( (b(p_j, p_m)) \) then

choose the one with smaller \( x \)-coordinate and insert it into \( Q \);

update the \( X \)-variables of PRECE \( (b(p_j, p_m)) \) and SUCCE \( (b(p_j, p_m)) \)

endwhile

End.

About the Correctness of Algorithm 1

We shall show that Algorithm 1 correctly computes the Left-Right Partial Voronoi Graph in \( O(n \log n) \) time and \( O(n) \) space, where \( n \) is the number of points in \( S \).

Lemma 3.2.4 : There are at most \( 3n \) intersection event-points inserted into the queue \( Q \), and \( Q \) contains no more than \( 4n \) event-points at any time.

[Proof : ] During the sweep process, there are only two kinds of event points: point site of \( S \) and intersection point. If the current event point is a point site of \( S \), then at
most two new bisectors are inserted into the binary search tree \(T\), i.e., the upper and lower bound of the current event point. Obviously, the two bounds do not intersect each other, but they may intersect with their neighbors in the tree \(T\), i.e., the upper boundary may intersect with its upper neighbor in the tree, and the lower bound may intersect with its lower neighbor in the tree. Therefore at most two intersections are inserted into the queue \(Q\).

If the current event is an intersection, then in the worst case, two new intersection points are created. However, Algorithm 1 chooses only the one with smaller \(x\)-coordinate. Hence, at most one new intersection may be inserted into \(Q\).

Since at each intersection point, a point site is deactivated, there may be at most \(n\) intersection points. Therefore, at most \(3n\) intersection event-points are inserted into \(Q\), and \(Q\) contains no more than \(4n\) event-points at any time. \(\square\)

**Theorem 3.2.2 (Correctness of Algorithm 1):** Algorithm 1 correctly computes the Left-Right Partial Voronoi Graph (or LR-PVG) of a set of \(n\) points in the plane in \(O(n \log n)\) time and \(O(n)\) space.

[Proof :] We prove this theorem in two parts:

**Part 1 (Correctness)** We use an inductive proof on the number of point sites in \(S_{\text{left}}\).

*Base Case*: \(|S_{\text{left}}| = 1\), there is only the first point site, i.e., the site with the least \(x\)-coordinate. Then \(p_{\text{min}} = \Lambda\) which implies that \(q_1 = q_3 = \Lambda\). Hence, the entire \(L(p_1)\) is inserted into DCEL which makes the half plane to the right of the sweepline through \(p_1\) as \(LR(p_1)\). This is certainly consistent with the definition of \(LR(p_1)\).

*Induction*: Assume that the LR-PVG\((S_{\text{left}})\) is correctly computed by Algorithm 1 when \(|S_{\text{left}}| = (k-1)\). Now we consider the case where \(|S_{\text{left}}| = k\) \((k > 1)\), i.e., the point site \(p_k\).
Case 1: $p_k$ is a point site in $S$. Since the region containing $p_k$ is the LR-PVG region associated with site $p_m$, where $p_m \in S_{left}(p_k)$. By the inductive assumption, for any point $u \in LR(p_m)$, we have $d(u, p_m) \leq d(u, q)$ for any $q \in S_{left}(p_m)$.

Without loss of generality, we only consider the second while loop of case 1 in Algorithm 1. Initially $q_u := p_m$. If the intersection $\text{ints}(b(q_u, p_k), L(p_k))$ does not lie on the vertical interval from $p_k$ to the intersection of the upper boundary of $LR(p_m)$ with the sweepline $L(p_k)$, then the vertical boundary of $LR(p_k)$ can be extended upwards. This is due to Lemma 3.2.2. Therefore, the execution of the while loop does not terminate and the next active point site immediately above $q_u$ is used to replace $q_u$. The while loop is repeated until a $q_u$ is found such that the intersection $\text{ints}(b(q_u, p_k), L(p_k))$ lies on the vertical interval from $p_k$ to the intersection of the upper boundary of $LR(p_m)$ and the sweepline $L(p_k)$. Lemma 3.2.3 guarantees that the line segment from $\text{ints}(b(p_k, q_u), L(p_k))$ to $p_k$ lies completely inside the vertical boundary of $LR(p_k)$.

Case 2: $p_k$ is an intersection. Suppose $p_k$ is the intersection of two bisectors $b(p_i, p_j)$ and $b(p_j, p_m)$. In case 2, the validities of $b(p_i, p_j)$ and $b(p_j, p_m)$ are tested. (1) If the two parent (the two bisectors which form the intersection) bisectors are both valid, then a new bisector $b(p_i, p_m)$ is generated. This is guaranteed by the property that from a LR-PVG vertex $v$ there is one and only one bisector issuing from that vertex to the right of the sweepline at $v$; (2) If one of them is deleted then $p_k$ is discarded and the surviving one is left intact; (3) If both are deleted then $p_k$ is discarded and the existence of an intersection between the two neighboring bisectors of $b(p_i, p_j)$ and $b(p_j, p_m)$ is tested. If such an intersection exists, it is inserted into the event queue. Therefore, $LR(p_k)$ has been correctly computed.

Part 2 (Complexity) Now we analyse the performance of Algorithm 1. Since the searching
information is always maintained in the balanced binary search tree $T$, whenever an event point is reached by the sweepline, the LR-PVG region which contains the current event point can be located in $O(\log n)$ time. Once this region is located, we begin to construct the upper bound and the lower bound of the LR-PVG region associated with that event point site. The determinations of both bounds can be done in at most $O(k_i \log n)$ time, where $k_i$ is the number of active PVG boundary edges encountered (hence deleted from the search tree $T$) in the while loops. The time for inserting a newly generated intersection or deleting a deactivated intersection from $Q$ is $O(\log n)$. By Lemma 3.2.4, the total number of event-points in $Q$ is bounded by $4n$ at any time, and by Lemma 3.2.1 and Theorem 3.2.1, the LR-PVG graph is a planar graph with at most $O(n)$ edges. Therefore, the total time that Algorithm 1 needs to compute the correct PVG graph is bounded by $O(\sum_{i=1}^{n} k_i \log n + 4n \times O(\log n)) = O(n \log n)$. Also since the sizes of the search tree $T$ and event queue $Q$ are no more than $O(n)$ at any time, only $O(n)$ space is needed by Algorithm 1. □

3 Constructing the Nearest Neighbor Voronoi Diagram

With the LR-PVG and RL-PVG having been computed by Algorithm 1, and the assumption that we have pre-sorted $S$ by the $x$-coordinates and have divided $S$ approximately into two equal parts, denoted as $S_{left}$ and $S_{right}$, we could proceed to construct the nearest-neighbor Voronoi diagram with the following procedure. We shall use $B(S_{left}, S_{right})$ to denote the dividing curve of $S_{left}$ and $S_{right}$.

Main Procedure: COMPUTING VORONOI DIAGRAM

1. Construct the LR-PVG($S$) and RL-PVG($S$);
2. Divide $S$ into two subsets $S_{right}$ and $S_{left}$ of approximately equal size;
3. Construct the dividing curve $B(S_{left}, S_{right})$ based on the LR-PVG($S$) and RL-PVG($S$);
4. Repeat Step 2 and Step 3 for both $S_{\text{right}}$ and $S_{\text{left}}$ until each of them becomes a set with a single point;

The major task in the Main Procedure is to merge LR-PVG($S$) with RL-PVG($S$) to form the final Voronoi diagram for the given set $S$. Therefore we construct the dividing curve (also called *merge curve*) based on the information extracted from LR-PVG($S$) and RL-PVG($S$). In constructing the dividing curve $B(S_{\text{left}}, S_{\text{right}})$, the usual scanning procedure (CCW-CW scheme) can still be applied, but as we will explain, extra care needs to be taken.

Before considering Step 3 in the Main Procedure, let us review some properties of the dividing curve $B(S_{\text{left}}, S_{\text{right}})$.

**Lemma 3.3.1:** Any dividing chain $\delta$ separating $S_{\text{right}}$ from $S_{\text{left}}$ is a monotonic curve with respect to any vertical axis. In particular, $\delta$ does not self-intersect.

[proof:] Suppose $\delta$ is not monotonic with respect to vertical axis $y$. Without loss of generality, assume that $S_{\text{right}}$ and $S_{\text{left}}$ are separated by a vertical line. Let $\delta$ be a sequence of vertices $q_1, q_2, \ldots, q_k$. By definition of the monotonicity of a curve (see Definition 3.2.2), if we traverse $\delta$ in the direction of decreasing $y$-coordinates, we must eventually reach a point $q_i$ ($1 \leq i \leq k$) where $\delta$ turns upward. (see Figure 8).

Let edge $\overline{q_{i-1}q_i}$ be the bisector $b(p_1, p_2)$, and $\overline{q_iq_{i+1}}$ be the bisector $b(p_1, p_3)$.
Since $y(q_{i+1}) > y(q_i)$ and $y(q_{i-1}) > y(q_i)$, we have $x(p_3) \leq x(p_1) \leq x(p_2)$ or $x(p_3) \geq x(p_1) \geq x(p_2)$. However, by the shape of $\delta$, $p_2$ and $p_3$ belong to the same set, contrary to our assumption that $S_{right}$ and $S_{left}$ are separated by a vertical line. So, a vertex like $q_i$ does not exist and $B(S_{left}, S_{right})$ is monotone with respect to $y$-axis. □

Lemma 3.3.2: On the dividing chain $B(S_{left}, S_{right})$, there must exist one bisector $b(p_l, p_r)$ with $p_l \in S_{left}$ and $p_r \in S_{right}$ (we call this bisector the starter bisector in the tracing process) such that $b(p_l, p_r)$ intersects the line segment joining $t_l$ with $t_r$, where $t_l$ and $t_r$ are the right extreme point of $S_{left}$ and the left extreme point of $S_{right}$ respectively.

[Proof:] The dividing curve must intersect the line joining $t_r$ to $t_l$, otherwise $t_r$ and $t_l$ will lie in the same side of the dividing curve, but this contradicts the nature of the dividing curve. □

Lemma 3.3.3: If the bisector $b(p_l, p_r)$ intersects the common area of $LR(p_l)$ and $RL(p_r)$, then the section of $b(p_l, p_r)$ inside their common area of $LR(p_l)$ and $RL(p_r)$ must be a final Voronoi edge.

[proof:] By the definition of $LR(p_l)$ and $RL(p_r)$, for any point $p$ on the section of bisector $b(p_r, p_l)$ which is in the area $LR(p_l) \cap RL(p_r)$, $d(p, p_l) = d(p, p_r) \leq d(p, q)$ for
Chapter 3 – Constructing the Voronoi Diagram with Two Sweeps

all \( q \in S_{\text{left}}(p_1) \cup S_{\text{right}}(p_r) \). Also for any point site \( s \) within the vertical strip defined by \( p_1 \) and \( p_r \), i.e., \( x(p_1) < x(s) < x(p_r) \), we must have \( d(p, s) \geq d(p, p_1) (=d(p, p_r)) \), for otherwise, \( p \) must have been contained inside the region \( LR(s) \) or \( RL(s) \), but \( p \) is inside the common area of \( LR(p_1) \) and \( RL(p_r) \). Therefore, we have \( d(p, p_1) = d(p, p_r) \leq d(p, q) \) for all \( q \in S \), thus the section of \( b(p_1, p_r) \) in the common area \( LR(p_1) \cap RL(p_r) \) is an edge on the Voronoi polygons \( V(p_1) \) and \( V(p_r) \). \( \square \)

Our idea of constructing the dividing curve is based on \( LR-PVG(S) \) and \( RL-PVG(S) \). It consists of two steps: (1) find out the starter bisector; (2) trace the dividing curve from the starter bisector in two passes (one pass 'upward', the other pass 'downward'). We describe how to implement these two steps in details in the following.

The Pre-processings

We perform a pre-processing to locate for each vertex in \( LR-PVG(S) \) the region containing that vertex in \( RL-PVG(S) \), and for each vertex in \( RL-PVG(S) \) the region containing that vertex in \( LR-PVG(S) \), using the result of Kirkpatrick [Kir83] or Edelsbrunner et. al. [EGS84]. Each of these pre-processings can be done in \( O(n \log n) \) pre-processing time, and then any point location in the planar subdivision can be effected in \( O(\log n) \) time and \( O(n) \) space.

Finding the Starter Bisector

By Lemma 3.3.2, the dividing curve intersects the line \( \overline{t_1 t_r} \) once. Since the line \( \overline{t_1 t_r} \) is segmented into many segments by the boundaries of \( LR \) or \( RL \) regions. To determine at which point the dividing curve intersects the line \( \overline{t_1 t_r} \), we have to examine all the sections of \( \overline{t_1 t_r} \). For each section \( c \) of \( \overline{t_1 t_r} \), there is a site pair \( \{p_r, p_1\} \) associated with it such that \( c \) is inside the common area of \( LR(p_1) \) and \( RL(p_r) \). If the current bisector \( b(p_1, p_r) \) intersects the segment \( c \) of \( \overline{t_1 t_r} \) inside the common area of \( LR(p_1) \) and \( RL(p_r) \), then
by Lemma 3.3.3, we have found the "starter bisector", which is b(p_r, p_r). Otherwise, we consider the next section c^[5] of \overline{t_{1}t_{r}} and its associated site pair \{p_r, p_1\}, and conduct the same test. By Lemma 3.3.2, it is inevitable that there is a bisector which intersects \overline{t_{1}t_{r}}. Hence the starter bisector will eventually be found.

The correctness of the above procedure in finding the starter bisector follows directly from Lemma 3.3.3.

Tracing Out the Dividing Curve

It is easily seen that the dividing curve can be obtained as a sequence of perpendicular bisector segments, \gamma_1, \gamma_2, \ldots, \gamma_i, \ldots, \gamma_m, where each \gamma_i is bounded by two Voronoi vertices v_i and v_{i+1} and m is the number of edges on the dividing curve.

The tracing procedure starts from the 'starter bisector' computed by the procedure we have just described. Since there are two directions in which the starter bisector could extend, the tracing of the dividing curve consists of two passes which are essentially the same except in their orientations. We shall only illustrate one of the two passes, specifically, the pass which traces the dividing curve from the the starter bisector upwards (recall that the dividing curve is monotone with respect to the y-axis).

Initially, \gamma_1 is the starter bisector b(p_1, p_r), and v_1 is the intersection between \gamma_1 and t_{1}t_{r}. The tracing process halts when we reach the ender bisector, i.e., the bisector of the two endpoints of the upper supporting line between CH(S_{right}) and CH(S_{left}). Throughout this tracing process, we always maintain b(p_1, p_r) as the most recently constructed bisector segment of the dividing curve.

Inductively, suppose that \gamma_i, v_i, and the p_1, p_r at the i-th stage have all been defined. We trace this dividing chain up to the (i+1)-th stage by defining the correct \gamma_{i+1}, v_{i+1}, and

[5] Here, the term "next of c" can be defined as the neighboring section of c which has not been examined.
the \( p_i, p_r \) at the \((i+1)\)-th stage. At the \(i\)-th stage, let \( \gamma_i \) be a portion of the bisector \( b(p_r, p_l) \) emanating upwards from \( v_i \). We let \( p_L \) (\( p_R \) respectively) be the intersection of the boundary of \( LR(p_i) \) (\( RL(p_i) \) resp.) and the ray \( \gamma_i \). We define a \emph{dummy break point} \( v \) of \( \gamma_i \) to be the first of \( p_R \) and \( p_L \) encountered as one moves from \( v_i \) along the direction of \( \gamma_i \).

In general, when tracing the dividing curve, there are three possible cases:

1. The dummy break point \( v \) is a non-vertex point on a non-vertical boundary of a LR-PVG or RL-PVG region;
2. The dummy break point \( v \) is a vertex of an LR-PVG or RL-PVG region;
3. The dummy break point \( v \) is a non-vertex point on the vertical boundary of a LR-PVG or RL-PVG region.

In the following, we consider each of the above three cases separately. For clarity sake, we shall assume that \( v \) is on exactly one of LR-PVG and RL-PVG in each case.

**Case 1:** The dummy break point \( v \) is a non-vertex point on non-vertical boundary of a LR-PVG or RL-PVG region  
Due to symmetry, we shall only consider the case where the dummy break point \( v \) is a non-vertex point on a non-vertical boundary edge of \( LR(p_1) \). In this case, \( v \) is a Voronoi vertex of the final Voronoi diagram, so we let \( v_{i+1} := v \).

Let's consider the example given in Figure 9. In this example, \( S_{\text{left}} = \{p_1, p_2, p_3\} \) and \( S_{\text{right}} = \{p_4, p_5, p_6\} \). The tracing of the dividing curve starts from \( b(p_3, p_4) \), and breaks at \( b(p_1, p_3) \) and \( b(p_4, p_5) \) respectively, and finally reaches \( b(p_2, p_5) \), which is a bisector on the lower boundary of \( RL(p_2) \). If the next section of the dividing curve (\( \gamma_4 \) in this case) goes along \( b(p_1, p_2) \), then the dividing curve will not separate \( p_1 \) from \( p_2 \), which contradicts the definition of the dividing curve.
Chapter 3 - Constructing the Voronoi Diagram with Two Sweeps

Figure 9 Shall \( \gamma_4 \) be defined as the portion of the ray emanating from \( v_4 \) in the direction of the bisector \( b(p_1, p_2) \) ?

Therefore, to define \( \gamma_{i+1} \), a test is needed to see whether or not the bisector defined according to the conventional CW-CCW scanning scheme separates \( p_1 \in S_{\text{left}} \) from \( p_r \in S_{\text{right}} \). We call this test a separation test.

In doing the separation test, we have two possible cases: the test succeeds and the test fails. In the following, we shall consider each of them separately.

(1.1) The separation test succeeds: Suppose \( v \) is on an edge \( b(p, p_I) \) of \( LR(p_i) \). Then \( \gamma_{i+1} \) is defined as the ray emanating upwards from \( v_{i+1} (= v) \) along the bisector \( b(p, p_r) \).

Lemma 3.3.4: Suppose the dummy break point of \( \gamma_i \) is on a non-vertical boundary of \( LR(p_i) \) at edge \( b(p, p_I) \), where \( p \in S_{\text{left}} \), then \( v_{i+1} = \text{ints}(\gamma_I, b(p, p_I)) \) and \( \gamma_{i+1} \) is the ray starting from \( v_{i+1} \) upwards along \( b(p, p_r) \).
[proof:] Since \( \text{ints}(\gamma_i, b(p, p_i)) \in LR(p_i) \) and \( \text{ints}(\gamma_i, b(p, p_i)) \in RL(p_r) \), so \( d(\text{ints}(\gamma_i, b(p, p_i)), p_i) = d(\text{ints}(\gamma_i, b(p, p_i)), p_r) = d(\text{ints}(\gamma_i, b(p, p_i)), p) \leq d(\text{ints}(\gamma_i, b(p, p_i)), q) \) for any other \( q \in S \), thus \( \text{ints}(\gamma_i, b(p, p_i)) \) is a Voronoi vertex on the dividing curve, i.e., \( v_{i+1} := \text{ints}(\gamma_i, b(p, p_i)) \). But \( p \in S_{\text{left}} \). Therefore, the ray emanating upwards from \( v_{i+1} \) along the bisector \( b(p, p_r) \) separates \( p \in S_{\text{left}} \) from \( p_r \in S_{\text{right}} \), and hence is \( \gamma_{i+1} \). □

![Diagram](image)

Figure 10 Four possible escapings.

(1.2) The separation test fails: Then \( \gamma_{i+1} \) is defined as the ray emanating upwards from \( v_{i+1} \) (= v) along the boundary bisector containing \( v_{i+1} \) (this bisector edge is called an escaping edge). There are four types of escaping, as illustrated in Figure 10: (a) The
Chapter 3 – Constructing the Voronoi Diagram with Two Sweeps

escaping occurs on the lower boundary of a RL-PVG region; (b) The escaping occurs on the upper boundary of a RL-PVG region; (c) The escaping occurs on the lower boundary of a LR-PVG region; (d) The escaping occurs on the upper boundary of a LR-PVG region. (c) and (d) are symmetric to (a) and (b) respectively. No matter which type of escaping happens, if $\gamma_{i+1}$ ends (i.e. the position of $v_{i+2}$) at a vertex of the LR-PVG region (not a corner), then we define $\gamma_{i+2}$ upwardly as one of the bisectors incident to $v_{i+2}$ such that it separates a point site in $S_{\text{left}}$ from another point site in $S_{\text{right}}$. (see (II) of Figure 11). We repeat this process until a $\gamma_{i+k}$ ($k \geq 1$) is defined such that $v_{i+k+1}$ is a corner of a PVG region, then we have returned to the case (1) or (2)

![Diagram](image)

Figure 11 (I). The test succeeds; (II) The test fails.
Chapter 3 – Constructing the Voronoi Diagram with Two Sweeps

Lemma 3.3.5: Suppose $\gamma_i = b(p_i, p_i)$ intersects the non-vertical boundary of $LR(p)$ at edge $b(p_i, p)$. If $p \in S_{right}$, then $v_{i+1} = \text{int}(\gamma_i, b(p_i, p))$ and $\gamma_{i+1}$ is the ray emanating upwards from $v_{i+1}$ along the bisector $b(p_i, p)$.

[proof:] See (c) of Figure 10. Similar to lemma 3.3.4, the intersection point $\text{int}(\gamma_i, b(p, p_i))$ is a Voronoi vertex on the dividing curve, and $v_{i+1} = \text{int}(\gamma_i, b(p, p_i))$. Also, since the bisector $b(p_i, p)$ separates $p \in S_{right}$ from $p_i \in S_{left}$ and $v_{i+1}$ is contained in this bisector, therefore $\gamma_{i+1}$ is the ray starting from $v_{i+1}$ pointing upwards along the bisector of $LR(p)$ which contains $v_{i+1}$. □

Similar arguments hold for the cases (a), (b) and (d) of Figure 11.

Case 2: The dummy break point $v$ is a vertex of a LR-PVG(S) or RL-PVG(S) region. Let $b(p, p_i)$ be the first bisector one encounters when one rotates around $v$ counterclockwise (resp. clockwise). If $p \in S_{left}$, then $\gamma_{i+1}$ is the bisector $b(p, p_i)$. Otherwise, starting from $b(p, p_i)$, we rotate around $v$ counterclockwise (resp. clockwise) (see Figure 12[5]), and scan all the LR-PVG edges incident to $v$ until an edge $e$ on the boundary of a LR-PVG region whose associated site belongs to $S_{left}$ (resp. $S_{right}$) is encountered. Then $\gamma_{i+1}$ is defined to be the ray emanating upwards from $v_{i+1}$ along the direction of $e'$ (see Figure 12), where $e'$ is the immediate predecessor of $e$ in the scanning process. We refer this scan around $v$ as an escape scan.

[5] In the figure, $\gamma_i$ itself is an escape edge. But in general, this does not have to be true.
When \( p \in S_{\text{left}} \), the case is essentially the same as case (1). For the case where \( p \in S_{\text{right}} \), we have the following lemma.

**Lemma 3.3.6**: Suppose the dummy break point \( v \) is a vertex of LR-PVG(S) (or RL-PVG(S)) and let \( b(p, p_t) \) be as defined above such that \( p \in S_{\text{right}} \). By rotating around \( v \) counterclockwise (resp. clockwise) and scanning all the LR-PVG (resp. RL-PVG) edges incident upon \( v \) until an edge \( e \) is found such that \( e \) is an edge on the boundary of a LR-PVG (resp. RL-PVG) region whose associated site belongs to \( S_{\text{left}} \) (resp. \( S_{\text{right}} \)), then \( v_{i+1} = v \) and \( \gamma_{i+1} \) is the ray emanating upwards from \( v_{i+1} \) along the bisector \( e' \), where \( e' \) is the immediate predecessor of \( e \) in the scanning process.

[proof:] We only consider the case when the dummy break point \( v \) is a vertex of a LR-PVG region (the case for RL-PVG is similar). Since \( p, p_r \in S_{\text{right}} \), \( b(p, p_r) \) cannot be \( \gamma_{i+1} \). Let \( e' = b(s, t) \) be the edge found by the escape scan method. If \( s \in S_{\text{right}} \) and \( t \in S_{\text{right}} \), then the rotation would not have stopped at \( b(s, t) \). If \( s \in S_{\text{left}} \) and \( t \in S_{\text{left}} \), then the rotation would have stopped before reaching \( b(s, t) \). Therefore, for the bisector \( b(s, t) \), we must have either \( s \in S_{\text{left}}, t \in S_{\text{right}} \) or \( s \in S_{\text{right}}, t \in S_{\text{left}} \). Since \( b(s, t) \) has
$v_{i+1} (=v)$ as an endpoint, and $v_{i+1}$ is a vertex on the dividing curve, therefore, $\gamma_{i+1}$ is the ray emanating upwards from $v_{i+1}$ along the bisector $b(s, t)$. □

Case 3: The dummy break point $v$ is a non-vertex point on the vertical boundary of a LR-PVG or RL-PVG region. In this case, $v$ is an intersection of $\gamma_i$ with the vertical boundary of LR($p_l$) (resp. RL($p_r$)). Before considering solving this case, we give the following preliminaries:

**Lemma 3.3.7** : If $\gamma_i = b(p_l, p_r)$ intersects the vertical boundary of a LR($p_l$) region (resp. RL($p_r$) region) not at the corners, then there must exist a section of $\gamma_i$ such that this section is the initial portion of the boundary of RL($p_l$) (resp. LR($p_r$)).

[Proof :] See Figure 13. Let $v$ be the point at which $\gamma_i$ meets the vertical boundary of LR($p_l$). First, we show that there must be a non-empty section $\overline{vq}$ of $\gamma_i$ with $q$ falling into the leftside of L($p_l$). Suppose to the contrary that there does not exist such a section of $\gamma_i$, i.e., $v = q$, then there exists a point site $p \in S_{\text{left}}(p_l)$ such that any point on the ray emanating upwards from $v$ along $\gamma_i$ is closer to $p$ than to $p_l$. Thus the upper corner of LR($p_l$) can not be above $v$, but this contradicts our assumption that $\gamma_i$ intersects L($p_l$) not at a corner. □
We assume that in the corresponding RL(p_l), \( \overline{vq} \) is an edge on the boundary of RL(p_l), then we have the following lemma:

**Lemma 3.3.8:** If \( q \in LR(s) \), then the next vertex (the actual \( v_{i+1} \)) of the dividing curve exists within the open line segment \( \overline{vq} \) if and only if \( d(q, s) < d(q, p_l) (=d(q, p_r)) \).

[proof :]

\( \Rightarrow \) If \( d(q, s) < d(q, p_l) (=d(q, p_r)) \), then there must exist a vertex of the dividing chain within the open line \( \overline{vq} \), this is because b(s, p_l) separates q to the half-plane which contains s.

\( \Leftarrow \) We shall prove that if \( d(q, s) \geq d(q, p_l) \), then the dividing chain does not branch out throughout the open line \( \overline{vq} \). We prove this by considering the following three cases (see Figure 14):
Chapter 3 - Constructing the Voronoi Diagram with Two Sweeps

Figure 14 Proof for lemma 3.3.8.

1. For any point site $p$ with $x(p) > x(p_1)$, by the definition of RL-PVG region, $d(q', p_1) = d(q', p) < d(q', p)$ for any interior point $q'$ on $\overline{pq}$.

2. For any point site $t$ with $x(q) < x(t) < x(p_1)$, the bisector $b(t, p_1)$ does not intersect the open line $\overline{pq}$. The reason is that should $b(t, p_1)$ intersect the open line segment $\overline{pq}$, then in the RL-PVG graph, the edge $\overline{pq}$ would not have been able to survive up to $q$.

3. For any point site $p$ inside the region $x(p) < x(q)$, the bisector $b(p, p_1)$ does not intersect the open line $\overline{pq}$. Suppose to the contrary that there exists a point $p$ in the region $x(p) < x(q)$ such that $b(p, p_1)$ intersects the open line $\overline{pq}$. Then we must have $d(p, q) < d(q, p_1)$. By the given condition $d(q, s) \geq d(q, p_1)$, we have $d(p, q) < d(q, p_1) \leq d(q, s)$. This implies that $q \notin LR(s)$, a contradiction.
Therefore, if \( d(q, s) \geq d(q, p_i) \), then the dividing chain does not branch out throughout the open line \( \overline{vq} \), where \( q \) is an endpoint of the edge \( \overline{v_iq} \) on the boundary of \( RL(p_i) \) and \( v \) is the point at which \( \gamma_i \) meets \( L(p_i) \). □

The above lemma shows that when the current section of the dividing chain reaches a vertical boundary of an LR-PVG region (similarly, an RL-PVG region) at point \( v \), by simply testing the condition in Lemma 3.3.8, we are able to tell whether or not the dividing chain branches out within the open line segment \( \overline{vq} \).

If the test fails, then include the entire line segment \( \overline{vq} \) into the dividing chain (we call this a \textit{jump} and the edge \( \overline{vq} \) a \textit{jump edge}. The test is called a \textit{jump test} and the condition in the test is called the \textit{jumping condition}), and assign \( q \) to \( v_{i+1} \), and hence extend \( \gamma_i \) to \( q \). To find the orientation of \( \gamma_{i+1} \), we rotate around \( v_{i+1} (=q) \) counterclockwise and scan all the RL-PVG edges incident upon \( v_{i+1} \) until an edge \( e \) incident upon \( v_{i+1} \) is found such that \( e \) is on the boundary of a RL-PVG region whose associated site belongs to \( S_{left} \), then \( \gamma_{i+1} \) is defined to be the ray pointing upwards from \( v_{i+1} \) along the bisector \( e \).

If the jump test succeeds, we locate the vertex \( v_{i+1} \) within the open line \( \overline{vq} \) (This is referred to as the dividing curve branching within the open line \( \overline{vq} \)). The point at which the dividing chain branches is called a \textit{branch point}. To find the branch point \( v_{i+1} \), we use the procedure \texttt{branch_point_location} which will be given later on. After the branch point (the actual \( v_{i+1} \)) is found, \( \gamma_{i+1} \) is readily defined (See Figure 15).
Figure 15 When \( v \) is an intersection of \( \gamma_1 \) with a vertical boundary.

In the following, we describe only the procedure for locating the branch point when \( \gamma_1 \) intersects a vertical boundary of LR(\( p_1 \)) at point \( v \). If \( \gamma_1 \) intersects a vertical boundary of RL(\( p_1 \)) at point \( v \), the procedure is similar.

**PROCEDURE** branch_point_location;

**Input:** \( \gamma_1 = b(p_1, p_2), L(p_1); \)

**Output:** the endpoint \( v_{n+1} \) of \( \gamma_1 \) which is within the open line segment \( vq \) and \( \gamma_{n+1}; \)

**Begin**

\( v \leftarrow \text{ints}(\gamma_1, L(p_1)); // \text{initial value of} \ v \)

Locate the region LR(\( s \)) which contains \( q \) in its interior;

1. **Repeat**
2. Locate the region LR(\( \alpha \)) which contains a non-null section of \( vq \) and \( v; \)
3. Let \( v' \) be the intersection of \( vq \) and the upper boundary of LR(\( \alpha \)); If there is no such intersection then let \( v' \) be \( q; \)
4. If \( d(v', \alpha) \leq d(v', p_l) \) then \( v_{i+1} \leftarrow v' \); \( \gamma_{i+1} \leftarrow b(\alpha, p_l) \); EXIT
   
   else \( v \leftarrow v' \); //extend \( \gamma_i \) to \( v \)

5. Until \( (\alpha = s) \); //where LR(s) contains \( q \)

6. End

Theorem 3.3.3: The above procedure computes the correct vertex \( v_{i+1} \) and \( \gamma_{i+1} \) on the dividing curve.

[proof:] In the procedure branch_point_location, line 2 locates the site \( \alpha \in S_{\text{left}}(p_l) \) such that LR(\( \alpha \)) contains \( v \) and a non-null section of \( \overline{vq} \) and computes the intersection point \( v' \). Note that \( v \) or \( v' \) are the two endpoints of the line segment \( \overline{vq} \cap \text{LR}(\alpha) \). Line 3 tests if \( v' \) is closer to \( \alpha \) than to \( p_l \). If it is, then by Lemma 3.3.8, the actual \( v_{i+1} \) must exist within the open line segment \( \overline{vq} \), which is actually \( v' \) and \( \gamma_{i+1} \) is the ray emanating from \( v_{i+1} \) along \( b(\alpha, p_l) \); otherwise, \( \gamma_i \) survives up to the intersection point \( v' \) and \( \gamma_i \) is thus extended to \( v' \). From Lemma 3.3.8, we know that if \( d(q, s) < d(q, p_l) \) the branch point must exist within the open line \( \overline{vq} \). Since when \( \alpha = s \), line 2 will ensure that \( v' = q \). Therefore, the execution of the repeat loop will terminate, i.e., the termination condition \( (\alpha = s) \) is correct. 

About the Correctness of the Tracing Procedure

Theorem 3.3.4: The tracing procedure computes the correct dividing curve.

[proof:] We prove this theorem using induction on the number of vertices and bisector segments constructed for the dividing curve \( B(S_{\text{left}}, S_{\text{right}}) \).

Base Case: As we see in the initialization of the tracing procedure, \( \gamma_1 = b(p_l, p_r) \), \( v_1 \) is the intersection of \( \gamma_1 \) with \( \overline{t_l t_r} \), where \( t_r \) and \( t_l \) are respectively the left-most point of
Chapter 3 – Constructing the Voronoi Diagram with Two Sweeps

$S_{\text{right}}$ and the right-most point of $S_{\text{left}}$, and $b(p_l, p_r)$ is the starter bisector. From Lemma 3.3.4, we know that $\gamma_1$ and $v_1$ are both correctly defined.

**Assumption:** Assume that the dividing curve has been correctly constructed upto the $i$-th stage, i.e., the bisector segment and vertex sequence $(v_1, \gamma_1, v_2, \gamma_2, ..., v_i, \gamma_i)$ is a portion of the dividing curve $B(S_{\text{left}}, S_{\text{right}})$. Now, we prove that the tracing procedure we have introduced computes the correct $v_{i+1}$ and $\gamma_{i+1}$.

There are at most three cases that $\gamma_i$ intersects the boundary of a LR-PVG or RL-PVG region, i.e., (1) $\gamma_i$ meets a non-vertical boundary at a non-vertex point; (2) $\gamma_i$ meets a non-vertical boundary at a vertex; (3) $\gamma_i$ meets a vertical boundary at a non-vertex point. But Lemma 3.3.4, Lemma 3.3.5 and Lemma 3.3.6 show that the tracing procedure computes the correct $v_{i+1}$ and $\gamma_{i+1}$ in case (1). Lemma 3.3.7 and Theorem 3.3.3 respectively show that the tracing procedure computes the correct $v_{i+1}$ and $\gamma_{i+1}$ in case (2) and case (3).

Therefore, the sequence $(v_1, \gamma_1, v_2, \gamma_2, ..., v_i, \gamma_i, v_{i+1}, \gamma_{i+1})$ are correctly computed. □

**About the Worst Case**

In the worst case, as illustrated in Figure 16, it is possible that the dividing chain will
cross some LR or RL region boundaries back and forth several times (as those boundary edges marked by 'x' in Figure 16).

If we test those marked edges each time when they are intercepted by the current section of the dividing chain, then in the worst case, the time complexity of even a single pass of the tracing procedure might be $O(n^2)$. Thus we need to find a way to avoid repeatedly testing those boundaries which have already been tested. Before considering how to solve this worst case problem, we shall give the following property for a LR-PVG or RL-PVG vertex.

**Lemma 3.3.9:** For any LR-PVG (resp. RL-PVG) vertex $v$, if there are more than one bisectors on the left (resp. right) of the vertical line $l$ passing through $v$, then there is one and only one bisector issuing from $v$ to the right (resp. to the left) of the vertical line through $v$. 50
Chapter 3 – Constructing the Voronoi Diagram with Two Sweeps

[proof:] Similar to [LP84]. □

Definition 3.3.1: For each vertex \( v \) of the LR-PVG, we call each of the edges incident to \( v \) and which lies to the left side of the vertical line \( l \) passing through \( v \) a parent edge of the only one edge incident to \( v \) and which lies to the right side the vertical line \( l \).

Now, let's consider how to avoid the repeated tests when the worst case happens. Each time when a jump test fails, a branch point has to be located. We begin with marking each of those LR-PVG boundary edges intercepted by \( \gamma \) with an 'x' sign except that the first and the last of them are each marked with a 'b' (see the example in Figure 16). We then continue the marking process using the following rules: (1) If all the parent edges of an edge are marked with 'x', then the edge is also marked with an 'x' (recall that Lemma 3.3.9 guarantees that there is only one bisector issuing from the intersection.

Figure 17 Illustration for lemma 3.3.9.
to the right); (II) If one of the parent edges of an edge is marked with a 'b', then that edge is also marked with a 'b'.

By completing the marking in this way, we have found two sequences of edges, each of which consists of the edges marked with 'b'. All these edges marked with 'b' form a front skeleton of \( \gamma_i \). The area enclosed by \( \gamma_i \) and its corresponding front skeleton is referred as the front hole of \( \gamma_i \), denoted by \( \text{FH}(\gamma_i) \).

Also we form another skeleton on the other side of \( \gamma_i \) by marking out the boundary of \( \text{V}(p_i) \) to the left of \( \gamma_i \). The area enclosed by this skeleton and \( \gamma_i \) is referred as the back hole of \( \gamma_i \). (see the shaded area in the example of Figure 16).

We now show how to compute the skeletons for the front hole and back hole separately.

**Computing the Front Hole** We shall describe the procedure for computing the front hole by the following algorithm:

PROCEDURE FRONT-HOLE \((\gamma_i)\):

1. Mark the first and the last boundary edge which intersect \( \gamma_i \) with label 'b';
2. Mark each of the edges intersecting \( \gamma_i \) between the two boundary edges (already marked with 'b') with a label "x";
3. Continue the marking by the following rules:
   a. For any vertex \( v \), if all the non-parent edges incident upon \( v \) are marked with 'x', then this parent edge incident upon \( v \) is marked with 'x';
   b. For any vertex \( v \), if exactly one of the non-parent edges incident upon \( v \) is marked with 'b', then the parent edge at is marked with 'b'; If two of the non-parent edges incident upon \( v \) are both marked with a 'b', then do nothing;

52
c. If an edge $e$ marked with 'b' intersects the vertical boundary of a LR-PVG region and $e$ is in the upper (lower, resp.) boundary of the front hole, then mark the lower (upper resp.) part of the vertical boundary intercepted by $e$ with a 'b';

d. If an edge $e$ marked with a 'x' intersects the vertical boundary of a LR-PVG region, then mark each of the two parts of the vertical boundary intercepted by $e$ with a 'b';

e. Repeat Steps 3.a through 3.d until there are no more edges to mark;

The examples for the rules 3.a through 3.d are given in the Figure 18 (a) through Figure 18 (d) respectively. Figure 19 illustrates the result of marking the edges in a front-hole using the above procedure.

Figure 18 The rules of conducting the marking.
Lemma 3.3.10: Let $\gamma_i = b(p_1, p_r)$. For any $p \in S_{left}$, if $LR(p)$ has some part of its region extended into the interior of the front hole of $\gamma_i$, then no point in that part of $LR(p)$ is included in the Voronoi cell $V(p)$ in the final Voronoi diagram.

[Proof:] Suppose to the contrary that there is a point $\alpha$ of $LR(p)$ in the interior of the front hole of $\gamma_i$ and $\alpha$ belongs to $V(p)$. Then by the definition of Voronoi polygon, the line segment $\overline{ap}$ must not intersect any Voronoi edge. However, $\overline{ap}$ intersects $\gamma_i$ which is certainly a Voronoi edge. A contradiction. $\Box$

The above lemma implies that there is no need to consider all those LR-PVG edges inside the front hole of $\gamma_i$ in constructing $\gamma_k$ and $v_{k+1}$ for any $k > i$. Therefore, whenever a jumping test fails, we mark every LR-PVG boundary edge intercepted (hence considered
Chapter 3 – Constructing the Voronoi Diagram with Two Sweeps

when locating the $v_{i+1}$ by $\gamma_i$ based on the two rules outlined above. Then we have a
front hole bounded by the front skeleton and the current $\gamma_i$.

Lemma 3.3.11: The procedure FRONT-HOLE marks the skeleton of the front hole.

[Proof:] The correctness of this lemma follows from Lemma 3.3.9 and 3.3.10. □

While doing the marking, whenever an edge $e$ is marked, we also record the
information about the fact that $e$ has been considered in locating $v_{i+1}$. This can be
easily done by adding one more field for each edge in the DCEL of LR-PVG, which
takes at most $O(n)$ extra space. Each of these fields is treated as a flag of the associated
dge. If the edge is considered (thus marked) in a branch point location process, we set
the flag of that edge to “true”, otherwise, we set it to “false”.

In subsequent stages, when a jumping test performed on a $\gamma_k$ (for some $k > i$)
fails, we call the branch point location procedure to locate $v_{k+1}$ as usual. However, in
running the procedure branch_point_location, if we encounter an edge marked
with 'b', we know that we are entering a front hole, say FH($\gamma_i$). To avoid reconsidering
those edges marked inside the front hole, we temporarily suspend the execution of the
branch_point_location procedure and traverse from $v_k$ along the sequence of
edges marked with 'b' to find the $\gamma_i$ (see Figure 20), and skip the front hole via
$\gamma_i$ and then exit along the other boundary of the front skeleton. Execution of the
branch_point_location procedure is resumed.
The description of skipping over the front hole $FH(\gamma_i)$ needs elaboration.

We shall call a point which we use as the starting point in executing procedure branch_point_location for an edge $e$ the entry point of $e$. (As we shall see in the following, the entry point of $e$ needs not be a point on the edge $e$). We shall denote the edge originated from $v \_k$ on which $v \_k+1$ (and hence $\gamma_k$) is determined by $v \_k \_q \_k$. Suppose $v \_k \_q \_k$ is an edge in RL-PVG which has to undergo the jumping test. We let the entry point of $v \_k \_q \_k$ be $v \_k$ if $v \_k$ is not inside any front hole and be the entry point of $\gamma_k-1$ otherwise.

1. $v \_k \_q \_k$ fails the jumping test: Then $q \_k$ cannot be inside any front hole or back hole. Starting from the entry point of $v \_k \_q \_k$, we execute the procedure branch_point_location to determine the branch point $v \_k+1$. Any front hole
encountered is skipped in the way as described in the preceding paragraph. We then
determine $\overline{v_{k+1}q_{k+1}}$.

b. $\overline{v_kq_k}$ passes the jumping test: We let $v_{k+1}$ be $q_k$, and then determine $\overline{v_{k+1}q_{k+1}}$ using
the escape scan method.

In general, suppose in the course of constructing the dividing curve, we encounter
an edge $\overline{v_kq}$ for which the jumping test has to be performed. Then by using the afore-
mentioned technique, we will determine a chain of edges $\{\overline{v_iq_i} | k \leq i \leq l\}$ such that $q_k = q_i$; $v_{i+1} = q_i$ for $k \leq i < l$ and all of the $\overline{v_iq_i}$ ($k \leq i < l$) pass the jumping test except $\overline{v_lq_l}$. Two cases are possible for $\overline{v_lq_l}$: (i) $\overline{v_lq_l}$ is the parent edge of $\overline{v_{l-1}v_l}$; (ii) $\overline{v_lq_l}$ is
a sibling edge of $\overline{v_{l-1}v_l}$.

In the first case, it is easily verified that $\overline{v_lq_l} = \overline{v_lv_{l+1}} = \gamma_l$ for $k \leq i \leq l$. Since step
(a) determines the branch point $v_{l+1}$ on $\overline{v_lq_l}$, we thus have $\overline{v_lv_{l+1}} = \gamma_l$. Therefore, the
chain of edges $\overline{v_lv_{l+1}}$ for $k \leq i \leq l$ extends the dividing curve from $v_k$ to $v_{l+1}$. Moreover,
in determining $v_{l+1}$, step (a) uses the entry point of $\overline{v_lq_l}$ (which is a $v_j$ for $j < l$ such
that $v_j$ is not inside any front or back hole) to skip over all the front holes that intersect
the chain $\{\overline{v_lv_{l+1}} | j \leq i < l\}$. After determining the break point $v_{l+1}$, all the unmarked
LR-PVG edges which intersect the chain are truncated and the marking method is used
to create a front and back hole for the chain.

In the second case, $q_l = v_{l+1}$ must hold true. Hence, $\overline{v_lq_l} = \overline{v_lv_{l+1}} = \gamma_l$ for $k \leq
i \leq l$, which implies that the chain of edges $\{\overline{v_lv_{l+1}} | k \leq i \leq l\}$ extends the dividing
curve from $v_k$ to $v_{l+1}$

**Computing the Back Hole** The purpose of forming the back hole is to avoid reconsider-
ating those LR-PVG boundary edges which intersect $\gamma_l$, when constructing the dividing
curve in the downward pass.
Chapter 3 – Constructing the Voronoi Diagram with Two Sweeps

The construction of the skeleton of the back hole starts from \( v_{i+1} \) — an endpoint of \( \gamma_i \). The skeleton is constructed as a sequence of bisector segments, \( \beta_1, \beta_2, ..., \beta_m \), where each \( \beta_i \) is on a bisector one of whose associated site is \( p_i \). We shall denote the end vertices of \( \beta_i \) by \( u_i \) and \( u_{i+1} \).

Initially \( u_1 = v_{i+1} \), \( \beta_1 \) is on the ray emanating from \( u_1 \) along \( b(p_i, p) \) in the orientation such that \( p_i \) lies to the left of \( \beta_1 \), where \( p \) is a site such that \( LR(p) \) contains \( u_1 \). (Note: In case \( u_1 \) is on the boundary of \( LR(p) \), then \( p \) is the site which is closest to \( p_i \) among all the sites that have \( u_1 \) on their boundaries).

In general, for any \( \beta_j \) (\( j \geq 1 \)), let \( \beta_j \) be on the bisector \( b(p_i, p) \). Then in finding the other endpoint \( u_{j+1} \) of \( \beta_j \), there are four possible cases:

1. \( b(p_i, p) \) intersects the non-vertical boundary of a LR-PVG region;
2. \( b(p_i, p) \) intersects the vertical boundary of a LR-PVG or RL-PVG region;
3. \( b(p_i, p) \) is along an edge in a RL-PVG region;
4. \( b(p_i, p) \) intersects the non-vertical boundary of a RL-PVG region;

In the following, we shall consider each case separately.

Let \( \beta_j \) be a section of the bisector \( b(p_i, p) \).

1. If \( b(p_i, p) \) first intersects a non-vertical boundary of a LR(p) region at edge \( b(p, p^*) \), then we define \( u_{j+1} := \text{ints}(b(p_i, p), b(p, p^*)) \) and \( \beta_{j+1} \) to be on the ray emanating from \( u_{j+1} \) along \( b(p_i, p^*) \) in the orientation such that \( p_i \) lies on the left side of \( \beta_{j+1} \).

2. If \( b(p_i, p) \) first meets a vertical boundary of a LR(p). Let \( v \) be the intersection point \( \text{ints}(b(p_i, p), L(p^*)) \). Then we perform a jumping test on the edge \( \overline{vq} \) (Note: this edge is an extension of the edge \( \overline{u_jv} \) along \( b(p_i, p) \)). If the test succeeds, then \( u_{j+1} := q \) and the
bisector which contains $\beta_{j+1}$ is the clockwise successor of $b(p_1, p)$ around $q$. Otherwise, $u_{j+1}$ is the branch point and $\beta_{j+1}$ is a section of $b(p_1, p^*)$ where $LR(p^*)$ contains $q$.

3. If $b(p_1, p)$ is along an edge $e$ on a RL-PVG region, then $u_{j+1}$ is defined as the other endvertex of $e$ which is not $u_j$, and $\beta_{j+1}$ is the non-parent edge incident on $u_{j+1}$ and is of the form $b(p_1, p^*)$ from some $p^* \in S$.

4. If $b(p_1, p)$ first intersects a non-vertical boundary of a RL$(p^*)$ region at an edge $e$, then we define $u_{j+1} := \text{ints}(b(p_1, p), e)$ and $\beta_{j+1}$ to be on the ray emanating from $u_{j+1}$ along the edge $e$ in the orientation such that $p_1$ lies on the left side of $\beta_{j+1}$.

Let $B_1$ and $B_2$ be respectively the last and the first LR-PVG edge which intersect $\gamma_i$, we mark each of them with a 'b'. Then the process of forming and marking the back hole of $\gamma_i$ is illustrated in the following procedure:

**PROCEDURE BACK-HOLE**

**Input:** $\gamma_i$ as the bisector $b(p_1, p)$; $v_{i+1}$ lies in $LR(p)$;

**Output:** A sequence of bisector sections: $\beta_1, \beta_2, ..., \beta_m, ...$, which form the skeleton of the back hole for $\gamma_i$, where $\beta_j$ is bounded by two vertices $u_j$ and $u_{j+1}$;

**Begin**

$u_1 := v_{i+1}; \ j := 1;$

$\beta_1$ — the ray emanating from $u_1$ along $b(p_1, p)$ in the orientation such that $p_1$ lies in the left side of $\beta_1$;

while $B_2$ has not been encountered do

**case 1:** $b(p_1, p)$ intersects the boundary edge $b(p, p^*)$ of $LR(p^*)$:

$u_{j+1} := \text{ints}(b(p_1, p), b(p, p^*)); \ p := p^*;$

$\beta_{j+1}$ := the ray emanating from $u_{j+1}$ along $b(p_1, p^*)$ in the orientation such that $p_1$ lies on the left side of $\beta_{j+1}$; /* define the next section of the skeleton */

59
Chapter 3 – Constructing the Voronoi Diagram with Two Sweeps

case 2: \( b(p_i, p) \) intersects a vertical boundary of \( LR(p^*) \):

\[
v := \text{ints}(b(p_i, p), L(p^*));
\]

Call \text{branch_point_location} to find \( u_{j+1} \); If the jumping test succeeds then define \( \beta_{j+1} \) as the ray originated from \( u_{j+1} \) and contains the clockwise successor of \( \beta_j \) in \( RL-PVG \); otherwise \( \beta_{j+1} \) is a section of \( b(p_i, p^*) \);

case 3: \( b(p_i, p) \) is along an edge \( e \) in \( RL-PVG \):

\[
u_{j+1} := \text{the other endvertex of } e; \]

\( \beta_{j+1} \) is the non-parent edge incident upon \( u_{j+1} \) and is of form \( b(p_i, p^*) \) for some \( p^* S \);

case 4: \( b(p_i, p) \) intersects the boundary edge \( e \) of \( RL(p^*) \):

\[
u_{j+1} := \text{ints}(b(p_i, p), e); p := p^*; \]

\( \beta_{j+1} := \text{the ray emanating from } u_{j+1} \text{ along } e \text{ in the orientation such that } p_i \text{ lies on the left side of } \beta_{j+1};// \text{define the next section of the skeleton} \]

\[
j := j + 1
\]

endwhile

End

Lemma 3.3.12: The above procedure computes the skeleton of the back hole.

[proof:] By induction on the number of segments of the back hole skeleton constructed by the above procedure.

Base Case: \( \beta_1 \) is determined to be on the ray emanating from \( v_{i+1} \) along \( b(p_i, p) \) in the orientation such that \( p_i \) lies to the left-hand side of \( \beta_1 \). Clearly, each of the points in \( LR(p) \) which is to the left of \( \gamma_1 \) has one of \( p \) and \( p_i \) as its closest endpoint. \( \beta_1 \) is thus correctly constructed. \( u_1 \) is obviously correct.

60
**Inductive Assumption:** Supposing that \( \beta_j \) is originated from \( u_j \) along bisector \( b(p_1, p) \) and that \( u_j, b(p_1, p) \) have both been correctly computed. We want to prove that \( u_{j+1} \) and the bisector containing \( \beta_{j+1} \) are also correctly computed by procedure \textsc{back-hole}.

**Case 1:** \( b(p_1, p) \) intersects a boundary edge \( b(p, p^*) \): in this case the intersection \( \text{ints}(\beta_j, b(p, p^*)) \) is equidistant to \( p_1, p \) and \( p^* \). Moreover, each of the points in \( \text{LR}(p^*) \) which lies to the left-hand side of \( \gamma_1 \) has one of \( p^* \) and \( p_1 \) as its closest site. Therefore, \( u_{j+1} := \text{ints}(\beta_j, b(p, p^*)) \) and that \( b(p_1, p^*) \) is considered as the bisector containing \( \beta_{j+1} \) are both correct.

**Case 2:** \( b(p_1, p) \) first meets the vertical boundary of a \( \text{LR}(p) \): Then the correctness of procedure \texttt{branch_point_location} ensures that \( u_{j+1} \) and the bisector contain \( \beta_{j+1} \) are both correctly computed.

**Case 3:** \( b(p_1, p) \) is along an edge in \( \text{RL-PVG} \): Clearly, one of the endpoint of that edge which is different form \( u_j \) has to be the endpoint \( u_{j+1} \) and \( \beta_j \) is the section of that edge bounded by \( u_{j+1} \) and \( u_j \). Moreover, as the clockwise successor of \( \beta_j \) around \( u_{j+1} \) is the bisector which has \( p_1 \) as an associated site, \( \beta_j \) is correctly determined.

**Case 4:** \( b(p_1, p) \) intersects the non-vertical boundary of a region \( \text{RL}(p^*) \) at edge \( e \): Clearly, \( u_{j+1} \) must be the intersection of \( e \) and \( b(p_1, p^*) \). Moreover, as \( e \) is on the bisector \( b(p_1, p^*) \) for some \( p^* \in S \), \( b(p_1, p^*) \) is correctly chosen as the bisector containing \( \beta_{j+1} \).

**Lemma 3.3.13:** Let \( \text{LR}(p) \) be any \( \text{LR-PVG} \) region that intersects \( \gamma_1 \). Then there exists a \( \beta_j \) such that \( \beta_j \) is a section of the bisector \( b(p, p_1) \).

[Proof:] The proof is based on the fact that in constructing the skeleton of the back-hole for \( \gamma_1 \), each \( \beta_j \) is a bisector of which one of the associated site is \( p_1 \) and that no \( \beta_j \) constructed in Case 3 can cut across any such \( \text{LR}(p) \) region.
curve after $\gamma_i$, we will never enter the back hole of $\gamma_i$ as well as all those LR(p) regions which intersect $\gamma_i$. This implies that all those edges in LR-PVG which are examined in determining $v_{i+1}$ for $\gamma_i$ will never be examined again in subsequent stages, i.e., each of them is examined a constant number of times.

The Algorithm for Computing the Dividing Curve

Before we construct the dividing curve, we do a planar location for each vertex of RL-PVG(S) in the planar subdivision induced by LR-PVG(S). Similarly, we do a planar location for each vertex of LR-PVG(S) in the planar subdivision induced by RL-PVG(S). These two pre-processing steps can be done in $O(n \log n)$ time and $O(n)$ space using the algorithm of Kirkpatrick [Kir83] or Edelsbrunner [EGS]. The dividing curve is then constructed as the follows:

**Algorithm 2: CONSTRUCTING THE DIVIDING CURVE**

Pre-processing: A planar location for each vertex of LR-PVG (resp. RL-PVG) graph in RL-PVG (resp. LR-PVG);

Input: LR-PVG(S) and RL-PVG(S) represented in DCELs;

output: A sequence of vertices, $\{v_1, v_2, \ldots\}$, forming the dividing curve $B(S_{\text{left}}, S_{\text{right}})$;

Begin

\[ t_i \leftarrow \text{left-most}(S_{\text{left}}); \quad t_r \leftarrow \text{right-most}(S_{\text{right}}); \]

determine the region RL$(p_i)$ containing $t_i$ in its interior;

\[ p_i \leftarrow t_i; \]

c $\leftarrow$ the first section of $t_i \overline{t_r}$ lying inside RL$(p_i) \cap RL(p_r)$; // thus c lies in the common area of LR$(p_i)$ and RL$(p_r)$

//Finding Starter Bisector:

while (c $\neq \Lambda$) and $\rightarrow$ (b($p_i$, $p_r$) intersects c inside the area RL$(p_i) \cap LR(p_r)$) do

62
Chapter 3 – Constructing the Voronoi Diagram with Two Sweeps

begin

\( p_r \leftarrow \text{NEXT}_r(p_r); // \text{RL}(p_r) \) is the next RL region intersecting \( \overline{t_1 t_r} \)

\( p_l \leftarrow \text{NEXT}_l(p_l); // \text{LR}(p_l) \) is the next LR region intersecting \( \overline{t_1 t_l} \)

\( c \leftarrow \text{NEXT}(c); // c \) is the next section of \( \overline{t_1 t_r} \)

end; // after this loop, the \( b(p_1, p_r) \) is the starter bisector

\( v_1 := \text{ints}(b(p_1, p_r), \overline{t_1 t_r}); i := 0; // \text{initially} v_1 \) is the starter point of the dividing curve

// Tracing Procedure :

repeat

1. \( i := i + 1; \)

2. \( d_L := \text{the first edge on LR}(p_l); // \text{in CCW} \)

3. \( d_R := \text{the first edge on RL}(p_l); // \text{in CW} \)

4. while \( (b(p_1, p_r) \cap d_L) = \emptyset \) do \( d_L := \text{NEXT}^1(d_L); \)

\( // \text{NEXT}^1(d_L) \) returns the next edge of the current polygon if scanning in CCW;

5. while \( (b(p_1, p_r) \cap d_R) = \emptyset \) do \( d_R := \text{NEXT}^2(d_R); \)

\( // \text{NEXT}^2(d_R) \) returns the next edge of the current polygon if scanning in CW;

6. case 1: if \( \gamma_i \) first intersects a non-vertical boundary edge at a non-vertex point then

   case 1.1: If \( \gamma_i \) first intersects \( d_R = b(p_1, p^* \) for some \( p^* \in S; \)

   \( // d_R \) is an edge in the upper or lower boundary of \( \text{RL}(p_1); \)

   If the separation test succeeds then

   \( v_{i+1} := \text{ints}(\gamma_i, d_R); \)

   create new bisector \( b(p_1, p^* \) and insert into DCEL structure;

   \( p_r \leftarrow p^* \)

   else // an escape occurs

   \( v_{i+1} := \text{ints}(\gamma_i, d_R); \)

63
Chapter 3 – Constructing the Voronoi Diagram with Two Sweeps

\[ p_1 \leftarrow p^*; \]

\[ \gamma_{i+1} \text{ is defined to be the ray originated from } v_{i+1} \text{ along } b(p_1, p_1); \]

**case 1.2:** If \( \gamma_1 \) first intersects \( d_L = b(p^*, p_1) \) for some \( p^* \in S_{k+t} \):

**If** the separation test succeeds then

//\( d_L \) is an edge in the upper or lower boundary of LR(\( p_1 \));

create new bisector \( b(p^*, p_1) \) and insert into DCEL structure;

\[ p_1 \leftarrow p^*; \]

\[ v_{i+1} \leftarrow \text{ints}(b(p_1, p_1), d_L); \]

else //an escape occurs

\[ v_{i+1} \leftarrow \text{ints}(b(p_1, p_1), d_R); \]

\[ p_r \leftarrow p^*; \]

\[ \gamma_{i+1} \text{ is defined to be the ray originated from } v_{i+1} \text{ along } b(p_1, p_1); \]

**case 2:** if \( \gamma_1 \) first intersects \( d_L \) (or \( d_R \)) at vertex \( v \) then:

\[ v_{i+1} \leftarrow v; \]

\[ \gamma_{i+1} \text{ is determined by the escape scan method}; \]

**case 3:** if \( \gamma_1 \) first intersects a vertical boundary then

**case 3.1:** \( \gamma_1 \) first intersects \( L(p_1) \) first;

//\( L(p_1) \) is the vertical boundary of LR(\( p_1 \));

\[ v \leftarrow \text{ints}(b(p_1, p_1), L(p_1)); \text{/the dummy break point} \]

Let \( q \) be the other endpoint of the edge \( b(p_1, p_1) \) in \( RL(p_1) \);

locate the site \( s \) such that \( LR(s) \) contains \( q \);

//this can be done with the information from the pre-processing;

If \( d(q, s) < d(q, p_1) \) then //test the jumping condition

64
Chapter 3 – Constructing the Voronoi Diagram with Two Sweeps

begin

Call branch-point-location to compute \( v_{i+1} \);

// the modified version

Call FRONT-HOLE;

Call BACK-HOLE

drop

else \( v_{i+1} := q \); \( \gamma_{i+1} \) is defined using the escape scan

method; // Due to Lemma 3.3.8, \( \gamma_i \) jumps to \( q \)

case 3.2 : \( \gamma_i \) first intersects \( L(p_i) \):

// \( L(p_i) \) is the vertical boundary of \( RL(p_i) \);

\( v \leftarrow \text{ints}(b(p_i, p), L(p_i)); // the dummy break point \)

Let \( q \) be the other endpoint of the edge \( b(p_i, p) \) in \( LR(p_i) \);

locate the site \( s \) such that \( RL(s) \) contains \( q \);

// this can be done with the information from the pre-processing;

If \( d(q, s) < d(q, p_i) \) then // test the jumping condition

begin

Call branch-point-location to compute \( v_{i+1} \);

// the modified version

Call FRONT-HOLE;

Call BACK-HOLE

drop

else \( v_{i+1} := q \); \( \gamma_{i+1} \) is defined using the escape scan

method; // Due to Lemma 3.3.8, \( \gamma_i \) jumps to \( q \)

65
until (neither LR-PVG nor RL-PVG intersects \( b(p_r, p_l) \));

End.

Performance Analysis of Algorithm 2

The Pre-processing Sorting the points of \( S \) lexicographically with many known optimal sorting algorithms can be done in \( O(n \log n) \) time and \( O(n) \) space [AHU74]. Performing the planar-point location can also be done in \( O(n \log n) \) time and \( O(n) \) space [Kir83].

The Tracing Procedure In finding the starter bisector, the line segment \( tr_t \) is tested section by section. with the DCEL data structures of LR-PVG(S) and RL-PVG(S), it takes only \( O(s) \) \( (O(t) \) resp.) time to find \( p_1 \), \( p_r \) resp.) of the next section based on the previously tested one, where \( s \) \( (t \) resp.) is the size of the polygon LR\( (p_1) \) \( (RL(p_r) \) resp.). It takes only constant time to test if \( b(p_1, p_r) \) intersects \( t_1 \overline{t_r} \). By the monotonicity of the LR-PVG or RL-PVG boundaries, we know that once a LR-PVG or RL-PVG edge is tested, it will not be tested again. Therefore, the time complexity of finding the starter bisector is bounded by the total number of edges contained in LR-PVG(S) and RL-PVG(S), i.e., \( O(n) \), where \( n \) is the size of \( S \).

Starting from the starter bisector \( b(p_1, p_r) \), the constructing of the dividing curve begins (again, we only consider the upward pass). In the Tracing Procedure of Algorithm 2, to find the edge \( d_L \) (resp. \( d_R \)) of LR\( (p_1) \) (resp. RL\( (p_r) \)) such that \( d_L \) (resp. \( d_R \)) first intersects \( b(p_1, p_r) \), we scan LR\( (p_1) \) (resp. RL\( (p_r) \)) in the counterclockwise (clockwise resp.) direction and test the edge one at a time for intersection. If there is no intersection found with an edge or if there is an intersection and this intersection is not chosen as the new vertex on the dividing curve, then we charge the time for the test to that edge itself which \( O(1) \) time. Therefore, to find the first intersection of \( b(p_1, p_r) \) and LR\( (p_1) \)
or $\text{RL}(p_r)$ takes at most $O(s+t)$ time, where $s$ and $t$ are the number of edges contained in $\text{LR}(p_l)$ and $\text{RL}(p_r)$ respectively.

After the two intersections are found, a comparison is made. The one which is closer to the intersection of the starter bisector and $\overline{t_1t_r}$ is chosen as the dummy break point.

(1) If the dummy break point $v$ is on an non-vertical boundary, then we take $v$ as the next vertex on the dividing curve, and we charge the test for the intersection with this edge to the new vertex; (2) If the dummy break point $v$ is a vertex of the LR-PVG or RL-PVG, to define the next section of the dividing curve, we use the escape scan method, i.e., rotating around $v$ and scanning all the RL-PVG (or LR-PVG) edges incident to $v$ in CCW (CW resp.) direction until an edge $e$ is found such that $e$ is on the boundary of a RL-PVG (or LR-PVG resp.) region associated with a site of $S_{left}$ (or $S_{right}$ resp.). We charge the test for $e$ to the next section of the dividing curve. Also we charge the test for each edge scanned to the edge itself; (3) If the dummy break point $v$ is on a vertical boundary of a LR-PVG (or RL-PVG) region, we do the jumping test: if the test succeeds, the next vertex on the dividing curve is found, and we charge the time for this jumping test to the newly found vertex; otherwise, we use the branch-point-location to locate the next vertex of the dividing curve. We charge the time for the marking process (both in the front hole and back hole marking) to the marked edges themselves.

In finding the endpoint $v_{k+1}$ of $\gamma_k$, if we encounter the front hole $FH(\gamma_i)$ for some ($i < k$), then starting from $v_k$, we traverse along the boundary of $FH(\gamma_i)$ until we find the intersection of $FH(\gamma_i)$ and $\gamma_k$. We charge each advance of the traverse to the degree of the vertex traversed, and the test for intersection to the edge itself.

In general, each edge of the LR-PVG or RL-PVG is charged at most four times in the process of constructing the Voronoi diagram.
Therefore, the Tracing Procedure totally takes $O(4 \times |\mathcal{E}| + \sum_{p \in V} d(p) + M)$ time at most, where $|\mathcal{E}|$ is the total number of edges in LR-PVG and RL-PVG; $V$ is the set of all LP-PVG and RL-PVG vertices; $d(p)$ is the degree of vertex $p$, and $M$ is the total number of edges in the final Voronoi diagram. By Lemma 3.2.1, $|\mathcal{E}| = O(n)$, and by Lemma 2.2.1, $M = O(n)$. Therefore, $\sum_{p \in V} d(p) = 2 \times |\mathcal{E}| = O(n)$. Hence the Tracing Procedure takes $O(n)$ time.

By the Main Procedure, we repeat computing each dividing curve with a top-down non-recursive method. Since both LR-PVG(S) and RL-PVG(S) are planar graphs with $O(n)$ edges and $O(n)$ vertices. The number of iterations is $O(\log n)$. As each iteration takes $O(n)$ time, the Voronoi diagram of a set of $n$ points in the plane can be computed in $O(n \log n)$ time with $O(n)$ space using the technique of two sweeps.
Chapter 4 Conclusions and Open Questions

We presented an optimal algorithm for computing the nearest neighbor Voronoi diagram for a set of \( n \) points in the plane using the plane-sweep method. The algorithm conceptually consists of two steps; the first step is to compute the Left-Right Partial Voronoi Graph (LR-PVG) and the Right-Left Partial Voronoi Graph (RL-PVG) using the sweepline method described in Algorithm 1. The second step (Algorithm 2) is to merge LR-PVG and RL-PVG together to form the final Voronoi diagram. In doing this merging process, we use a top-down no-recursive version of the divide-and-conquer technique. It is shown in this thesis that the first step takes at most \( O(n \log n) \) time and \( O(n) \) space. Therefore, the algorithm takes at most \( O(n \log n) \) time and \( O(n) \) space.

There are several open problems which are worth investigating. Perhaps the most promising one is using the sweepline method to construct the Voronoi diagram for general geometric objects.

Another open question is to extend this algorithm to the case when the metric of distance is other than Euclidean, such as \( L_p \) with \( 1 \leq p \leq \infty \), or convex polygonal distance functions. For example, when each point in the given point set is associated with a weight, the Voronoi diagram is the so called weighted Voronoi diagram. Conceptually, the extension of this algorithm to the weighted Voronoi diagram is not difficult. The generalization of this algorithm to other metrics of distance is hoped to be addressed.

A final open question involves the numerical instability of the algorithm. This problem has been noticed for many geometric algorithms. For example, two almost parallel (but finally intersect) bisectors might be treated as parallel, thus bounded regions might be treated as unbounded regions. These numerical errors can lead to the topological
changes of the actual Voronoi diagram, and hence might make the algorithm fail to compute the correct Voronoi diagram.
Bibliography


Appendix A

This appendix contains an example which describes the process of constructing the transformed Voronoi diagram of a set $S$ of eight points in the plane, where $S = \{p_1, p_2, \ldots, p_8\}$. For convenience, we sweep the given point set from the left to the right.

Figure 21 when the sweepline is at $p_1$. 

76
Figure 22 when the sweepline is at \( p_1 \).

Figure 23 when the sweepline is at \( p_4 \).
Figure 24 when the sweepline is at $p_i$.  

Figure 25 when the sweepline is at $p_j$.  

78
Figure 26 when the sweepline is at \( p_4 \).

Figure 27 when the sweepline is at \( p_7 \).
Figure 28 when the sweepline is at pa.
Appendix B

This appendix contains an example which describes the process of constructing the LR-PVG graph of a set $S$ of nine points in the plane, where $S = \{p_1, p_2, \ldots, p_9\}$.

Figure 29 when the sweepline is at $p_1$. 
Figure 30 when the sweepline is at \( p_1 \).

Figure 31 when the sweepline is at \( p_2 \).
Figure 32 when the sweepline is at $p_4$.

Figure 33 when the sweepline is at $p_5$. 
Figure 34 when the sweepline is at $p_4$.

Figure 35 when the sweepline is at $p_7$.  

84
Figure 36 when the sweepline is at $p_5$.

Figure 37 when the sweepline is at $p_5$. 
Figure 38 The final LR-PVG(S) graph.
Subject Index

1-dimensional weighted Voronoi diagram .... 4

A
abstract Voronoi diagram ................. 2
average time complexity .................. 1

B
back hole .................................. 52
back hole .................................. 57
bisecting Jordan curves .................. 2
branch point ................................ 46

c
constrained (multiplicative) weighted Voronoi
diagram ................................. 5
convex distance function .................. 2
CW-CCW scanning scheme ................. 2

D
d-dimensional and presented an algorithm

which is optimal when d is even ........... 4
DCEL ...................................... 20
divide-and-conquer ......................... 1, 2, 12
dividing curve ............................ 12
dummy break point ......................... 37, 67

E
entry point ................................ 56
escape scan ............................... 41, 67
escaping ................................. 39
Euclidean metric ........................ 2
Euclidean space .......................... 2
event-point schedule ...................... 13

F
front hole ................................ 52
front skeleton ............................ 52

G
general position assumption ............... 1
generalizations of the Voronoi diagram .... 1
tmetric transformation ...................... 14

H
horizontal merging ......................... 6

I
incremental technique ..................... 1
iterative greedy method 1
<table>
<thead>
<tr>
<th>J</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>jump</td>
<td>46</td>
</tr>
<tr>
<td>jump test</td>
<td>46</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>K</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>k-th order Voronoi polygon</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>L</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$L_1$-metric</td>
<td>2</td>
</tr>
<tr>
<td>Laguerre metric</td>
<td>3</td>
</tr>
<tr>
<td>Left-Right Partial Voronoi Graph</td>
<td>20, 21</td>
</tr>
<tr>
<td>$L_p$-metric</td>
<td>2</td>
</tr>
<tr>
<td>LR-PVG region</td>
<td>21</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>M</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Manhattan distance metrics</td>
<td>2</td>
</tr>
<tr>
<td>minimum spanning tree</td>
<td>5</td>
</tr>
<tr>
<td>monotone with respect to a straight line</td>
<td>21</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>N</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>nearest neighbor Voronoi diagram</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Partial Voronoi Graph</td>
<td>20</td>
</tr>
<tr>
<td>path-connected</td>
<td>2</td>
</tr>
<tr>
<td>plane-sweep</td>
<td>1</td>
</tr>
<tr>
<td>Primitive Geometric Functions</td>
<td>26</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Right-Left Partial Voronoi Graph</td>
<td>20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>scanning scheme</td>
<td>11, 12</td>
</tr>
<tr>
<td>separation test</td>
<td>38</td>
</tr>
<tr>
<td>spokes</td>
<td>5</td>
</tr>
<tr>
<td>standard median finding algorithm</td>
<td>10</td>
</tr>
<tr>
<td>star-shaped</td>
<td>24</td>
</tr>
<tr>
<td>starter bisector</td>
<td>34, 66</td>
</tr>
<tr>
<td>stater set</td>
<td>5</td>
</tr>
<tr>
<td>straight line dual</td>
<td>10</td>
</tr>
<tr>
<td>supporting lines</td>
<td>11</td>
</tr>
<tr>
<td>sweepline method</td>
<td>13</td>
</tr>
<tr>
<td>sweepline status</td>
<td>13</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>T</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>the farthest neighbor Voronoi diagram</td>
<td>3</td>
</tr>
<tr>
<td>the numerical instability</td>
<td>69</td>
</tr>
<tr>
<td>The straight line dual</td>
<td>23</td>
</tr>
<tr>
<td>the straight line visible distance</td>
<td>5</td>
</tr>
<tr>
<td>top-down non-recursive</td>
<td>20</td>
</tr>
<tr>
<td>Tracing Procedure</td>
<td>63</td>
</tr>
<tr>
<td>transition points</td>
<td>13</td>
</tr>
<tr>
<td>V</td>
<td>W</td>
</tr>
<tr>
<td>--------------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>vertical merging</td>
<td>weighted Voronoi diagram</td>
</tr>
<tr>
<td>Voronoi cell</td>
<td></td>
</tr>
<tr>
<td>Voronoi circle</td>
<td>window-contour merging</td>
</tr>
</tbody>
</table>

Voronoi polygon ...................... 1

W

89
Author Index

A

Aurenhammer ....................... 3, 4

C

C.A. Wang and Y.H. Tsin .............. 4

Chew ................................. 4

Chew and Drysdale .................... 2

D

D. T. Lee .............................. 1, 11

Drysdale .............................. 5

Drysdale and D. T. Lee ............... 5

E

Edelsbrunner and Seidel .............. 3

F

Fortune ............................... 1, 13

H

Hwang ................................. 2

K

Kirkpatrick ........................... 5, 62

Klein .................................. 2

L

Lee and Lin ............................ 4

Lee and Wong ........................... 2

P

Preparata and Hong .................. 11

S

Shamos ............................... 11

Shamos and Hoey ..................... 1

Sharir ................................. 5

W

Wang and Schubert ................... 4

Y

Yap ..................................... 6
Vita Auctoris

Sichao Wang was born in 1964 in Shaanxi Province, The People's Republic of China. In 1986, he graduated from Fudan University with a Bachelor of Computer Science. In May 1991 he completed his Master of Computer Science in the School of Computer Science at the University of Windsor.