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.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

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

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. 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 photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, tests publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30.

Canada

The University of Alberta

A New Generalized Voronoi Diagram In The Plane

by

CAO AN WANG

A thesis
submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree
of Doctor of Philosophy

Department of Computing Science

Edmonton, Alberta
Spring, 1988

ISBN 0-315-42964-X

THE UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR: CAO AN WANG

TITLE OF THESIS: A New Generalized Voronoi Diagram In The Plane

DEGREE FOR WHICH THIS THESIS WAS PRESENTED: Doctor of Philosophy

YEAR THIS DEGREE GRANTED: 1988

(Signed) C. A. Wang

Permanent Address:

Dated March 4, 1988

THE UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research, for acceptance, a thesis entitled **A New Generalized Voronoi Diagram In The Plane** submitted by **CAO AN WANG** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy**

L K Schubert

Co-Supervisor : Len Schubert

Barry Joe

Co-Supervisor : Barry Joe

Joseph O'Rourke

Ronald L. Rivest

Date March 4, 1988

To My Wife and Parents

ABSTRACT

In this thesis, a new generalized Voronoi diagram in the plane, called a *bounded Voronoi diagram*, is defined. This generalization is one of the natural extensions of the previously existing Voronoi diagrams in the plane. Three important cases of bounded Voronoi diagrams are discussed. They are:

(1) The bounded Voronoi diagram for a monotone chain.

(2) The bounded Voronoi diagram for a simple polygon.

(3) The bounded Voronoi diagram for a set of non-crossing line segments.

Algorithms for constructing these bounded Voronoi diagrams are presented. All of them are optimal.

Bounded Voronoi diagrams are powerful tools which help to efficiently solve many geometric problems.

ACKNOWLEDGEMENTS

I wish to express my sincere gratitude to my co-supervisors, Drs. *Len Schubert* and *Barry Joe*, for their invaluable help in my thesis. They spent a lot of time, even their own spare time, to read through the thesis sentence by sentence.

I would like to thank Drs. *Joe Culberson* and *Andy Liu* for their carefully reading and suggestions, especially on Chapters 1 to 4.

I particularly appreciate the supports from my supervisory committee on many occasions.

I would like to express my appreciation to Dr. *Raimund Seidel* for his many helpful suggestions on the thesis.

I would like to thank Drs. *Stanley Cabay* and *Peter Tsin* for their friendly advice and moral support.

I would like to thank my classmate *Ambrish Mathur* for reading the early version of Chapter 5.

Finally, without my wife *Weifong* and my daughter *Xin*'s full understanding and inspiration, and without my parents *Paling* and *Siuching*'s encouragement, my PhD program would never have been completed.

The research was supported by graduate assistantships provided by the Department of Computing Science of the University of Alberta, by NSERC Operating Grant A8818 (L.K.Schubert), and by NSERC operating Grant A9186 (B.Joe).

Table of Contents

List of Figures

Chapter 1

Introduction

Although the advent of geometry can be traced back several thousand years, computational geometry is still a young sub-discipline of the design and analysis of algorithms. Computational geometry was not systematically studied until the appearance of the dissertation of M. Shamos in 1978. Shamos showed that the available knowledge in classical geometry does not directly provide efficient methods for solving many problems. He suggested that the main task of computational geometry is to design efficient algorithms (if possible, optimal algorithms) for solving geometric problems.

The problems of computational geometry, besides being theoretically interesting, arise in areas such as computer graphics (for example, hidden line and hidden surface elimination problem), pattern recognition (for example, finding the diameter and the convex hull of a set of points), VLSI (for example, reporting the intersections of a set of line segments), robotics (for example, constructing the Voronoi diagram of a set of polygonal objects and finding the shortest path among obstacles), image processing (for example, finding the contour of a set of pixels), and operations research (for example, finding the Euclidean minimum spanning tree of a set of points).

In the following, we first give a brief review of computational geometry, then present the problems which are related to the thesis. We finally show the organization of the thesis.

1.1. A brief review of computational geometry.

Due to the wide range of applications of computational geometry, a large body of literature involving computational geometry scattered in various journals and books has appeared in the past ten years. However, according to the nature of the geometric objects involved, computational geometric problems can mainly be classified into the following five categories [Lee84]:

(1) convex hull problem (convexity)

(2) intersection problem

(3) geometric searching problem

(4) optimization problem

(5) proximity problem

Each category can be briefly described as follows.

(1) Convex hull problem.

Given a set S of n points in d-dimensional space, find its convex hull (the smallest convex set that contains S). The problem in two and three dimensional space has been extensively studied, and a lower bound of $\Omega(n \log n)$ time has been proved [Sham78].

(2) Intersection problem.

Typical examples of problems in this class are: given a set of n line segments in the plane, determine whether or not any two segments intersect (this is called 'intersection detection', and a lower bound of $\Omega(n \log n)$ time has been established); report the number of line segments that intersect (this is called 'intersection counting', and a lower bound of $\Omega(n \log n)$ time has been established); report the pairs of line segments that intersect (this is called 'intersection reporting', and a lower bound of $\Omega(n \log n + k)$ time has been established, where k is the number of intersecting points).

Several extensions of this problem have been studied. Some examples are: reporting the intersections of a set of rectangles in the plane; detecting and finding the intersection of two convex polygons, of a convex polygon and a nonconvex polygon, or of two nonconvex polygons; detecting the intersection of two 3-dimensional polyhedra.

(3) Geometric searching problem.

A searching problem consists of querying a certain database and gathering a response from the database. The database contains certain geometric objects and is organized so that the searching process can be performed quickly. The measure of a search algorithm consists of the following factors:

query time: the time needed to respond to a query,

storage: the space required by the structured database,

preprocessing time: the time needed to organize the database,

update time: the time to insert or delete an entry.

One of the basic problems in this class is subdivision search. A planar subdivision is a collection of polygons covering the plane. Given a planar subdivision with n edges, find the region of the subdivision that contains a query point. An algorithm with O(log n) query time, O(n log n) preprocessing time, and O(n) storage is known for this problem [Kirk83,Ede86,Sar86].

(4) Optimization problem.

This class involves geometric problems of a combinatorial nature. Example problems are: the Euclidean traveling salesman problem; the disk cover problem (in which n points in the plane are given along with p disks of radius r, and it is required to determine if there exists a placement of these p disks so as to cover all n points); the linear programming problem; and the smallest enclosing circle problem (in which n points in the plane are given, and the smallest circle that encloses all the points is to be found).

(5) Proximity problems.

Geometric objects are used to model physical objects in the real world, and the relative positions of these objects in space. Studying the relative positions among geometric objects is an important problem. Some of the basic problems in this class are finding the closest pair of a set of points in the plane; finding the nearest neighbor vertices of a simple polygon in the plane; finding all nearest neighbors of a set of points in the plane; finding the Euclidean minimum spanning tree of a set of points in the plane; triangulating a set of points in the plane; constructing the Delaunay triangulation of a set of points in the plane; and many modified versions of those problems. Most of the problems in this class are related to a basic geometric structure called the Voronoi diagram.

1.2. Critical analysis of the problems related to this thesis.

Voronoi diagram

The Voronoi diagram, which is also called Dirichlet tessellation by mathematicians, or Thiessen polygons by geographers, is one of the most important geometric structures for partitioning space. Since space partitioning is a fundamental problem in computational geometry, the problem of constructing Voronoi diagrams has attracted a lot of attention.

Formally, the Voronoi diagram of a set of n points in the plane, called a *standard Voronoi diagram*, is a collection of n Voronoi regions $V(p_i)$ for all $1 \le i \le n$, such that

$$V(p_i) = \{x \mid d(x,p_i) \le d(x,p_j),\ 1 \le j \le n,\ x \text{ in the plane}\}$$

where d(x,y) represents the Euclidean distance between x and y.

M. Shamos and D. Hoey first introduced the Voronoi diagram as a powerful tool in the solution of numerous geometric problems [Sham75]. For example, once the Voronoi diagram for a set of points in the plane is available, the closest pair problem,

all nearest neighbors problem, the Delaunay triangulation problem, and the minimum spanning tree problem in the plane, can all be solved in time proportional to the cardinality of that set of points, by obtaining the straight line dual of the Voronoi diagram. The power of the Voronoi-diagram is not limited to the class of proximity problems. We shall see in Chapter 6 that many problems from other classes can be solved efficiently when the Voronoi diagram is available. Examples are finding the convex hull of a set of line segments, constructing the Euclidean minimum spanning tree of a set of line segments, determining the separability of two sets of line segments, and doing a nearest neighbor search in a set of line segments (see Fig.1.1).

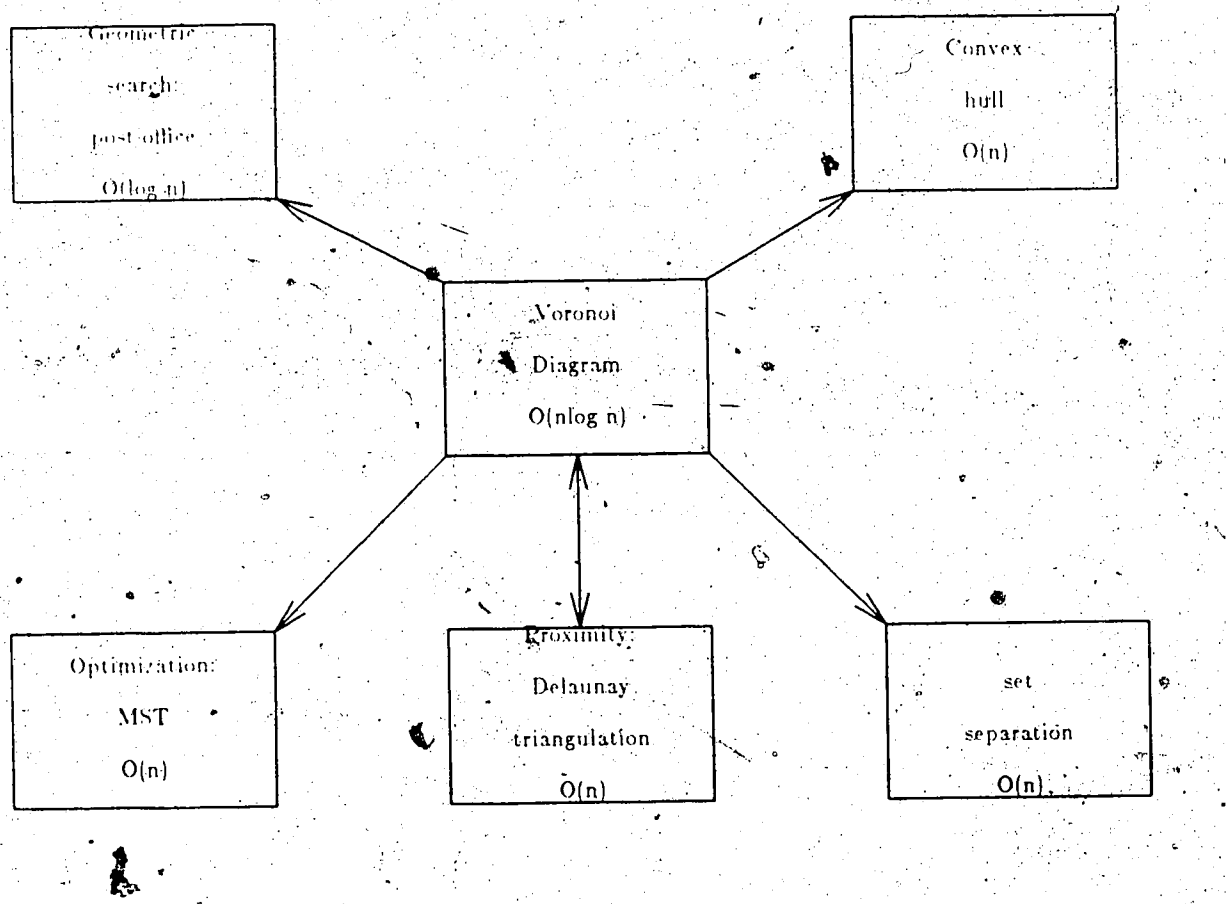


Figure 1.1 Voronoi diagrams are a powerful tool.

Many extensions to the standard Voronoi diagram have been investigated. The main extensions can be classified as follows:

(1) The Voronoi diagram for a set of n points in the plane under an L_p metric [Lee80].

(2) The kth-order Voronoi diagram for a set of points in the plane [Sham78].

(3) The weighted Voronoi diagram for a set of points in the plane [Aur84].

(4) The higher-dimensional Voronoi diagram for a set of points [Avi83,Ede83].

(5) The Voronoi diagram for a set of objects in the plane, where the objects are straight line segments, or simple curve segments [Kirk79,Yap85,Fort86,Fort87].

The last class of Voronoi diagrams is one of the obvious extensions of the standard Voronoi diagram. Formally, it can be defined as follows.

The Voronoi diagram for a set of n objects $l_1, l_2, \ldots, l_n$ in the plane is a collection of n Voronoi regions $V(l_i)$ for all $1 \le i \le n$, such that

$$V(l_i) = \{x \mid d(x,l_i) \le d(x,l_j),\ 1 \le j \le n,\ x \text{ in the plane}\},$$

where $d(x,l_i)$ represents the shortest Euclidean distance between x and the object l_i.

In this class of Voronoi diagrams, first R. Drysdale [Dry79], then D. Lee and R. Drysdale [Lee79] investigated the Voronoi diagram for a set of n line segments in the plane. They proposed an $O(n \log^2 n)$ algorithm to construct it. Later, D. Kirkpatrick [Kirk79, Chaz85,Lee82a] improved the time bound of this problem to $O(n \log n)$ time by the use of a divide and conquer method. He called the Voronoi diagram of a set of line segments the *skeleton* of that set. Recently, C. Yap [Yap85, Fort86,Fort87] extended the input objects of a Voronoi diagram to a set of n simple curve segments and showed that it can be constructed in $O(n \log n)$ time.

Delaunay triangulation

Triangulating a set of points in the plane is another important problem in computational geometry. Many geometric problems can be solved efficiently if triangulations of the set are available. Two examples are: finding the shortest path between two given points in a polygonal region and decomposing a simple polygon into convex subpolygons. The Delaunay triangulation of a set of points in the plane is a special kind of triangulation. It can be simply described as the triangulation of a set of data points in the plane such that the interior of the circumcircle of each triangle does not contain any data point. Although the Delaunay triangulation of a set of points in the plane can be obtained from the corresponding Voronoi diagram of that set, directly constructing the Delaunay triangulation of the set is necessary in many cases [Lee80a]. D.Lee and B. Schachter proposed two algorithms for constructing the Delaunay triangulation of a set of n points. One algorithm takes $O(n^2)$ time, the other takes $O(n \log n)$ time.

There are several important unsolved problems in Delaunay triangulations and Voronoi diagrams:

(1) The generalized version of the problem is the Delaunay triangulation of a set of line segments. In the triangulation, the vertices of each triangle are the endpoints of some input line segments, the edges of each triangle do not cross any input line segment, and the interior of the circumcircle of each triangle contains no endpoint visible from all three vertices of the triangle. D. Lee and A. Lin proposed an $O(n^2)$ algorithm to construct it [Lee85]. However, the algorithm is not known to be optimal. Hence, a natural question is can we design a faster algorithm to construct it.

(2) It has been shown that the straight line dual of the standard Voronoi diagram for a set of points in the plane is the Delaunay triangulation of that set of

points[Sham75]. As long as one of the two diagrams is available, the other can be constructed in time proportional to the cardinality of that set. Now, the question whether or not the duality of the Voronoi diagram and the Delaunay triangulation still holds when the objects are a set of line segments. Some efforts to answer this question have been made in the past, and it has been shown that the straight line dual of the Delaunay triangulation of a set of n line segments is not the skeleton of that set[Lee85]. Thus, although we know that the skeleton of a set of n line segments can be constructed in O(n log n) time, we still do not know whether or not the Delaunay triangulation of a set of n line segments can be constructed in O(n log n) time. (Remark: L. Chew recently found an O(n log n) algorithm to construct the generalized version of Delaunay triangulation [Chew87].) Consequently, the problem will be how to define the straight line dual of the Delaunay triangulation of a set of line segments and how fast we can find it. We believe that the Delaunay triangulation of a set of line segments can be obtained from the bounded Voronoi diagram of that set, which we shall define in Chapter 2, in time proportional to the cardinality of the set. We also show that the bounded Voronoi diagram of a set of n line segments can be constructed in O(n log n) time, and hence that the corresponding Delaunay triangulation can be found in O(n log n) as well.

(3) Finally, the existing Voronoi diagrams do not deal with obstacles. But in the real world, obstacles are often present. Thus, we need to know how to define the Voronoi diagram so that it can be used to represent the information about obstacles.

As illustrated in Fig.1.1, the significance of studying these geometric structures is due to their wide applications in solving many problems in computational geometry. Thus, the two structures have attracted a lot of attention. The main goal of this thesis is to answer the above previously unsolved problems.

1.3. Outline of the thesis.

We shall begin the thesis by giving the definition of the bounded Voronoi diagram, then discuss the diagrams in two different cases.

Chapter 2 is a study of the bounded Voronoi diagram. The definition of the bounded Voronoi diagram is given, and two special cases are discussed. The properties which are different from those of the conventional Voronoi diagram are presented. Conceptually, bounded Voronoi diagrams divide the input information into 'Voronoi data' and 'obstacles'. This concept can be extended to the other classes of Voronoi diagrams.

Chapter 3 presents an algorithm to construct the bounded Voronoi diagram for a monotone chain (see the definition in that chapter). We apply a divide and conquer method and obtain an $O(n \log n)$ time algorithm. The emphasis of this chapter is on the merge procedure for the bounded Voronoi diagrams of the two subchains, because this procedure is the backbone of the algorithms in later chapters.

Chapter 4 discusses the construction of the bounded Voronoi diagram of a simple polygon. This algorithm is based on the following observation. The interior and the exterior of the bounded Voronoi diagram of a simple polygon can be constructed independently. A divide and conquer method is applied to construct the interior bounded Voronoi diagram of a polygon. The algorithm takes $\Theta(n \log n)$ time, where n is the number of vertices in the polygon.

Chapter 5 studies the construction of the bounded Voronoi diagram for a set of line segments. The design of an efficient algorithm for constructing this diagram relies on a fast merge algorithm. An optimal $\Theta(n \log n)$ time algorithm is presented, where n is the number of input line segments.

Chapter 6 presents some problems which can be efficiently solved when the corresponding bounded Voronoi diagram is available, and it also summarizes the

results in the thesis and discusses some open problems remaining in this area.

Chapter 2

Preliminaries.

In this chapter, we first give some general notation and definitions used in the thesis. We then define the *Bounded Voronoi Diagram* for the endpoint set of a set of non-crossing line segments with the line segments themselves as obstacles in the plane and briefly describe its properties. We also discuss two special cases of bounded Voronoi diagrams: for a monotone chain and for a simple polygon. We finally describe the data structures used in our algorithms for constructing the diagrams.

Throughout the rest of this thesis, we use the following notational conventions (although it will always be clear from context what kind of object a symbol refers to). We use bold upper case for unordered sets (i.e., $\mathbf{C},\mathbf{L},\mathbf{P},\mathbf{S},\ldots$), italic upper case for sets on which an orientation relation is imposed (i.e., $C, L, P, \ldots$), bold letters with superscripts for unordered subsets (i.e., $\mathbf{C}^1,\mathbf{L}^1,\mathbf{P}^1,\mathbf{S}^1,\ldots$), italic letters with superscripts for oriented subsets ($C^1, L^1, P^1, \ldots$), italic letters with subscripts for distinguishing the different elements in a set (which may be themselves sets) (i.e., $x_1, l_1, p_1, p_i, r_1, r_i, \ldots$), and Roman upper case for some standard types of geometric objects (i.e., P for a polygon, L for a line).

A straight line determined by two distinct points p and q is an unordered set of points,

$$\mathbf{L} = \{ x \mid x = \alpha p + (1-\alpha) q; \ \alpha \in \mathbb{R} \};$$

if $0 \le \alpha \le 1$, $\mathbf{L}$ is a straight line segment, and it is usually written as $\overline{pq}$; if $\alpha \ge 0$ (or $\alpha \le 1$), $\mathbf{L}$ is a ray starting at q (or p).

Definition: A *finite linear figure* $\mathbf{F}$ is the union of a finite number of line segments (with line segments regarded as point sets). A *vertex* v of a figure $\mathbf{F}$ is a point of $\mathbf{F}$ such that for no circular region $\mathbf{R}$ centered at v is $\mathbf{R} \cap \mathbf{F}$ a diameter of $\mathbf{R}$. An *edge* of $\mathbf{F}$ is a straight line segment $e \subseteq \mathbf{F}$ joining two vertices of $\mathbf{F}$ and not

containing interior vertices. If some edges of a linear figure are replaced by rays, or straight lines, then the linear figure is *infinite*. A linear figure is *simple* if the degree of each vertex is at most two (the *degree* of a vertex is the number of edges sharing that vertex).

Definition: A *(finite or infinite) chain* is a (finite or infinite) linear figure which is also simple and connected. A chain is *endless* if all vertices are of degree two. A chain is *closed* (i.e., a simple polygon) if it is finite and endless, and *open* otherwise. A chain is *semi-infinite*, if one of the extreme edges of an open finite chain is replaced by a ray.

By the Jordan curve theorem, endless chains bipartition the plane (whether they are closed or open). We can impose an 'orientation' on a chain by specifying an ordered pair of distinct points belonging to one of its edges (e.g., the endpoints of the edge, if it is finite). This defines a 'direction of traversal' in an obvious way. For closed chains, i.e., simple polygons, we always choose the orientation such that the interior of the polygon lies to the right of the direction of traversal.

Definition: A chain C *does not cross* a point set S (e.g., a linear figure), if there exists an endless chain C' such that $C \subseteq C'$ and $S-C'$ lies entirely on one side of C'. We write this as $C \not\subset S$. The negation of the relation is written replacing $\not\subset$ by $\subset$. C *overlaps* S if some subset of $C \cap S$ is a finite line segment. Note that line segments which have a common endpoint or which overlap do not cross.

For simplicity, we assume in the rest of the thesis that the input line segments are finite and do not cross or overlap, and they are in general positions, so that no more than three endpoints are co-circular, no more than two endpoints are co-linear, no center of a circumcircle of three data points falls on an input line segment, and in general, there are no 'coincidental' overlaps between constructed lines, constructed points, and input line segments. (If the assumption of general positions is removed, then extra time is required to detect whether or not there are coincidental overlaps, and if they

do occur, then perturbation methods can be applied to deal with the coincidental situations. If the input line segments cross each other, then a preprocessing step, for example a plane sweep, is required to obtain all the intersection points. If the intersection points are regarded as data points, then the case will be dealt with in Chapter 5.)

Note that with the assumed restrictions, the union of input line segments yields a linear figure whose edges are the input line segments and whose vertices are their endpoints. Thus the correspondence between sets of input line segments and linear figures is one-to-one, allowing us to talk more or less interchangeably about sets of input line segments and linear figures.

Definition: The *straight-line distance* between two points, x and y, with **L** (a linear figure, or the corresponding set of line segments) as the obstacle set is defined as follows.

$$d_L(x,y) = \begin{cases} d(x,y) & \text{if } \overline{xy} \notin \mathbf{L}, \\ \text{undefined} & \text{otherwise} \end{cases}$$

where $d(x,y)$ is the Euclidean distance of two points and $\overline{xy}$ is the line segment delimited by x and y. (Remark: If $d(x,y)$ represents the geodesic distance [Aro87] of points x and y, the resulting Voronoi diagram is the same as the diagram defined below.)

In the following, we first define a 'bounded Voronoi diagram' for a set of line segments. We then discuss it in two special cases and briefly describe the properties of these diagrams, and finally, we present the data structures used in our algorithms for constructing the bounded Voronoi diagrams.

Definition: The *bounded Voronoi diagram* (or BV-diagram) of a set of line segments **L** with endpoints **P** is a collection of regions (which we call BV-regions) $\mathbf{Vor}(\mathbf{P},\mathbf{L}) = \{\mathbf{V}(p_i) \mid p_i \in \mathbf{P}\}$, such that

$$\mathbf{V}(p_i) = \{ y \mid d_L(y,p_i) \le d_L(y,p_j), \text{ for all } p_j \in \mathbf{P}, i \neq j, y \in E^2 \}$$

(We take $\alpha \le \beta$ to be true *iff* α is defined, and β is either undefined or at least as

great as α.) (Refer to Fig.2.1.) The *BV-boundary* of a BV-diagram is the linear figure formed by the union of BV-region boundaries and input line segments. Those edges of the BV-boundary which are not portions of input line segments are called *BV-edges* (*BV-rays*), while the rest are called *pseudo-edges*. The vertices of the BV-boundary are called *BV-vertices*, and these are divided into *bound* BV-vertices, lying on input line segments, and *free* BV-vertices, not lying on line segments. (Because of the 'general positions' assumption, vertices where three BV-edges meet cannot fall on input line segments.)

By definition, a free BV-vertex is determined by three data points, and a bound BV-vertex is determined by two data points and an input line segment (obstacle). Thus, a BV-edge is determined by four elements (two data points and two obstacles, or three data points and an obstacle, or four data points), a BV-ray is determined by three elements (two data points and an obstacle, or three data points), and a pseudo-edge is determined by four elements (three data points and an obstacle). In the thesis, we are often concerned with only two of the four elements determining a BV-edge or a pseudo-edge. That is, we are concerned with the two elements which determine the line overlapping the BV-edge or the pseudo-edge. Thus, we often use $(v_i, v_j; p_s, p_q)$ to represent a BV-edge, where $\overline{v_i v_j}$ lies on the perpendicular bisector of p_s and p_q, and $(v_i, v_j; p_s, l_s)$ to represent a pseudo-edge, where $\overline{v_i v_j}$ lies on l_s and is visible from p_s.

Note in the thesis that we only consider straight line distances in the definition of the Voronoi diagrams. If we replace straight line distances by geodesic distances in the definition of Voronoi diagrams, then BV-diagrams become geodesic Voronoi diagrams [Aro87].

Note also that in the above definition, we restrict the set of data points **P** to be the set of endpoints of **L**. In more general BV- diagrams, we might allow the data points to be different from the endpoints of **L**. However, it seems difficult to design

efficient algorithms to construct these BV- diagrams. Note that the BV- regions in the diagrams may not be connected to each other (i.e., some regions may be invisible from all data points, so that no part of them belongs to any Voronoi region), and the number of isolated BV- regions may be $\Omega(|\mathbf{P}|^4)$. For now, we require the endpoints to be the data points.

Three cases of BV- diagrams are illustrated below.

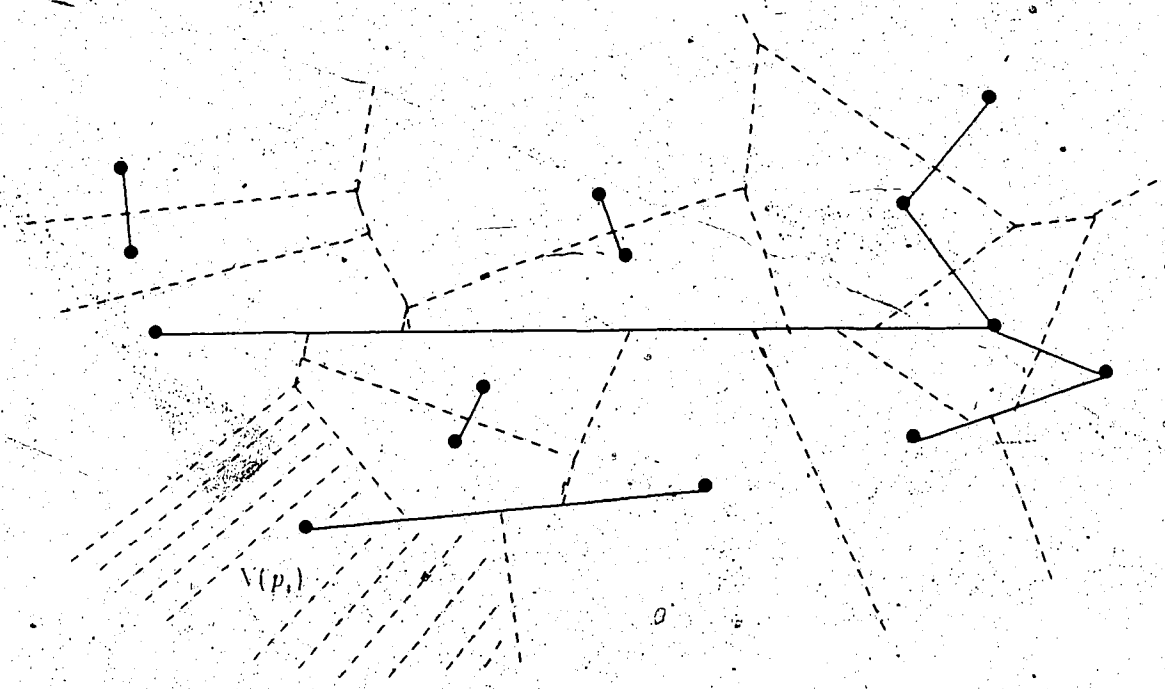


Figure 2.1 The BV- diagram of L.

(a). The general case of a BV- diagram for a finite set of non-crossing line segments **L** in the plane (see Fig.2.1). Note that the input line segments are all included in the BV-boundary by definition.

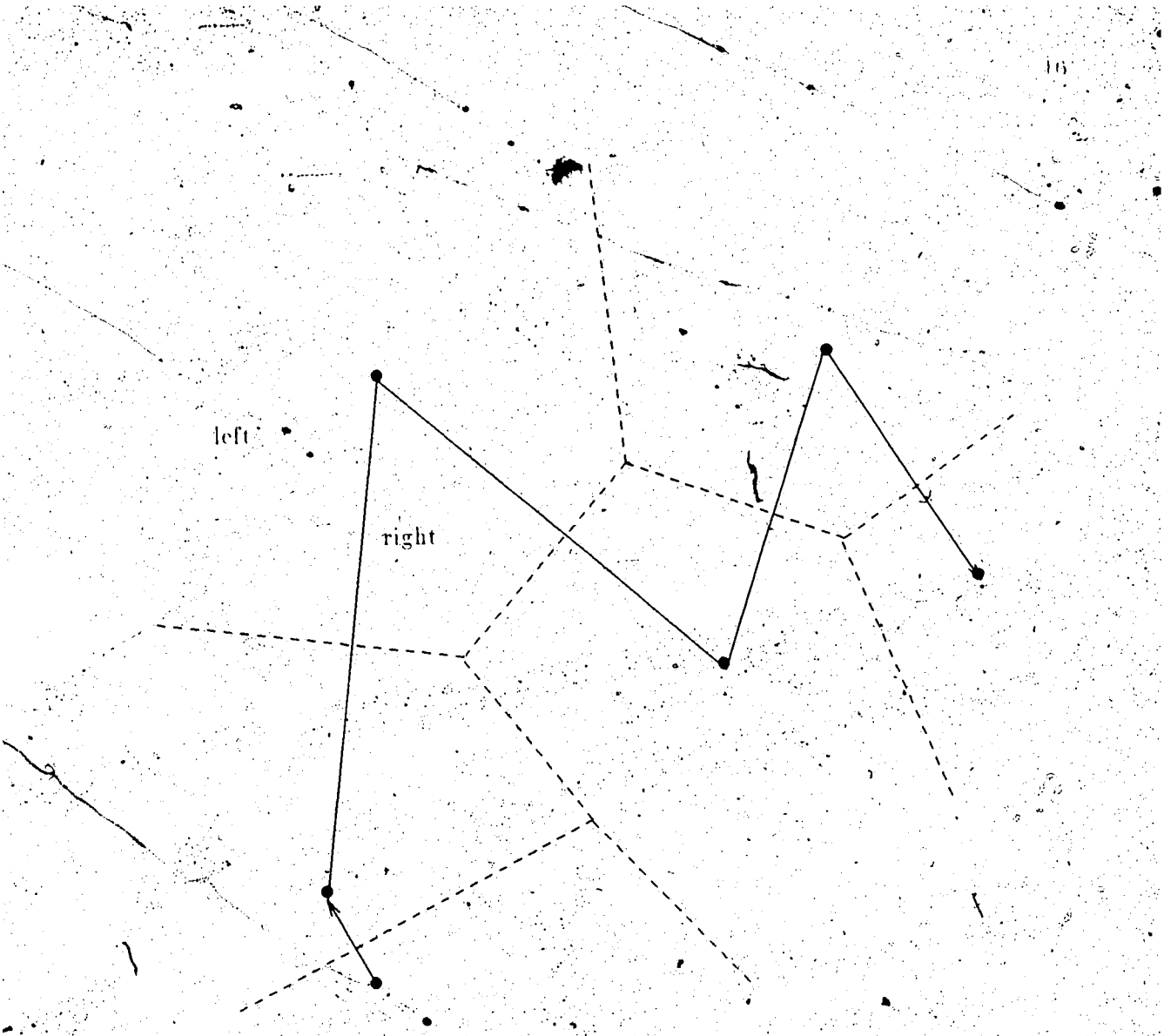


Figure 2.2 The BV- diagram of a chain.

(b) A BV- diagram for a finite chain $C = (p_1, \ldots, p_n)$. This is the BV- diagram of the vertex set $\{p_1, \ldots, p_n\}$ with C as the obstacle. Note that each BV-region $V(p_i)$ for $1 < i < n$ is split into two subregions. If we choose the orientation of the chain as indicated by the arrows in the figure, we then can define the right and the left regions for vertices other than the endpoints of the chain (see Fig.2.2).

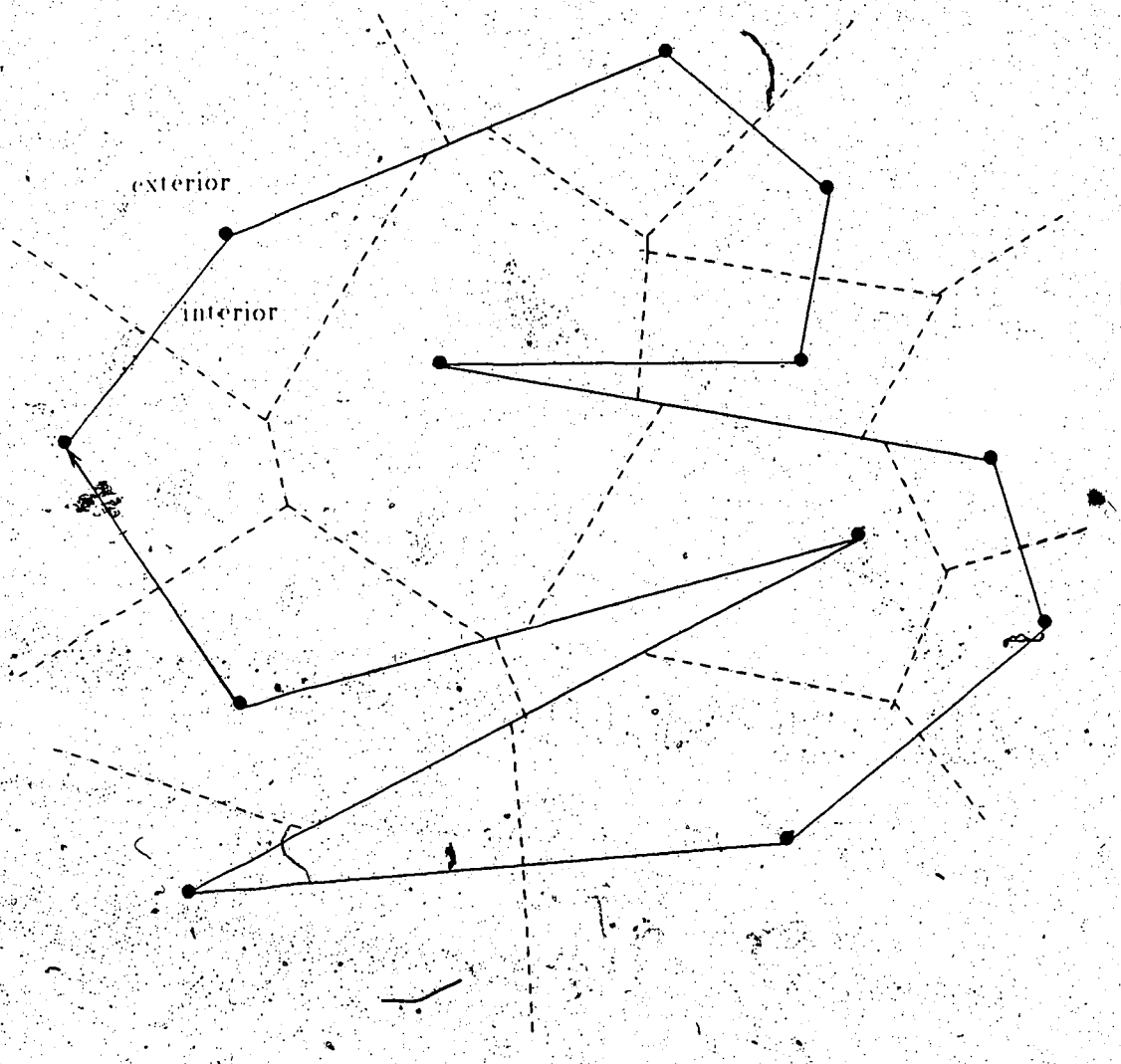


Figure 2.3 The BV- diagram of a simple polygon.

(c) A BV- diagram for a simple polygon P, i.e., the BV-diagram of the vertex set **P** of P with the boundary B of P as the obstacle. Note that each BV-region is split into an interior region and an exterior region (see Fig.2.3).

We briefly list some properties of BV- diagrams which are different from those of standard Voronoi diagrams. We also prove a lemma which is important for constructing BV-diagrams. Let $\mathbf{P}=\{p_1,\ldots,p_m\}$ be the endpoint set of **L**. In the BV- diagram of **L**, a BV-region $V(p_i)$, where $p_i \epsilon \mathbf{P}$, may not be convex because of its truncation at the

obstacles (see the shaded region of Fig.2.1). However, $V(p_i)$ must be star-shaped since every point belonging to it is *visible* from p_i. (A point is *visible* from a point $p_i \in \mathbf{P}$ if $\overline{xp_i} \notin \mathbf{L}$.) The boundary of $V(p_i)$ need not be the locus of points equidistant from p_i and its neighboring data points, because it may consist in part of pseudo- edges coinciding with segments of obstacles. A pseudo-edge has at most one endpoint that is a data vertex (see examples in Fig.2.3), while the other endpoint (or both of them) is determined by an incident BV-edge. While free BV-vertices are always of degree three (as in standard Voronoi diagrams), bound BV-vertices may be data points and hence of the same degree (1,2,3,...) as those data points. Bound BV-vertices which are not data points are either of degree three (formed by a BV-edge incident on an input line segment, where at least one of the data points determining that BV-edge is not an endpoint of the input line segment in question), or of degree four (formed by two BV-edges incident on an input line segment, where those two BV-edges together form part of the perpendicular bisector of the input line segment). Note that because of the general positions' assumption, there cannot be a bound vertex of degree four such that its two incident BV-edges are not part of the perpendicular bisector of the input line segment. Because of the inclusion of input line segments in the BV-boundary, the intersection of a BV-region with the BV-boundary may include line segments internal to the BV-region, besides the boundary segments of the BV-region. Two BV- regions are *neighbors* if they share a BV- edge or pseudo-edge. Note that since we restrict the data points to be the endpoints of input line segments, any point $x \in E^2$ must be visible from at least two data points. Therefore, any point $x \in E^2$ must belong to some BV-region by definition.

Lemma 2.1 *Let (v_i, v_j, p_x, p_y) be a BV-edge of BV-diagram Vor(**P**;**L**), where **L** is the set of input line segments and **P** is the set of their endpoints. Then, p_x and p_y are visible from each other.*

Proof:

Suppose for contradiction that p_x and p_y are not visible from each other. Then, some $l \in \mathbf{L}$ must cross $\overline{p_x p_y}$. Let p_s and p_t be the endpoints of l and let x be an arbitrary point on $\overline{v_i v_j}$. Then, one of p_s and p_t must lie inside the triangle $p_x x p_y$ since otherwise l will also cross one of $\overline{p_x x}$ and $\overline{p_y x}$, and by definition, $\overline{v_i v_j}$ would not be a BV-edge shared by p_x and p_y. Thus, since one of p_s and p_t, say p_s, will lie inside the triangle. (Without loss of generality, we assume that l is chosen so that x is visible from p_s). Then, x must be closer to p_s than to one of p_x and p_y. However, by the definition of BV-edges, x must be closer to p_x and p_y than to p_s. □

The number of edges in a BV-diagram of n line segments is bounded by $O(n)$. To see this, note that each pseudo-edge must share a bound vertex with a BV-edge. Hence, the number of pseudo-edges in a BV-diagram is proportional to the number of BV-edges. Moreover, each BV-edge is shared by a pair of data points. If we connect the two data points of each pair with a line segment (Delaunay edge), then these line segments form a planar graph on n data points (endpoints). By Euler's theorem, the number of edges of the planar graph is bounded by $O(n)$. Thus, the number of BV-edges is bounded by $O(n)$ because of the one to one correspondence between BV-edges and Delaunay edges. Hence, the number of edges in a BV-diagram is bounded by $O(n)$.

BV- boundaries are planar graphs, hence they can be represented by either the *Doubly Connected Edge List* (DCEL) [Mul78] or the *Quad Edge* (QE) data structure [Gui85]. In addition, for fast access to the line segments incident at a data point and the BV- edges associated with a data point, we use the following data structures, which can be obtained from DCEL or QE in time proportional to the number of input line segments. Vor(**P**:**L**) is represented as an array indexed by data points. Each cell of the array contains a list of the Voronoi boundary elements (i.e., BV- edges and pseudo-edges) associated with the indexed data point. Each element includes:

(a) the data point which together with the indexed data point determines the BV-edge (or, in the case of a pseudo-edge, the edge of L together with the indexed data point determines the pseudo-edge); and

(b) the endpoints of the BV-edge or pseudo-edge; and

(c) flags which indicate whether the BV-boundary element is a pseudo-edge and whether the two data points determining the BV-boundary element form an edge of **L**, and pointers which point to the edges of **L** incident at the indexed data point, and which point to the edges of L_2 containing the pseudo-edges.

We use pV to denote the above data structure. L is an array indexed by the obstacle line segments, where each cell of the array contains a list of those data points (if any) which determine pseudo edges lying within the indexed obstacle line segment. More data structure details will be given later.

Chapter 3

Finding the BV- diagram of a monotone chain.

3.1. The main idea of the construction.

The main purpose of discussing the BV- diagram of a monotone chain (see the definition below) is to introduce a merge algorithm which deals with obstacles. A monotone chain can be divided into two halves such that they can be separated by a line which passes through exactly one point of the chain. This property is called pseudo-linear separability (see the definition below). This property allows us to design a fast merge algorithm which will be used in the two special cases of BV- diagrams (chains and polygons).

The main idea of our algorithm for constructing the BV- diagram of a monotone chain C is to apply a divide and conquer method to C. Clearly, the time complexity of a divide and conquer approach (when used recursively) is determined by the time of its merge procedure as well as the depth of recursion. We shall show that the time complexity of our merge procedure is proportional to the total number of vertices in the two half chains which are to be merged. The depth of recursion is $\log_2 n$, hence, the time complexity of our algorithm is bounded by $O(n \log n)$, where n is the total number of vertices in the monotone chain.

In order to merge two original BV- diagrams $Vor(P^1;C^1)$ and $Vor(P^2;C^2)$ into a resultant BV- diagram $Vor(P^1 \cup P^2;C^1 \cup C^2)$, where C^1 and C^2 are two chains and P^1 and P^2 are their vertex set respectively, we must determine which original BV- edge or pseudo-edge (or part of it) will be deleted in the resultant BV- diagram and which new BV- edge or pseudo-edge, which does not belong to the original BV- diagrams, will be added into the resultant BV- diagram. Merging the BV-diagrams of two arbitrary chains is quite difficult. To see this, note that by Lemma 2.1, the data points associated with an original BV- edge must be visible from each other. However,

one chain may obstruct the line of sight of two data points sharing a BV-edge in the other chain, causing the BV-edge to vanish from the resultant diagram. Hence, when we merge the BV- diagrams of two chains, we must ignore all of those original BV-edges whose associated data points are no longer visible from each other. Hence, we have to test the intersections between the line segments connecting the data points associated with the original BV- edges and the chains to identify those BV- edges. Using a brute force method, the test would take quadratic time.

Moreover, one chain may also distort the original BV- diagram of the other chain, since two data points in one chain, originally not determining a BV- edge, may now determine one due to the presence of the other chain. This means that some new BV- edges must be created in the original BV- diagram before we execute a merge procedure. This work may also take more than linear time. Clearly, solving the above two problems will take more than time proportional to the cardinality of the two chains.

In the resultant BV-diagram, those BV-edges which are determined by some data points in one chain and some data points in the other chain, and those pseudo-edges which are part of one chain while truncating the BV-region of some data point of the other chain, form the 'shared BV- boundary'. (A formal definition for a special case of (pseudo-) linearly separable chains will be given later.) In the case of two non-(pseudo-) linearly separable chains, the shared BV- boundary may not include all the new BV- edges which must be added to the resultant BV-diagram. The new edges may occur between the data points of the same chain. Thus, finding the shared BV- boundary of two such chains is not equivalent to finding all the new BV-edges and new pseudo-edges in their resultant BV- diagram. That is, the conventional method for merging two standard Voronoi diagrams does not generalize to this case.

However, we shall show in Lemma 3.1 that the shared BV- boundary of two

linearly separable chains is an endless open chain. Further, we shall show in Lemma 3.2 that when two input chains are linearly separable, all new BV-edges and pseudo-edges (with respect to the given original BV-edges and pseudo-edges) in the resultant BV-diagram consist of the shared BV-boundary, and all original BV-edges (or pseudo-edges) which must be deleted in the resultant BV-diagram can be detected according to the information obtained during the construction of the shared BV-boundary. In this sense, we can say that constructing the resultant BV- diagram of two linearly separable chains is *equivalent* to finding their shared BV- boundary.

Definition: Two linear figures are *pseudo-linearly separable* if they can be separated by a straight line crossing neither of them. A straight line which separates two linear figures is called a *separator* of the two figures. Two linear figures are *linearly separable* if they are pseudo-linearly separable and disjoint from some separator.

We would like to construct the shared BV- boundary of two linearly separable chains by locating a starting segment and then constructing successive segments in the order determined by their interconnections.

There are three problems which must be solved in such a construction.

a) How to find the starting segment of the shared BV- boundary (which we call the start problem).

b) How to determine the successive segments of the shared BV- boundary. Each BV-edge of the shared BV- boundary is determined by a pair of data points visible from each other, where each data point in the pair belongs to a different chain, and each pseudo-edge is determined by a data point and an input line segment from opposite chains such that the pseudo-edge is visible from the data point. Thus, the key point is how to ensure that each segment of the shared BV-boundary which we construct satisfies the visibility require-

ment (which we call the visibility problem).

c) How to determine when the construction of the shared BV- boundary is finished (which we call the termination problem).

In the following, we shall first give several definitions and lemmas which are necessary for solving the above problems, then present the algorithms to merge the BV- diagrams of two linearly or pseudo- linearly separable chains. We finally show how to construct the BV- diagram of a monotone chain.

Let C^1 and C^2 be two (pseudo-) linearly separable finite chains, and $\mathbf{P}^1=\{p_1,\ldots,p_n\}$ and $\mathbf{P}^2=\{q_1,\ldots,q_m\}$ be their vertex sets (data point sets), respectively.

Definition: The *shared BV- boundary* of C^1 and C^2 is the linear figure formed from the union of *shared BV- edges* and *shared pseudo-edges*; the shared BV- edges are those determined by some $q_i \in \mathbf{P}^2$ and some $p_j \in \mathbf{P}^1$, and the shared pseudo-edges are those which are part of one chain and truncate the BV-region of a data point of the other chain. (Note that an input line segment may contain several shared pseudo-edges.) The vertices of the shared BV- boundary are called *shared BV- vertices*. Clearly, if some data point p_k is shared by C^1 and C^2 (i.e., C^1 and C^2 are pseudo linearly separable), then the boundary of $V(p_k) \in Vor(\mathbf{P}^1 \bigcup \mathbf{P}^2; C^1 \bigcup C^2)$ belongs to the shared BV- boundary.

We shall see in Lemma 3.1 that the vertices of the shared BV- boundary of two linearly separable chains must have degree two. We shall also see in Lemma 3.5 that some (free or bound) vertex in the shared BV- boundary of two pseudo- linearly separable chains may have degree three (see Figure 3.12, where the 'connecting point' (free) is of degree three, and see Figure 3.2, where v_i (bound) is of degree three).

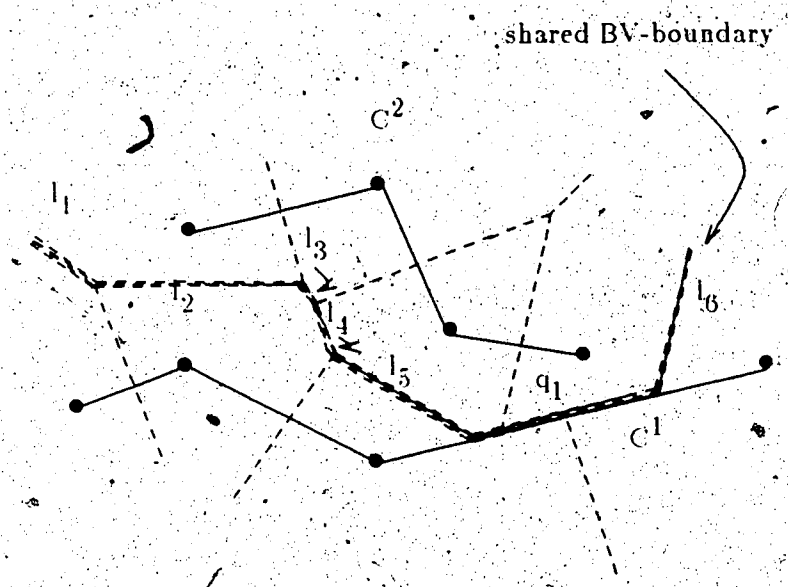


Figure 3.1 The shared BV- boundary of two linearly separable chains.

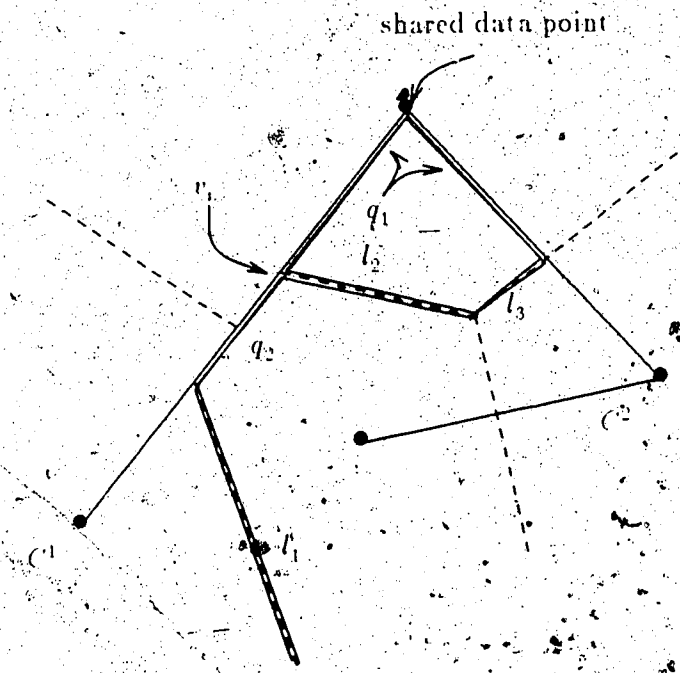


Figure 3.2 The shared BV- boundary of two pseudo-linearly separable chains (right [illegible]

Definition : A chain C is *monotone* with respect to a straight line l if the order of the perpendiculars from the vertices of C to l along l is the same as that of the vertices in C.

Definition: A *bisector ray* of a directed line segment $l = \overrightarrow{pq}$, denoted by $b(r_0;p,q)$, is defined as half of the perpendicular bisector of l, starting at the midpoint r_0 of l, and pointing to the right hand side with respect to the direction of $\overrightarrow{pq}$.

Definition : Given a standard Voronoi diagram, a *spoke* is the line segment connecting a Voronoi vertex to one of its associated data points. Thus, a Voronoi vertex has three incident spokes by the general position assumption.

3.2. Finding the BV- diagram of two linearly separable chains.

In the following two sections, we first show how to construct the shared BV-boundary of two linearly separable chains, and establish that merging the BV-diagrams of two chains is equivalent to finding their shared BV- boundary. We then discuss how to construct the shared BV- boundary of two pseudo-linearly separable chains sharing exactly one of their extreme vertices.

From now on, we use the term *input-chains* to distinguish them from the chains in BV- diagrams which are called *BV- chains*, and use $p_i, p_j, p_z, \ldots$ to represent the data points and $v_i, v_m, v_n, \ldots$ to represent the Voronoi (BV-) vertices.

Lemma 3.1 generalizes a corresponding result for standard Voronoi diagrams (e.g., [Prep85], Theorem 5.13 (iii)).

Lemma 3.1: *The shared BV- boundary of two linearly separable input chains C^1 and C^2 is an endless open BV- chain.*

Proof: (Refer to Fig.3.3)

By definition, the shared BV- boundary must be a portion of the BV-boundary of the two input chains, and hence it is a linear figure. We shall show that any line

perpendicular to an arbitrarily chosen separator of C^1 and C^2 intersects the shared BV-boundary exactly once. Therefore, the shared BV-boundary of C^1 and C^2 is an endless BV-chain.

(1) The shared BV-boundary is a collection of endless chains and it separates the BV-regions of data points of one input chain from those of the other input chain. To see this, note that each BV-region in the resultant BV-diagram (after merging) is a cell (topologically a closed disc). If a BV-region of a data point from one input chain and a BV-region of a data point from the other chain intersect, they intersect in an edge, or a ray, or a line. The union of these edges (rays or lines) must be a collection of endless chains. (An alternative argument is as follows. In the resultant BV-diagram, we assign the red color to the BV-regions of data points from one input chain and the blue color to those from the other input chain in the resultant BV-diagram. Then, the resultant BV-diagram is a two-colorable map. The boundary shared by the regions with opposite colors forms endless chains [Bol79].)

(2) Let S(x) represent any line perpendicular to an arbitrarily chosen separator L of C^1 and C^2, passing through $x \epsilon \mathbf{L}$. Then, S(x) crosses the shared BV-boundary exactly once.

(a) Suppose for contradiction that S(x) for some $x \epsilon \mathbf{L}$ crosses the shared BV-boundary more than one time. Since no segment of the shared BV-boundary can overlap S(x) for any $x \epsilon \mathbf{L}$ due to the linear separability of C^1 and C^2, S(x) then is divided into several intervals by the crossover points. There must exist at least two consecutive crossover points on S(x), say y and z, where y and z can lie on opposite sides of the separator or on the same side of the separator. Without loss of generality, let S(x) be horizontal and C^1 lie on the right hand side of L and C^2 on the left hand side of L. We assume that the intervals are placed on S(x) from left

to right as $[-\infty,y]$, $[y,z]$, $[z,+\infty]$. Then, no matter which interval x lies on, it must be the case that some point, say $y' \in (y,z)$, belongs to the BV-region associated with a data point, say p_j of C^1 (respectively C^2), and a point say $z' \in [-\infty,y) \cup (z,+\infty]$, belongs to the BV-region associated with a data point, say p_i of C^2 (respectively C^1). (To see this, note that if y or z belongs to a BV-edge, then this follows the definition of BV-edges; if y or z belongs to a pseudo-edge, then this is implied by the linear separability of C^1 and C^2.) Then, there must exist one of the following two cases for y' and z': both y' and z' lie on the same side of x on S(x), or y' lies on the left side of x and z' lies on the right side of x on S(x), and one of the following two cases for y and z: y and z lie on the same side of L or they lie on the opposite sides of L. We only show the case that y and z lie on opposite sides of the separator and y' and z' lie on the same side of the separator (refer to the right part of Fig.3.3). For the other combinations, the proof is similar to the following one. Now, we shall prove that there cannot exist such pair of points on S(x) by showing $z' \in V(p_j)$ which contradicts the given fact, hence the shared BV-boundary cannot cross S(x) more than once. To do so, we show that (i) both p_i and p_j are visible from y' and z', and (ii) $\overline{z'p_i}$ is longer than $\overline{z'p_j}$.

(i) Note that no input line segment can cross $\overline{y'p_i}$ (since it cannot cross the separator), otherwise either it will also cross $\overline{z'p_i}$ or one of its endpoints will be closer to z' than p_i to z', and both conclusions will contradict $z' \in V(p_i)$. Thus, p_i is visible from both y' and z'. This implies that p_j must not lie outside of the circle with $\overline{y'p_i}$ as radius since $y' \in V(p_j)$. Note also that no input line segment can cross $\overline{z'p_j}$, otherwise either it will also cross $\overline{y'p_j}$ (this contradicts $y' \in V(p_j)$) or one of its endpoints will be closer to z' than p_i to z' (this contradicts $z' \in V(p_i)$). Thus, z' is visible from both p_i and p_j.

(ii) $\overline{z'p_i}$ is longer than $\overline{z'p_j}$ because p_i lies on the circle with $\overline{y'p_i}$ as radius, p_j lies

on or inside the circle, and p_j and z' lie on the same side of L. Thus, $z' \in V(p_j)$ by definition and this contradicts the given fact that $z' \in V(p_i)$. Therefore, no S(x) can cross the shared BV-boundary more than once.

(b) S(x) for all $x \in \mathbf{L}$ must cross the shared BV-boundary at least once. To see this, suppose that S(x) for some $x \in \mathbf{L}$ does not cross the shared BV-boundary. Then by (1), there must exist a S(x) crossing the shared BV-boundary more than once, which contradicts part (a) of (2). □

The shared BV-boundary of C^1 and C^2 cannot cross C^1 or C^2. To see this, we suppose for contradiction that the shared BV-boundary crosses C^1 and/or C^2. Then, there must exist two points of S(x) for some $x \in \mathbf{L}$, which lie on the opposite sides of a line segment of $C^1 \cup C^2$ and which belong to BV-regions of data points from the same input chain. Then, one of the two points must not be visible from the corresponding data point. This contradicts the definition of BV-region.

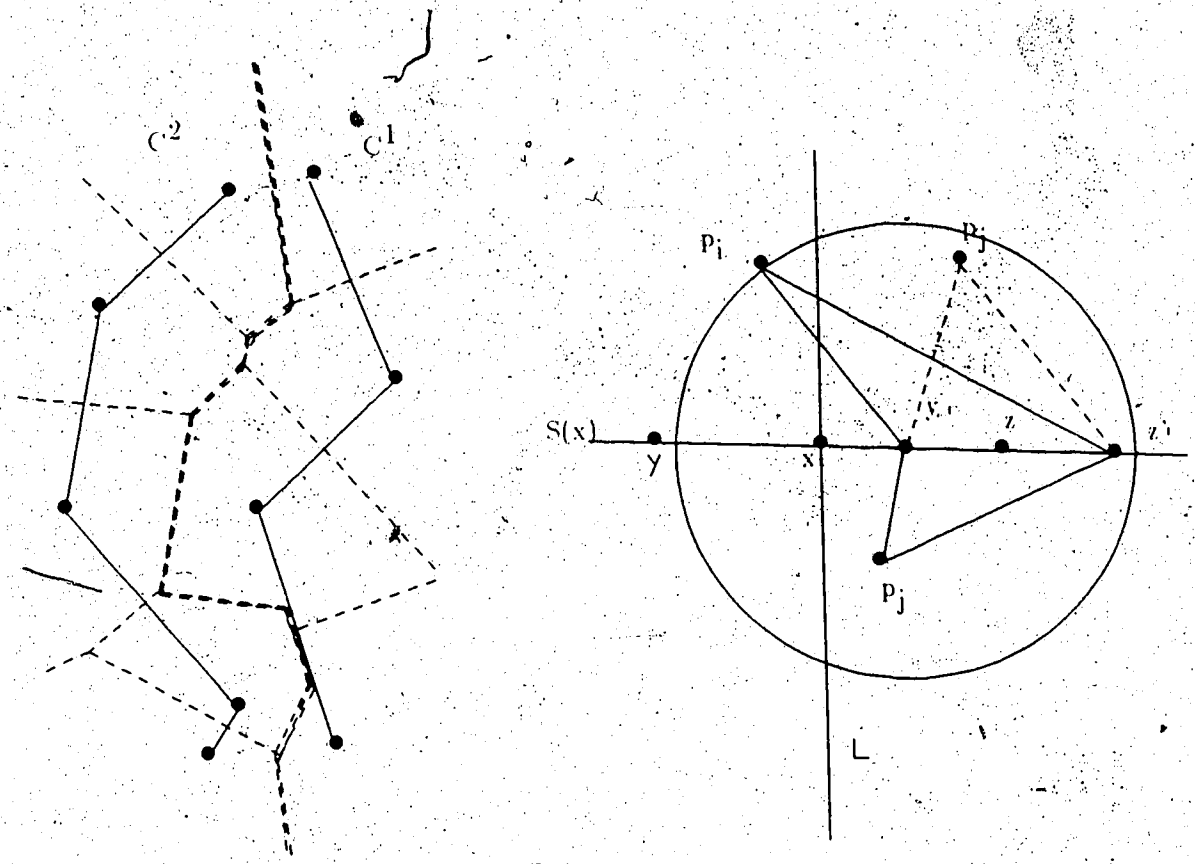


Figure 3.3 The shared BV- boundary is an endless open chain.

We shall show in the following lemma that when the two input chains are linearly separable, the resultant BV- diagram of the two input chains can be obtained from the original BV-diagrams by deleting the portions of the BV-edges and pseudo-edges of C^1 (C^2) which which lie on the opposite side of the shared BV-boundary from C^1 (C^2), and adding the edges of the shared BV-boundary.

Lemma 3.2: *Merging the BV-diagrams of C^1 and C^2 is equivalent to finding the shared BV- boundary of C^1 and C^2, where C^1 and C^2 are two linearly separable chains.*

Proof:

We shall prove that (1) all the new BV- edges (pseudo-edges) which will

appear in the resultant BV- diagram are included in the shared BV- boundary, and (2) all the original BV- edges (pseudo-edges) (or a segment of a such edge) which will disappear from the resultant BV- diagram can be deleted according to information provided by the construction of the shared BV- boundary. (By a 'new' BV- edge we mean that the BV- edge is determined by two data points which have not determined an original BV- edge. Thus a BV- edge, which is a segment of an original BV- edge, is not a new BV- edge.) Note that a new pseudo-edge is created if at least one new BV-edge sharing a bound BV-vertex with that pseudo-edge is created. An original pseudo-edge changes if at least one original BV-edge sharing a bound BV-vertex with that pseudo-edge disappears or is shortened. Thus, we only need to consider the BV-edges.

(1) Any BV- edge determined by a data point $p_i \epsilon \mathbf{P}^1$ and a data point $p_j \epsilon \mathbf{P}^2$ must belong to the shared BV- boundary by definition. Thus, we only need to consider whether or not there exists a new BV- edge, say (v_m, v_n, p_s, p_t) such that both p_s and p_t belong to $\mathbf{P}^1$ (symmetrically, the following proof is applicable when both data points belong to $\mathbf{P}^2$, and the proof is also applicable when it is a new BV-ray). Now, to show this cannot arise, assume that p_s and p_t do not determine a BV-edge in the BV-diagram of $\mathbf{C}^1$. If p_s and p_t are not visible from each other, then obviously no (original or new) BV-edge will be created. Suppose that p_s and p_t are visible from each other. The interior of the circle with $\overline{p_s p_t}$ as the diameter must contain some data points of $\mathbf{C}^1$ sharing some BV-edges with p_s and p_t so that p_s and p_t do not share any BV-edge. Note that p_s, p_t, and these data points determine a convex hull. The convex hull cannot be crossed by $\mathbf{C}^2$ due to the linear separability of the input chains. Then, the distances among the data points of the convex hull

are not changed when C^2 is present. It follows that p_s and p_t do not determine any BV-edge in the resultant BV-diagram. Therefore, no new BV- edge (or BV- ray) can be created between any two data points of C^1 when C^2 is present.

(2) If an original BV- edge (or part of it) of C^1 (respectively, a BV- edge of C^2) disappears from the resultant BV- diagram, then the original BV- edge must be crossed by the shared BV- boundary, or one of the spokes associated with this original BV- edge must be crossed by the shared BV- boundary. To see this, we suppose that there is a deleted original BV- edge such that it is not crossed by the shared BV- boundary and none of its spokes is crossed by the shared BV- boundary. Note that the shared BV- boundary must be an endless open chain by Lemma 3.1, hence the chain will partition the plane into two parts such that each part contains an input chain. The deleted original BV- edge must lie on one of the two parts. If it lies in the part containing its associated data points, then no edge of C^2 lying on the other part can cross the BV-edge and the line segment connecting the associated data points due to the linear separability. Hence, this original BV- edge should not disappear. Thus, any deleted BV-edge must lie on the part not containing its associated data points. Then, at least one of its spokes must be crossed by the shared BV- boundary. (Thus, by examining the crossing between the shared BV- boundary and the original BV- edges as well as their spokes, we can determine which original BV- edge (or part of it) should be deleted in the resultant BV- diagram.) □

To simplify the description of the merge procedure, we first consider the case of two linearly separable input chains, then consider the case of two pseudo- linearly separable input chains. (Note that the input chains are not required to be monotone as long as the original BV- diagrams are given.)

Let us consider the three problems (see pp.23-24) involved in merging two BV-diagrams of two linearly separable input chains. By Lemma 3.1, the shared BV-boundary is an endless open chain which does not cross the input chains. Thus, the start and termination problems can be solved by the conventional "convex hull" method [Sham78]. By Lemma 3.2, all new BV-edges and pseudo-edges appearing in the resultant BV-diagram form the shared BV-boundary. The visibility requirement for the shared BV-boundary of the two input chains is ensured by the linear separability. We shall show in Lemma 3.3 that each BV-region can be partitioned into at most three convex sub-regions. We shall also show that the original BV-edges and pseudo-edges intersecting the shared BV-boundary can be found quickly. We shall now show how to construct the successive segments of the shared BV-boundary according to the interconnections of its elements.

We can find the convex hull of C^1 and C^2 in O(n log n) time, where n is the total number of vertices in the two input chains [Sham78]. Because of the linear separability, the convex hull must contain two specific edges (which are called *common tangents*) such that the endpoints of each edge come from a different input chain. The perpendicular bisectors of the two edges can be chosen as the starting and ending segments (rays) of the shared BV-boundary. If the convex hulls of the two input chains are available, the start and termination problems can be solved in logarithmic time by using a balanced binary search tree [Prep85, pp.121].

Let the two input chains be $\mathbf{C}^1$ and $\mathbf{C}^2$, and let their BV- diagrams be $Vor(\mathbf{P}^1;C^1)$ and $Vor(\mathbf{P}^2;C^2)$. Let $(v_m,v_n;p_s,p_t)$ represent an original BV- edge and $(v_m,v_n;l_t,p_s)$ represent a pseudo-edge $\overline{v_m v_n}$ that is shared by p_s and p_t (l_t). Note that if a BV- edge is delimited by only one BV- vertex, then it is a BV- ray; if a BV- edge is not delimited by any BV- vertex, then it is a BV- line. We also let $b(v_i;p_x,p_y)$ (or $b(v_i;l_x,p_y)$) represent a ray of the shared BV- boundary that starts at v_i and is perpen-

dicular to $p_x p_y$ (or is along l_x and is determined by p_y). $b(v_i, v_{i+1}; p_x, p_y)$ or $b(v_i, v_{i+1}; l_x, p_y)$ represents an edge of the shared BV- boundary. The merge algorithm will begin at a starting ray of the shared BV- boundary, and then traverse this ray to construct the rest of the shared BV- boundary.

Assume that we are currently traversing a ray $b(v_i; p_x, p_y)$ of the shared BV- boundary. $b(v_i; p_x, p_y)$ will meet some original BV- edge, say $(v_m, v_n; p_x, p_t)$ or some edge of the input chains according to one of the following three cases. (Here, we also assume that the original BV-edges or pseudo-edges which meet the shared BV-ray are given. We shall show in Lemma 3.3 how to find these edges.)

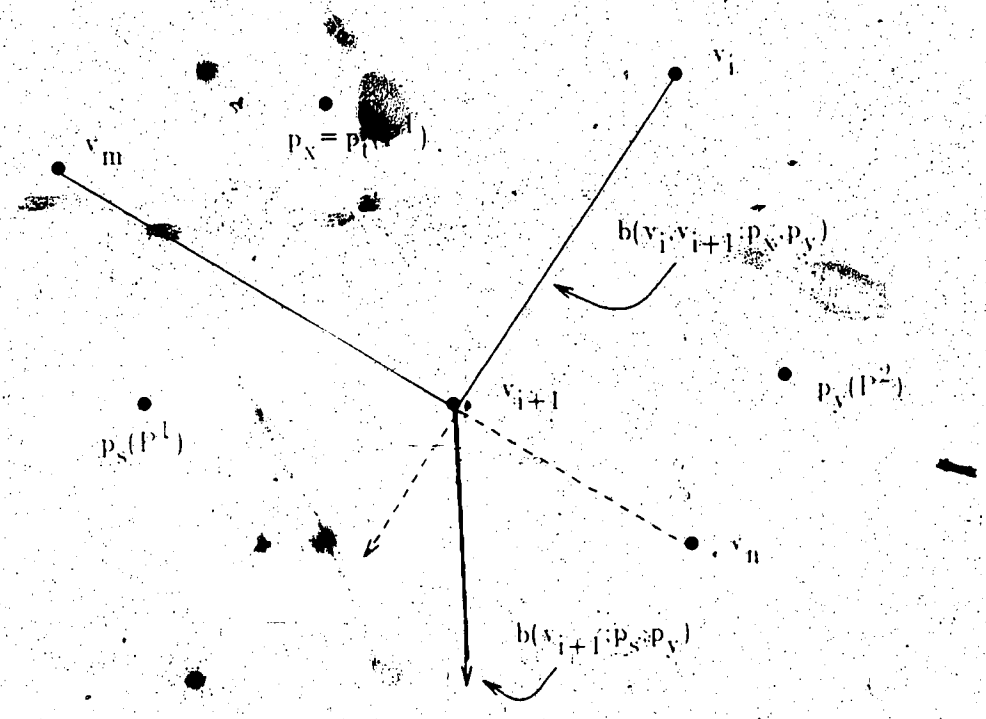


Figure 3.4 Case a.

Case (a). Assume that $b(v_i; p_x, p_y)$ is shared by a pair of data points, $p_x \in \mathbf{P}^1$ and $p_y \in \mathbf{P}^2$, so that $b(v_i; p_x, p_y)$ does not coincide with any segment of $C^1 \cup C^2$. Assume further that $b(v_i; p_x, p_y)$ meets a BV- edge $(v_m, v_n; p_x, p_t)$ in $\mathrm{Vor}(\mathbf{P}^1; C^1)$ or $\mathrm{Vor}(\mathbf{P}^2; C^2)$ (shown in Fig.3.4).

Without loss of generality, let both data points of $(v_m, v_n; p_s, p_t)$, p_s and p_t belong to $\mathbf{P}^1$. Note that one of p_s and p_t must coincide with one of p_x and p_y, say $p_x = p_t$. Clearly then the remaining two vertices, p_s and p_y will determine a new BV-ray. (This case is the same as the standard Voronoi diagram.) We execute the following operations similar to those of merging two standard Voronoi diagrams. Find the intersection point v_{i+1} of $b(v_i; p_x, p_y)$ and $(v_m, v_n; p_s, p_t)$; ignore the segment of $b(v_i; p_x, p_y)$ which leaves v_{i+1} and ignore the segment $\overline{v_{i+1}v_n}$; and regard the ray $b(v_{i+1}; p_s, p_y)$ directed along the bi-sector ray $p_s p_y$ as a new ray of the shared BV-boundary (where $b(v_{i+1}; p_s, p_y)$ is called a *branch*). We call the above operations, *find the next branch* of the shared BV-boundary. Note that $b(v_i, v_{i+1}; p_x, p_y)$, $(v_m, v_{i+1}; p_s, p_t)$, and the branch $b(v_{i+1}; p_s, p_y)$ form a fork at v_{i+1}, and the fork partitions the original BV-regions associated with $p_t\ (= p_x)$, p_y, and p_s into three new BV-regions. It is easy to verify that the points inside each new BV-region are closer to one of the data points $p_t (= p_x)$, p_s, and p_y than to the others. Hence, the branch (or some initial segment of it) must belong to the shared BV-boundary by definition.

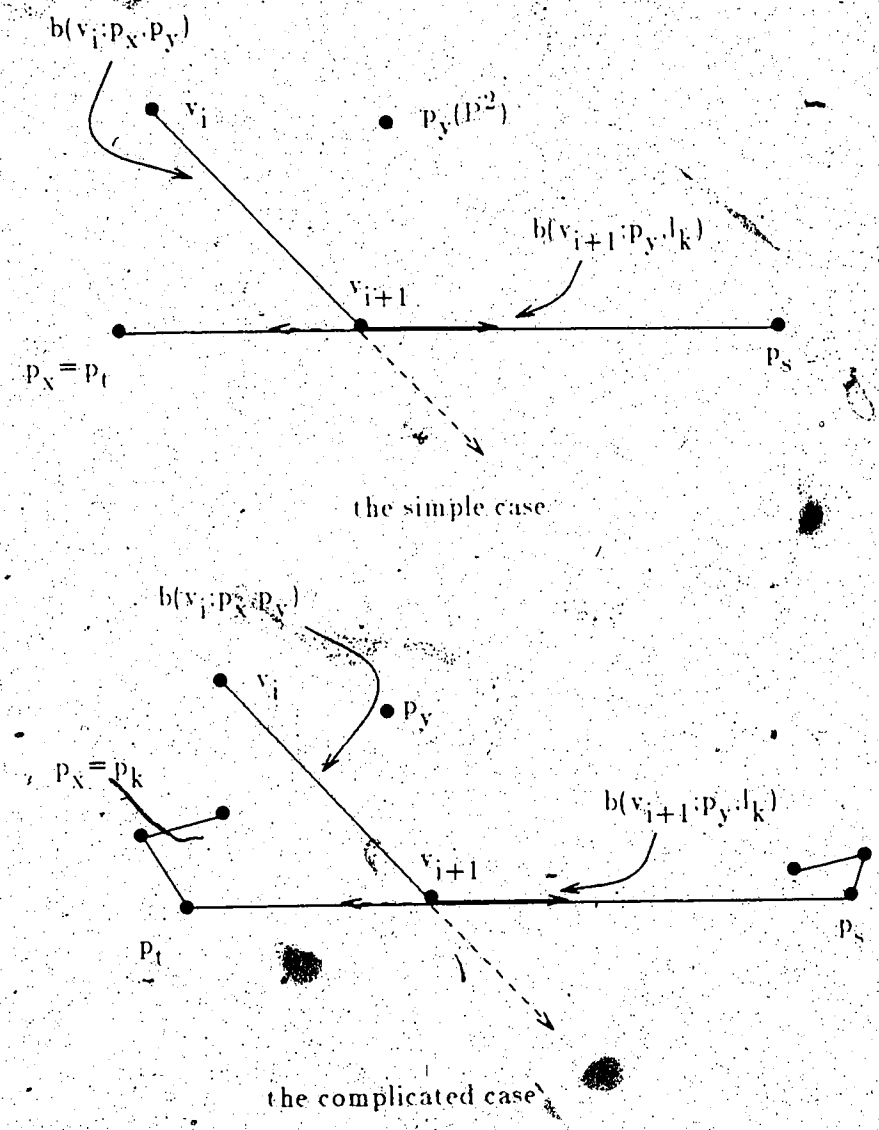


Figure 3.5 Case b.

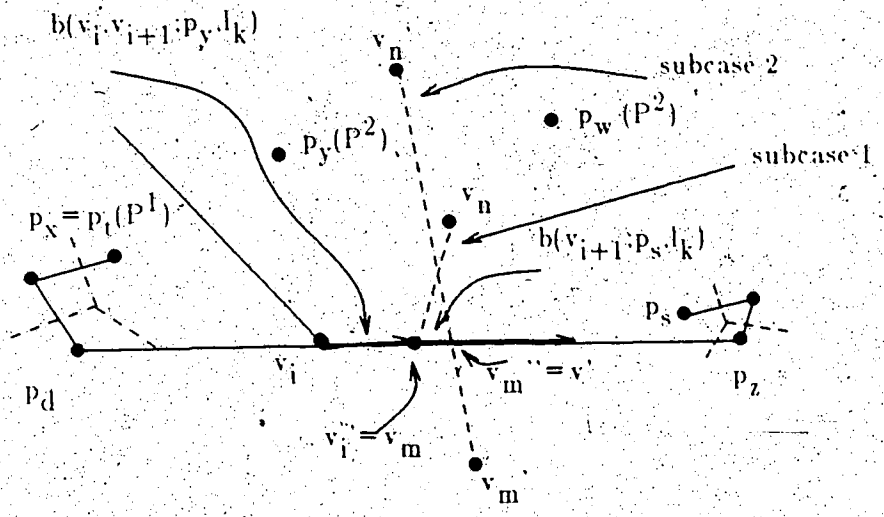
Case (b). Assume again that $b(v_i:p_x,p_y)$ is shared by a pair of data points, $p_x \in P^1$ and $p_y \in P^2$, so that $b(v_i:p_x,p_y)$ does not coincide with any segment of $C^1 \cup C^2$. But now assume that $b(v_i:p_x,p_y)$ meets an edge of the original BV- boundary which is a segment of an edge in $C^1 \cup C^2$ (shown in Fig.3.5).

First we discuss a simple case. Let this edge be $l_k = \overline{p_t p_s} \subset (C^1 \cup C^2)$, $p_x = p_t$, and the intersection point of $b(v_i:p_x,p_y)$ and $\overline{p_t p_s}$ be v_{i+1}. Note that $b(v_i:p_x,p_y)$ cannot

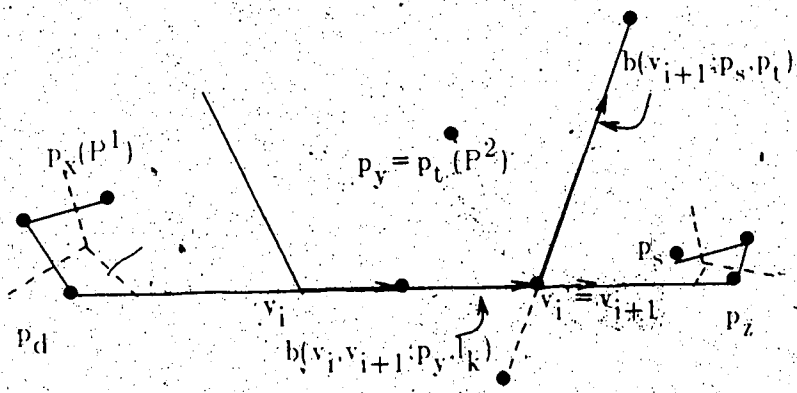
cross $\overline{p_s p_t}$ by Lemma 3.1. hence we ignore the segment of $b(v_i;p_x,p_y)$ which crosses $\overline{p_s p_t}$. We also ignore the segment of the edge from v_{i+1} to p_t, since this segment of the edge cannot contain a segment of the shared BV- boundary. To see this, assume that a segment of $\overline{v_{i+1}p_t}$ becomes an edge of the shared BV- boundary. This new shared BV-edge then must connect to another shared BV- edge leaving $\overline{p_t p_s}$ because any edge of shared BV- boundary cannot have a BV-vertex with degree one. Note that the leaving shared BV- edge must cross one of the BV- regions associated with p_y or with p_x. But, that implies there exists new BV- regions for p_x and for p_y other than the original BV- regions sharing $b(v_i;p_x,p_y)$, contrary to the assumption that $b(v_i;p_x,p_y)$ is the shared BV- ray determined by p_x and p_y. Now, we regard the ray starting at v_{i+1} and pointing to p_s as a new segment of the shared BV- boundary (a branch). We also replace the data points p_t by l_k. We call the operations described above, find the next branch". The original BV- regions of p_x, p_t, and p_y are repartitioned by $\overline{p_t v_{i+1}}$, $b(v_{i+1};p_y,l_k)$, and $b(v_i,v_{i+1};p_x,p_y)$ into two new BV-regions such that points on a new region are closer to one of data points p_t $(=p_x)$ and p_y than to the others. Note that p_x and p_y belong to the two different input chains and $b(v_i,v_{i+1};p_x,p_y)$ is the loci of points equidistant to p_x and p_y respectively, hence $b(v_i,v_{i+1};p_x,p_y)$ is a shared BV-edge. Moreover, the BV-region associated with $p_y \in \mathbf{P}^2$ is truncated by $b(v_{i+1};p_y,l_k)$ which is a portion of an edge belonging to $\mathbf{C}^1$, hence part of $b(v_{i+1};p_y,l_k)$ is a shared pseudo-edge.

In a more complicated case, $l_k = \overline{p_t p_s}$ may coincide with several original pseudo-edges from the same partition. Thus, in general, $p_x \neq p_t$ as shown in Fig.3.5b. When $b(v_i;p_x,p_y)$ meets $\overline{p_t p_s}$, we shall regard the ray starting at the intersection point v_{i+1} and pointing to p_s as the branch and l_k and p_y as the two elements determining the branch. Note that p_t and p_k belong to the same partition, hence we ignore the segment of the edge from v_{i+1} to p_t by the same argument as that in the above simple case. Then, $b(v_i,v_{i+1};p_y,p_k)$ and some initial segment of $b(v_{i+1};p_y,l_k)$ are segments of

the shared BV- boundary.



case 1



case 2

Figure 3.6 Case c.

Case (c). Assume that $b(v_i;p_y,l_k)$ coincides with an edge, say $l_k = \overline{p_d p_z}$ in $C^1 \cup C^2$. We also assume that $b(v_i;p_y,l_k)$ meets an original BV- edge, not coinciding with any

edge in $C^1 \cup C^2$ (refer to Fig.3.6).

Let the intersection point of $b(v_i;p_y,l_k)$ and the original BV-edge be v_i' (see subcase 1 of case 1). When we traverse $b(v_i;p_y,l_k)$ and reach v_i', we shall consider three cases.

(1) If l_k of $b(v_i;p_y,l_k)$ belongs to C^1 (respectively, C^2) and the original BV-edge, say $(v_m,v_n;p_s,p_t)$ also belongs to C^1 (respectively, C^2), then we continue to traverse $b(v_i;p_y,l_k)$ (or $b(v_i;p_w,l_k)$), and we ignore $(v_m,v_n;p_s,p_t)$ since $(v_m,v_n;p_s,p_t)$ no longer partitions $V(p_s)$ and $V(p_t)$ properly due to the existence of p_y (subcase 1). We also create a new BV-line $(:p_y,p_s)$ or $(:p_w,p_s)$ and find the crossover point with $\overline{p_dp_s}$ since when the shared BV-ray reaches the crossover point, it may leave $\overline{p_dp_s}$.

(2) If l_k of $b(v_i;p_y,l_k)$ belongs to C^1 (respectively, C^2) and the original BV-edge, say $(v_m,v_n;p_y,p_w)$ belongs to C^2 (respectively, C^1), we delete the part of it crossing $\overline{p_dp_s}$, replace $(v_m',v_n';p_y,p_w)$ by $(v_m'',v_n';p_y,p_w)$ since $(v_m'',v_n';p_y,p_w)$ still partitions $V(p_y)$ and $V(p_w)$ properly (subcase 2). We replace p_y by p_w, and continue to traverse $b(v_i;p_w,l_k)$.

(3) If $b(v_i;p_y,l_k)$ reaches the crossover point of the currently created BV-line, say $(:p_s,p_t)$, where $p_t \in \mathbf{P}^2$ and $p_s \in \mathbf{P}^1$, then we consider the following two subcases. If $p_y \neq p_t$, then we ignore the BV-line, and create a new BV-line $(:p_y,p_s)$. If $p_y = p_t \in \mathbf{P}^2$ and $p_s \in \mathbf{P}^1$, then the shared BV-ray will leave $\overline{p_dp_s}$. We regard the ray starting at the meeting point and along the leaving edge as the branch (see case 2, where v_{i+1} is the meeting point, and $b(v_{i+1};p_s,p_t)$ is regarded as the branch); ignore the segment of $b(v_i;p_y,l_k)$ from v_{i+1} pointing to p_s and ignore the part of $(:p_s,p_y)$ crossing $\overline{p_dp_s}$. By the same argument used in the case (b), $b(v_i,v_{i+1};p_y,l_k)$ and $b(v_{i+1};p_s,p_t)$ (or some initial segment of it) are the shared pseudo-edge and BV-edge. We call the operations described above as 'find the next branch'. (Note that $(:p_s,p_t)$ can be created in constant time when $b(v_i;p_w,l_k)$ reaches v_i' in cases (1) and (2).)

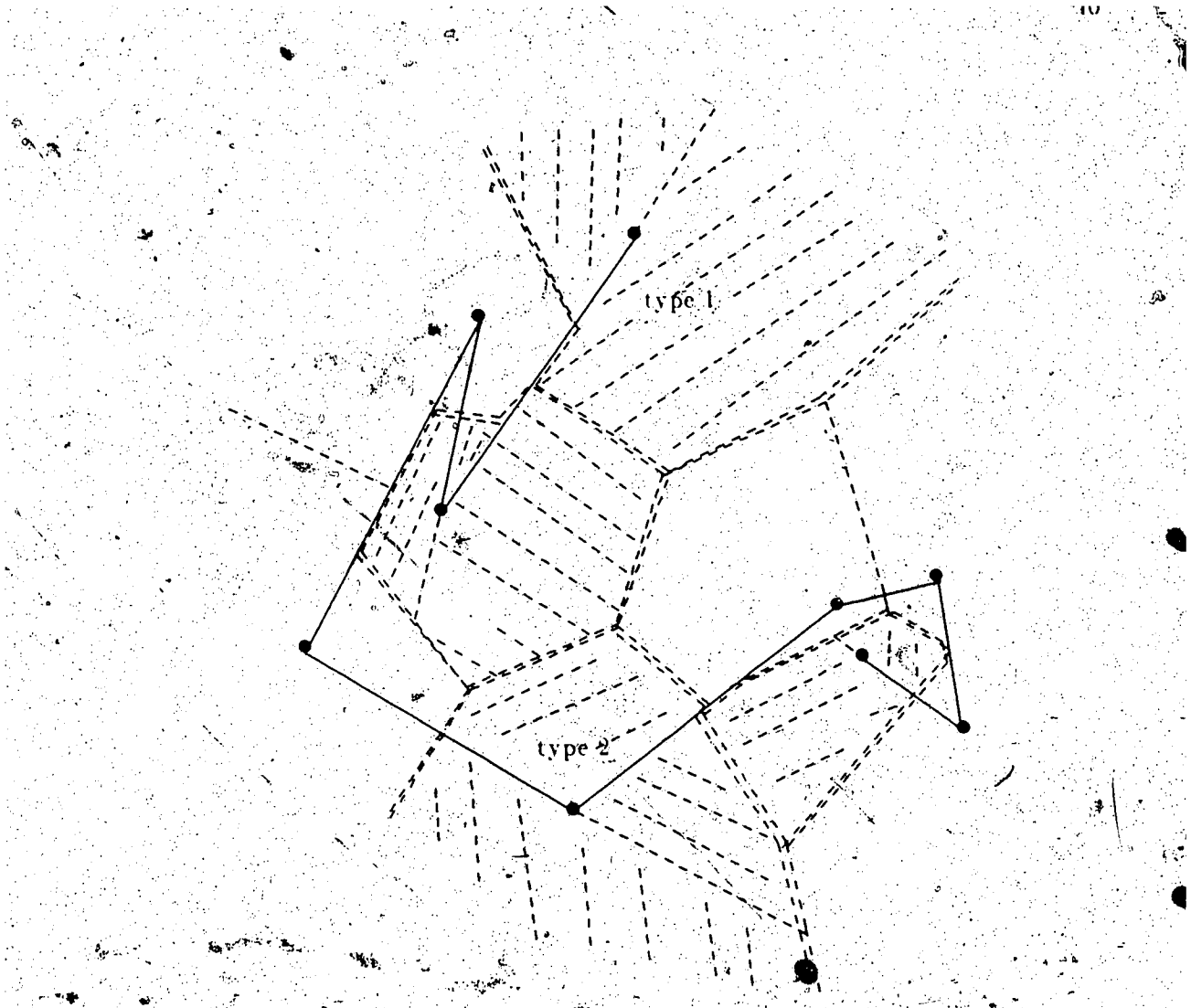


Figure 3.7 The different types of BV-regions and their convex sub-BV-regions.

Lemma 3.3: *Every BV-region of the BV-diagram of a finite chain can be partitioned into two or three convex polygons (closed or open) in time proportional to the size of the BV-region.*

Proof:

There exist exactly two types of data points in an input chain: extreme data points and non-extreme data points. If a data point is an extreme point, then the BV-region associated with that data point consists of one connected region. Otherwise, it consists of two connected sub-regions separated by the edges incident at

the data point (refer to Fig.3.7). The interior angle between a BV-edge and a pseudo-edge of a BV-region or sub-BV-region that share an BV-vertex is clearly less than 180°. The interior angle between two adjacent BV-edges of a BV-region or sub-BV-region must be less than 180°, since the two BV-edges are portions of the perpendicular bisectors of the associated data point and its neioghboring data points. Thus, if a vertex on the boundary of a BV-region or a sub-BV-region has an interior angle $> 180°$, then the reflex vertex must be the associated data point (the interior angle at an extreme data point can be considered to be 360°). Hence, if we extend one of the obstacles incident at the associated data point, the extending line will partition the BV-region or the sub-BV-region into two smaller convex subregions (either open or closed). The partition can be done in time proportional to the size of the BV-region or the sub-BV-region by scanning its boundary. □

Now, we are ready to present the merge algorithm for constructing the shared BV-boundary of two linearly separable chains.

In the algorithm, for simplicity, we always assume that all BV-regions of the original BV-diagrams have been partitioned into convex sub-BV-regions. This processing takes time proportional to the size of the original BV-diagrams. In the data structure, each data point is associated with two or three lists of BV-edges, pseudo-edges, and an additional line segment (or ray) (which partitions the BV-region or the sub-BV-region into convex sub-BV-regions). Each list represents the boundary of a convex sub-BV-region. The additional line segment is associated with two pointers pointing to the two lists (which represent the two convex BV-subregions sharing that additional line segment). Thus, in constant time, we can traverse from one convex sub-BV-region to the other convex sub-BV-region through the additional line segment.

Suppose C^1 and C^2 are linearly separable. (u,w,r,s) represents an unexamined

original BV- edge (or BV-ray or pseudo-edge). $(u,w;r,s) \in C^1 \cup C^2$ (or $b(v;p,q) \in C^1 \cup C^2$) denotes an edge of the original BV- diagrams (or an edge of the shared BV-boundary) coinciding with an edge in $C^1 \cup C^2$. (The negation of the relation is denoted by $\notin$). The phrase *find the next branch* means to find the next segment of the shared BV- boundary according to the above three cases as well as the subcases.

$b(v;p,q)$ with p and q as the associated data elements represents the ray of the shared BV- boundary which we currently traverse. The statement $b(v;p,q)$ = the branch' means replacing p or q of the original branch by the new data elements which share the new branch and activating the branch as the ray of shared BV- boundary which we are traversing. $CH(C^1 \cup C^2)$ denotes the convex hull of C^1 and C^2 (a linear figure).

The algorithm starts at the ray $b(v;p,q)$ which is a portion of the perpendicular bisector of $\overline{pq} \in CH(C^1 \cup C^2)$, where $p \in C^1$ and $q \in C^2$. v must lie on one of the BV-subregions of $V(p)$ and one of the BV-subregions of $V(q)$. Now, the algorithm finds which BV-boundary segment of $V(p)$ and $V(q)$ will first intersect $b(v;p,q)$. Note that each BV-subregion is convex and the shared BV-boundary is monotone, hence the algorithm can find the intersecting segment by scanning the boundaries of the two BV-subregions without backtracking (refer to Preparata and Shamos's book for details [Prep85]). If the segment first intersecting $b(v;p,q)$ is an additional line segment, the algorithm will continue to check for the intersection of $b(v;p,q)$ and the boundary of the BV-subregion sharing that additional segment (in this case, no branch is created and no data element is replaced). If the segment first intersecting $b(v;p,q)$ is a BV-edge or a pseudo-edge, then the algorithm will find the new branch according to the cases we described previously. The new branch is regarded as the shared BV-ray. When an intersection is found, the previous branch becomes a BV-element of the resultant BV-diagram, and an appropriate portion of the original BV-element is deleted. In this

manner, the algorithm merges the BV-diagrams. The algorithm will terminate when $\overline{pq}$ is an edge of the convex hull.

Algorithm 1

Input: $Vor(\mathbf{P}^1; C^1), Vor(\mathbf{P}^2; C^2), C^1, C^2, CH(C^1), CH(C^2)$

Output: $Vor(\mathbf{P}^1 \cup \mathbf{P}^2; C^1 \cup C^2)$

Method:

1. find the starting segment of the shared BV- boundary, say $b(v;p,q)$ by the 'convex hull method';
2. *Repeat*
   (a) find the BV-edge (or BV-ray or pseudo-edge) $(u,w;r,s)$, which first intersects $b(v;p,q)$;
   (b) find the next branch according to the following cases:
   $b(v;p,q) \circ C^1 \cup C^2$ meets $(u,w;r,s) \phi C^1 \cup C^2$
   or $b(v;p,q) \phi C^1 \cup C^2$ meets $(u,w;r,s) \phi C^1 \cup C^2$
   or $b(v;p,q) \phi C^1 \cup C^2$ meets $(u,w;r,s) \circ C^1 \cup C^2$;
   (c) $b(v;p,q) \leftarrow$ the branch;

Until ($\overline{pq} \in CH(C^1 \cup C^2)$).

Lemma 3.4: *Given $Vor(\mathbf{P}^1; C^1)$ and $Vor(\mathbf{P}^2; C^2)$, where $\mathbf{P}^1$ and $\mathbf{P}^2$ are the two vertex sets of C^1 and C^2 respectively, and C^1 and C^2 are two linearly separable input chains, and also given $CH(C^1)$ and $CH(C^2)$, Algorithm 1 takes $O(|\mathbf{P}^1 \cup \mathbf{P}^2|)$ time to produce $Vor(\mathbf{P}^1 \cup \mathbf{P}^2; C^1 \cup C^2)$.*

Proof:

(1) Algorithm 1 produces the shared BV- boundary of $C^1 \cup C^2$.

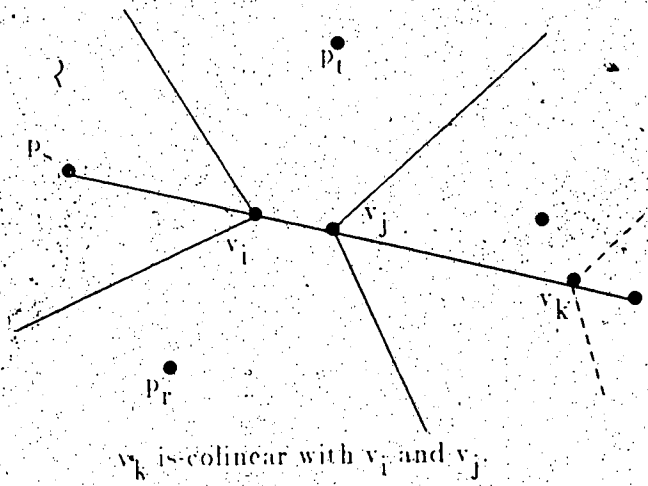
Since C^1 and C^2 are linearly separable, constructing the shared BV-boundary of them is equivalent to merging the BV-diagrams of them by Lemma 3.2, and the shared BV- boundary is an endless open BV- chain by

Lemma 3.1. Note that the three cases as well as the subcases described previously show that each branch created by the algorithm forms the locus of points equidistant to the closest data points of two input chains or it is a part of an edge of one input chain and which truncates the original BV-region of a data point of the other input chain. Note that the visibility requirement is satisfied. Thus, each segment created by the algorithm satisfies the definition of shared BV- boundary. The while loop of Algorithm 1 exhausts all segments of the shared BV- boundary since the linear figure is connected. Thus, Algorithm 1 merges the two BV- diagrams.

(2) Algorithm 1 takes $O(|\mathbf{P}^1 \cup \mathbf{P}^2|)$ to construct the shared BV- boundary. Each original BV- edge of the BV- diagrams can be intersected by the shared BV- boundary at most twice. To see this, assume that an original BV-edge is intersected by the shared BV- boundary more than two times. Note that each intersection point must be a BV- vertex of the resultant BV-diagram. The assumption implies there are more than two co-linear BV-vertices on the BV-edge, contrary to the property that a BV- region can not have more than two consecutive co-linear vertices on a BV-edge. Furthermore, the shared BV- boundary can overlap with an edge of the input chains $C^1 \cup C^2$ at most once. To see this, note that the two vertices of an edge must belong to the same input chain since the two input chains are linearly separable. Thus, only an even number of segments of shared BV- boundary can meet that edge since all the shared bound BV-vertices (which are not the data points) must be of degree two. Note also that the shared BV-boundary does not cross any edge of C^1 and C^2 by Lemma 3.1, hence the number must be either zero or two (see Figure 3.8). Thus, the shared BV- boundary can intersect the original BV- edges and coincide with the edges of $C^1 \cup C^2$ at most $O(|\mathbf{P}^1 \cup \mathbf{P}^2|)$ times. Now, the remaining problem is to show that

each segment of shared BV- boundary can be constructed in amortized constant time. Note that we have attached three pieces of information to each element in p-V, thus, updating the associated data element of a branch and determining if a segment of shared BV- boundary coincides with an edge take constant time. Since each BV-subregion is convex, by the method shown in [Prep85, pp.211-214], all the original BV- edges or pseudo-edges intersecting the shared BV- boundary can be found in time proportional to $O(|P^1 \cup P^2|)$. The time for solving the starting and termination problems is bounded by $O(|P^1 \cup P^2|)$ if the convex hulls of the two input chains are given. Hence, the shared BV- boundary can be constructed in time proportional to $O(|P^1 \cup P^2|)$.

(3) During the merging, we can delete the appropriate portions of all the original BV-edges and pseudo-edges which intersect the shared BV-boundary or whose spokes intersect the shared BV-boundary in time proportional to the size of the input chains. We can add the segments of the shared BV-boundary to the original BV-diagrams by a single scan on the original BV-regions during the construction of the shared BV-boundary. □



v_k is colinear with v_i and v_j.

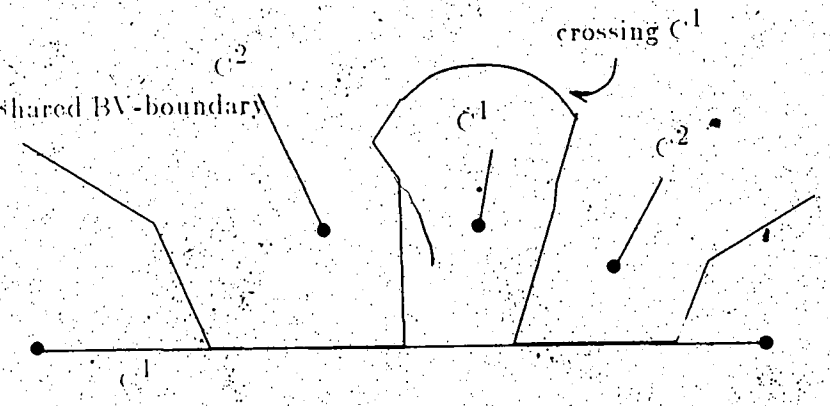


Figure 3.8. The above two cases cannot occur.

3.3. Finding the BV-diagram of two pseudo-linearly separable chains

In the case that two input chains are pseudo-linearly separable, the merge procedure is more complicated than that of two linearly separable input chains. The complication is due to the shared data points of the two input chains. If two input chains share a data vertex, then by definition, the boundary of the resultant BV- region associated with that shared data point must belong to the shared BV- boundary of the two input chains. The resultant BV- region is the overlapped portion of the two original BV- regions associated with that data point (see the shaded area of Fig.3.9).

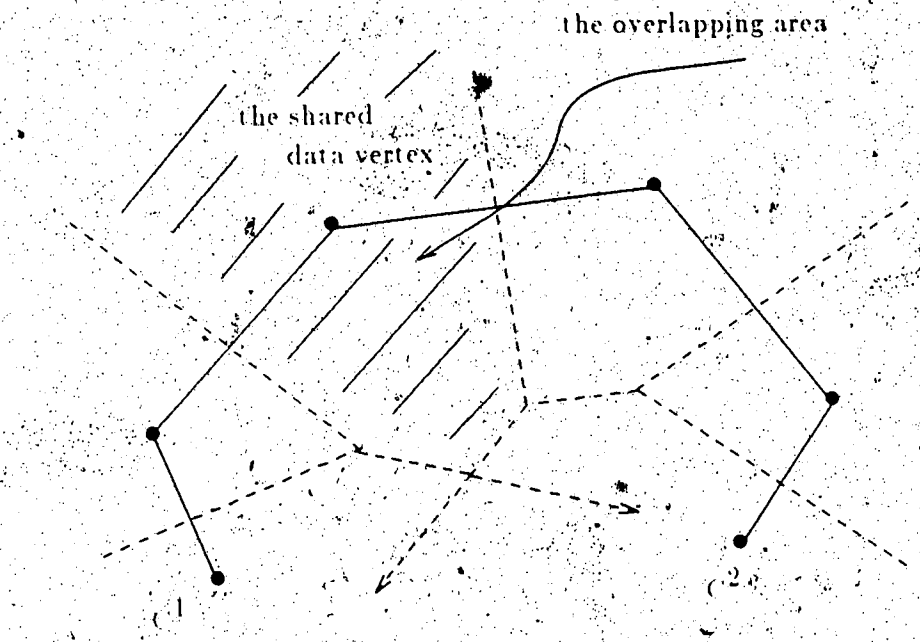


Figure 3.9 The resultant BV- region of a shared data point (shaded).

In the following, we shall first show what the shared BV- boundary of two pseudo-linearly separable input chains is. We then present the algorithm to construct it.

Definition: A *traversal* of a finite linear figure is a sequence of vertices $v_1, \ldots, v_n$ of the figure such that $\{\overline{v_i v_{i+1}} \mid 1 \le i < n\}$ is the set of edges of the figure. (Note that 're-traversal' of edges is allowed.) The notion of traversal can be extended in an intuitively obvious way to apply to infinite figures, by allowing fictitious vertices at infinity among the v_i. A linear figure has a traversal iff it is connected. A traversal $v_1, \ldots, v_n$ of a linear figure *does not cross* obstacle **L** (a point set, such as a linear figure) iff no subfigure $\bigcup_{i=k}^{m} \overline{v_i v_{i+1}}$, $1 \le k \le m < n$, crosses **L**. A linear figure **F** is *subdivided* (*relative to obstacle* **L**) iff there exists a traversal of **F** which does not cross **L**.

[redacted]na 3.5, we shall show that the shared BV-, boundary of two pseudo-linearl[redacted]able input chains (sharing exactly one of their extreme data points) forms

two undivided linear figures separated by the union of the input chains. Each undivided linear figure is either an infinite chain or a polygon boundary and a semi-infinite chain with a vertex in common. In Lemma 3.6, we shall show that the common vertex (connecting point) of the polygon boundary and the semi-infinite chain can be found in time logarithmically proportional to the size of the p_i-V^1 and p_i-V^2, where p_i is the common data point of the two chains, and p_i-V^1 and p_i-V^2 represent the boundaries of the two original BV- regions of p_i, each BV- region belonging to a different chain.

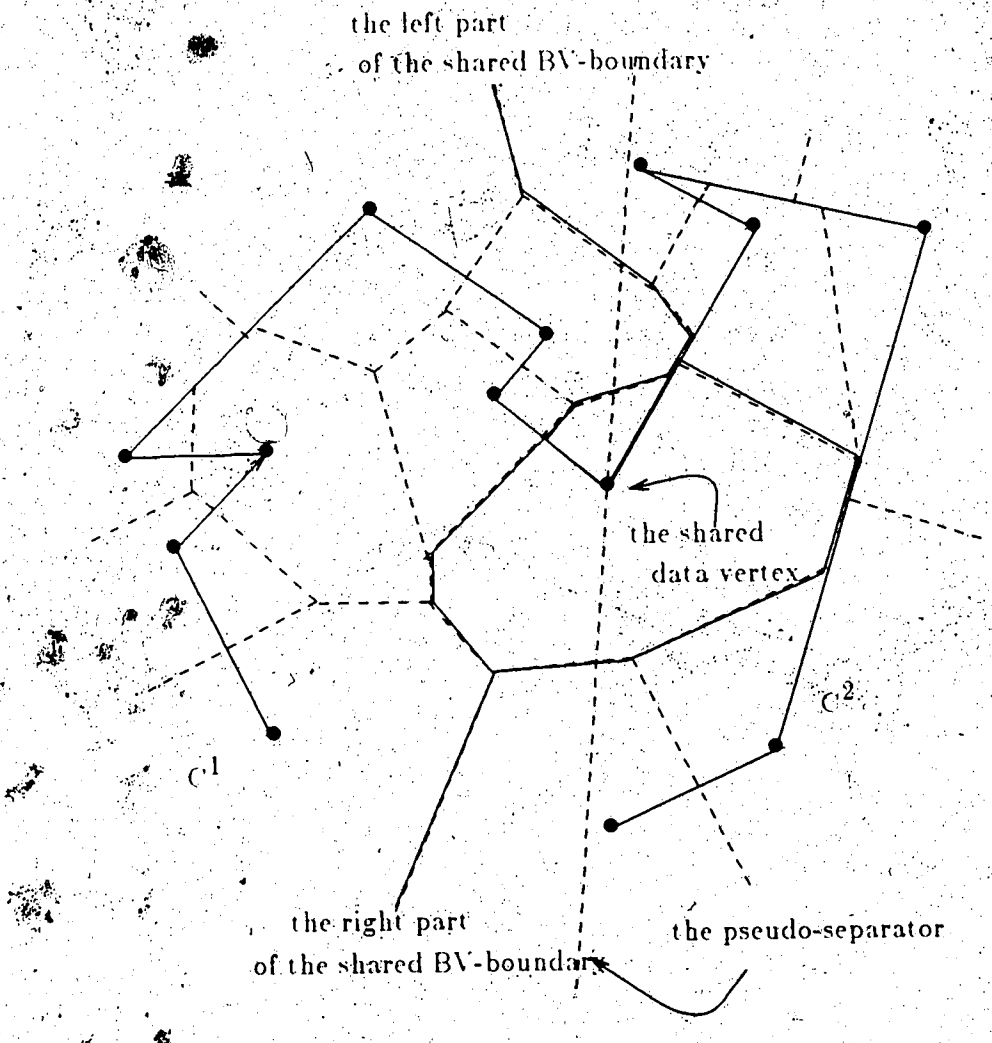


Figure 3.10 The shared BV- boundary of two chains.

Lemma 3.5. *The shared BV- boundary of C^1 and C^2 sharing exactly one of their extreme data points consists of two undivided linear figures separated by $C^1 \cup C^2$, where C^1 and C^2 are two pseudo-linearly separable input chains.*

Proof:

We prove this by a perturbation method. Let the shared data point of C^1 and C^2 be p_i. We first perturb C^1 and C^2 from any one of the separators such that p_i is split into two sufficiently close data points $p_i' \in C^1$ and $p_i'' \in C^2$. If the separator contains other data points of C^1 and C^2, then these data points must also be perturbed slightly in the appropriate direction. Thus, the new chains $C^{1'}$ and $C^{2''}$ are linearly separable. The shared BV-boundary of the new chains is an endless open chain which does not cross $C^{1'}$ and $C^{2''}$ by Lemma 3.1. We then undo the perturbation in the resultant BV-diagram. This will involve the removal of the BV-edge shared by p_i' and p_i'' in the resultant BV-diagram and the addition of the rest of the boundaries of $V(p_i')$ and $V(p_i'')$ in the resultant BV-diagram to the shared BV-boundary of $C^{1'}$ and $C^{2''}$. The result of undoing will be the shared BV-boundary of C^1 and C^2. Note that the shared BV-boundary of $C^{1'}$ and $C^{2''}$ must connect to the boundaries of $V(p_i')$ and $V(p_i'')$ in the resultant diagram. Note also that the removal of the BV-edge shared by p_i' and p_i'' does not change the connectivity among the boundaries of $V(p_i')$ and $V(p_i'')$ and the shared BV-boundary of $C^{1'}$ and $C^{2''}$ except that we now regard p_i' and p_i'' as p_i. In other words, the boundaries of $V(p_i')$ and $V(p_i'')$ after the removal of the BV-edge shared by p_i' and p_i'' now become the boundary of $V(p_i)$.

(1) The shared BV- boundary of C^1 and C^2 consists of two undivided linear figures separated by $C^1 \cup C^2$. To see this, note that the union of the two chains $C^1 \cup C^2$ becomes one obstacle. Thus, although the shared BV- boundary of C^1

and C^2 may lie on both sides of some edge of $C^1 \cup C^2$ and some shared pseudo-edges and shared bound BV-vertices may lie on the input edges, it does not cross $C^1 \cup C^2$ in terms of an undivided linear figure. Note also that the extreme segments of the shared BV-boundary of $C^{1'}$ and $C^{2''}$ are rays, and they do not intersect. Thus, they do not intersect after undoing the perturbation.

(2) The part of the shared BV- boundary lying on the either side of $C^1 \cup C^2$ is either a polygon boundary and a semi-infinite chain with a vertex in common, or an infinite chain through the shared data point of the input chains.

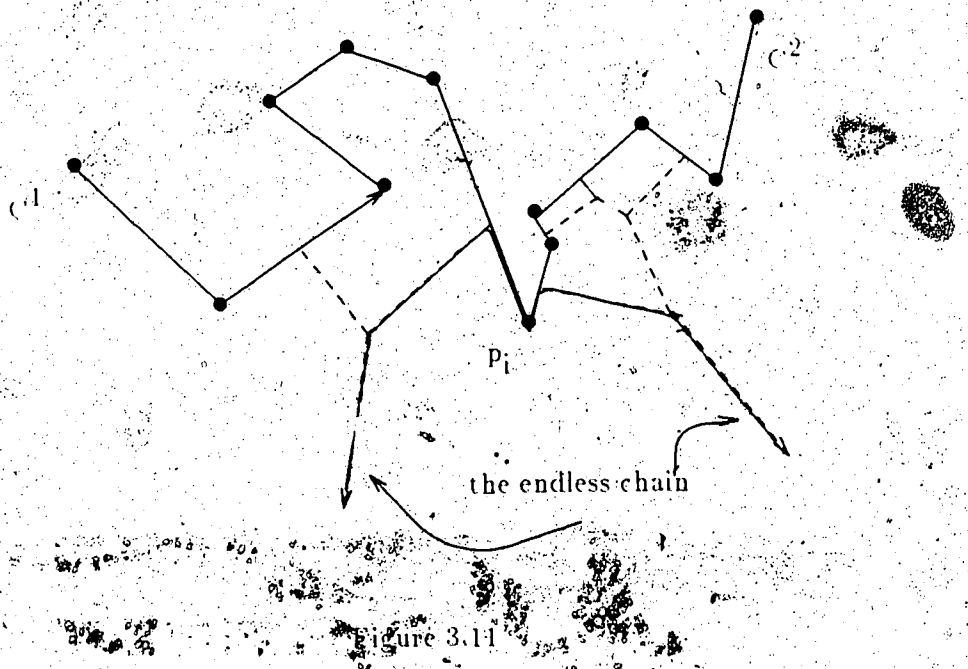


Figure 3.11

(a) If the shared vertex p_i belongs to the convex hull of $C^1 \cup C^2$, then one of the two subregions of $V(p_i)$ separated by $C^1 \cup C^2$ must be open. The shared BV-boundary is the boundary of this open subregion (refer to Fig.3.11, where one part of the shared BV- boundary is an infinite BV-chain).

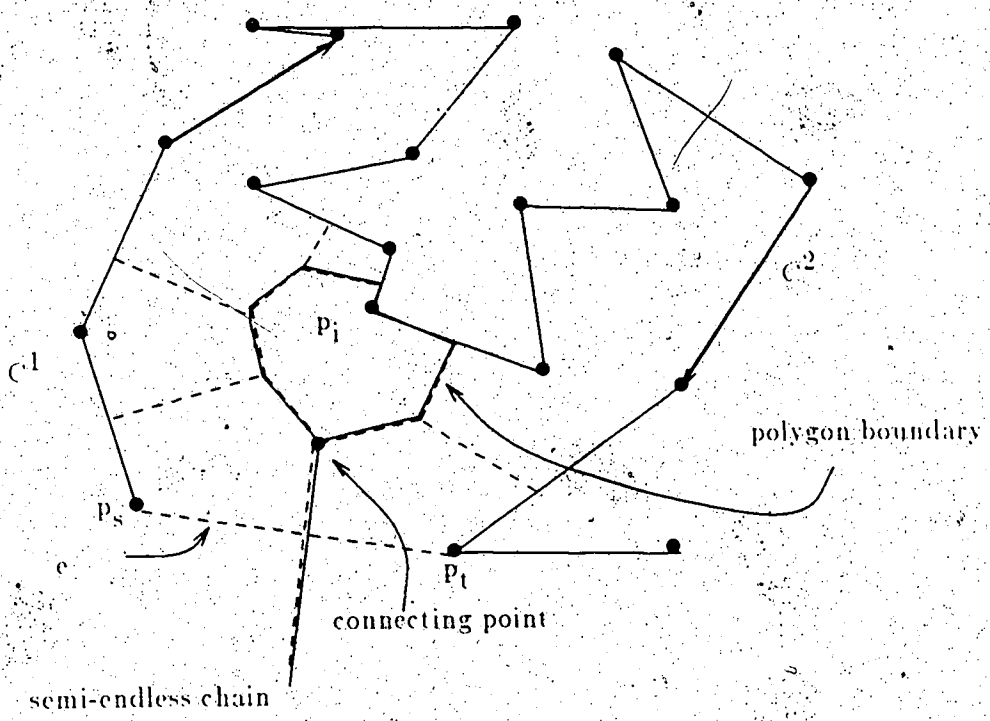


Figure 3.12

(b) If p_i does not belong to the convex hull of $C^1 \cup C^2$, then both subregions of $V(p_i)$ are closed (refer to Fig.3.12). If p_i belongs to the convex hull, then one of the two subregions separated by $C^1 \cup C^2$ must be closed (refer to Fig.3.11). Moreover, the undoing of the perturbation does not change the connection between the boundaries of $V(p_i')$ and $V(p_i'')$ and the shared BV-boundary of $C^{1'}$ and $C^{2''}$ after the removal of the BV-edge shared by p_i' and p_i'', except that $V(p_i')$ and $V(p_i'')$ become $V(p_i)$. Note that the subregion of $V(p_i)$ is closed, and the shared BV-boundary of $C^{1'}$ and $C^{2''}$ after the removal is a semi-infinite BV-chain. The two parts must connect. Thus, the shared BV-boundary must be a polygon boundary and a semi-infinite chain with a vertex in common. □

Lemma 3.5 shows that the shared BV- boundary of two pseudo- linearly separable

input chains sharing exactly one of their extreme data points must be two undivided linear figures. In order to construct them in linear time, we must be able to find the starting segment of the shared BV- boundary in at most linear time.

In the case that the undivided linear figure is an infinite chain, the chain must pass through the shared data point p_i and all the vertices in the chain are of degree two. We regard the pseudo- edges incident at p_i as the two starting edges of the shared BV- boundary. Thus, the rest of the chain can be constructed in linear time by a single traversal of each semi- infinite chain. (Clearly, the BV- edges incident at p_i can be found in constant time if p_i is given.)

In the case that the undivided linear figure is a polygon boundary and a semi-infinite chain with a point in common, we must be able to find the connecting (common) point in at most linear time to ensure the figure can be constructed in linear time. Note that by Lemma 3.5, the connecting point must be shared by $V(p_i)$ and the semi-infinite chain, and it is a vertex of the shared BV-boundary. Thus, it must be a BV-vertex (free or bound) of $V(p_i)$ since the shared BV-boundary must be a subset of the BV-diagram. Thus, the connecting point must be the intersection point of the two original BV-region boundaries of p_i. We shall see in the following lemma that the intersection point of the two original BV-region boundaries of a shared vertex p_i can be found in logarithmic time.

Lemma 3.6: *Given the boundaries of $V^1(p_i)$ and $V^2(p_i)$, where p_i is shared by two pseudo-linearly separable chains C^1 and C^2, and p_i is the only point lying on a separator, then the intersection points of the boundaries of $V^1(p_i)$ and $V^2(p_i)$ can be found in time logarithmically proportional to the size of p_i - V^1 and p_i - V^2.*

Proof:

We only show how to find the intersection point of the two BV- region boundaries of p_i lying on one side of $C^1 \cup C^2$ in the case described in the paragraph before this lemma, because the analysis for the two sides are similar. Note that only p_i lies on a separator, therefore no BV-edge from $V^1(p_i)$ will overlap some BV-edges from $V^2(p_i)$. Note also that p_i lies on the convex hull of C^1 and the convex hull of C^2, hence $V^1(p_i)$ and $V^2(p_i)$ must be open. Moreover, the extreme edges(rays) from the different BV-regions are pointing in 'opposite' directions so that there exists exactly one intersection point. Now, note that the (bound or free) BV- vertices in $V^1(p_i)$ and $V^2(p_i)$ are ordered according to the polar angles originating at p_i, yielding an effective order for the BV-edges and pseudo-edges. Thus we can find the intersection point in time logarithmically proportional to the size of p_i-V^1 and p_i-V^2 by applying a binary search on the boundaries using the method of Chazelle and Dobkin [Chaz80b]. □

The 'branch' of the shared BV- boundary can be determined in constant time according to the three cases shown before.

By an argument similar to Lemma 3.2, it can be seen that merging the BV-diagrams of two pseudo- linearly separable input chains is equivalent to finding the shared BV-boundary of them. Based on Lemma 3.3, Lemma 3.5 and Lemma 3.6, we can design an algorithm to merge the BV- diagrams of two pseudo-linearly separable chains sharing exactly one of their extreme data points in linear time.

The main idea of our algorithm is as follows:

(1) Locate the starting segment of the shared BV- boundary on each side. We first find the branches by examining the intersection point (if any) of the original BV-region boundaries of the shared data point. (If no intersection point is found, we regard the BV-region boundary of the shared data point as the shared BV-boundary.)

(2) Construct the successive shared BV- edges (pseudo-edges) in the order determined by their interconnections in each branch.

(3) Terminate the construction of each branch. Note that each part of shared BV-boundary terminates as a ray, which is determined by a pair of vertices belonging to an edge of the convex hull of the input chains.

In the algorithm, the phrase 'find the next branch' means to do the the following operations : find the intersection point of two meeting segments; ignore the appropriate segments of the two intersecting line segments, update the associated data points as well as decide the next branch. The phrase 'find the branches around the shared data point' means to find the starting segment according to the two types of linear figures. All the notations and the assumptions used in Algorithm 2 follow those in Algorithm 1 except that C^1 and C^2 are pseudo-linearly separable, s represents the data point shared by the two input chains.

Algorithm 2

Input : $\mathrm{Vor}(\mathbf{P}^1; C^1)$ and $\mathrm{Vor}(\mathbf{P}^2; C^2)$, C^1 and C^2, $\mathrm{CH}(C^1)$ and $\mathrm{CH}(C^2)$, s.

Output: $\mathrm{Vor}(\mathbf{P}^1 \bigcup \mathbf{P}^2; C^1 \bigcup C^2)$.

Method:

1 If $s \in \mathrm{CH}(C^1 \bigcup C^2)$, then find the infinite BV-chain by traversing the boundary of the open subregion of $V(s)$;

find the one or two branches around s, say b_1 and possibly b_2.

2 *For* $b = b_1$, (possibly) b_2 *Do*

(a) $b(v; p, q) \leftarrow b$;

(b) *Repeat*

(1) find the BV-edge (BV-ray, pseudo-edge), say $(u, w; r, s)$, which

first intersects b(v:p,q):

(2) find the next branch according to the following cases:

b(v:p,q) $\circ$ $C^1 \cup C^2$ meets (u,w:r,s) ϕ $C^1 \cup C^2$

or b(v:p,q) ϕ $C^1 \cup C^2$ meets (u,w:r,s) ϕ $C^1 \cup C^2$

or b(v:p,q) ϕ $C^1 \cup C^2$ meets (u,w:r,s) $\circ$ $C^1 \cup C^2$;

(3) b(v:p,q) – the branch:

Until $\overline{pq} \in CH(C^1 \cup C^2)$

End-Do

Lemma 3.7: *Given* $Vor(\mathbf{P}^1;C^1)$ *and* $Vor(\mathbf{P}^2;C^2)$, *where* $\mathbf{C}^1$ *and* $\mathbf{C}^2$ *are the two pseudo-linearly separable input chains sharing one of their extreme data points and* P^1 *and* P^2 *are their vertex sets respectively, Algorithm 2 takes* $O(|\mathbf{P}^1 \cup \mathbf{P}^2|)$ *time to produce* $Vor(\mathbf{P}^1 \cup \mathbf{P}^2;C^1 \cup C^2)$.

Proof:

Note that merging the BV- diagrams of C^1 and C^2 is equivalent to finding the shared BV- boundary between them by a slightly modified Lemma 3.2. By Lemma 3.5 and Lemma 3.6, the shared BV- boundary consists of two undivided linear figures, and the starting segments of the figures can be found in sublinear time. By a proof similar to the one of Lemma 3.4, Algorithm 2 exhausts all segments of the shared BV- boundary of C^1 and C^2, and the time complexity is $O(|\mathbf{P}^1 \cup \mathbf{P}^2|)$. □

Obviously, if we recursively bi-partition a monotone chain, then any two half subchains are pseudo- linearly separable, i.e., they share only a vertex on the pseudo-separator. Note that the start of constructing the shared BV- boundary can be found in linear time, the connecting point can be found in logarithmic time, hence it is easy

to design a divide and conquer algorithm to obtain the BV- diagram of a monotone chain with n vertices in $O(n \log n)$ time.

Theorem 3.1: *Given a monotone chain C with n vertices, the BV- diagram of C can be constructed in $O(n \log n)$ time.*

Proof: By a divide and conquer method. □

Definition: A simple polygon P is *monotone* with respect to one of its edges e_i if the order of the perpendiculars of the vertices of P to e_i is the same order as that of the vertices in P.

Corollary 3.1: *The BV- diagram of an n-gon P that is monotone with respect to one of its edges e_i (which is given) can be found in $\Theta(n \log n)$ time.*

Proof:

We first divide the boundary of P into the edge e_i and the boundary C-$\{e_i\}$. Using the monotonic property of C-$\{e_i\}$, we can apply a divide and conquer method to C-$\{e_i\}$ for constructing the BV- diagram in $O(n \log n)$ time. Then, we add e to C-$\{e\}$ to form a polygon, we ignore all BV- edges crossing e_i and regard the perpendicular bi-sector ray of e_i as the left BV- edge of e_i. The algorithm is optimal [Agg87]. □

Chapter 4

Finding the BV-diagram of a simple polygon.

In this chapter, we shall discuss a special case of BV-diagrams, namely, the BV-diagram for a simple polygon P. Many simple polygon problems in Computational Geometry such as the closest pair of vertices, all nearest neighbors, finding the Delaunay triangulation of a simple polygon P can be solved efficiently if the BV-diagram of P is available. The efficiency of solving these problems relies on the fast construction of the BV- diagram of a simple polygon. In the following, we consider a divide and conquer method to construct the diagram. We shall first show the basic idea of our method for constructing the BV- diagram of a simple polygon P, then show how to merge the BV- diagrams of two sub-polygons of P.

4.1. The basic idea for finding the BV-diagram of P.

When we apply a divide and conquer method to P to construct its BV- diagram, we need to design a merge procedure. Clearly, we will face the same three problems as we have mentioned in Chapter 3.

The critical point for designing a merge procedure is how fast we can solve the visibility problem. Solving this problem in the case of a simple polygon is not trivial. Note that two boundary segments of a simple polygon may not be pseudo-linearly separable due to nonconvexity. When we merge two boundary segments, the visibility requirement may not be satisfied.

Our method to solve the visibility problem is based on the following observation. The BV-boundary of a simple polygon P consists of the interior and the exterior parts, where each part is an undivided linear figure. The interior part is a collection of closed polygon boundaries and the exterior is a collection of open and closed polygon boundaries. These two parts can be independently constructed since they are separated by the boundary of P. Moreover, the convex hull of P and the polygon boundary B of P

form several closed regions, each of which is a smaller simple polygon containing an edge of the convex hull (not belonging to B of P). The exterior part of the BV- boundary of P is the collection of BV- boundaries of these smaller polygons (we shall define the BV-boundary, which is called pseudo- BV- boundary, of a smaller polygon later).

The BV- boundary of each smaller simple polygon can be constructed independently too. To see this, note that the BV- edges of crossing an edge of the convex hull do not intersect the BV- edges crossing the other edges of the convex hull due to the convexity. The BV- boundaries of the simple small polygons are separated by B. Thus, we can construct the BV- boundary of P by independently constructing the interior of the BV- boundary of P and the BV-boundaries of the smaller polygons.

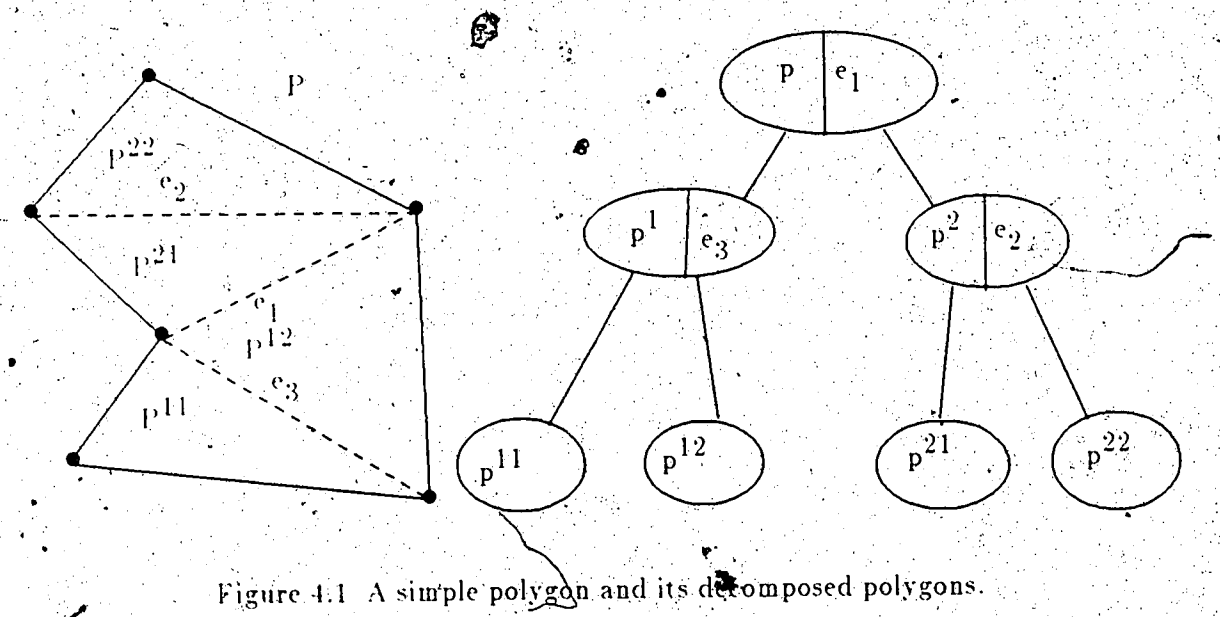


Figure 4.1 A simple polygon and its decomposed polygons.

Our method for constructing the interior of the BV- boundary of P is based on Chazelle's *polygon cutting theorem* [Chaz82]. An n-gon P can be recursively decomposed into a set of subpolygons in $O(n \log n)$ time such that a parent polygon is separated by a *cutting line segment* into a pair of child polygons and each child polygon contains at least one third of the vertices of the parent polygon. Obviously,

the smallest subpolygons are triangles. An example of decomposing a polygon into a set of smaller polygons is shown in Fig.4.1. Note that the depth of recursion is $O(\log n)$ if we recursively decompose an n-gon. Thus, if we can merge the BV- boundaries (which are called 'pseudo- BV-boundaries') of two subpolygons in time proportional to the size of their parent polygon, we obtain an $O(n \log n)$ algorithm for constructing the interior of the BV- boundary of P.

The exterior BV-boundary of P can be obtained by constructing the BV-boundaries of those smaller polygons by the same method as constructing the interior of the BV-boundary of P. Note that each data point can be assigned to at most two smaller polygons, and the BV-boundary of a smaller m-gon can be constructed in O(m log m) time. Thus, the exterior of the BV-boundary of P can be constructed in O(n log n) time too.

4.2. Merging the pseudo BV-diagrams of two subpolygons of P.

In this section, we first give some definitions (including 'pseudo-BV-diagrams') and lemmas; we then show that the shared pseudo-BV- boundary of two subpolygons (see the definition below) sharing an edge is an undivided linear figure; we finally present the merge procedure.

Definition: An *open polygon* P° is a polygon P along with a subset of edges of P which are regarded as obstacles. An edge (without its endpoints) of P° which does not belong to the set of obstacles is called an *open edge* of P°. We denote the set of open edges of P° as E°. The non-open edges are called *obstacle edges*.

Definition: The *pseudo- interior* BV- diagram of P° is the interior part of the BV- diagram of P (i.e., $\{V(p_i)|p_i \in P\} \bigcap R'(P)$, where $R'(P)$ consists of the interior of P and the non-open edges of P). The *pseudo- BV-boundary* of P° is the interior part of the BV-boundary of P (after removal of the pseudo-edges coinciding with the edges of E°).

Let $H(e_i)$ denote a half-plane lying on right hand side of the straight line extending edge e_i, and let $\overline{H}(e_i)$ denote the other half-plane.

Definition : A *pseudo*-BV- diagram of an open polygon P° contains two types of diagrams: the pseudo- interior BV- diagram of P° and a set of *pseudo-exterior* BV-diagrams of P°. A pseudo-exterior BV- diagram with respect to an open edge e_i is a collection of BV- regions. $Vor(P^\circ : B-\{e_i\}) = \{V(p_j) \mid p_j \in (P \bigcap H(e_i))\}$, where

$$V(p_j) = \{x \mid d_{B-\{e_i\}}(x, p_j) \leq d_{B-\{e_i\}}(x, p_k),\ x \in \overline{H}(e_i),\ p_k \in (P \bigcap H(e_i))\}$$

and $d_{B-\{e_i\}}(x, p_j)$ is the straight line distance between x and p_j with $B-\{e_i\}$ as obstacles and B is the boundary of polygon P. Clearly, the pseudo-exterior BV- diagram lies in $\overline{H}(e_i)$.

A clearer alternative view of the pseudo-BV-diagram of an open polygon P° is as follows. Given an open polygon P°, we can construct a specific surface, which is a set of half-planes (sheets) connecting to the plane on which the open polygon lies. We call $\overline{H}(e_i)$ an *edge induced half-plane* (induced by e_i). For each open edge of P°, we attach the corresponding half-plane on the straight line extending the open edge such that line segments crossing the open edge will lie on the half-plane. The straight line distance defined in Chapter 2 is defined in the union of the half-plane and the open polygon. Note that line segments crossing different open edges will enter different half-planes. Clearly, the BV-diagram of an open polygon is defined on the specifically constructed planes. In contrast to the previous definition, the pseudo- interior BV-diagram is the portion of the BV-diagram (which is defined on the edge induced half-planes and the main plane) lying on the main plane, each pseudo-exterior BV-diagram is the portion of the BV-diagram lying on the half-plane induced by the corresponding open edge.

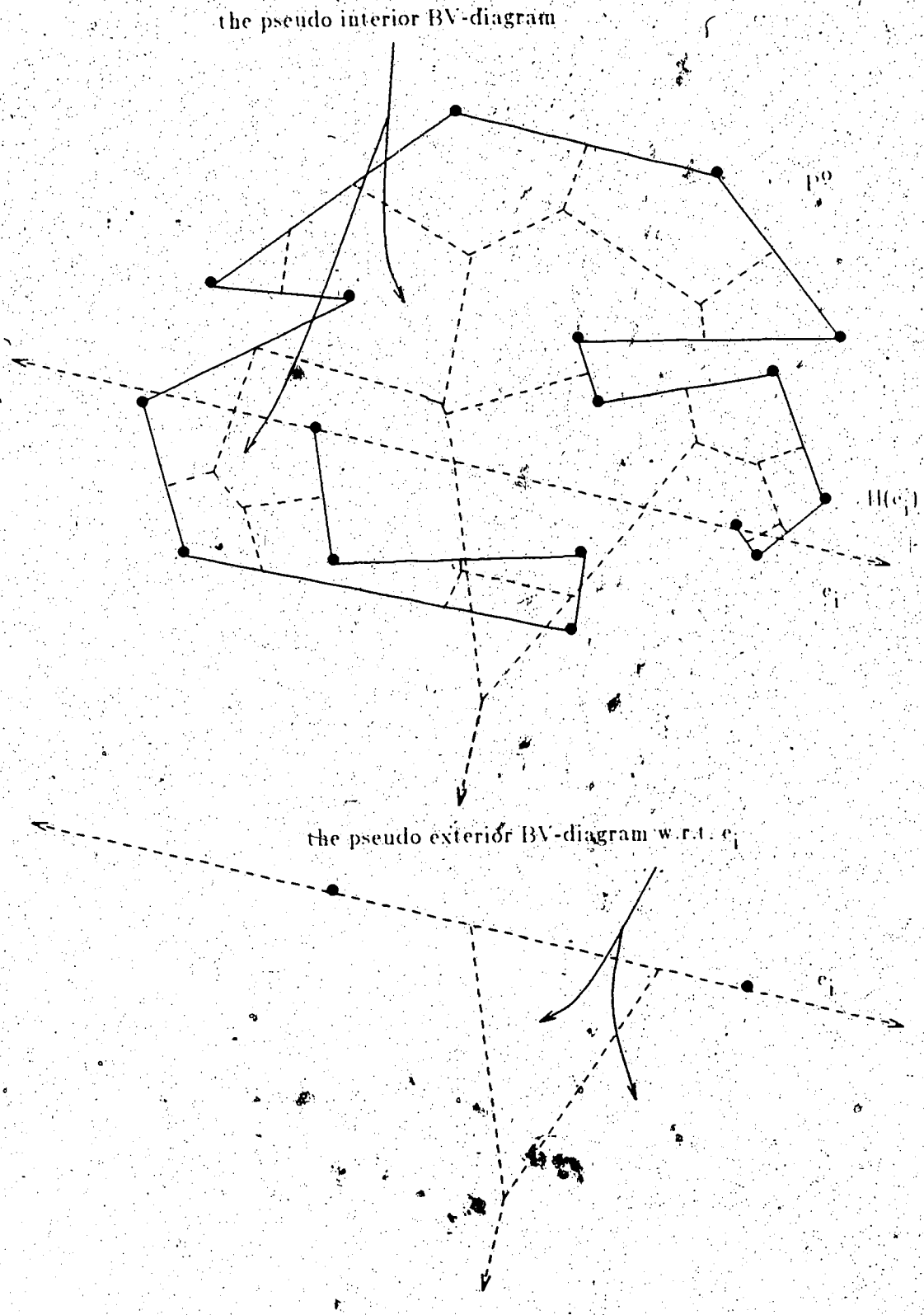


Figure 4.2 The pseudo-BV-diagram of P^o w.r.t. e_i.

In the following, we will use this concept to define some geometric objects and to prove some lemmas. An example of the pseudo-BV- diagram of an open polygon P^o containing an open edge e_i is shown in Fig.4.2. The upper part of the figure shows that a pseudo-exterior BV- diagram w.r.t. e_i superimposes on its pseudo- interior BV-diagram (which lies on the half-plane induced by e_i).

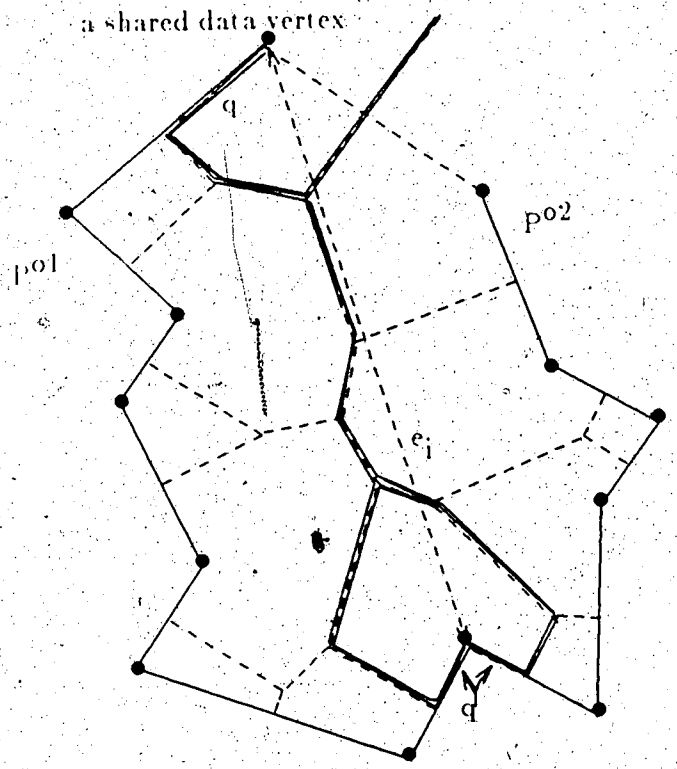


Figure 4.3 The shared pseudo-BV- boundary of two open polygons w.r.t. e_i.

Definition: Let P^{o1} and P^{o2} be two open polygons sharing an open edge e_i, and let the open edge sets be $\mathbf{E}^{o1}$ and $\mathbf{E}^{o2}$, respectively. Let B^1 and B^2 be the boundary of P^{o1} and P^{o2} respectively, and let $\mathbf{P}^1$ and $\mathbf{P}^2$ be the set of vertices of P^{o1} and P^{o2} respectively. The *shared* pseudo-BV- boundary of P^{o1} and P^{o2} with respect to the shared edge e_i is the linear figure formed from the union of *shared BV- edges* and *shared pseudo-edges* of the BV-diagrams of P^{o1} and P^{o2}; the shared BV- edges are

those determined by some $p_k \in \mathbf{P}^1$ and some $p_j \in \mathbf{P}^2$, and the shared pseudo-edges are those which are part of a non-open edge of one polygon and truncate the BV-region of a vertex of the other polygon. The vertices of the shared BV- boundary are called *shared BV- vertices*. Clearly, if some data point p_k is shared by $\mathbf{B}^1$ and $\mathbf{B}^2$, then the boundary of $V(p_k) \in Vor(\mathbf{P}^1 \bigcup \mathbf{P}^2; \mathbf{B}^1 \bigcup \mathbf{B}^2)$ belongs to the shared BV- boundary (see Fig.4.3).

By an analysis similar to the case of merging the BV- diagram of two chains, one can see that merging the pseudo-BV- diagrams of two open polygons is equivalent to finding the shared pseudo-BV- boundary of them because of the pseudo-linear separability within the union of the interior of the two open polygons.

The following lemma shows that we only need to consider those BV- edges which intersect the cutting line segment in our merge procedure.

Lemma 4.1: *Let $e_i = \overline{p_s p_t}$ be a cutting line segment of a polygon P, let $\mathbf{P}^1$ and $\mathbf{P}^2$ be the verter sets of the two subpolygons P^{o1} and P^{o2} sharing e_i respectively. If $p_j \in \mathbf{P}^1$ and $p_k \in \mathbf{P}^2$ determine an edge of the pseudo-BV- diagram of P^{o1} and P^{o2}, then $(\overline{p_j p_k} \bigcap e_i) \neq \emptyset$.*

Proof: By contradiction.

Assume that $p_j \in \mathbf{P}^1$, $p_k \in \mathbf{P}^2$ determine an edge of the pseudo-BV- diagram of P, and $(\overline{p_j p_k} \bigcap e_i) = \emptyset$. Then, some portion of $\overline{p_j p_k}$ lies outside P, since e_i is a cutting line of P. Note by Lemma 2.1, that the corresponding Delaunay edge of any BV-edge of the pseudo-interior BV-diagram of P^{o1} and P^{o2} must lie inside the subpolygons P^{o1} and P^{o2}. Hence, p_j and p_k cannot determine an edge of the pseudo-BV- diagram of P. This contradicts the assumption. □

Lemma 4.1 implies when we merge the BV- diagrams of two subpolygons of P sharing e_i, we only need to merge the pseudo-BV- diagram of one subpolygon with the

pseudo-exterior BV- diagram w.r.t. e_i of the other subpolygon and vice versa. The following lemma shows that the shared pseudo-BV- boundary of two open polygons sharing e_i is an undivided linear figure. The linear figure may contain two parts: the first part is the union of the boundaries of the resultant BV- regions associated with the shared data points, and the second part is the shared BV- chain determined by the rest of the data points in the two open polygons.

Lemma 4.2 : *Given two open polygons* P^{o1} *and* P^{o2} *sharing an open edge* e_i, *the shared pseudo-BV- boundary of* P^{o1} *and* P^{o2} *with respect to* e_i *is an undivided linear figure.*

Proof:

The shared pseudo-BV- boundary is a linear figure and must be connected. It consists of two polygon boundaries (either open or closed) connected to each other either by a shared edge or by a finite chain (see Fig.4.5, where two polygon boundaries connect to each other by a chain, and Fig.4.1, where two polygon boundaries connect to each other by sharing an edge). To see this, we first perturb P^{o1} and P^{o2} from $e_i = \overline{p_j p_k}$ slightly. In this way, we split p_j into $p_{j1} \in P^{o11}$ and $p_{j2} \in P^{o22}$, and p_k into $p_{k1} \in P^{o11}$ and $p_{k2} \in P^{o22}$ such that the points in each pair are sufficiently close to each other, where P^{o11} and P^{o22} are two new open polygons, and the differences between the new and old polygons are the two split vertices. We now attach a half-plane (a sheet) to the line extending $\overline{p_{j1}p_{k1}}$ (respectively, $\overline{p_{j2}p_{k2}}$). Thus, the boundaries of P^{o11} and P^{o22} are linearly separable within the area $P^{o11} \cup P^{o22} \cup E$, where E is the union of the two attached half-planes and the rectangle $p_{j1}p_{j2}p_{k2}p_{k1}$. By an argument similar to Lemma 3.1, the shared pseudo-BV-boundary of P^{o11} and P^{o22} consists of an endless open chain. Now, we undo the perturbation in the resultant BV-diagram of P^{o11} and P^{o22}. By an

argument similar to Lemma 3.5, one can see that the shared pseudo- BV-boundary of P^{o1} and P^{o2} consists of two polygon boundaries (either open or closed) connecting to each other either by a shared edge or by a finite chain.

□

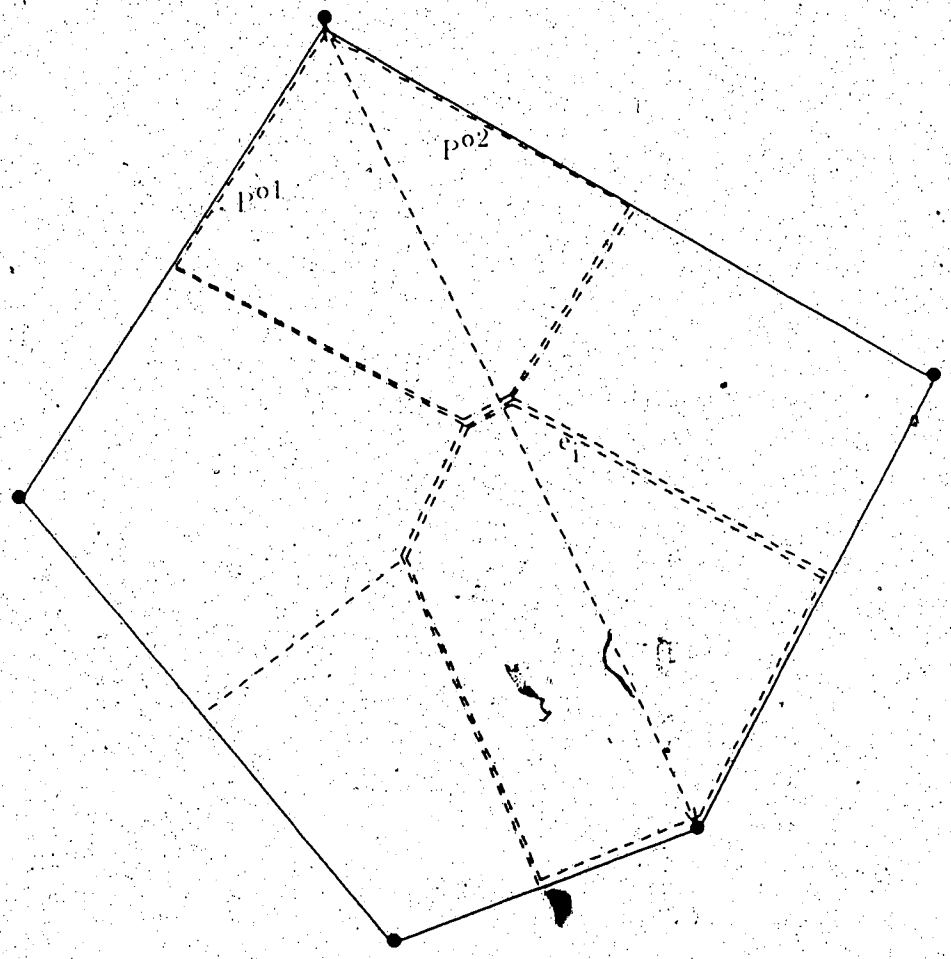


Figure 4.4 Two polygon boundaries share an edge

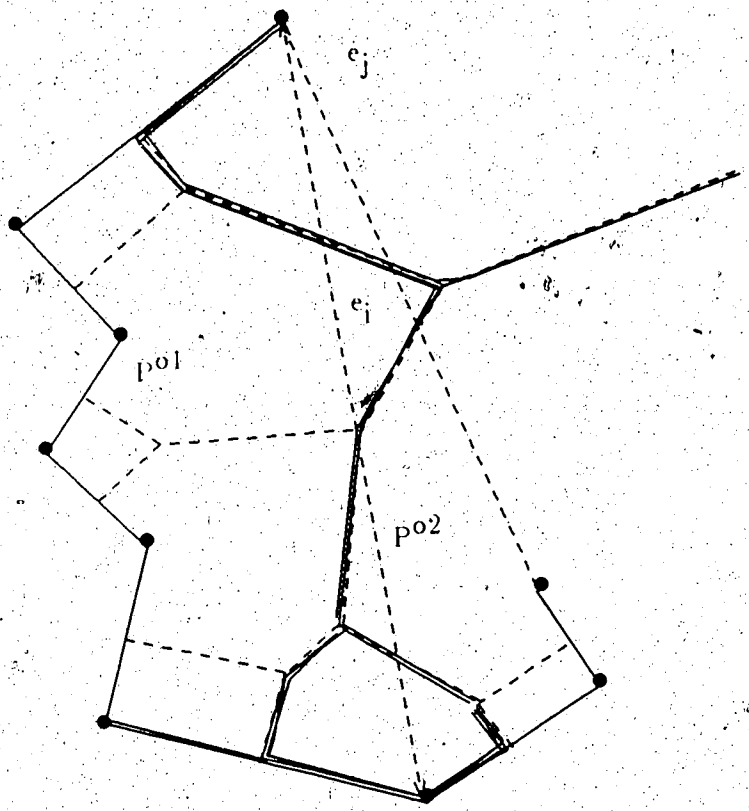


Figure 4.5 Two polygon boundaries are connected by a chain

Lemma 4.3 : *Given two open polygons* P^{o1} *and* P^{o2} *sharing an open edge* e_i, *merging the pseudo-BV-diagrams of the two polygons is equivalent to finding their shared pseudo- BV- boundary.*

Proof :

Note that e_i (with its endpoints) serves as a separator of the two polygon boundaries within region $\mathbf{P}^{o1} \cup \mathbf{P}^{o2}$, since the two polygon boundaries only share the endpoints of e_i. Note also that the BV-edges in the different edge-induced half-planes do not intersect. This means that we need not consider whether or not extending line e_i separates the area outside $\mathbf{P}^{o1} \cup \mathbf{P}^{o2}$. Thus, the lemma follows the argument similar to that used in Lemma 3.2. □

By Lemma 3.6, the shared (bound or free) BV-vertices of degree three can be found in sublinear time. We can construct the shared pseudo-BV-boundary of P^{o1} and P^{o2} by an algorithm similar to Algorithm 2.

Let e_i be the shared open edge of P^{o1} and P^{o2}; p_s and p_t be the vertices of e_i; $\mathbf{E}^{o1}$ and $\mathbf{E}^{o2}$ be the open edge sets of P^{o1} and P^{o2}. Let $B^o=(B^{o1}\text{-}\mathbf{E}^{o1}) \bigcup (B^{o2}\text{-}\mathbf{E}^{o2})$, and let $\mathbf{Vor}(\mathbf{P}^{o1}; B^{o1}\text{-}\mathbf{E}^{o1})$ and $\mathbf{Vor}(\mathbf{P}^{o2}; B^{o2}\text{-}\mathbf{E}^{o2})$ be the two pseudo-BV- diagrams, each of which contains a set of pseudo-exterior BV-diagrams of P^{o1} (or P^{o2}) with respect to their open edges and the pseudo-interior BV- diagram of P^{o1} (or P^{o2}). In our data structure, each data point is associated with two or three lists of BV-edges, pseudo-edges, and an additional line segment (including the BV-edges in both pseudo-interior and pseudo-exterior BV-diagrams); each list represents the boundary of a convex BV-subregion associated with that data point. For each data point, the associated BV-edges (BV-rays) are labelled if they cross or lie outside an open edge. Thus, the asociated BV-edges (BV-rays) labelled by different open edges do not merge with each other. The algorithm works similary to Algorithm 2. It first finds the branches of the two BV-regions associated with the two shared data points respectively. If these two BV-regions share a BV-edge (this can be found by checking if the BV-rays of $V^1(p_s)$ and $V^2(p_s)$ (respectively, $V^1(p_t)$ and $V^2(p_t)$) overlap), then the algorithm reports the boundaries of $V(p_s)$ and $V(p_t)$, and stops. Otherwise, the algorithm will choose one data point and regard its branch as the starting BV-ray of the pseudo-shared BV-boundary. The construction of the rest of the pseudo-shared BV-boundary is the same as Step 2 of Algorithm 2. The algorithm will terminate when the BV-ray of the pseudo-shared BV-boundary overlaps the branch of the other shared data point. The rest of the notation in the algorithm follows that in Algorithm 2.

Algorithm 3

Input: $\mathbf{Vor}(\mathbf{P}^{o1};B^{o1}\text{-}\mathbf{E}^{o1})$, $\mathbf{Vor}(\mathbf{P}^{o2};B^{o2}\text{-}\mathbf{E}^{o2})$; B^{o1}, B^{o2}; $\mathbf{E}^{o1}$, $\mathbf{E}^{o2}$; e_1

Output: $\mathbf{Vor}(\mathbf{P}^{o1} \bigcup \mathbf{P}^{o2};B^{o}\text{-}\mathbf{E}^{o})$

Method:

1 find the starting segments of the shared pseudo-BV-boundary around p_1 and q_1, say $b(v;p,q)$ and $s(v_1;p_1,q_1)$; if no branches are found (i.e., $V(p_1)$ and $V(p_1)$ share an edge), then report the boundaries and stop;

2 *While* $(b(v;p,q) \neq s(v_1;p_1,q_1))$ *Do*

(a) find the BV-edge (BV-ray or pseudo-edge) say, $(u,w;r,s)$, which first intersects $b(v;p,q)$;

(b) find the next branch according to the following cases:

$b(v;p,q) \circ B^{o}$ meets $(u,w;r,s) \neq B^{o}$

or $b(v;p,q) \neq B^{o}$ meets $(u,w;r,s) \neq B^{o}$

or $b(v;p,q) \neq B^{o}$ meets $(u,w;r,s) \circ B^{o}$;

$b(v;p,q) \leftarrow$ the branch;

End-Do.

Lemma 4.4 : *Given* $Vor(\mathbf{P}^{o1};\mathbf{B}^{o1})$ and $\mathbf{Vor}(\mathbf{P}^{o2};\mathbf{B}^{o2})$, where $\mathbf{B}^{o1}$ and $\mathbf{B}^{o2}$ *are the boundaries of the two open polygons* P^{o1} *and* P^{o2} *sharing an open edge* e_1 *and* $\mathbf{P}^{1}$ *and* $\mathbf{P}^{2}$ *are the vertex sets of* P^{o1} *and* P^{o2} *respectively. Algorithm 3 produces the shared pseudo-BV- boundary of* P^{o1} *and* P^{o2} *in time proportional to* $|\mathbf{P}^{1} \bigcup \mathbf{P}^{2}|$.

Proof:

(a) Algorithm 3 produces the shared pseudo-BV- boundary of the two open

polygons. By Lemma 4.2, the shared pseudo-BV- boundary must be an undivided linear figure. Note that each segment created by Algorithm 3 belongs to the shared pseudo-BV- boundary by an argument similar to the one used in Lemma 3.4. The 'while loop' of Algorithm 3 exhausts all the segments of the shared pseudo-BV- boundary of P^{o1} and P^{o2}.

(b) Algorithm 3 takes $O(|P^1 \cup P^2|)$ to construct the shared pseudo-BV- boundary by an argument similar to the one used in Lemma 3.7. □

By Lemma 4.3, we can construct the resultant pseudo-BV-diagram of two open polygons from their shared pseudo-BV-boundary. We now design a divide and conquer algorithm to construct the pseudo-BV- diagram of an n-gon P. The algorithm recursively decomposes the polygon P into subpolygons, such that each subpolygon contains at least one third of the vertices of the parent polygon. Then the algorithm constructs the pseudo-BV-diagram of each triangle (the smallest subpolygon) and merges pairs of them. The resultant diagrams are the pseudo-BV- diagrams of the polygons in their immediate parent. Repeat the merge procedure for the pseudo-BV- diagrams of the new pair of subpolygons which we are currently merging until the pseudo-BV-diagram of P has been found.

The only remaining problem is to prove that the resultant pseudo-BV- diagram of P is the interior of the BV- diagram of P.

Lemma 4.5 : *The pseudo-BV-diagram of P, produced by Algorithm 3, is the interior of the BV- diagram of* P.

Proof:

Note that the polygon P has no open edges. Thus, the lemma follows by the definition of pseudo-BV- diagram and the definition of the interior of the BV-diagram. □

Now, it is not difficult to construct both the interior and the exterior BV-diagrams of an n-gon P. To do so, we first find the convex hull of P. Note that the convex hull of P and the boundary of P determine several smaller polygons, each of which contains a convex hull edge of P as the open edge. We then obtain their pseudo-BV- diagrams by Algorithm 3. The BV- diagram of P is the collection of the interior of the BV-diagram of P, the bisector rays of the convex hull edges belonging to P and pointing to the outside of the convex hull, and the pseudo-BV- diagrams of those smaller polygons (with the edges belonging to the convex hull as open edges).

Theorem 4.1: *The BV-diagram of a simple n-gon* P *can be constructed in* $\Theta(n \log n)$ *time.*

Proof:

Let the smaller polygons determined by the convex hull of P and the boundary of P be $P^1, \ldots, P^k$. The triangulation of P (P^*) and the construction of its partitioning tree can be found in $O(|\mathbf{P}| \log |\mathbf{P}|)$ ($O(|\mathbf{P}^1| \log |\mathbf{P}^1|)$ time [Chaz82]. The pseudo-BV- diagram of each smaller polygon can be constructed independently, and the construction for each smaller polygon say, P^i, takes $O(|P^i| \log |P^i|)$ time by Lemma 4.1. Note that each vertex of P can be assigned to at most two smaller polygons. Constructing the pseudo-BV- diagrams of k smaller polygon takes $O(|P^1| \log |P^1| + |P^2| \log |P^2| + \ldots |P^k| \log |P^k|) \leq O(|P^1| + \ldots |P^k|) \log n\ ) = O(n \log n)$ time. The construction of the interior of the BV- diagram of the n-gon takes $O(n \log n)$ time. Thus, the time complexity for constructing the BV-diagram of the n-gon P takes $O(n \log n)$ time. Recently, Aggarwal et al proved that $\Omega(n \log n)$ is the worst case lower bound for constructing the BV-diagram of an n-gon [Agg87]. Thus, the algorithm is optimal. □

Chapter 5

Finding the bounded Voronoi diagram for a set of line segments

5.1. A preview of the problem

In this chapter, we shall investigate the BV-diagram of a set of non-crossing and non-overlapping line segments, where the data points are the endpoints of these line segments and the obstacles are these line segments themselves. We frequently call the input line segments, *obstacles*.

In the rest of the chapter, we use **L** to denote a set of non-crossing and non-overlapping obstacles, and **P**, the set of the endpoints of **L**. We also assume that $|\mathbf{L}| = n$ and $|\mathbf{P}| = m$, thus $n \leq m \leq 2n$.

The main idea of our approach is to use divide and conquer to construct Vor(**P**;**L**).

The divide step is as follows. We first find the convex hull of **P**, CH(**P**); we then sort the data points of **P** by their x-coordinates (we assume that the x-coordinates are distinct); we finally draw a vertical line between each pair of two consecutive data points in the sorted list. For example, if p_i and $p_j \in \mathbf{P}$ are consecutive in the list, we draw a line which is parallel to the y-axis, and which linearly separates p_i and p_j.

Thus, the m-1 vertical lines divide CH(**P**) into m polygons (pentagons, quadrilaterals, and triangles). We call these polygons *initial vertical polygons* (v-polygons). Each initial v-polygon contains exactly one data point and many *obstacle segments* (a part of an obstacle is an obstacle segment if one of its two data points lies on the initial v-polygon). We call the collection of the data points and the obstacle segments lying on an initial v-polygon *Voronoi data* (v-data) *of the initial v-polygon* (refer to the upper part of Fig.5.1, where the leftmost v-polygon is a triangle which contains a data point and two obstacle segments; the next v-polygon is a pentagon which contains a data point and one obstacle segment; the shaded area is the active sub-v-gon of the v-

gon which we shall define later). Thus, the result of the divide step is m v-polygons and m sets of corresponding v-data.

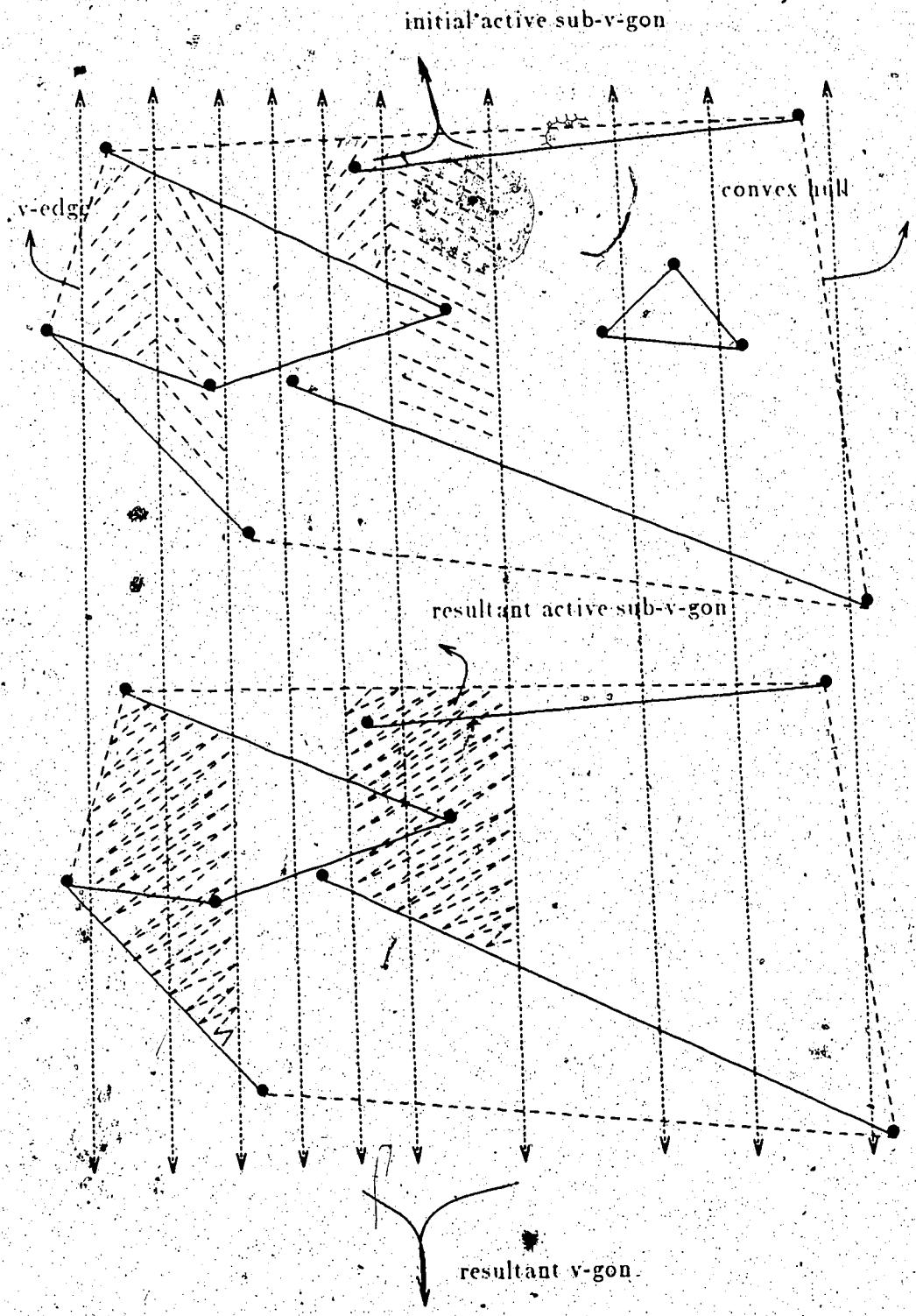

initial active sub-v-gon
v-edge
convex hull
resultant active sub-v-gon
resultant v-gon

Figure 5.1

The conquer step is divided into $\log_2 m$ stages. In the first stage, we construct the pseudo-BV-diagram (which we shall define later) of the set of v-data of each initial v-polygon, which, for simplicity, is called the pseudo-BV-diagram of the initial v-polygon. In the second stage, we combine adjacent pairs of initial v-polygons (once for a v-polygon) to form a new v-polygon for each pair (therefore, forming a new set of v-data), and merge the pseudo-BV- diagrams of the initial v-polygons in each pair to get the pseudo-BV-diagram of the new v-polygon. In an arbitrary stage, say the k-th stage, we combine adjacent pairs of old v-polygons (which are the result of the (k-1)-th stage) to form a new v-polygon for each pair and merge the pseudo-BV-diagrams of the two old v-polygons in each pair to get the pseudo-BV-diagram of the new v-polygon. After $\log_2 m$ stages, the resultant v-polygon is CH(**P**) and the resultant pseudo-BV-diagram of the v-polygon is the BV-diagram of **L**.

We shall see that the whole divide and conquer process takes $O(m \log m)$ time. The critical point for this method is a fast merge procedure for each conquer stage (this is, the time of the merge procedure in each conquer stage must be proportional to m). The merge procedure must deal with two problems:

(1) How to combine two old adjacent v-polygons to get a new v-polygon (the necessity of discussing this problem will become clear later), and

(2) how to merge the pseudo-BV-diagrams of the two old v-polygons to get the pseudo-BV-diagram of the new v-polygon.

In the following two sections, we shall discuss the above two problems respectively.

5.2. Combining two adjacent vertical polygons

Definition: Let $(p_1 \ldots p_m)$ be a list of data points of **P** sorted by their x-coordinates. A vertical line linearly separating two sets of data points in the sorted list is called a *e-line*. A polygon which lies on CH(**P**), and each of whose edges is either a part of a e-line or an edge (or a part of an edge) of CH(**P**) is called a *vertical polygon* (v-polygon) (note that CH(**P**) is a special case of v-polygon). A v-polygon is denoted by P^v. The left (right) vertical edge of P^v (if any) is denoted by e_l (e_r). The collection of the data points, the obstacles, and the obstacle segments lying on P^v is the set of *Voronoi data* (v-data) of P^v.

The vertical edge e_l (e_r) of a v-polygon P^v may cross several obstacles. A segment of e_l (e_r) which is delimited by two adjacent crossover points is called a *segment* of a v-edge. The portion of an obstacle with one data point lying on a v-polygon is called an *obstacle segment* of that v-polygon.

Definition: An obstacle crossing a v-polygon (i.e., some part of the obstacle lies inside the v-polygon and both its endpoints lie outside the v-polygon) is called a *crossing-obstacle* of that v-polygon. A v-polygon is divided by its *crossing*-obstacles into several smaller polygons, which are called *sub-v-polygons*.

Definition: We attach an edge-induced halfplane (sheet) on each segment of the v-edges of the boundary of P^v, and the sheets and P^v form a special space (refer to Section 4.4 and to Fig.5.2. where we show the space for a sub-v-polygon). The *pseudo-BV-diagram* of the set of v-data of P^v is the BV-diagram of the data points and the obstacles (obstacle segments) of the set defined on the special space. (The pseudo-interior BV-diagram is the portion of the pseudo-BV-diagram lying on P^v, and the pseudo-exterior BV-diagrams are the portions lying on the sheets.)

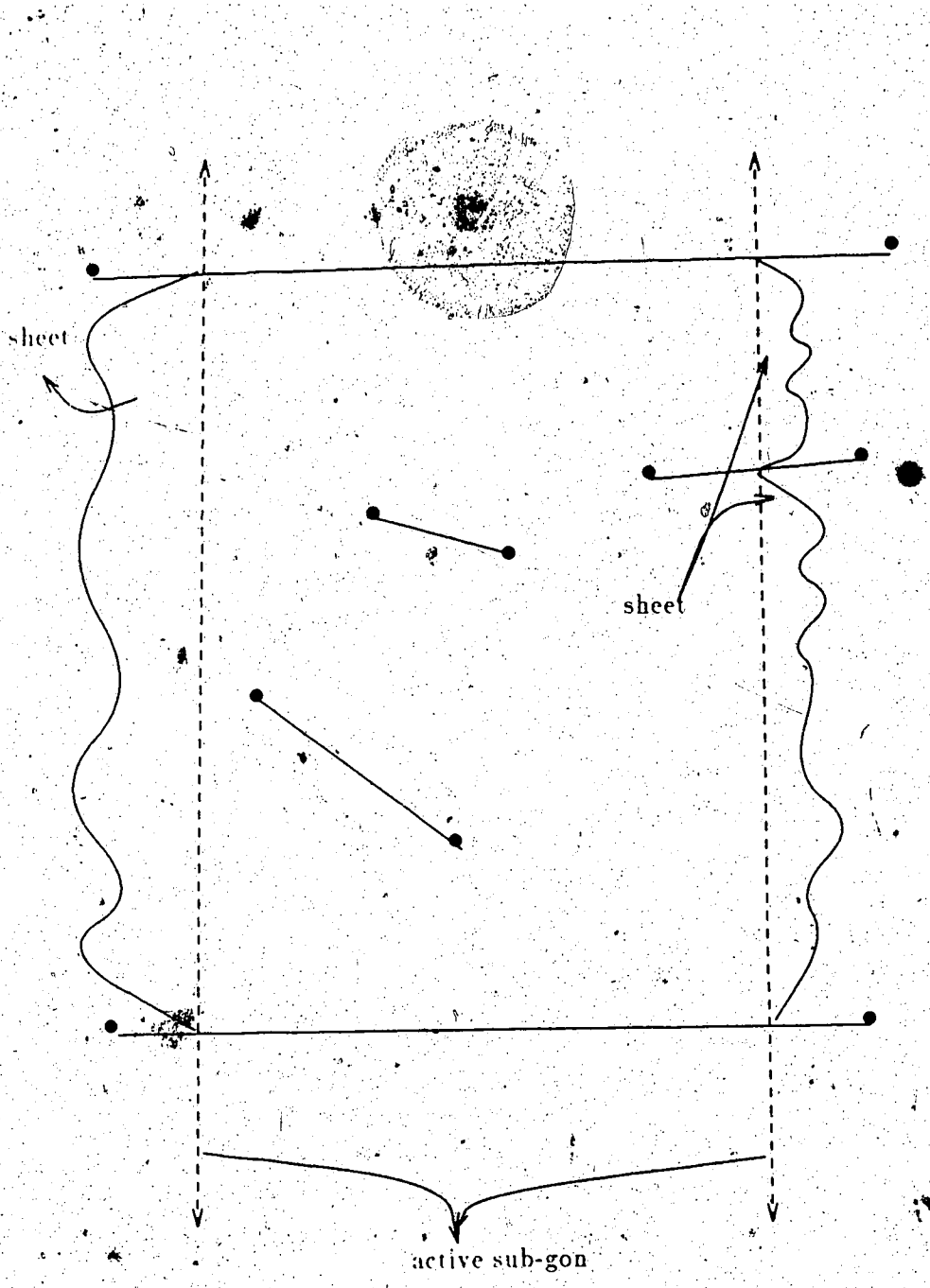


Figure 5.2

Note that each v-polygon may contain O(n) sub-v-polygons since it can be crossed by O(n) obstacles. If we keep all the information about these sub-v-polygons during the conquer step, we could have $O(n^2)$ pieces of information. This will yield inefficient algorithms. However, further investigations show that we only need to combine a limited number of sub-v-polygons to merge the pseudo-BV-diagrams of two v-polygons. That is, many sub-v-polygons do not contain any data point, hence they may not affect the resultant pseudo-BV-diagram. (A similar technique was first suggested by C. Yap [Yap85], and later used by P. Chew [Che87].)

Let us consider two adjacent v-polygons separated by a v-line, say L_s. For convenience, we color all elements of the v-polygon which lie on the left hand side of L_s blue, and those on the right hand side of L_s red. We call the combined (sub-) v-polygons *resultant (sub-) v-polygon*. We also call a (resultant) sub-v-polygon *active* if the set of corresponding v-data contains at least one data point. In contrast with this, a non-active (resultant) sub-v-polygon contains no data point. We call the pseudo-BV-diagram of the resultant v-polygon *resultant pseudo-BV-diagram* (refer to Fig.5, where the shaded areas are active sub-v-gons).

In order to merge the pseudo-BV-diagrams of a blue v-polygon and a red v-polygon, we must first determine which blue sub-v-polygon should combine with which red sub-v-polygon. We have the following two observations:

(1) It is sufficient to combine each pair of blue and red sub-v-polygons such that the blue e_r and the red e_l of the two sub-v-polygons overlap. (We call the two sub-v-polygons in such a pair *neighboring sub-v-polygons*. Note that one blue (red) sub-v-polygon may combine with many red (blue) sub-v-polygons. Therefore, one blue (red) sub-v-polygon can belong to many neighboring sub-v-polygons.) If a blue e_r does not overlap a red e_l, then the corresponding blue sub-v-polygon and the red sub-v-polygon must be separated by either some crossing obstacle or some

obstacle segment of the resultant v-polygon. In either case, the data points (if any) of the blue sub-v-polygon and the data points (if any) of the red sub-v-polygon are not visible from each other, and they cannot determine any new BV-element in the resultant pseudo-BV-diagram.

However, there may still exist O(n) such neighboring sub-v- polygons in a resultant v-polygon. The following observation shows that we need not consider all these pairs of neighboring sub-v-polygons.

(2) It is sufficient to consider only those pairs such that at least one of the two neighboring sub-v-polygons in a pair is active.

If both the blue and the red sub-v- polygons in a pair are active, the two sub-v-polygons must contain data points by definition. Then, some blue data point and some red data point of the two sub-v-polygons must be visible from each other since the blue e_r and the red e_l do not belong to **L**. Hence, some blue data point and some red data point of the two sub-v-polygons must determine some new BV-element in the resultant pseudo-BV-diagram by definition.

If the blue (red) sub-v-polygon is active and its neighboring red (blue) sub-v-polygon is not active, then some blue (red) data points of the sub-v-polygon must determine some new BV-element in the red (blue) sub-v-polygon by definition.

If both the blue and red sub-v-polygons in a pair are non-active, then no BV-element will be created since the resultant sub-v-polygon contains no data point and since it is separated by two crossing obstacles from the rest of the resultant v-polygon.

The above two observations show that in order to merge the pseudo-BV-diagrams of a blue v-polygon and a red v-polygon, we only need to combine every active blue (red) sub-v-polygon with its neighboring red (blue) sub-v-polygons.

In the following, we shall define the closest line segments of a data point along the y-axis direction, then present a procedure for finding the active sub-v-polygons of a v-polygon. A v-polygon is represented by two lists of the v-edges of its active sub-v-polygons since they contain sufficient information for combining v-polygons. That is, the left list which contains all the left v-edges (and the segments of these left v-edges) of the active sub-v-polygons and the right list which contains all the right v-edges (and their segments) of the active sub-v-polygons. The edges (and their segments) on the left (right) list are ordered according to the y-coordinates of their endpoints.

Definition: A line segment $l \in L$ is said to be *closest to* a data point $p_j \in P$ along the y-axis direction if a ray radiating from p_j in the positive or negative y- direction crosses l, say at s, and no other line segment crosses $\overline{sp_j}$. The line segments closest to p_j along the x-axis direction can be defined similarly.

Procedure CombiningVgons

Input: A blue v-polygon and a red v-polygon

Output: A new v-polygon

Method:

(1) If the two input polygons are initial v-polygons.

(a) Find the active initial sub-v-polygons. Each initial v-polygon contains exactly one data point, and the active initial sub-v-polygon of each initial v-polygon is bounded by the two line segments (crossing-obstacles or convex hull edges) closest to the data point along the vertical direction and the two v-lines closest to the data point by the definition. (If the data point, say p_j, lies on CH(P) and p_j is not the leftmost or the rightmost data point, then the active sub-v-polygon is bounded by the two convex hull edges incident at p_j, the line segment closest to p_j along the vertical direction, and the two v-lines closest to p_j. If p_j is the leftmost or rightmost data point, then the active sub-v-polygon is bounded by the two convex hull edges incident at p_j and the v-line closest to p_j.) The line segments closest to p_j for all $p_j \epsilon \mathbf{P}$ can be found by the plane-sweep method [Ben79]. The v-lines closest to p_j for all $p_j \epsilon \mathbf{P}$ are recorded in the divide step. Thus, the two vertical edges e_l, e_r, and the upper and bottom edges of an active initial sub-v-polygon can be found by finding the intersections of these lines. The segments of e_l and e_r can be found by finding the intersections of the v-lines and the obstacles incident at the data point.

(b) Combine an active initial blue (red) sub-v-polygon and its neighboring initial red (blue) sub-v-polygon to form new active sub-v-polygons. Note that e_r of the blue active sub-v-polygon and e_l of the red active sub-v-polygon have been computed. Note also that the blue e_r and the red e_l must coincide with their separator L. Thus we can determine whether or not the blue e_r overlaps the red e_l by examin-

ing their y-coordinates from point b upwards along L_s, where b is the crossover point of the lower part of $CH(\mathbf{P})$ on L_s. If the blue e_r and the red e_l overlap, then by observation (2), the two sub-v-polygons will form a new active sub-v-polygon, otherwise they will form two new active sub-v-polygons. In the former case, we keep the endpoints of the blue e_l (and its segments, if any) and the blue $e_{l'}$ in the new left list, and the endpoints of the red e_r (and its segments, if any) and the red $e_{r'}$ in the new right list. $e_{l'}$ is the vertical edge of a neighboring blue non-active sub-v-polygon of the red active sub-v-polygon ($e_{l'}$ can be determined in constant time by finding the intersections of the two crossing-obstacles of the blue non-active sub-v-polygon and the blue left v-line), and $e_{r'}$ is the vertical edge of a neighboring red non-active sub-v-polygon of the blue active sub-v-polygon. In the latter case, we keep the endpoints of the blue e_l (and its segments, if any) in the left list and the red $e_{r'}$ (and its segments, if any) in the right list, and keep the red e_r (and its segments) in the right list and the blue $e_{l'}$ (and its segments) in the left list. e_r, $e_{r'}$, and their segments (respectively, e_l, $e_{l'}$, and their segments) are kept in the new list according to the y-coordinates of their endpoints.

(2) If the two input v-polygons are not initial v-polygons.

Let us assume they are two adjacent v-polygons in the k-th stage of the conquer step. Note that because we combine the adjacent v-polygons pairwise in each stage of the conquer step, the (k-1)-th stage must result in $m/2^{k-1}$ pairs of v-polygons ($k \geq 1$), and each pair contains (i) a blue v-polygon and a red v-polygon separated by a v-line L_s, and (ii) the left and the right lists of the v-edges (and their segments) of the active blue sub-v-polygons and the lists for the active red sub-v-polygons. We now consider one of these pairs of v-polygons. Note that the blue e_r's and the red e_l's in a pair must coincide with L_s. We can determine the new active sub-v-polygons in a pair by examining the overlaps of the blue e_r's and

t[illegible]red e_l's upwards along L_s by observation (2). We start a new active sub-v-poly[illegible]n if an overlap occurs. We continue the new active sub-v-polygon as long as the overlaps of blue e_r's and red e_l's continue to occur. We terminate a new active sub-v-polygon if no new blue e_r and no new red e_l overlap with the previous blue e_r and red e_l (that is, we encounter a crossing obstacle of the resultant v-polygon). In this manner, we combine the corresponding blue and red sub-v-polygons together to form new active sub-v-polygons. We record the blue e_l and their segments of the new active sub-v-polygon in the new left list, the red e_r and their segments, in the new right list. (This takes time proportional to the number of segments of the v-edges in the new v-polygon. Hence, the number is bounded by the number of data points and the number of obstacle segments in the new active sub-v-polygon). Since we examine the overlaps sequentially along L_s, the active blue e_l's (respectively the active red e_l's) are ordered in the right (left) list of the new v-polygon. We execute the above combining process for all pairs of adjacent v-polygons. By observations (1) and (2), the above process produces all the new active sub-v-polygons of the resultant v-polygon.

Lemma 5.1: (1) *All the active initial sub-v-polygons of* **L** *can be found in O(m log m) time.* (2) *All the active resultant sub-v-polygons in the k-th stage of the conquer step can be found in O(m) time.*

Proof:

(1) The active initial sub-v-polygon of each initial v-polygon is bounded by the two line segments (crossing-obstacles or convex hull edges) closest to the data point along the vertical direction and the two v-lines closest to the data point by definition. The line segments closest to p_j for all $p_j \in \mathbf{P}$ can be found by the plane-sweep method in O(m log m) time [Ben79]. The v-lines closest to p_j for all $p_j \in \mathbf{P}$ can be found in O(m log m) time in the divide step. The two vertical edges e_l, e_r,

and the upper and bottom edges of each active initial sub-v-polygon can be found in constant time by finding the intersections of their corresponding lines. Thus, all the active initial sub-v-polygons can be found in O(m log m) time. The segments of e_l (e_r) can be found in time proportional to the obstacles incident at the data point by finding the intersections of the v-lines and the obstacles incident at the data point. Note that the number of these segments for all the active initial sub-v-polygons is bounded by m since each obstacle can have at most two obstacle segments.

(2) We shall prove by induction that all the active resultant sub-v-polygons in the k-th stage of the conquer step can be obtained in O(m) time.

Base step.

Combining an active initial blue (red) sub-v-polygon and its neighboring initial red (blue) sub-v-polygon to form new active sub-v-polygons takes time proportional to the number of obstacle segments of the two sub-v-polygons. This is because finding the overlaps of the v-edges of the active sub-v-polygons along their separator L_s takes constant time and constructing the left and right lists of the new v-polygon takes time proportional to the number of obstacle segments (each pair of adjacent obstacle segments must correspond to either a active or a non-active neighboring sub-v-polygon).

Induction step.

We consider the k-th stage of the conquer step. Note that in the (k-1)-th stage there are $m/2^{k-1}$ pairs of v-polygons, where each pair contains (i) a blue v-polygon and a red v-polygon separated by a v-line L_s, and (ii) the left and the right lists of the vertical edges of the active blue sub-v-polygons and the lists for the active red sub-v-polygons. We consider one of these pairs and assume that the blue v-polygon has t active sub-v-polygons with t' v-edge segments and the

red v-polygon has s active sub-v-polygons with s' v-edge segments. Since the blue e_r's in the old left list and the red e_l's in the old right list are ordered according to the y-coordinates of their endpoints, the testing for overlaps of the edges of the active sub-v-polygons along L_s can be done without backtracking. Note that constructing the left and right lists of the new active sub-v-polygons takes time proportional to the number of the v-edge segments in the old active sub-v-polygons. Thus, the combining process takes time proportional to s'+t', since there exist only s'+t' v-edge segments of the active sub-v-polygons. The resultant v-polygon is the polygon in the k-th stage by observations (1) and (2). We find the new active sub-v-polygons for each of $m/2^{k-1}$ pairs of v-polygons. The time complexity for finding all the new active sub-v-polygons is bounded by O(m). This is because each obstacle can have at most two obstacle segments, hence the total number of obstacle segments in the k-1th stage is bounded by O(m), and the lemma follows from the above argument. □

5.3. Merging the pseudo-BV-diagrams of two v-polygons

Let $\mathbf{L}^1$ be the set of blue obstacles and blue obstacle segments (the set of blue v-data) contained by a blue v-polygon, and $\mathbf{L}^2$ be the set of red obstacles and red obstacle segments (the set of red v-data) contained by a red v-polygon. Let $\mathbf{P}^1$ and $\mathbf{P}^2$ be the corresponding endpoint sets of $\mathbf{L}^1$ and $\mathbf{L}^2$ respectively. $\mathbf{P}^1$ and $\mathbf{P}^2$ are linearly separated by a vertical line L_s.

Definition: The *shared BV-boundary* of a set of blue v-data $\mathbf{L}^1$ and a set of red v-data $\mathbf{L}^2$ is the linear figure formed from the union of *shared BV-edges (rays)* and *shared pseudo-edges*; the shared BV-edges (rays) are those determined by some $q_i \in \mathbf{P}^2$ and some $p_j \in \mathbf{P}^1$, and the shared pseudo-edges are those which are part of an obstacle or obstacle segment of $\mathbf{L}^1$ ($\mathbf{L}^2$) and truncating the BV-region of a data point of $\mathbf{P}^2$ ($\mathbf{P}^1$). The vertices of the shared BV-boundary are called *shared BV-vertices* (refer to

Fig. 5.3).

We shall give some details of our data structure for the BV-diagram of a set of line segments. In the previous chapters, we dealt with chains. Each data point in a chain can be shared by at most two edges (obstacles). Therefore, each data point can associate with at most three convex sub-BV-regions (refer to Lemma 3.3). In this chapter, the input is a set of n non-crossing and non-overlapping line segments. Each data point may be shared by up to n line segments, and hence each data point may associated with up to O(n) convex sub-BV-regions. If we use the previous data structure, we may encounter some difficulty in designing an efficient merge algorithm. This is because when we need to traverse the boundary of some sub-BV-regions of a data point (which occurs in our merge algorithm), we must traverse the whole boundary of the BV-region associated with the data point, and this may take O(n) time in the worst case. If we traverse the boundary of some sub-BV-region of such a data point many times, and each time we traverse the boundary of some different sub-BV-region, then we may traverse $O(n^2)$ BV-elements (BV-edges, BV-rays, and pseudo-edges), and yield an inefficient merge algorithm. In order to avoid this difficulty, we regard a data point, say p_i, shared by k obstacles as k sub-data points. That is, p_{i1}, p_{i2},....p_{ik}. Each sub-data point corresponds to one obstacle. By a proof similar to Lemma 3.3, we can show that the number of convex sub-regions in the BV-diagram of a set of n line segments is bounded by O(n). Thus, each sub-data point is associated with at most two convex sub-BV-regions because the total number of sub-data points is bounded by 2n, and each sub-data point is assigned to exactly one obstacle. Now, if we need to traverse the boundary of some sub-BV-region of a data point shared by several obstacles, we need not traverse the boundaries of the all sub-BV-regions of the data point. (In the following, sub-BV-region is simply written as BV-region and sub-data point as data point.)

In the following, we shall first show that the shared BV-boundary of a blue v-polygon and a red v-polygon consists of several undivided linear figures (refer to the definition in Section 3.3, a linear figure is undivided with respect to obstacle l iff there exists a traversal of the linear figure which does not cross l), and each undivided linear figure is a finite, semi-infinite, or infinite BV-chain. We shall then show that the shared BV-boundary of a blue v-polygon and a red v-polygon can be constructed in time proportional to the number of data points and the number of obstacle segments contained in the two v-polygons.

Lemma 5.2: *The shared BV-boundary of a blue v-polygon and a red v-polygon is an infinite BV-chain if the v-data of the two v-polygons are linearly separable.*

Proof:

Note that the proof of Lemma 3.1 only relies on the property of linear separability of two input chains. The proof does not rely on the ordering property or the connectivity property of the data points of the two input chains. Thus, we can replace the two sets of input chains by two sets of obstacles and obstacle segments in Lemma 3.1, and the proof for the modified Lemma 3.1 is still valid. □

Definition: An obstacle is called a *cutting obstacle* of a v-polygon if it crosses the separator L_s of the v-polygon, where L_s separates the blue v-polygon from the red v-polygon of this v-polygon, and if at least one of the two endpoints of the obstacle lies on the v-polygon.

We shall show what the shared BV-boundary of a blue and a red v-polygon looks like. Note by definition that the shared BV-boundary cannot cross obstacles. Therefore, the shared BV-boundary must be separated by crossing obstacles. That is, we only need to discuss the portion of the shared BV-boundary of each resultant active sub-v-polygon (since the resultant non-active sub-v-polygons do not create any BV-element). There are two cases of resultant active sub-v-polygons: (1) a resultant active

sub-v-polygon contains no cutting obstacles, and (2) a resultant active sub-v-polygon contains some cutting obstacles.

(1) We further consider two subcases: (a) a resultant active sub-v-polygon contains only blue (red) data points, and (b) it contains both blue and red data points.

(a) In this case, the whole red (blue) sub-v-polygon belongs to some blue (red) data points by definition. Hence, the portion of the shared BV-boundary consists of two pseudo-edges by definition, where each of the pseudo-edges coincides with a crossing obstacle and lies on the red (blue) sub-v-polygon.

(b) In this case, the portion of the shared BV-boundary must consist of a BV-chain and two pseudo-edges (they may or may not connect to each other) by an argument similar to the proof of Lemma 5.2.

(2) We further consider two subcases: (a) a resultant active sub-v-polygon contains only blue (red) data points, and (b) it contains both blue and red data points.

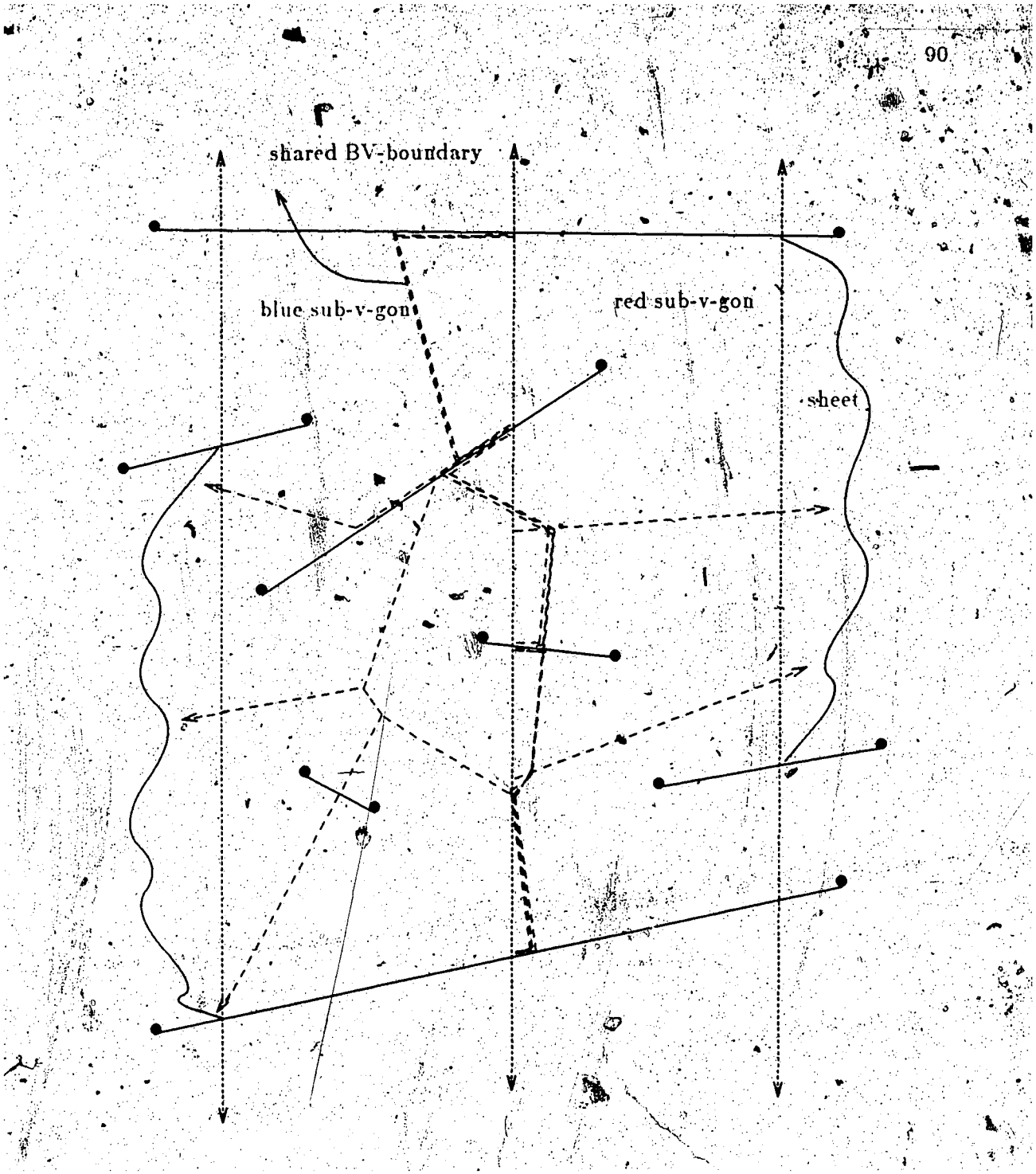
(a) In this case, the whole red (blue) sub-v-polygon belongs to some blue (red) data points by definition. Then, the portion of the shared BV-boundary consists of several pseudo-edges by definition, where each of the pseudo-edges coincides with a crossing or cutting obstacle and lies on the red (blue) sub-v-polygon.

(b) In this case, we only discuss the case that each blue (red) sub-v-polygon of the resultant active sub-v-polygon contain at least a data point (otherwise, this blue (red) sub-v-polygon will contain only pseudo-edges by the above analysis). Before we discuss the case, we give a definition.

Definition: A *cutter* of the shared BV-boundary of a blue and a red v-polygon is an infinite chain which has two types: (a) it consists of a cutting obstacle of the resultant v-polygon, say $l_c = \overline{p_i p_j}$, and two rays $p_i\vec{x}^-$ and $p_j\vec{x}^+$, where $p_i\vec{x}^-$ starts at p_i and extends horizontally to negative infinity and $p_j\vec{x}^+$ starts at p_j and extends horizontally to positive infinity, and (b) it consists of an obstacle segment (if the cutting obstacle has only one data point, say p_i, lying on the resultant v-polygon), and two rays $p_i\vec{x}^-$

and $p_i p_j$. We denote the cutter of a cutting obstacle l_c as l_c.

For simplicity, we only discuss the case that a resultant active sub-v-polygon contains exactly one cutting obstacle since the case of more than one cutting obstacle is a straightforward extension of the simple case.


shared BV-boundary
blue sub-v-gon
red sub-v-gon
sheet

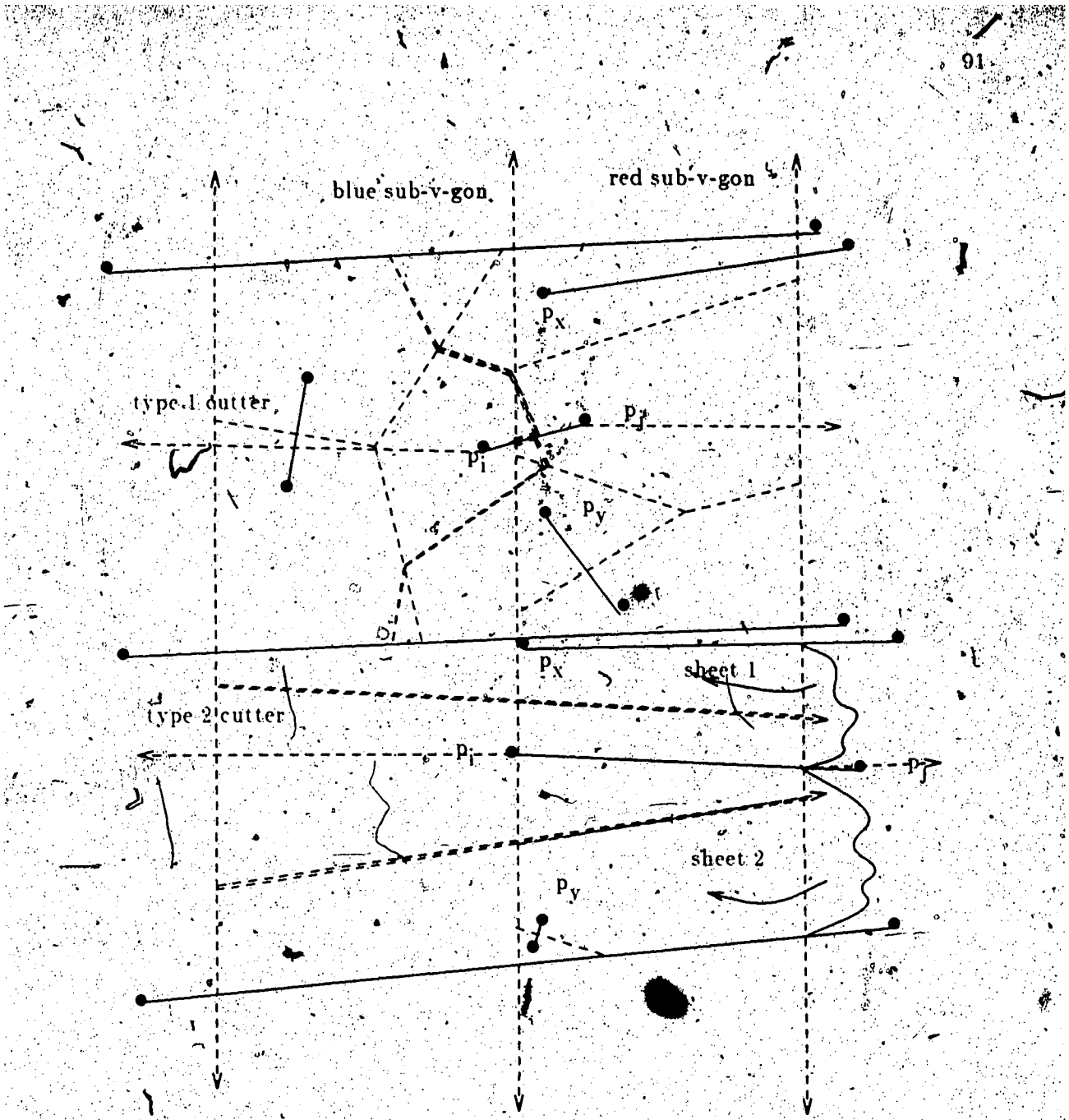

blue sub-v-gon
red sub-v-gon
p_x
type 1 cutter
p_j
p_i
p_y
p_x
sheet 1
type 2 cutter
p_i
p_j
sheet 2
p_y

Figure 5.4

Lemma 5.3: *Let l_c be the only cutter of the resultant active sub-v-polygon of a blue sub-v-polygon and a red sub-v-polygon, where $l_c = \overline{p_i p_j}$. The shared BV-boundary of the blue and red sub-v-polygons consists of two parts (BV-chains) which are undivided with respect to l_c.*

Proof:

We shall show that (1) the shared BV-boundary consists of two parts undivided with respect to l_c, and (2) each part is either a BV-chain or a BV-chain and a pseudo-edge on l_c that may or may not connect to each other.

(1) We consider two subcases: (a) both p_i and p_j of l_c lie on the resultant sub-v-polygon (refer to the upper part of Fig.5.4) and (b) only one of p_i and p_j lies on the resultant sub-v-polygon (refer to the lower part of Fig.5.4).

(a) Without loss of generality, let p_i be a blue data point. Any shared BV-edge cannot cross cutting obstacle l_c by the definition of undividability. Any shared BV-edge cannot cross rays $p_i\vec{x}^-$ and $p_j\vec{x}^+$ (we only discuss one of them because of the symmetry of the two cases). Suppose some shared BV-edge crosses $p_i\vec{x}^-$ (let the crossover point be c), where the shared BV-edge is determined by an arbitrary red data point, say p_r, and an arbitrary blue data point, say p_k. Then c must be a point equidistant to p_r and p_k. Note that the x-coordinate of p_r must be larger than that of p_i, hence p_i must lie inside the circle with $\overline{cp_r}$ as radius which contradicts the definition of shared BV-edges. Therefore, no shared BV-edge can cross $p_i\vec{x}^- \cup l_c \cup p_j\vec{x}^+$.

(b) By the same argument as in (a), no shared BV- (pseudo-) edge (rays) can cross l_c and ray $p_i\vec{x}^-$. Moreover, two shared BV-edges, one of them determined by a pair of data points lying above the cutter and the other determined by a pair of data points lying below the cutter, cannot connect. This is because they will lie on

different sheets if they cross the v-edge of the polygon. Therefore, no shared-BV-edge can cross the cutter.

(Remark: If no red data point lies on one side (or both sides) of the line extending l_c, then the blue p_i or some blue data point must determine some shared pseudo-edge with the red portion of l_c by definition. The pseudo-edge lying on one side of the line extending l_c must not connect to the shared BV-edges and pseudo-edges lying on the other side since they are undivided with respect to l_c.)

(2) Let us consider one of the two parts of the shared BV-boundary separated by the cutter of a cutting obstacle $l_c = \overline{p_i p_j}$. Let the part be denoted as SVB_1. We imagine that a sheet (an edge-induced halfplane $\overline{H}(l_c)$) is attached on the line extending l_c, where l_c now is not regarded as an obstacle. Note by definition that each BV-edge (ray) of SVB_1 must be determined by a red data point and a blue data point such that the line segment spanning them does not cross any obstacle. This is true particularly for l_c since $l_c \in \mathbf{L}$ and each pseudo-edge of SVB_1 must be determined by a red (blue) data point and a blue (red) obstacle such that the line segment spanning the data point and any point of the pseudo-edge does not cross any obstacle. Note also that the attached sheet does not contain any data point and obstacle. Then, all the red and blue data points and obstacles which originally determine SVB_1 except l_c will determine a new shared BV-boundary SVB_1' on the space $S \bigcup \overline{H}(l_c)$, where S is the original space. Note that these data points and obstacles are linearly separable in $S \bigcup \overline{H}(l_c)$, hence the shared BV-boundary SVB_1' is an infinite BV-chain by Lemma 5.2. Let us consider the difference between SVB_1 and SVB_1'. Note that SVB_1 and SVB_1' are determined by the same set of data points and obstacles except l_c, and note also that if we add l_c to the space $S \bigcup \overline{H}(l_c)$, we only need remove the portion of SVB_1' lying on $\overline{H}(l_c)$ and add a shared-pseudo-edge to SVB_1' which is on l_c, because there is no data point

on the sheet $\overline{H}(l_c)$. Thus, the portion of SVB_1' lying on the original space S must be the same as SVB_1 except the shared pseudo-edge on l_c. The part of SVB_1' lying on S is a BV-chain, hence SVB_1, which is separated by the cutter, consists of a BV-chain and a pseudo-edge on l_c which may or may not connect to the BV-chain. □

Lemma 5.4: *The shared BV-boundary of a blue v-polygon and a red v-polygon consists of several undividable BV-chains separated by the cutters of the cutting-obstacles and separated by the crossing-obstacles of the resultant v- polygon.*

Proof:

The lemma follows from Lemma 5.3 and the fact that the shared BV-boundary is separated by the crossing-obstacles of the resultant v-polygon. □

Let v-line L_s separate a blue sub-v-polygon P^1 and a red sub-v-polygon P^2, and let $Vor(\mathbf{P}^1;\mathbf{L}^1)$ and $Vor(\mathbf{P}^2;\mathbf{L}^2)$ be the two pseudo-BV-diagrams of P^1 and P^2 respectively. Let l_1 and l_2 cross L_s, where l_1 (l_2) can be an obstacle or an obstacle segment. We assume that the crossover points of l_1 and l_2, say c_1 and c_2 are visible from each other. Thus, l_1 and l_2 are two *adjacent* cutting obstacles of the resultant sub-v-polygon. Let $CH(\mathbf{P}^i)$ and $CH(\mathbf{P}^j)$ be two convex hulls, where $\mathbf{P}^i$ and $\mathbf{P}^j$ are the set of blue data points and the set of red data points respectively such that the BV-regions of these blue and red data points cross $\overline{c_1c_2}$ (refer to Fig.5.5).

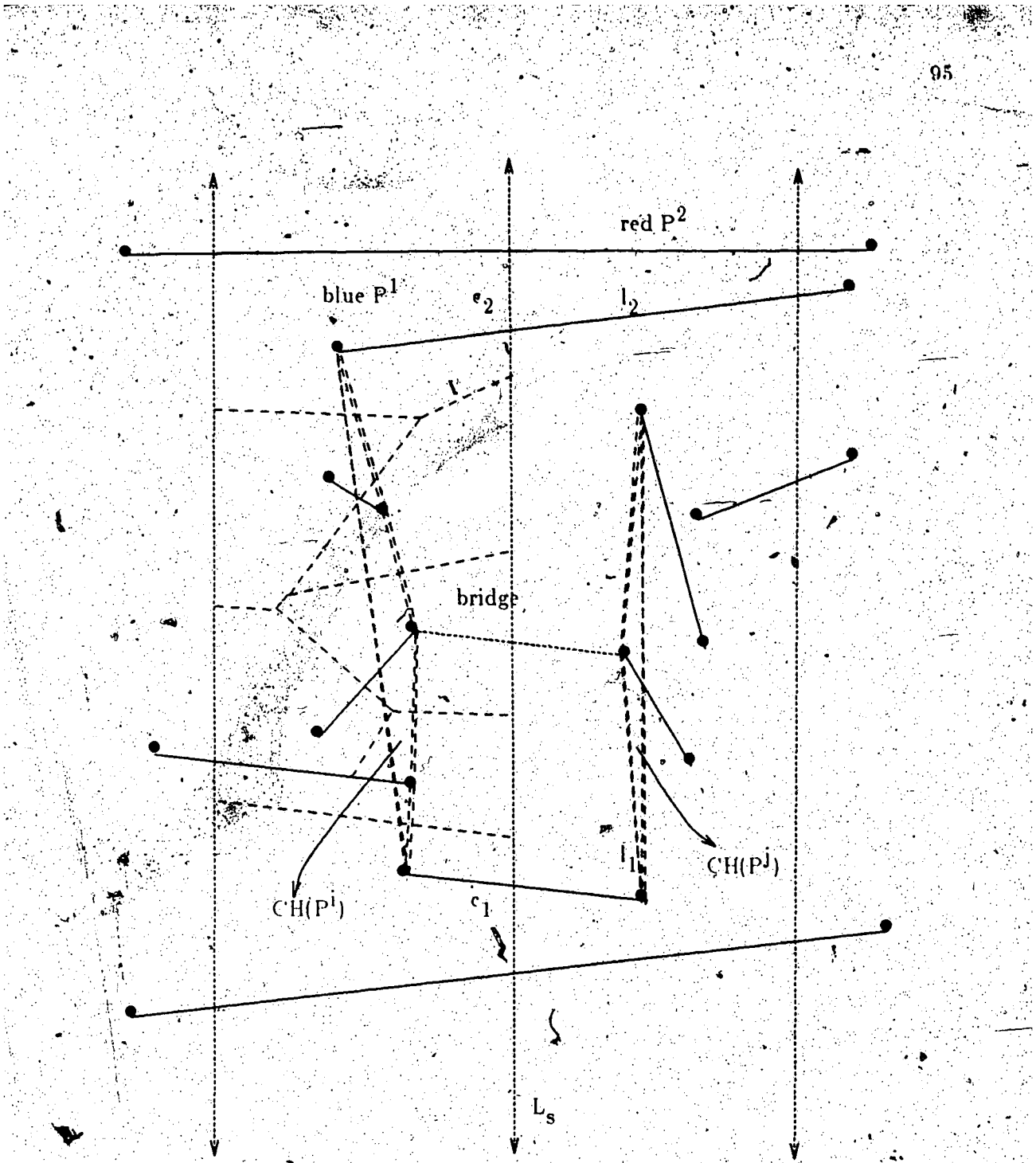


Figure 5.5

We call an edge $e=\overline{p_x p_y}$ a bridge of $CH(\mathbf{P}^i)$ and $CH(\mathbf{P}^j)$ if $p_x \epsilon \mathbf{P}^i$ and $p_y \epsilon \mathbf{P}^j$, and e does not cross the boundaries of $CH(\mathbf{P}^i)$ and $CH(\mathbf{P}^j)$. In the following, we present a procedure to find a bridge of $CH(\mathbf{P}^i)$ and $CH(\mathbf{P}^j)$.

Procedure FindingBridges

Input: A blue and a red sub-v-polygon and their pseudo-BV-diagrams, the crossover points c_1 and c_2 of two adjacent cutting or crossing obstacles

Output: a bridge e

Method:

It is easy to find $\mathbf{P}^i$ and $\mathbf{P}^j$ by checking the BV-regions which crosses $\overline{c_1 c_2}$.

Since $CH(\mathbf{P}^i)$ and $CH(\mathbf{P}^j)$ are two linearly separable convex polygon, there must exist a bridge, say $e=\overline{p_x p_y}$, between them such that p_x is the data point with the maximal x-coordinate of $\mathbf{P}^i$ and p_y is the data point with the minimal x-coordinate of $\mathbf{P}^j$. Hence, they can be found in a simple scan on the x-coordinates of these data points.

(In the trivial case that $\mathbf{P}^i$ (or $\mathbf{P}^j$) contains no data point, then no bridge exists since the portion of the shared BV-boundary consists of only pseudo-edges.)

Lemma 5.5: *Given a blue sub-v-polygon* P^1, *a red sub-v-polygon* P^2, *and their pseudo-BV-diagrams* $Vor(\mathbf{P}^1,\mathbf{L}^1)$ *and* $Vor(\mathbf{P}^2,\mathbf{L}^2)$, *and two adjacent cutting or crossing obstacles* l_1 *and* l_2, *a bridge* e *of* $CH(\mathbf{P}^i)$ *and* $CH(\mathbf{P}^j)$ *can be found in time proportional to the number of BV-elements (BV-edges, BV-rays, and pseudo-edges) associated with the data points of* $\mathbf{P}^1 \cup \mathbf{P}^2$.

Proof:

We first consitler the time complexity of constructing $\mathbf{P}^i$ (respectively $\mathbf{P}^j$). The data point whose BV-region contains c_1 can be found in constant time. (We shall see in Lemma 5.7 that when we compute the adjacent cutting or crossing obstacles for two v-polygons which will be merged, we also compute the BV-regions

containing the crossover points of the separator L_s and the cutting or crossing obstacles. This can be done in time proportional to the number of data points in the blue and red v-polygons). The sequence of data points can be found in time proportional to the number of BV-elements of the BV-region crossing $\overline{c_1c_2}$ since the neighboring BV-region can be found in constant time and the BV-edges crossing $\overline{c_1c_2}$ can be identified in constant time. Obviously, the number of these BV-regions is bounded by $|\mathbf{P}^i|$.

We now consider the time complexity of constructing a bridge e. Note that determining the data point of $\mathbf{P}^i$ with maximal x-coordinate and the data point of $\mathbf{P}^j$ with the minimal x-coordinate can be done in time proportional to the size of the two sets. Hence, e can be found in time proportional to the number of BV-elements associated with the data points of $\mathbf{P}^i \bigcup \mathbf{P}^j$. □

Definition: A ray is a *starting segment* of a BV-chain if its initial part belongs to one of the segments of the BV-chain.

Let P^1 and P^2 be the blue sub-v-polygon and the red sub-v-polygon, respectively. Let $Vor(\mathbf{P}^1;\mathbf{L}^1)$ and $Vor(\mathbf{P}^2;\mathbf{L}^2)$ be their pseudo-BV-diagrams. Let l_1 and l_2 be two adjacent cutting or crossing obstacles, and let $CH(\mathbf{P}^i)$ and $CH(\mathbf{P}^j)$ be the two convex hulls defined as in Lemma 5.5. Then, by Lemma 5.3, there exists a BV-chain separated by the cutters of l_1 and l_2 which is part of the shared BV-boundary of P^1 and P^2. Let the BV-chain be SVB_1. In the following, we present a procedure to find the two starting segments of SVB_1, in the case that each of $\mathbf{P}^1$ and $\mathbf{P}^2$ contains at least one data point (since otherwise, the shared BV-boundary consists of pseudo-edges which are easy to find).

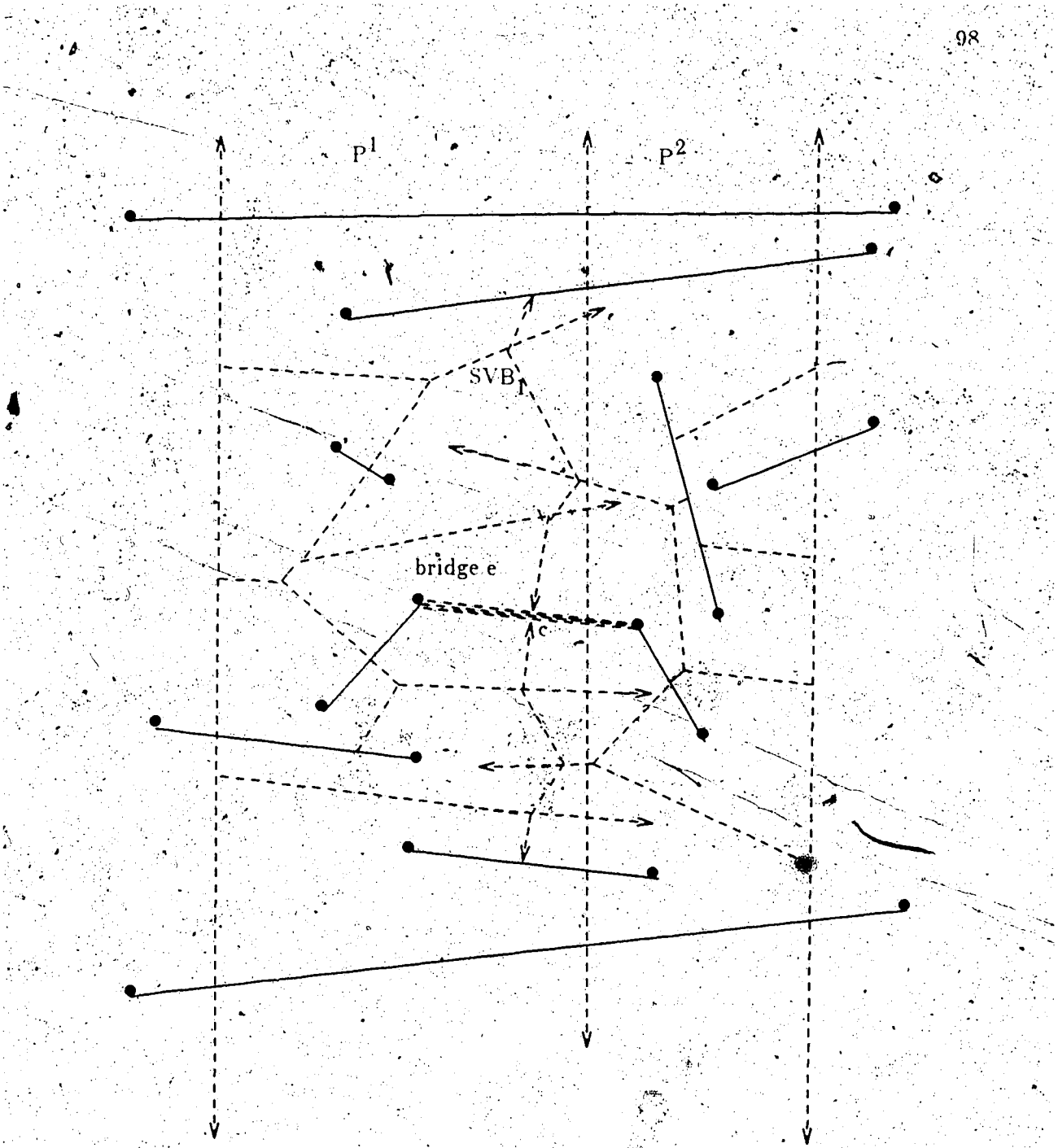


Figure 5.6

Procedure FindingStartingSegment

Input: A blue and a red sub-v-polygon and their pseudo-BV-diagrams and a bridge e

Output: the starting segments of SVB_1

Method:

Note that SVB_1 must cross e since its endpoints have distinct colors.

Now, we show some properties of c. By definition, c must be equidistant to a blue data point, say p_a, and to a red data point, say p_b, where c is visible from p_a and p_b, and p_a is the data point closest to c in $\mathbf{P}^1$ and p_b is the data point closest to c in $\mathbf{P}^2$. Thus, c must lie on BV-regions $BV(p_a)$ and $BV(p_b)$ and c also must be on the perpendicular bisector of $\overline{p_a p_b}$. By definition, e does not cross any obstacle, hence c must not belong to any obstacle.

According to the above three properties of c, we can find c by a scan on e from one end of e to the other end. To do so, we order the blue BV-regions and the red BV-regions on e according to their crossover points of their BV-edges (rays) on e. The ordering can be found by searching on the boundaries of these BV-regions crossing e. Thus, e is divided into a sequence of segments such that each segment is contained by a blue BV-region and a red BV-region. We now examine each segment to determine if the segment intersects the perpendicular bisector of the two data points whose BV-regions contain this segment. If the segment intersects the bisector, then the two rays radiating from the intersecting point in the bisector directions are the starting segments by definition. Otherwise, we examine the next segment in the sequence. Since c must exist on e, the search must terminate.

Lemma 5.6: *Given* $\mathrm{Vor}(\mathbf{P}^1{:}\mathbf{L}^1)$ *and* $\mathrm{Vor}(\mathbf{P}^2{:}\mathbf{L}^2)$, *and given a bridge e of* $\mathrm{CH}(\mathbf{P}^1)$ *and* $\mathrm{CH}(\mathbf{P}^2)$, *the starting segments of the BV-chain* SVB_e *can be found in time proportional to the number of the BV-elements (BV-edges, BV-rays and pseudo-edges) whose BV-regions cross e.*

Proof: (refer to Fig.5.6)

Note that finding the sequence of the segments on e, where each segment is contained by a blue and a red BV-region, takes time proportional to the number of BV-elements whose BV-regions cross e. This is because $\mathrm{Vor}(\mathbf{P}^1{:}\mathbf{L}^1)$ and $\mathrm{Vor}(\mathbf{P}^2{:}\mathbf{L}^2)$ are given and hence the sequence of the crossover points of the blue and red BV-edges (rays) on e can be found in time proportional to the number of BV-elements whose BV-regions cross e, and because the segments can be determined by merging the two ordered lists of crossover points (which takes time proportional to the number of the crossover points in the lists). Note also that determining if a segment intersects the corresponding bisector takes constant time. Note finally that the search of e need not backtrack. Thus, the starting segment can be found in time proportional to the number of BV-elements whose BV-regions cross e. □

Lemma 5.7: *Given a blue v-polygon* P^1 *and a red v-polygon* P^2, *and given* $\mathrm{Vor}(\mathbf{P}^1{:}\mathbf{L}^1)$ *and* $\mathrm{Vor}(\mathbf{P}^2{:}\mathbf{L}^2)$, *the starting segments of all BV-chains of SVB can be found in time proportional to the number of data points and obstacle segments in the two v-polygons, where SVB denote the shared BV-boundary of* P^1 *and* P^2.

Proof:

By Lemma 5.3, SVB consists of several BV-chains (and possibly some pseudo-edges on cutting obstacles which may be disjoint from the BV-chains), and each BV-chain is separated by the cutters of the two cutting obstacles or crossing-obstacles of the resultant v-polygon. To find the starting segments of each BV-

chain, we must first find the adjacent cutting or crossing obstacles.

To do so, we scan on L_s from the point on the bottom of CH(**P**) upwards to examine each active resultant sub-v-polygon (which are available after combining the two old sub-v-polygons). We record all obstacles (which cross L_s) and their crossover points (i.e., c_1, c_2,....) into a list for all active sub-v-polygons. We also chase Vor($\mathbf{P}^1$:$\mathbf{L}^1$) and Vor($\mathbf{P}^2$:$\mathbf{L}^2$) to find the BV-regions which contain the crossover points (this information is available from the two pseudo-BV-diagrams). Then, the adjacent cutting or crossing obstacles can be found in constant time from the list.

By procedure **FindingBridges** and procedure **FindingStartingSegment**, we can find the starting segments for each BV-chain which corresponds to two adjacent cutting or crossing obstacles.

The scan takes time proportional to the number of data points and obstacle segments in the resultant v-polygon since the number of cutting or crossing obstacles is bounded by the number of data points and obstacle segments in the resultant v-polygon and the scan does not backtrack. The starting segments for each BV-chain can be found in time proportional to the number of BV-elements of the BV-regions crossing $\overline{c_1c_2}$ of L_s. Thus, all the starting segments can be found in time proportional to the number of BV-elements of BV-regions crossing L_s. This is because each BV-region can cross L_s at most one time, and each BV-region can cross at most one bridge. The number of BV-elements whose BV-regions cross L_s and the bridges is proportional to the number of data points in the resultant v-polygon. □

Now, we can design a merge algorithm to find the shared BV-boundary of a blue v-polygon and a red v-polygon. In the algorithm, P^1 and P^2 are two adjacent v-polygons. Attached to each are two lists (the left and the right lists) which keep the

information of the v-edges (and their segments, if any) of the active sub-v-polygons. We also assume that the lists contain additional information that is attached to each segment of the v-edge of the active sub-v-polygons and indicates which data points define the BV-regions containing the endpoints of the segment. $\mathbf{L}^1$ and $\mathbf{L}^2$ are two sets of obstacles and obstacle segments lying on $\mathbf{P}^1$ and $\mathbf{P}^2$ respectively. $\mathbf{P}^1$ and $\mathbf{P}^2$ are two sets of corresponding data points respectively. $Vor(\mathbf{P}^1;\mathbf{L}^1)$ and $Vor(\mathbf{P}^2;\mathbf{L}^2)$ are the pseudo-BV-diagrams. The rest of the notation and assumptions follow from those of Algorithm 1.

Algorithm 4

Input: $\mathrm{Vor}(\mathbf{P}^1; L^1)$, $\mathrm{Vor}(\mathbf{P}^2; \mathbf{L}^2)$, P^1, and P^2.

Output: $\mathrm{Vor}(\mathbf{P}^1 \bigcup \mathbf{P}^2; L^1 \bigcup L^2)$ and $P^1 \bigcup P^2$.

Method:

1 find the new active sub-v-polygons by procedure **CombiningVgons**;

2 find the starting segments of the BV-chains of the shared BV- boundary by procedures **FindingBridges**, **FindingStartingSegment**, and by the process described in the proof of Lemma 5.7, and push them on a stack N;

3 *While* $N \neq \phi$ *Do*

b$(v;p,q)$ ← POP(N);

Repeat

(a) find the BV-edge (or BV-ray or pseudo-edge) $(u,w;r,s)$, which first intersects b$(v;p,q)$;

(b) find the next branch according to the following cases:

b$(v;p;q)$ o $L^1 \bigcup L^2$ meets $(u,w;r,s) \notin L^1 \bigcup L^2$

or b$(v;p,q) \notin L^1 \bigcup L^2$ meets $(u;w;r,s) \notin L^1 \bigcup L^2$

or b$(v;p,q) \notin L^1 \bigcup L^2$ meets $(u,w;r,s)$ o $L^1 \bigcup L^2$;

(c) b$(v;p,q)$ ← the branch;

Until (the branch $o(L^1 \bigcup L^2)$ meets a v-line or the branch meets a cutting obstacle or a BV-ray)

find the pseudo-edge lying on the cutting obstacle, if any;

End Do.

Lemma 5.8: *Given a blue v-polygon* P^1 *and a red v-polygon* P^2, *and their corresponding pseudo-BV-diagrams* $Vor(\mathbf{P}^1;L^1)$ *and* $Vor(\mathbf{P}^2;L^2)$, *Algorithm 4 takes* $O(|\mathbf{P}^{1'} \cup \mathbf{P}^{2'}|)$ *time to produce* $Vor(\mathbf{P}^1 \cup \mathbf{P}^2; L^1 \cup L^2)$, *where* $\mathbf{P}^{1'}$ ($\mathbf{P}^{2'}$) *represents the union of the data point set and obstacle segment set in the blue (red) v-polygon.*

Proof:

By Lemma 5.4 and Lemma 3.4, Algorithm 4 produces the pseudo-BV-diagram of $\mathbf{L}^1$ and $\mathbf{L}^2$. By Lemma 5.2, Lemma 5.7, and Lemma 3.4, Algorithm 4 takes time proportional to $O(|\mathbf{P}^{1'} \cup \mathbf{P}^{2'}|)$ to produce $Vor(\mathbf{P}^1 \cup \mathbf{P}^2; L^1 \cup L^2)$. □

In the following, we present an example to summarize the merge algorithm (refer to Fig.5.7). Given a blue v-polygon and a red v-polygon as shown in the upper part of Fig.5.7 and given their corresponding blue pseudo-BV-diagram and the red pseudo-BV-diagram as shown in Fig.5.8, we now use procedure **CombiningVgons** to find the new active v-polygon. The blue v-gon contains the left list (e_{l1}, e_{l1}', e_{l3}) and the right list (e_{r1}, e_{r1}', e_{r3}, e_{r3}') and the red v-gon contains the left list ($\overline{e_{l1}}$, $\overline{e_{l1}}'$, $\overline{e_{l1}}''$) and the right list ($\overline{e_{r1}}$). The new v-polygon contains the left list (e_{l1}, e_{l1}', e_{l1}'', e_{l1}''') and the right list (e_{r1}, e_{r1}', e_{r1}''). We then use procedure **FindingBridges** to find the bridges of the resultant v-polygon. The upper part of Fig.5.9 shows the bridges, where there are four cutting obstacles in the resultant v-gon, hence there exist five segments of L, separated by the obstacles. Among them, the first, the third, and the fifth segment from bottom correspond to degenerate cases (only one data point is involved). We finally use procedure **FindingStartingSegement** to find the starting segment of each separated portion of the shared BV-boundary. At this point, we can find the shared BV-boundary of the pseudo-BV-diagrams of the two v-gons by Step 3 of the Algorithm 4. Step 3 is very similar to that of Algorithm 1 except for the termination condition. The bottom part of Fig.5.9 shows the shared BV-boundary.

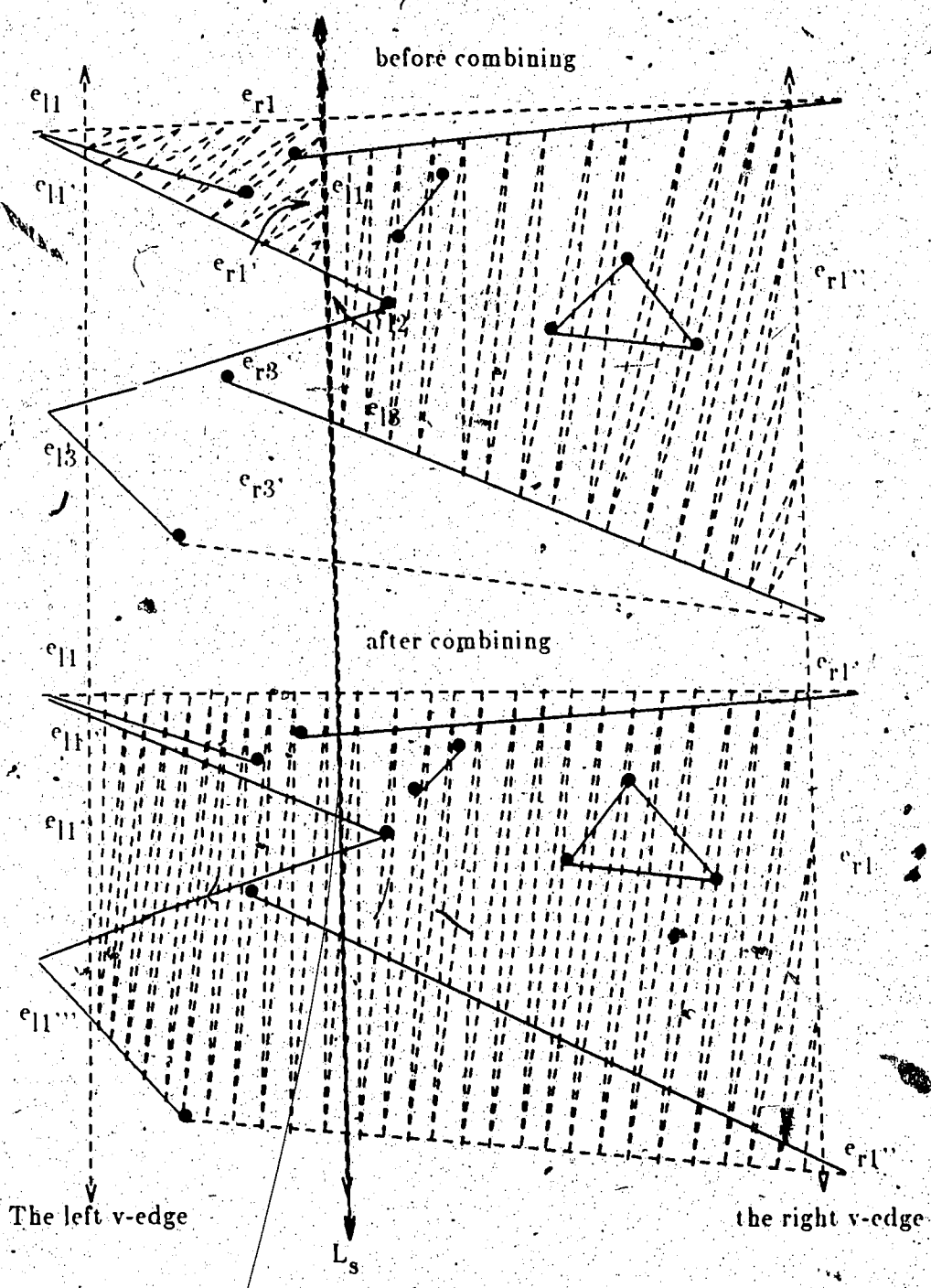

before combining
e_{l1}
e_{r1}
$e_{l1'}$
e_{l1}
$e_{r1'}$
$e_{r1''}$
e_{r3}
e_{l3}
$e_{r3'}$
after combining
e_{l1}
$e_{r1'}$
$e_{l1'}$
e_{l1}
e_{r1}
$e_{l1'''}$
$e_{r1''}$
The left v-edge
the right v-edge
L_s

Figure 5.7

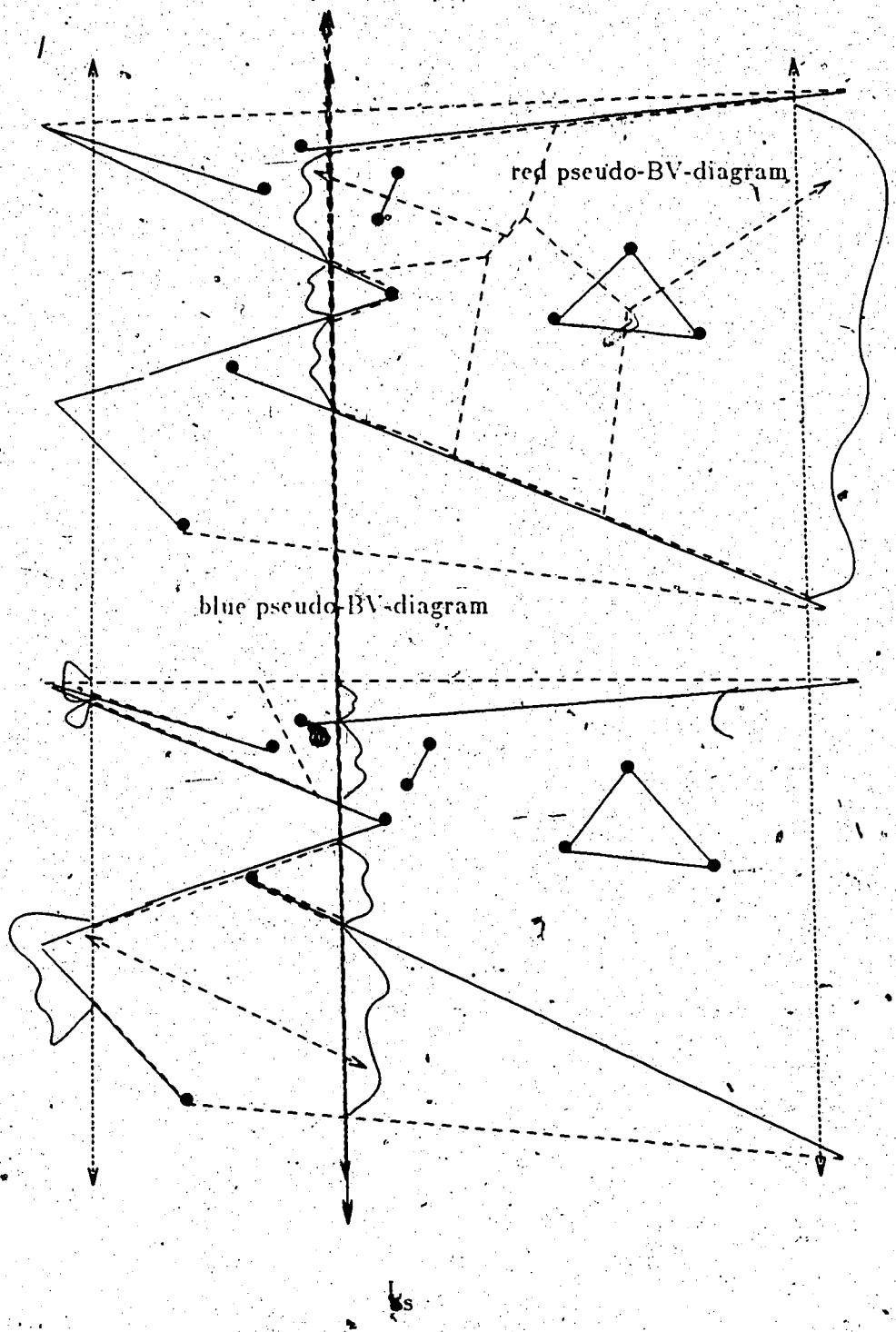


Figure 5.8

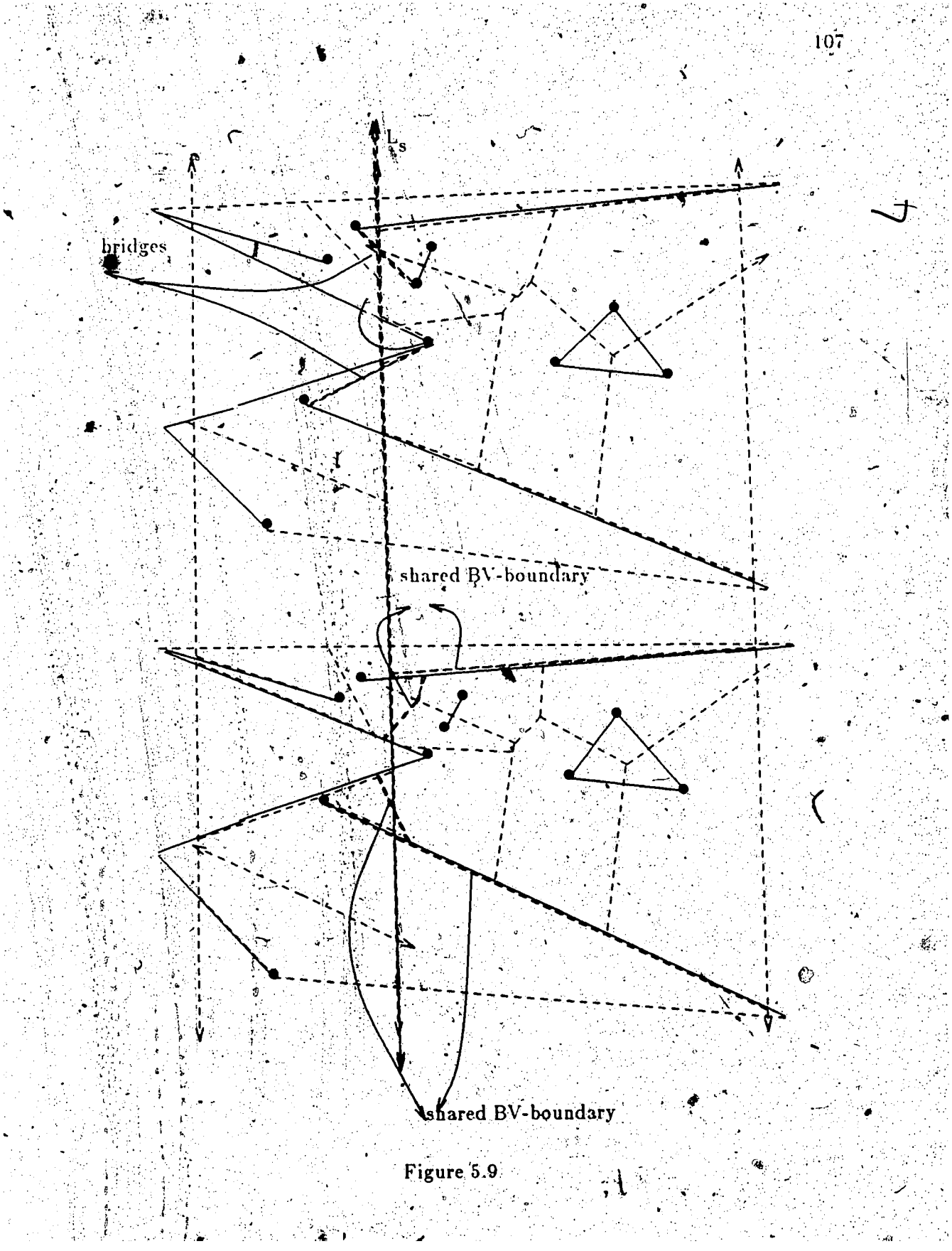


Figure 5.9

It is not difficult to design an O(n log n) divide and conquer algorithm for constructing the BV-diagram of a set L of n obstacles. This is because we have a linear time merge algorithm.

Theorem 1: *The BV-diagram of a set of n obstacles can be constructed in*

$O(n \log n)$ *time.*

Proof: By the divide and conquer method, and by Lemma 5.8. □

Theorem 2: *The construction of the BV-diagram of a set of n obstacles takes at least*

$\Omega(n \log n)$ *time.*

Proof:

We shall show that the problem of sorting n distinct real numbers can be transformed in linear time to the problem of constructing the BV-diagram of n non-overlapping and non-crossing line segments. To do so, we first transform n distinct real numbers to n line segments in the plane as follows. Suppose that s is such a real number. We can create a line segment l_s in the plane in constant time, such that the x- and y- coordinates of the two endpoints of l_s are (s,0) and (s,1), respectively. We do the above transformation for each real number, which will result in n line segments parallel to the y-axis. We then construct the BV-diagram of these n line segments. The output of the BV-diagram is a sequence of O(n) BV-edges and pseudo-edges. Note that if two data points both with zero y-coordinate share a BV-edge, then the two data points must be adjacent on the x-axis. Note also in the BV-diagram that if a data point is not the endpoint of a line segment with the largest or the smallest x-coordinate, then it has three data points that are Voronoi neighbors, otherwise it has two data points which are Voronoi neighbors. Thus, by examining the y-coordinates of the neighboring data points (at most three) of each data point, we can obtain a sorted sequence of data points with zero y-coordinates in O(n) time. Thus, we obtain a sorted list of n dis-

tinct real numbers. Since the sorting problem has $\Omega(n \log n)$ lower bound, the construction of the BV-diagram of a set of n non-crossing line segments must takes at least $\Omega(n \log n)$ time. □

Chapter 6

Concluding remarks

In this thesis, a new generalized Voronoi diagram in the plane, called the BV-diagram (BV- diagram) is defined. The BV- diagram is one of the natural extensions of the previously existing Voronoi diagrams.

In the BV- diagram, the input information contains the "data points" and "obstacles". The data points are the input information which determine the existence of the Voronoi components and the obstacles are the input information which determine the shape of the Voronoi components. The concept of dividing the input information into data and obstacles can be extended to the other classes of Voronoi diagrams: for example, the kth-order BV- diagram, the BV- diagrams in three or higher dimensional space, the bounded weighted Voronoi diagram, and the BV- diagram in L_p metrics.

Three important cases of the BV- diagrams are investigated, namely, the BV-diagram for a monotone chain, for a simple polygon, and for a set of non-crossing line segments. The algorithms to construct these diagrams are presented. All of them are optimal.

Many problems in computational geometry can be solved efficiently if the BV-diagram of a set of line segments is available (by using the same approach as for the standard Voronoi diagram). They include:

(1) Given a set **L** of n line segments in the plane and the BV-diagram of that set, the pair of closest and straight line reachable endpoints of **L** can be found in O(n) time.

(2) Given a set **L** of n line segments in the plane and the BV-diagram of that set, the pair of closest and straight line reachable endpoints for each endpoint in **L** can be found in O(n) time.

(3) Given a set **L** of n line segments in the plane and the BV- diagram of that set, the Euclidean minimum spanning tree of that set **L** can be found in O(n) time.

(4) Given a set **L** of n line segments in the plane, the closest and straight line-reachable endpoint of a query point q can be found in O(log n) time with O(n) space and O(n log n) preprocessing time.

(5) Given a set **L** of n line segments in the plane and the BV-diagram of that set, the convex hull of that set **L** can be found in O(n) time.

(6) Given a set **L** of n line segments in the plane and the BV- diagram of that set, the Gabriel graph of that set **L** can be found in O(n) time. (Remark: The Gabriel graph of a set of line segments is a graph such that each edge e_g is determined by two endpoints in **L** and no other endpoint of **L** lies inside the circle with e_g as diameter and is visible from the two endpoints of e_g.)

A subject of further research is to prove that the Delaunay triangulation of a set of n non-crossing line segments is the pseudo-dual of the BV-diagram of that set, so that the Delaunay triangulation can be obtained from the BV-diagram in O(n) time. Our approach is to prove that every Delaunay edge of the Delaunay triangulation must correspond to either a BV-edge (ray) or a pseudo-edge of the BV-diagram lying on the plane or a BV-edge (ray) of the pseudo-exterior BV-diagram lying on some sheet attached to an obstacle. We can maintain the latter information during the construction of the BV-diagram with extra linear time and space.

There are many problems remaining open under the concept of the BV- diagram (where the endpoints of the input line segments are not necessarily the data points). Some open problems are listed as follows.

(1) Given a set of n data points inside an n-gon, can we design an $O(n^2)$ algorithm to construct the BV- diagram of that set (where the n-gon is regarded as the obstacle)?

(2) Given a set of n data points and n line segments in the plane can we design an $O(n^4)$ algorithm to construct the BV- diagram of that set (where the n line segments are regarded as the obstacles)?

(3) Find the BV- diagram of a set of points with a set of non-crossing facets as obstacles in three or higher dimensional space.

(4) Find the kth-order BV- diagram of a set of points with n line segments as obstacles.

(5) Find the weighted BV- diagram of a set of points with n line segments as obstacles.

REFERENCES.

[Agg87] Aggarwal P., Raghavan P., and Tiwari P., (1987) "Lower bounds for computing the closest pair in simple polygons and related problems", Manuscript.

[Aho76] Aho A., Hopcroft J., and Ullman J., (1976) "*The Design and Analysis of Computer Algorithms*", Addison-Wesley.

[Aro87] Aronov B., (1987) "On the geodesic Voronoi diagram of point sites in a simple polygon", *Proc. of 3rd ACM Symposium on Computational Geometry*, Waterloo, Canada, pp.39-49.

[Asa85] Asano T., Asano T., Guibas L., Hershberger J., Imai H., (1985) "Visibility polygon search and Euclidean shortest paths", *Proc. 26th Annual IEEE Symposium on Foundation of Computer Science*, pp. 155-164.

[Aur84] Aurenhammer F. and Edelsbrunner H., (1984), "An optimal algorithm for constructing the weighted Voronoi diagram in the plane", *Pattern Recognition*, 17(2), pp.251-257.

[Avi83] Avis D. and Bhattacharya B., (1983), "Algorithms for computing d-dimensional Voronoi diagrams and their duals", *Advances in Computing Research*, Edited by Preparata F., I.JAI press, pp.179-209.

[Ben76] Bentley J. and Shamos M., (1976), "Divide and conquer in multidimensional space", *Proc. of the Eight Annual ACM Symposium on Theory of Computing*, pp. 220-230.

[Ben78] Bentley J. and Friedman J., (1978) "Fast algorithms for constructing minimum spanning tree in coordinate spaces", *IEEE Trans. on Computers*, Vol.C-27, No. 2, pp. 97-105.

[Ben79] Bentley J. and Ottmann T., (1979), "Algorithms for reporting and count-

ing geometric intersections", *IEEE Trans. on Computers*, Vol. C-28, pp. 643-647.

[Ben80] Bentley J., (1980) "Multidimensional divide and conquer", *Communications of ACM*, Vol.23, No.4, pp.214-219.

[Bol79] Bollobas B., (1979), "*Graph Theory, An Introductory Course*", Springer-Verlag, New York.

[Bro79] Brown K., (1979), "Voronoi diagrams from convex hulls", *The Information Processing Letters*, Vol.9, No.5, pp.223-228.

[Cha79] Chand D. and Kapur S., (1970), "An algorithm for convex polytopes", *Journal of ACM*, Vol.17, No.k, pp.78-86.

[Chaz80a] Chazelle B., (1980), "Computational geometry and complexity", Technical Report, Carnegie-Mellon University.

[Chaz80b] Chazelle B. and Dobkin D., (1980), "Detection is easier than computation", *Proc. of 11th ACM Annual Symposium on Theory of Computing*, pp.153-161.

[Chaz82] Chazelle B., (1982), "A theorem on polygon cutting with applications", *Proc. of 23rd IEEE Annual Symposium of Foundation of Computer Science*, pp.339-349.

[Chaz83] Chazelle B., Guibas, and Lee D. T., (1983), "The power of geometric duality", *Proc. of 24th IEEE Annual Symposium on Foundation of Computer Science*, pp.217-225.

[Chaz85] Chazelle B. and Edelsbrunner H.,(1985), "An improved algorithm for constructing k-th order Voronoi diagrams", *Proc. of 1st ACM Symposium on Computational Geometry*, pp.228-234.

[Che76] Cheriton D. and Tarjan R., (1976), "Finding minimum spanning trees", *Journal of ACM*, pp.724-742.

[Chew85] Chew L., Drysdale R., (1985), "Voronoi diagrams based on convex distance functions", *Proc. of 1st ACM Symposium on Computational Geometry*, pp.235-244.

[Chew87] Chew L., (1987), "Constrained Delaunay triangulations", *Proc. of 3rd ACM Symposium on Computational Geometry*, pp.215-222.

[Chi83] Chin F. and Wang C., (1983), "Optimal algorithms for the intersection and the minimum distance problems between planar polygons", *IEEE Transactions on Computers*, Vol.C-32, pp.1203-1207.

[Chi84] Chin F. and Wang C., (1984), "Minimum vertex distance between separable convex polygons", *Information Proceeding Letters*, Vol.18, pp.41-45.

[Dob83] Dobkin D. and Kirkpatrick D., (1983), "Fast detection of polyhedral intersection", *Theoretical Computer Science*, Vol.27, pp.241-253.

[Dob85] Dobkin D. and Kirkpatrick D., (1985), "A linear algorithm for determining the separation of convex polyhedra", *Journal of Algorithms*, pp.241-253.

[Dry79] Drysdale R., (1979), "Generalized Voronoi diagrams and geometric searching", Stan-cs-79-705 computer science technical report, Stanford University (PhD thesis)

[Ede82] Edelsbrunner H., (1982), "Intersection problems in computational geometry", PhD Dissertation, IIG, Technische University, Graz, Austria, Rep 93.

[Ede83] Edelsbrunner H.,O'Rourke J., Seidel R., (1983), "Constructing arrangements of lines and hyperplanes with applications", *Proc. of 24th IEEE Annual Symposium on Foundation of Computer Science*, pp.83-91.

[Ede86] Edelsbrunner H.,O'Rourke J., Seidel R., (1986), "Constructing arrangements of lines and hyperplanes with applications", *SIAM J. Computing.*, Vol.15, No.2, pp.341-363.

[Ede86] Edelsbrunner H.,Guibas L., and Stolfi J., (1986), "Optimal point location in a monotone subdivision", *SIAM J. Computing.*, Vol.15, No.2, pp.317-340.

[Fort86] Fortune S. (1986), " A sweepline algorithm for Voronoi diagrams", *Proceedings of thee 2nd Symposium on Computational Geometry*, NY, pp.313-322.

[Fort87] Fortune S. (1987), " A sweepline algorithm for Voronoi diagrams", *Algorithmica*, pp.153-174.

[Gar78] Garey M. Johnson D., Preparata F., and Tarjan R., (1978), "Triangulating a simple polygon", *Information Processing Letters*,Vol.7,No.4, pp.175-180.

[Gin81] ElGindy H., Avis D., (1981), "A linear algorithm for computing the visibility polygon from a point", *J. of Algorithm*, 2, pp.186-197.

[Gui85] Guibas L., Stolfi J., (1985), "Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams", *ACM Trans. on Graphics*, Vol.4, No.2, pp.74-122.

[Kirk79] Kirkpatrick D., (1979), "Efficient computation of continuous skeletons", *Proceedings of the Twentieth Annual Symposium on the Foundations of Computer Science*, pp. 18-27.

[Kirk83] Kirkpatrick D., (1983), "Optimal search in planar subdivisions", *SIAM J. Computing*, Vol.12, No.1, pp.28-35.

[Law76] Lawson C., (1976), "C^1-compatible interpolation over a triangle", Jet Propulsion Lab. *Tech. Memo.*, 33-770.

[Lee78] Lee D. T., (1978), "Proximity and reachability in the plane", Coord. Sci. Lab., Univ. Illinois Technical report, R-831.

[Lee79] Lee D. T., Drysdale R., (1981), "Generalized Voronoi diagrams in the plane", *SIAM J. Computing*, 10, 1, pp.73-87.

[Lee80a] Lee D. T., Schachter B., (1980), "Two algorithm for constructing Delaunay triangulations", *Int. J. Comput. Inform. Sci.*, Vol. 9, No. 3, pp. 219-242.

[Lee80b] Lee D. T., (1980), "Two dimensional Voronoi diagram in the L_1 metric", *J. ACM*, pp. 604-618.

[Lee82a] Lee D. T., (1982), "On K-nearest neighbor Voronoi diagrams in the plane", *IEEE Trans. Computers*, pp. 478-487.

[Lee82b] Lee D. T., Preparata F., (1982), "Euclidean shortest paths in the presence of rectilinear barriers", *Networks*, Vol 14, pp.393-410.

[Lee84] Lee D. T., Preparata F., (1984), "Computational geometry - a survey", *IEEE Trans. On Computers*, Vol. C-33, pp. 1072-1101.

[Lee85] Lee D. T., Lin A., (1985), "A generalized Delaunay triangulation for a set of line segments in the plane", *Discrete and Computational Geometry*, pp.84-100.

[Mul78] Muller D., Preparata F., (1978), "Finding the intersection of two convex polyhedra", *Theoretical Computer Science* 7(2), pp.217-236.

[Prep77] Preparata F. and Hong S., (1977), "Convex hull of finite sets of points in two and three dimensions", *Commun. ACM*, Vol. 20, pp. 87-93.

[Prep85] Preparata F., Shamos M., (1985), "*Computational Geometry*", Springer Verlag.

[Sar86] Sarnak N., Tarjar R., (1986), "Planar point location using persistent search trees", *Commun. ACM*, pp.669-679.

[Sham75] Shamos M., Hoey D.,(1975), "Closest point problems", *Proc. 16th IEEE Annu. Symp. Found. Comput. Sci.*, pp 151-162.

[Sham78] Shamos M., (1978), "Computational geometry", PhD Thesis, Department of Computer Science, Yale University.

[Sei81] Seidel R., (1981), "A convex hull algorithm optimal for point sets in even dimensions", Rep. 81-14, Dept. of Computer Science, University of British Columbia.

[Sei82] Seidel R., (1982), "The complexity of Voronoi diagrams in higher dimensions", *Proc. 20th Allerton Conference on Comm., Control and Comput.*, pp.94-95.

[Tou82] Toussaint G., Avis D., (1982), "On a convex hull algorithm for polygons and its applications to triangulation problems", *Pattern Recognition*, Vol. 15, pp 23-29.

[Yao80] Yao A. (1980), "Finding the Euclidean minimum spanning tree in d-space", *SIAM J. on Computing*, pp.114-156.

[Yap 85] Yap C. (1985), "An O(n log n) algorithm to construct the Voronoi diagram of a set of simple curves", Technical report, NYU.