HIERARCHICAL DECOMPOSITION OF POLYGONS WITH APPLICATIONS

by

Hossam A. ElGindy

School of Computer Science
McGill University

A dissertation
submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of
DOCTOR OF PHILOSOPHY

ABSTRACT

Computational geometry is the branch of design and analysis of algorithms which deals with the computational aspects of problems involving geometrical objects; i.e., analyzing the complexity of geometrical problems and exploiting the properties of geometrical objects to develop efficient algorithms. The increase in algorithmic studies of different geometric problems resulted in the development of useful problem solving techniques; e.g., decomposition, divide-and-conquer, dynamization and plane-sweep. Also, it was observed that some geometric objects admit more efficient solutions than others; e.g., a monotone polygon compared to an arbitrary simple polygon when the problem at hand is to compute the shortest route between two points.

In this thesis we apply the decomposition technique together with a hierarchical method for arranging the decomposition components in the design of efficient solutions for computing the weak visibility polygon from an edge and computing the shortest route between two points inside a simple polygon, and routing networks inside a rectilinear polygon. Using the same method we present a polynomial algorithm for checking the feasibility of embedding a graph in the plane such that it represents the visibility graph of a simple polygon for a specialized class of simple polygons.

RESUME

Les algorithmes géométriques forment une branche de l'analyse des algorithmes qui traite de l'aspect calculatoire de problèmes impliquant des objets géométriques, c'est-à-dire l'étude de la complexité de ces problèmes et l'utilisation des propriétés des objets géométriques pour développer des algorithmes efficaces. L'étude de l'aspect algorithmique de divers problèmes géométriques a conduit à l'élaboration de techniques de résolution de problèmes, tel que la décomposition, le diviser-pour-régner, la dynamisation et le balayage du plan. On observe également que certains objets admettent des solutions plus efficaces que d'autres; par exemple, un polygone monotone, en comparaison à un polygone simple arbitraire, lorsqu'il s'agit de calculer le plus court chemin entre deux points.

Dans cette thèse, nous employons la technique de décomposition et une méthode hiérarchique de classement des composantes de la décomposition pour calculer le polygone de faible visibilité d'une arête, le plus court trajet entre deux points à l'intérieur d'un polygone simple et les réseaux d'acheminement à l'intérieur d'un polygone rectilinéaire. En utilisant la même méthode, nous présentons un algorithme polynomial pour vérifier la faisabilité de l'enchâssement d'un graphe dans le plan de manière à ce qu'il représente le graphe de visibilité d'un polygone simple, pour une classe spécifique de polygones simples.

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Prof. David Avis, for his support and advice during the preparation of this thesis.

I would like to thank Prof. Godfried Toussaint for his suggestions and for many useful discussions. I also appreciate the healthy environment provided by my collegues during my study at McGill Univesity.

TO THE LOVE OF MY LIFE

Table of Contents

Chapter 1
Introduction 1

Chapter 2
Hierarchical Description of Simple Polygons 4

2.1. Definitions and Notation. 4
2.2. A Linear Time Algorithm for Computing the Hierarchy. 4

Chapter 3
Computing the Weak Visibility Polygon from an Edge in a Simple Polygon 11

3.1. Introduction. 11
3.2. Additional Terminology. 12
3.3. Visibility Preliminaries. 13
3.4. Two Algorithms for Computing the Weak Visibility Polygon from an Edge in Monotone Polygons. 15
3.5. An Efficient Algorithm for Computing the Weak Visibility Polygon from an Edge in a Simple Polygon. 23
3.6. Conclusion and Applications. 26

Chapter 4
Recognizing Visibility Graphs of Simple Polygons 52

4.1. Introduction. 52
4.2. Embedding Maximal Outerplanar Graphs as Visibility Graphs. 52
4.3. Recognizing the Visibility Graph of a Convex Fan. 57
4.4. Discussion of the General Problem. 63

Chapter 5
Routing inside Simple Polygons 83

5.1. Introduction. 83
5.2. Routing in a General Rectilinear Channel. 84

Chapter 6
Shortest Route inside Simple Polygons 87

6.1. Introduction. 87
6.2. An Algorithm for Computing the Shortest Route inside a Monotone Polygon. 88
6.3. An Algorithm for Computing the Shortest Route inside a Simple Polygon. 91
6.4. Applications. 94

Bibliography 104

1. Introduction

Geometric objects such as points, line segments, rectangles and polygons are used to model objects in a variety of applications areas. In *pattern recognition*, points in multi-dimensional space are used to represent multi-attribute descriptions of objects [Tou]. In *VLSI design systems*, rectangles are used to model components of a circuit to simplify the hierarchical building of complicated designs [M&C]. In *concurrent transaction systems*, hyperrectangles are used to describe the progress made toward the completion of each transaction [Y&Pap&K,L&Pap]. In *image processing* and *computer graphics*, polygons are used to describe curves by an appropriate selection of points on or near the curve [Pav,N&S]. These geometric abstractions provide clearer understanding of the corresponding problems and of the most suitable computer representation for the data, and thus lead to the design of efficient algorithmic solutions.

Some geometric objects admit more efficient solutions than others. Examples are: $\Omega(n\log n)$ time is required to solve the hidden-line problem for a set of n line segments [Asa] while a linear running time algorithm has been presented in [E&A] for solving the hidden-line problem inside a simple polygon, and $\Omega(n\log n)$ time is required to compute the convex hull of a set of n points [Sha] while a linear running time algorithm has been presented in [M&A] for computing the convex hull of a simple polygon.

Within the class of polygons, when the problem involves "neat" shapes, one can often use their properties to design optimal solutions. Examples are checking query point inclusion in star-shaped [Sha] or monotone [L&P1] polygons, triangulating L_2-convex [E&A&T] or monotone [G&J&P&T] polygons, and computing the intersection of convex polygons [Sha, O&C&O&N]. Optimality of algorithms for the above problems suggests the decomposition of a given shape into smaller and more simple components, and then solve the problem for each component separately, e.g., triangulating the interior of an arbitrary polygon [G&J&P&T]. The strategy of decomposing a given shape into simpler components is also used in shape matching and

recognition. Examples of shape recognition methods based on decompositions are [Fu, F&P, Mar, Shap&H], which decompose a given shape into *spiral*, *star-shaped* or *convex* components and then recognize each component separately A survey of algorithms for computing such decompositions can be found in [Tou, Cha, Kei]. This dissertation deals with the use of the *decomposition strategy* in developing efficient solutions for graph embedding, visibility, network routing and shortest route problems. However, good solutions for these problems require the use of components' adjacency relation together with efficient methods for processing each component. In chapter 2 we describe a hierarchical method for describing simple polygons based on a selected decomposition which allows an efficient processing of a given simple polygon and present a linear running time algorithm for building such a description Using such description, processing the vertices of a component may be completely avoided if they do not contribute to the final result. Each chapter from three to six is devoted to describing the use of this hierarchical description in solving a different problem.

Avis and Toussaint [A&T] introduced the notion of weak visibility and presented a linear algorithm for checking the weak visibility of a simple polygon from an edge. In chapter 3 we present an $O(n \log n)$ algorithm for computing the subset of a n-sided simple polygon that is weakly visible from an edge. The algorithm uses a hierarchical description of the polygon based on a decomposition into simpler components, *monotone* polygons, in which the weakly visible vertices can be reported efficiently. When the given polygon belongs to the specialized class of *monotone* polygons, a linear algorithm is presented. Application of the algorithm to a problem in *motion planning* is described in the concluding section of chapter 3

Recently, *combinatorial* descriptions of geometrical objects have been used in recognizing shapes Such a system computes a combinatorial description of a given shape, a process which does not usually require extensive computing, and then attempts to match it with a stored model. Examples of such descriptions are: the *signature* of a curve [ORou] and the *visibility graph* of a simple polygon [A&E and Shap&H] which have been used to check the similarity of the corresponding objects Chapter 4 is concerned with the reverse operation; i.e., constructing

a geometrical object from its combinatorial description. We characterize the *minimal* visibility graph on a set of vertices, and give an algorithm for embedding a graph with a given hamiltonian circuit in the plane such that it represents the visibility graph of a simple polygon for a specialized class of simple polygons, the class of *convex fans*. The algorithm makes use of a hierarchical description of convex fans based on decomposition by special non-intersecting subset of its diagonals, the *maximal diagonals*. The problem of recognizing visibility graphs corresponding to arbitrary simple polygons is also discussed.

Chapter 5 addresses the problem of routing VLSI circuits; i.e., specifying the paths of wires on the chip. Due to the intractability of the routing problem, shown in [Szy and S&Y], many researchers directed their attention towards the design of good approximation algorithms. Examples of good heuristics for routing networks when their terminals lie on the boundary of a rectangle are given in [B&B&L, R&B&M, R&F]. In this chapter we describe the use of a hierarchical description of a simple polygon, based on a decomposition into rectangles, in enhancing the performance of algorithms; e.g. in PI system [Riv], for routing networks inside a rectilinear polygon, where terminals lie on the boundary of the polygon. We use this description to provide the routing order for the rectangles and to efficiently report the networks to be routed in each rectangle. Routing of networks in each rectangle is performed using any of the existing heuristics. The objective of the algorithm is to provide an interactive tool, for a human designer, which is capable of efficiently performing the tedious and time consuming task of detailed routing.

Algorithms for reporting the shortest route between two points inside a simple polygon are presented in chapter 6. Similar to the algorithm for computing the weak visibility region from an edge, the shortest route algorithm uses a hierarchical description of the polygon based on a decomposition into monotone polygons, in which the shortest route can be computed in linear time. In addition to the known applications of such algorithms; e.g. computing the trajectory of a racing car [Cha2] and wire routing in integrated boards [High], we describe its use in the design of efficient solutions of some optimization and separability algorithms.

2. A Hierarchical Description of Simple Polygons

2.1. Definitions and Notation

A *polygon* P with n vertices is a closed path $p_1, p_2, \ldots, p_n, p_{n+1} (p_{n+1} = p_1)$ where p_i has (x_i, y_i) as its x- and y-coordinates. The i*th* edge (or side) of P, denoted by e_i, is the closed line segment joining the pair of vertices p_i and p_{i+1}. The *boundary* of P, denoted by BD(P), is the sequence of edges $e_1, e_2, \ldots, e_n$. A polygon is said to be *simple* when no two nonconsecutive edges intersect. We assume that the vertices are given in clockwise order so that the interior of the simple polygon always lies to the right as its boundary is traversed. Between a pair of points s and t on the boundary of P, we define the chain CN(s,t) to be the subset of BD(P) so that the interior of P always lies to the right as CN(s,t) is traversed from s to t.

Let u and v be two points inside a simple polygon P. u is *visible* from v if the closed line segment joining them does not intersect the exterior of P. A closed line segment connecting two visible points in BD(P) is a *chord* of P. A chord connecting two vertices p_i and p_j, where $i < j$, will be denoted by $CHORD_{p_i p_j}$. An *arm* of the polygon P, denoted by $ARM_{p_i p_j}$, is the subset of the interior of P which is enclosed by the chain $CN(p_i, p_j)$ and the chord $CHORD_{p_i p_j}$. Note that the whole polygon can be viewed as an arm. A set of arms of P is said to be *nested* if, given any pair $ARM_{p_{i_1} p_{j_1}}$, $ARM_{p_{i_2} p_{j_2}}$ of the arms, their intersection is either $ARM_{p_{i_1} p_{j_1}}$, $ARM_{p_{i_2} p_{j_2}}$, or the empty set. It follows directly from the *Jordan curve theorem* that a set of arms forms a nested set if and only if the chords which generate the arms do not *properly* intersect.

2.2. A Linear Time Algorithm for Computing the Hierarchy

A set of chords of a simple polygon P, whose members do not *properly* intersect, decomposes the interior of P into a set of non-overlapping components. The algorithm by Lee and Preparata [L&P1] for decomposing a simple polygon into a set of monotone components is an example of a method for efficiently reporting such a set of chords and the corresponding set of non-overlapping components.

The hierarchical description of P, denoted by HIER-DESC(P), is based on a recursive partition of the polygon P into smaller arms, by removing the components of the decomposition one at a time until each arm has been reduced to one of the components. The top of the hierarchy represents the whole polygon which can be viewed as an arm. Nodes in the next level represent non-overlapping arms of the polygon. The set difference between an arm represented by a node in the hierarchy and union of the arms stored as its sons in the hierarchy is a component of the selected decomposition of P. For the example shown in Figure 2.1, the component D_4 is the set difference between the arm $ARM_{p_3 p_{16}}$ and the union of the two arms stored at the next level $ARM_{p_3 p_7} \cup ARM_{p_{11} p_{14}}$. Since each arm of the polygon uniquely corresponds to a chord, it suffices to label each node with the chord corresponding to that arm, as shown in Figure 2.2. Note that we can also use this representation as an adjacency tree of the components of P, where each node represents a component and two nodes are connected if the corresponding components share an edge.

How to build such a hierarchy? We start by storing the whole polygon in the top node and then remove the component of P which contains the edge e_n in its boundary. The removal of this component partitions the remainder of the polygon into a set of non-overlapping arms. Each arm, represented by the corresponding chord, is stored as a son of the top node. We then process (or expand) each new node, in a recursive fashion, until all the leaf nodes represent components of P.

We now give a complete description of the algorithm for building HIER-DESC(P). For convenience, we assume that the decomposition contains more than one component. This case can be handled by a straightforward sequence of steps.

procedure BUILD-HIER-DESC (P)

input:

1. A list of the components of the simple polygon P, where each component is represented by a sequence of chords in its boundary starting from the chord with the smallest first vertex. Components are initially unmarked.
2. A list of non-intersecting chords, where each element in the list points to the two components which contain that chord in their boundary. Chords are initially are unmarked.
3. A pointer, STRT, to the component which contains the edge e_n in its boundary.

output:

A pointer, HIER-DESC(P), to the top of the hierarchical description of the simple polygon P.

Initialization Steps

store the edge e_n in a node pointed to by HIER-DESC(P)
cur-node ← HIER-DESC(P);
cur-component ← STRT;

General Step

while cur-node does not contain e_n or
cur-component has an unmarked chord ***do***

while cur-component has an unmarked chord ***do***

if cur-component is unmarked ***then***

mark cur-component
for each unmarked chord of cur-component ***do***
insert a node containing this chord as a son of cur-node;

end if

cur-node ← son of cur-node containing an unmarked chord of cur-component;
mark the chord of the cur-component which is represented by cur-node;
cur-component ← the other component having the chord represented by cur-node in its boundary;
mark the chord of the cur-component which is represented by cur-node;

end while

cur-component ← the other component having the chord represented by cur-node in its boundary;
cur-node ← parent-cur-node;

end while

end BUILD-HIER-DESC

Analysis of the procedure BUILD-HIER-DESC

First we prove the correctness of the procedure in the following two lemmas; then the linearity is proved in Theorem 2.3.

Lemma 2.1 At any time during the execution of the procedure BUILD-HIER-DESC, each leaf node represents an arm which satisfies the following conditions:

a) is a subset of the arms represented by all its ancestors

b) intersections of its interior with interiors of the arms represented by the rest of the nodes are empty.

Proof Initially the hierarchy contains a single node, which corresponds to the whole polygon, and the conditions of the lemma are satisfied.

Assume that conditions a) and b) are satisfied prior to processing (conceptually, deleting) a new component of P. The boundary of the new component contains the chord which is stored in the current node. Therefore, deletion of the new component results in a set of non-overlapping arms, which are then stored as sons of the current node. Each new arm is a subset of the arm represented by the current node. Since set inclusion is a transitive relation, it follows directly that the arm represented by each new node is also a subset of the arms represented by all its ancestors. It is easy to see that condition b) is satisfied for the new nodes. Thus the lemma follows.

Q.E.D.

Lemma 2.2 Upon termination of the procedure BUILD-HIER-DESC, the leaf nodes represent the non-overlapping components of the simple polygon P.

Proof Initially all the chords are unmarked. When the arm corresponding to a chord is partitioned into smaller arms, the procedure marks that chord. Therefore, it is sufficient to show that upon termination of the procedure all the chords are marked. This follows from the

fact that the procedure climbs back out of a node only after all the chords of the corresponding component are marked. Thus the lemma follows.

Q.E.D.

Theorem 2.3 Procedure BUILD-HIER-DESC builds a hierarchical representation of a simple polygon in linear time.

Proof Performance of the procedure depends on the number of times each chord is processed. The procedure processes each chord three times only. In the first time, a node which represents the corresponding arm of the polygon is inserted. In the second time, prior to processing components of the corresponding arm, the chord is marked in the lists of the two components which contain it in their boundaries. After processing all the components of the corresponding arm and climbing back to a higher level, a chord is never reprocessed. Therefore, the procedure runs in $O(n)$ time as the number of chords is less than or equal to n-3.

Q.E.D.

Figure 2.1

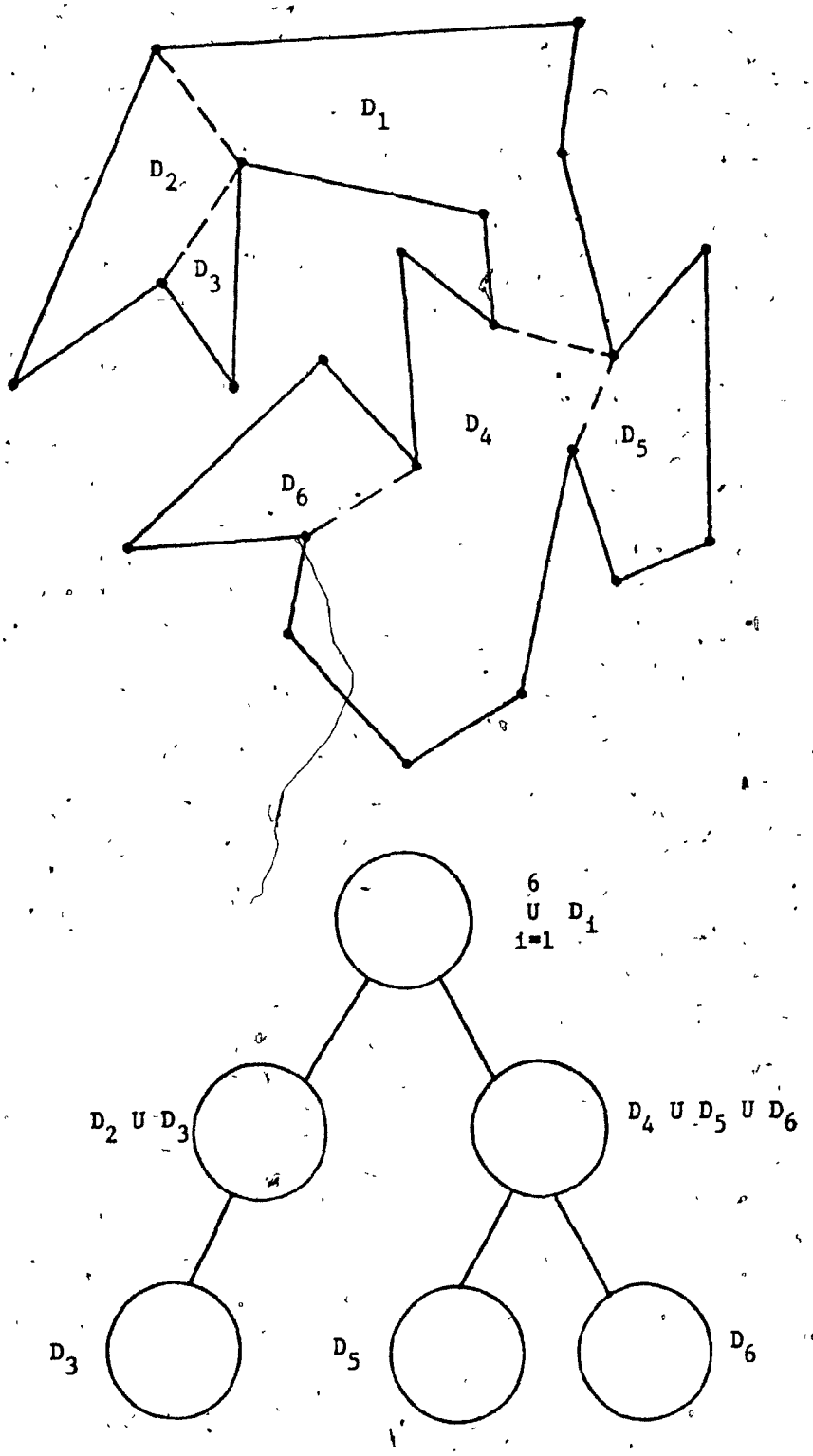
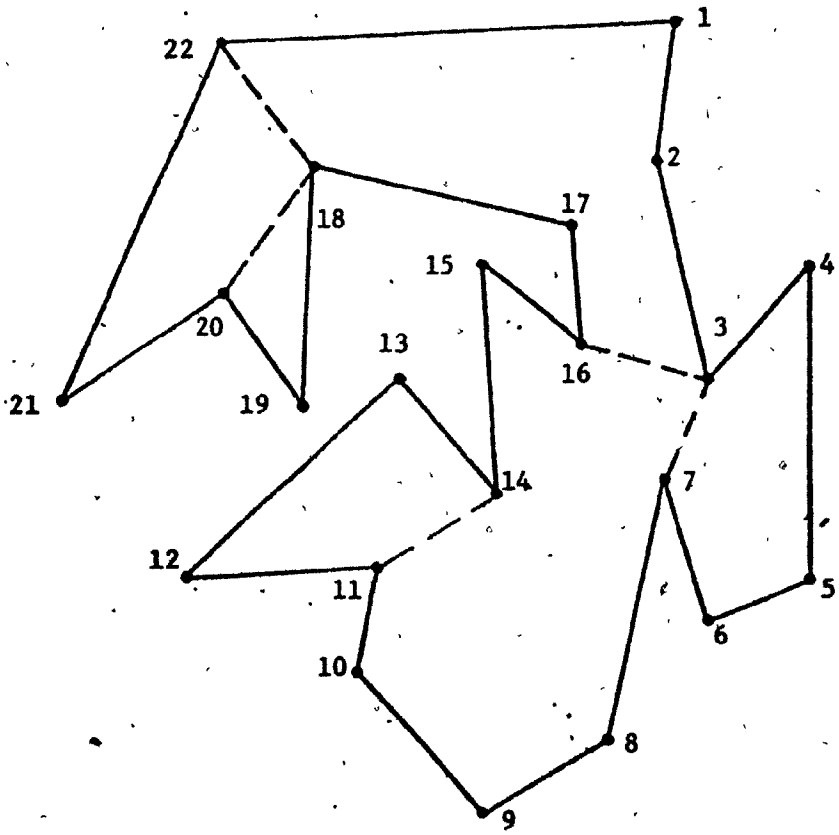
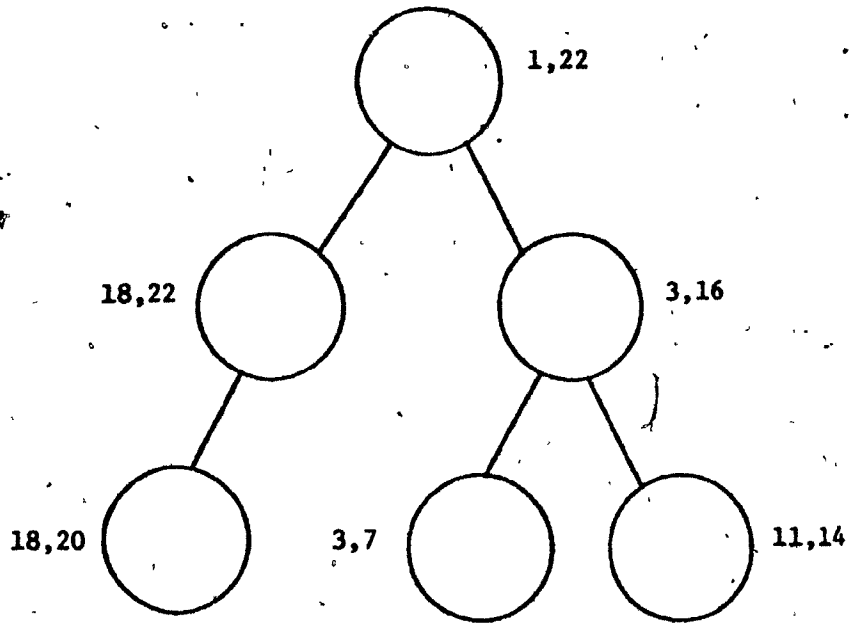


Figure 2.2


1
22
2
17
18
15
4
20
3
13
16
21
19
14
7
12
11
5
6
10
8
9


1,22
18,22
3,16
18,20
3,7
11,14

3. Computing the Weak Visibility Polygon from an Edge in a Simple Polygon

3.1. Introduction

Computing the visibility region from a single point or in a preferred direction is a recurring problem in path planning [Don,L&W] and separability [S&T,T&E,T&S] in robotics, in computer generated pictures of three dimensional objects [C&L,Rap,Yao], in pattern recognition [D&B], and in VLSI routing [Asa2] problems In addition, an algorithm for solving this problem can be used in solving other geometric problems (e g triangulating an L-convex polygon) efficiently [E&A&T] The two dimensional version of this problem can be described as follows

> Given a set of disjoint simple polygons $P=\{P_1, P_2, ..., P_k\}$, and a point X in the exterior of P. Report the visibility polygon of X (i.e., the subset of the plane that is visible from the point X)

With the increasing size of the problem instances to be processed, the need for efficient algorithms arises ElGindy and Avis [E&A] and Lee [Lee] presented linear running time algorithms for computing the visibility polygon when the set P contains one simple polygon Asano [Asa] proved an $\Omega(n + k \log k)$ lower bound for computing the visibility polygon from X when the polygons in the set P are convex, where n is the total number of vertices in P, and presented an algorithm which achieves that bound; In addition, Asano presented two algorithms for solving the general two dimensional hidden-line problem that run in $O(n + n \log k)$ time.

Recently, Avis and Toussaint [A&T] studied the problem of edge visibility in simple polygons. They introduced three notions of polygonal visibility from an edge complete, strong and weak visibility (refer to Figure 3 1 for illustration), and then presented a linear running time algorithm for checking visibility of a simple polygon from an edge under any of these notions Checking the weak visibility of a simple polygon from an edge is motivated in [A&T] by deciding if the interior of the polygon can be *watched* by a *mobile guard* patrolling an edge of the polygon In this chapter we present an algorithm for computing the weak visibility region from an edge, inside an n sided polygon, which has a complexity of $O(n \log n)$, and then describe

its use in the design of efficient solutions for planar *separability* and *reachability* problems. The method used depends on representing the polygon in a hierarchical form based on a decomposition into a set of simpler components, monotone polygons, in which the weak visibility polygon from an edge can be computed efficiently. We also present a linear running time algorithm for computing the weak visibility polygon from an edge in monotone polygons.

3.2. Additional Terminology

A simple polygon P is said to be in *standard form* with respect to its *ith* edge if all of its vertices lie on the same side of or on the line through the edge e_i. A chain is *monotone* with respect to a line LINE if orthogonal projections of the chain points on LINE are in the same order as the points on the chain. A simple polygon P is said to be monotone if there exist two vertices, p_i and p_j, and a line LINE such that the chains CN(p_i, p_j) and CN(p_j, p_i) are monotone with respect to the line LINE.

Let v and u be two points inside a simple polygon P. The *visibility polygon* of v is the subset of the interior and the boundary of P that is visible from the point v. Also u is *weakly visible* from the edge e_i if there exists a point w on the edge e_i such that u is visible from w. The *weak visibility polygon* from e_i is the subset of the interior and the boundary of P that is weakly visible from e_i. We define the *right intercept* of u on e_i, denoted by r(e_i, u), as the furthest clockwise point on e_i such that u and r(e_i, u) are visible. The vertex of P that lies on the line segment joining u and r(e_i, u) is called the *right anchor* of u with respect to the e_i, and denoted by ra(e_i, u). Similarly, the *left intercept* of u on e_i, denoted by l(e_i, u), is the furthest counterclockwise point on e_i such that u and l(e_i, u) are visible. The vertex of P that lies on the line segment joining u and l(e_i, u) is called the *left anchor* of u with respect to the edge e_i, and is denoted by la(e_i, u). A pair of vertices p_j and p_k, where p_j precedes p_k in a clockwise traversal of BD(P), is called a *gap* with respect to the edge e_i if they are weakly visible from e_i and the set of vertices $\{p_{j+1}, p_{j+2}, \ldots, p_{k-1}\}$ are not weakly visible from the edge e_i.

3.3. Visibility Preliminaries

Let $Q=\{q_1, q_2, \ldots, q_k\}$ be the set of intersection points of the straight line passing through the edge e_i and the edges of the polygon P sorted in order of their occurrence on the straight line. If the straight line is collinear with an edge, then their intersection is represented by the two endpoints of the edge. Let the points q_{j-1} and q_j represent intersection of the straight line with the edge e_i. Refer to Figure 3.2 for illustration.

If the pair of points q_j and q_{j+1} defines a chord of P (i.e. the line segment (q_j, q_{j+1}) lies inside P), then the subset of the corresponding arm, $ARM_{q_j, q_{j+1}}$, which is weakly visible from the edge e_i is only visible from the point p_{i+1} (equivalently, q_j). Using the algorithm for computing the visibility polygon from a point, presented in [E&A], this subset can be computed in linear running time. The same applies to the pair of intersection points q_{j-2} and q_{j-1}.

Using the following result,

> **Lemma 3.1** [E&A] (See Figure 3.3) Let POL be a simple polygon and e_i be an edge in its boundary such that the vertices pol_{i-1} and pol_{i+2} lie on the same side of the line passing through e_i. If y is a point in the exterior of POL which is collinear with the the edge e_i, then the line segment joining y to a point in the interior of POL intersects the chain $CN(pol_{i+1}, pol_i)$.

we will show that the remainder of the arms corresponding to pairs of points in Q which define chords of P are not weakly visible from the edge e_i.

Pairs of non-consecutive points in Q do not form chords of the polygon. Each arm of P corresponding to two consecutive points in Q, which define a chord of P and are neither q_j nor q_{j-1}, together with each point of the edge e_i satisfy the conditions of lemma 3.1. Therefore, these arms are not weakly visible from the edge e_i and their deletion does not affect the weak visibility polygon from the edge e_i. The deletion of these arms results in a smaller simple polygon P*, shown hashed in Figure 3.2, which is in standard form with respect to the edge e_i.

Since arms corresponding to pairs of consecutive points in Q form a nested set, we only have to delete those arms that are maximal. Simple modifications of the algorithm for computing the visibility polygon from a point [E&A] lead to a linear running time method for reporting and deleting the necessary arms (i.e , computing the polygon P^*).

Let the pair p_j and p_k of vertices, be a gap with respect to the edge e_i. Extend the line segment joining $la(e_i, p_j)$ to p_j until it intersects $BD(P^*)$ in g_j and the line segment joining $ra(e_i, p_k)$ to p_k until it intersects $BD(P^*)$ in g_k Refer to Figure 3.4 for illustration. By definition, the chain $CN(g_j, g_k)$ does not contain any vertex of the polygon P and thus is a subset of an edge of P The line segment joining g_j and g_k is called a *gap filler*

Lemma 3.2 The line segment joining g_j and g_k is weakly visible from e_i.

Proof There are three cases as shown in Figure 3.5.

In case (a), BD(P) intersects the line segments connecting p_j to $l(e_i, p_j)$ and p_k to $r(e_i, p_k)$ only at $la(e_i, p_j)$ and p_j respectively. Therefore the two triangles $(p_j, l(e_i, p_j), r(e_i, p_k))$ and (p_j, g_j, g_k) lie in the polygon P. For every point Y in the line segment connecting g_j to g_k, we draw a line through p_j to a point Z in the edge e_i The line segment connecting Z to Y does not intersect the exterior of P, and thus Y is weakly visible from e_i. Case (b) is proved similarly

In case (c), the line segment joining p_j and $r(e_i, p_j)$ intersects the chain $CN(p_{i+1}, p_{j-1})$ only at $ra(e_i, p_j)$, and the line segment joining p_k and $r(e_i, p_k)$ intersects the chain $CN(p_{i+1}, p_{k-1})$ only at $ra(e_i, p_k)$. If we assume that $l(e_i, p_j)$ follows $r(e_i, p_k)$ on the edge e_i, then $ra(e_i, p_k)$ must be a vertex in the chain $CN(p_{j+1}, p_{k-1})$ which contradicts the fact that p_j and p_k form a gap with respect to the edge e_i Therefore $r(e_i, p_j)$ and $l(e_i, p_j)$ precede $r(e_i, p_k)$ on the edge e_i, and the line segments connecting p_j to $l(e_i, p_j)$ and p_k to $r(e_i, p_k)$ intersect at a point, say X. The two triangles $(X, l(e_i, p_j), r(e_i, p_k))$ and (X, g_j, g_k) lie in the polygon P.

Similar to the proof of case (a), we can show that every point in the line segment connecting g_j to g_k is weakly visible from e_i

Q.E.D.

It follows from the previous result that one can use the left and right intercepts of the weakly visible vertices to compute the gap fillers in a single scan of the boundary of P Due to their simplicity, details of this step will be omitted in the description of algorithms through the remainder of the chapter

3.4. Two Algorithms for Computing the Weak Visibility Polygon from an Edge in Monotone Polygons.

In this section we present two algorithms for computing the weak visibility region from an edge in monotone polygons The first algorithm WVE-MON-1 runs in linear time, and is thus optimal However, the method used leads to a quadratic running time algorithm when applied to an arbitrary simple polygon The second algorithm WVE-MON-2 is slower, runs in $O(n\log n)$ time, but the method used results in an algorithm with the same complexity when applied to an arbitrary simple polygon The two algorithms are designed to deal with simple polygons that are monotone with respect to a vertical line, and the weak visibility polygon is to be calculated from an edge in the chain joining the vertex with the minimum y coordinate to the vertex with the maximum y coordinate Using the algorithm for reporting a line with respect to which a simple polygon is monotone, presented in [P&S], together with simple transformations we can process an arbitrary monotone polygon to satisfy these conditions in linear running time For convenience, vertices of the monotone polygon are labeled such that weak visibility polygon is to be calculated from the edge e_n Refer to Figure 3 6 for illustration.

Algorithm WVE-MON-1

Given a simple monotone polygon $M = \{m_1, m_2, \ldots, m_n\}$, the algorithm WVE-MON-1 proceeds as follows:

Step 1 The algorithm computes the connected subset of M, denoted by M^* and drawn in solid lines in Figure 3.6, which is in standard form with respect to the edge e_n, and then partitions M^* into three components M_A, M_B, and M_C. Vertices of M_A lie above both m_1 and m_n, vertices of M_B lie below both m_1 and m_n, and vertices of M_C have their y coordinates between those of m_1 and m_n. It also computes the two extreme points $EXTREME_A$ and $EXTREME_B$. The algorithm processes the subpolygons $M_1 = M_A \cup M_C$ and $M_2 = M_B \cup M_C$ separately. Due to the monotonicity of M, vertices of the subpolygon M_C are weakly visible from the edge e_n. However, they are processed as part of both M_1 and M_2 since they may be *left* or *right anchor points* of the vertices of M_A or M_B respectively.

For the subpolygon M_1, the algorithm scans the list of vertices in increasing order of their y coordinates maintaining two *deque* data structures LD and RD. Each *deque* contains a convex chain that is also monotone with respect to a vertical line. The *deque* RD consists of vertices of CN(m_1, $EXTREME_A$) that are the prospective *right anchor points* of the vertices of M_1 to be processed, and the *deque* LD consists of vertices of CN($EXTREME_A$, m_n) that are the prospective *left anchor points* of the vertices of M_1 to be processed. The front element in RD is the last examined vertex of the chain CN(m_1, $EXTREME_A$) which is weakly visible from e_n, and the front element in LD is the last examined vertex of the chain CN($EXTREME_A$, m_n) which is weakly visible from e_n. The rear element in RD is the right anchor point of the front element in LD; and RL is defined as the directed line joining the front element in LD to its right anchor point. The rear element in LD is the left anchor point of the front element in RD, and LL is defined as the directed line joining the front element in RD to its left anchor point. (Refer to Figure 3.7 for illustration.)

When a new vertex in the chain CN($EXTREME_A$, m_n) is encountered, the algorithm checks its position relative to LL and RL. If it lies to the left of the directed line RL, then it is not weakly visible from e_n and is immediately rejected. If it lies to the right of the directed line LL, then the algorithm terminates the scanning of vertices since the new vertex and the remainder of the vertices are not weakly visible from the edge e_n. Otherwise, the algorithm removes vertices from the front of LD until the remaining chain together with the new vertex form a convex chain, and adds the new vertex to the front of LD. The algorithm then removes vertices from the rear of RD until the new vertex lies to the right of the directed line passing through the last two vertices in RD.

A vertex in the chain CN(m_1, $EXTREME_A$) is processed in a similar fashion with the roles of LD and RD, and LL and LR interchanged.

A complete description of this step is given in procedure MON-SCAN-1, shown in Figure 3.8. Processing the subpolygon M_2 is a mirror image of processing M_1, and its description is omitted.

Step 2 The algorithm performs a clockwise traversal of the boundary of M^* and computes gap fillers for each one of the encountered gaps.

Analysis of the algorithm WVE-MON-1

We prove correctness of the algorithm in the following lemmas, and then prove its linearity in theorem 3.8. In the following lemmas we will deal with the subpolygon M_1. The case of the subpolygon M_2 is similar.

Lemma 3.3 During the scanning of the monotone subpolygon M_1, if a vertex m_i which belongs to the chain CN($EXTREME_A$, m_n) lies to the left of the directed line RL, then it is not weakly visible from the edge e_n.

Proof (refer to Figure 3.9.) Let m_j be the front element in the *deque* LD, which is the last

examined vertex in the chain CN($EXTREME_A$, m_n) that is weakly visible from the edge e_n. Extend the line segment connecting m_i to m_j until it intersects the chain CN(ra(e_n, m_j), $EXTREME_A$) in a point, say m_j'. The arm $ARM_{m_j\ m_j'}$, which contains e_n in its boundary, together with the vertex m_i satisfy the conditions of lemma 3.1. Therefore, m_i is not weakly visible from the edge e_n.

Q.E.D.

Lemma 3.4 During the scanning of the monotone subpolygon M_1, if m_i is the first vertex in the chain CN($EXTREME_A$, m_n) which lies to the right of the directed line LL, then the vertex m_i and the remainder of the vertices of M_1 are not weakly visible from e_n.

Proof (refer to Figure 3-10.) Let m_j be the front vertex in the deque RD. Extend the line segment connecting m_i to m_j until it intersects the chain CN(m_i, m_n) in a point, say m_j'. The arm $ARM_{m_j'\ m_j}$, which contains e_n in its boundary, together with the vertex m_i satisfy the conditions of lemma 3.1. Therefore, m_i is not weakly visible from the edge e_n. It follows from lemma 3.3 that remainder of vertices of M_1 which lie to the right of the directed line joining m_j to the m_i are not weakly visible from e_n. Each remaining vertex of M_1 that lies to the left of the directed line joining m_j to the m_i also lies to the right of the directed line LL. Therefore, remainder of the vertices of M_1 are not weakly visible from e_n, and the lemma follows.

Q.E.D.

Lemma 3.5 During the scanning of the monotone subpolygon M_1, if a vertex m_i which belongs to the chain CN(m_1, $EXTREME_A$) lies to the right of the directed line LL, then it is not weakly visible from the edge e_n.

Proof Similar to lemma 3.3.

Lemma 3.6 During the scanning of the monotone subpolygon M_1, if m_i is the first vertex in

the chain CN(m_1, $EXTREME_A$) which lies to the left of the directed line RL, then the vertex m_i and the remainder of the vertices of M_1 are not weakly visible from e_n.

Proof Similar to lemma 3.4.

Lemma 3.7 During the scanning of the monotone subpolygon M_1, if a vertex m_i lies to the left of the directed line LL and to the right of the directed line RL, then it is weakly visible from the edge e_n.

Proof During the execution of the procedure MON-SCAN-1, vertices stored in the two *deques* RD and LD satisfy the following properties:

(a) vertices in RD and LD are weakly visible from the edge e_n.
(b) the front elements in RD and LD are the last examined vertices of CN(m_1, $EXTREME_A$) and CN($EXTREME_A$, m_n) that are found to be weakly visible from e_n.
(c) the rear elements in RD and LD are the right and left anchor points of the front elements in LD and RD, respectively.

First we concentrate on proving that the vertex m_i is weakly visible when the above properties are satisfied. Let u and w be the front elements of RD and LD, respectively. By construction, vertices of the chain CN(m_1, u) lie to the right of the directed line connecting u to its right intercept and to the right of the directed lie connecting w to its right intercept. If r(u, e_n) follows r(w, e_n) on the edge e_n, then the right anchor point ra(w, e_n) must be one of the processed vertices in the chain CN(u, w). This contradicts the fact that u and w are the *last visible* vertices on the chains CN(m_1, $EXTREME_A$) and CN($EXTREME_A$, m_n) prior to the processing of m_i, and the monotonicity of M_1. It follows from this argument and the fact that l(u, e_n) precedes r(u, e_n) on the edge e_n that the directed line connecting u to its left intercept intersects the directed lie connecting w to its right intercept at a point, say X, and that the two triangles (r(w,e_n),X,l(u,e_n)) and (u' ,w' ,X) lie completely inside the polygon M_1 (Refer to Figure 3.11 for illustration). From m_i draw a line through X to a point on the edge e_i, say Y. The line segment connecting m_i to Y lies inside M. Therefore m_i is weakly visible from the edge e_n.

Now it remains to show that the three properties are satisfied after the vertex m_i is processed. The proof is inductive. Initially the deque RD contains the single vertex m_1, the deque LD contains the single vertex m_n and properties (a)-(c) are satisfied. Assuming the lemma is satisfied prior to processing the vertex m_i, we will show that the algorithm processes the vertex m_i correctly and that properties (a)-(c) are satisfied again after processing the vertex m_i.

If m_i lies to the left of RL or to the right of the line LL, then it is not weakly visible from the edge e_n, as shown in lemmas 3.3 - 3.6. The deques are not updated and the three properties are satisfied.

If m_i is a vertex in the chain CN(m_1, $EXTREME_A$) and lies to the right of RL and to the left of the line LL, then the algorithm deletes vertices from the rear of LD until a vertex, say m_j, is found such that all the vertices in LD lie to the left of the directed line joining m_i to m_j. Extend the line segment joining m_i to m_j until it intersects the boundary of M_1 in a point, say m_j'' (Refer to Figure 3.12). In the first part of the proof we showed that m_i is weakly visible from e_n. Therefore, m_j'' is $I(m_i, e_n)$ and the rear element in LD is the left anchor point of m_i. The algorithm then adds m_i to the front of the deque RD which satisfies the properties (a)-(c). The case of a vertex in the chain CN($EXTREME_A$, m_n) is proved in a similar fashion with the roles of LD and RD, and LL and RL interchanged.

Q.E.D.

Theorem 3.8 The algorithm WVE-MON-1 computes the weak visibility polygon from an edge correctly in linear running time.

Proof Correctness of the output has been proved in lemmas 3.1 to 3.7. Performance of the algorithm depends on the number of times each vertex of the polygon is processed.

A vertex may be added to the front of a *deque* once. Therefore, at most n vertices may be inserted in each deque. When the algorithm checks the weak visibility of a vertex and computes its intercepts on the edge e_n, it may delete vertices from one end of each deque. Since deleted vertices are not considered again, checking a front or a rear element of a deque data structure, and deleting it can be performed in constant time. It follows that the total running time of the algorithm is linear in the number of the vertices of the polygon.

Q.E.D.

Algorithm WVE-MON-2

This algorithm proceeds in the same steps as the algorithm WVE-MON-1, but it differs in the data structure used to store the prospective anchor points during the second step. The algorithm WVE-MON-2 uses two *balanced search trees* LTree and RTree where each tree contains a convex chain that is also monotone with respect to a vertical line. The *tree* RTree consists of vertices of CN(m_1, $EXTREME_A$) that are the prospective *right anchor points* of the vertices of M_1 to be processed. The element with the largest y coordinate in RTree, denoted by RTreeMax, is the last examined vertex of the chain CN(m_1, $EXTREME_A$) which is weakly visible from e_n, and the element in RTree with the smallest y coordinate, denoted by RTreeMin, is the right anchor point of LTreeMax. RL is defined as the directed line joining LTreeMax to its right anchor point. The oriented half-lines passing through the edges of the convex chain in LTree together with the line RL induce a convex subdivision of the plane. Therefore, processing a new vertex reduces to a simple version of the point location problem (Refer to Figure 3.13). The *tree* LTree and the line LL are defined similarly.

For the subpolygon M_1, the algorithm scans the vertices in increasing order of their y coordinates. When a new vertex in the chain CN(m_1, $EXTREME_A$) is encountered, the algorithm checks its position relative to LL and RL. If it lies to the left of the directed line LL and to the right of the directed line RL, then the algorithm searches the planar subdivision generated by the directed half-lines passing through the edges of the convex chain stored in

RTree for the region containing the new vertex and computes the corresponding right anchor point. Also the planar subdivision generated by the directed half-lines passing through the edges of the convex chain stored in LTree is searched for the corresponding left anchor point. The algorithm discards vertices in the tree RTree above the right anchor point and the vertices in the tree LTree below the left anchor point, and then inserts the new vertex into RTree. If the vertex lies to the right of the directed line LL or to the left of the directed line RL, then its is processed as in MON-SCAN-1.

A vertex in the chain CN($EXTREME_A$, m_n) is processed in a similar fashion with the roles of LTree and RTree, and LL and RL interchanged.

A complete statement of the procedure MON-SCAN-2 is shown in Figure 3.14.

Algorithms WVE-MON-1 and WVE-MON-2 differ only in the data structure used to store the prospective anchor points. Therefore, correctness of the algorithm WVE-MON-2 follows from the lemmas 3.1-3.7. Performance analysis of WVE-MON-2 is given in the following theorem.

Theorem 3.9 The procedure WVE-MON-2 computes the weak visibility polygon from an edge in a monotone polygon correctly in O($n$log$n$) running time.

Proof A vertex may be added to the balanced tree once. Therefore, the tree can contain at most n vertices Since locating the region that contains a new vertex and updating the tree (i.e. discarding part of the balanced search trees RTree and LTree) can be performed in O(log$n$) time [AHU, p. 155], it follows that the total running time of the algorithm is O($n$log$n$).

Q.E.D.

3.5. An Efficient Algorithm for Computing the Weak Visibility Polygon from an Edge in Simple Polygons

We now describe the algorithm for computing the weak visibility from an edge. For convenience, we assume that vertices of the simple polygon P are labeled such that weak visibility polygon is to be calculated from the edge e_n.

The algorithm preprocesses the polygon P in three passes. First, it deletes arms of the polygon which cannot be weakly visible from e_n. This results in a simple polygon P^* that is in standard form with respect to the edge e_n. In the second pass, it decomposes the polygon P^* into a set of components that are monotone with respect to a vertical line, known as the regular decomposition, using the algorithm presented in [L&P1]. In the third pass, it uses the procedure BUILD-HIER-DESC to compute the hierarchical representation of P^* based on the regular decomposition.

Similar to the algorithms in the previous section, it then scans the vertices of P^* with the help of the HIER-DESC(P^*), to compute the right and left intercepts of the weakly visible vertices. Finally, it computes gap fillers. During the scanning step, the algorithm traverses the hierarchy HIER-DESC(P^*) and processes the vertices of the components encountered. The algorithm starts at the component which contains the edge e_n in its boundary and processes the monotone polygon as in WVE-MON-2. That is, computes the two subpolygons M_1 and M_2 and then process the sorted list of vertices for each subpolygon, separately.

In processing the subpolygon M_1, when a vertex in the chain CN(p_1, $EXTREME_A$) that is the upper end-point of a chord connecting two vertices non-consecutive on the boundary of P^* is encountered, the algorithm checks its position relative to the directed line LL. If it lies to the right of LL (as shown in Figure 3.15 a), then the arm corresponding to the chord found is not weakly visible from e_n, and the algorithm continues to process the remainder of the vertices of M_1 as in MON-SCAN-2. Otherwise, the algorithm computes the *two* extreme points of the chord that are weakly visible from e_n, Lower and Upper, as shown in Figure 3.15 b. It then

searches RTree and LTree for the anchor points of Upper and splits the chains stored in RTree and LTree such that:

$RTree_1$ ($RTree_2$) contains vertices of the chain below (above) and including the right anchor point found.

$LTree_1$ ($LTree_2$) contains vertices of the chain above (below) and including the right anchor point found.

Now the algorithm starts to process the new component which is the difference between the arm corresponding to the encountered chord and the union of the arms corresponding to its sons in the hierarchy. Vertices of the new component with y-coordinates less than that of Lower can not be weakly visible from e_n, and are not processed. The remaining vertices of the new component are processed as in MON-SCAN-2, in order of increasing y-coordinates, using the vertices of the trees $RTree_2$ and $LTree_2$ as the initial set of prospective anchor points. After the vertices of the arm corresponding to the encountered chord are processed, it continues to process the remainder of the vertices in M_1 using vertices of the trees $RTree_1$ and $LTree_1$ as the prospective anchor points.

The case of a chord connecting two non-consecutive vertices in the chain CN ($EXTREME_A$, p_n) is handled in a similar way, with the roles of RTree and LTree and LL and RL interchanged. A complete description of the method for processing M_1 is given in procedure SCAN-M1, shown in Figure 3.16. We now give the complete algorithm, procedure WVE.

procedure WVE

1. Compute the connected subset of the polygon, P*, which is in standard form with respect to the edge e_n.

2. Decompose the polygon P* into a set of monotone components.

3. BUILD-HIER-DESC (P*).

4. Fetch the monotone component in the top of the HIER-DESC(P*), M.

   Locate the two vertices with the maximum and minimum y-coordinates,

$EXTREME_A$ and $EXTREME_B$ respectively;

Decompose M into the two subpolygons
M_1 and M_2 as described in section 3.4.

RTree ← Λ; Insert p_1 into RTree;
LTree ← Λ; Insert p_n into LTree;
SCAN-M1 (M_1, LTree, RTree, $EXTREME_A$)

RTree ← Λ; Insert p_1 into RTree;
LTree ← Λ; Insert p_n into LTree;
SCAN-M2 (M_2, LTree, RTree, $EXTREME_B$)

5. Scan the weakly visible vertices in clockwise order and compute the gap fillers.

end WVE

We now prove the main result of this chapter.

Theorem 3.10 Procedure WVE computes the weak visibility polygon from an edge in an arbitrary simple polygon correctly in O($n$log$n$) running time.

Proof In lemmas 3.1-3.7 we proved that the algorithm processes a vertex in a monotone polygon correctly. In this theorem we only prove correctness of the transition step from one monotone component to another.

Let $CHORD_{p_i p_{i+1}}$ be a chord connecting two non-consecutive vertices of the polygon P^* in the chain CN(p_1, $EXTREME_A$), and let Lower and Upper be the two extreme points of the chord that are weakly visible from e_n. Let NEW-COMP be the monotone component of the arm $ARM_{p_i p_{i+1}}$ which contains the chord $CHORD_{p_i p_{i+1}}$. Vertices of NEW-COMP *above* the point Lower together with the vertices stored in the trees $LTree_2$ and $RTree_2$ form a monotone polygon. Therefore, correctness of the procedure SCAN-M1 follows from lemmas 3.1-3.7 as in the procedure MON-SCAN-2. Vertices in the tree $RTree_1$ can only be right anchor points of vertices of NEW-COMP which lie to the left of the directed line joining Upper to its right anchor point, which are shown not to be weakly visible from the edge e_n in lemma 3.3.

Therefore, *only* vertices of the chain stored in the tree $RTree_2$ can be prospective right anchor points of vertices in NEW-COMP. The fact that *only* vertices in the tree $LTree_2$ can be prospective left anchor points of vertices in NEW-COMP can be shown in a similar way.

The case of Lower and Upper being vertices in the chain CN($EXTREME_A$, p_n) is proved in a similar way, and is omitted.

Since the algorithm uses the same data structure as the algorithm WVE-MON-2, where the anchor points of a vertex can be computed in O(log$n$) time, it follows that the total running time is O($n$log$n$).

Q.E.D.

3.6. Conclusion and Applications

Recently, two different algorithms have been developed for computing the weak visibility polygon from an edge in simple polygons. Ghosh and Shyamasundar [G&S] claimed a linear running time algorithm. The main idea of the proposed algorithm is that the weak visibility polygon from an edge e_i in a simple polygon is the union of the visibility polygons from a *few* vertices; namely p_i, p_{i+1} and the vertices of the polygon that are visible from either p_i or p_{i+1}. Unfortunately their algorithm does not always work correctly. A counterexample is given in Figure 3.17, where a vertex that is weakly visible from the edge e_i is not visible from the selected set of vertices.

A different approach, which follows the steps of the algorithm for checking the weak visibility of a polygon from an edge in [A&T], is presented by Lee and Lin [L&L]. In this algorithm, vertices of the polygon are scanned twice to compute their left and right intercepts, whose relative positions on the edge are later used to determine the weakly visible region. During each scan, the algorithm achieves O($n$log$n$) running time in the worst case by maintaining the prospective anchor points in a *concatenable queue* and associating a *concatenable queue* with each scanned vertex. The algorithm described in the previous section

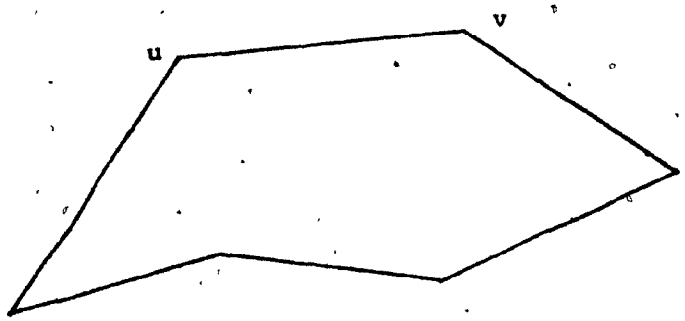
differs from this approach in that the weak visibility of a vertex is immediately determined when the vertex is encountered during the scanning step (step 4 of procedure WVE) and that the scanning process is terminated when it is clear that the remaining vertices are not weakly visible. An example of problems which require the use of these features in the design of efficient solutions is the *k-reachability* problem. This problem is a fundamental problem in printed circuit routing [High], and in computing the reachable workarea of a robot arm.

In a *k-reachability* problem we are given a simple polygon P, an interior point q and a constant k, it is required to compute the subset of P that is reachable from q by an interior path of at most k bends. The basic idea is to compute the k-reachable region one bend at a time. First we compute the 0-reachable region from q (i.e., the *visibility polygon* of q). We then compute the 0-reachable region from each edge of the visibility polygon of q which is not part of the boundary of P (i.e., the *weak visibility polygon* from that edge). The latter step is then repeated k-1 times to compute the k-reachable region of q (Refer to Figure 3.18 for an illustration of 0, and 1-reachable regions). An implementation of this method, which uses a weak visibility algorithm as a black box, leads to an $O(kn\log n)$ algorithm for computing the k-reachable region of a point inside a n sided polygon However modifying the procedure SCAN-M1 by adding recursive calls to itself, with the the weakly visible part of LL or RL as the edge from which the weak visibility to be computed, prior to each *terminate* statement leads to an $O(n\log n)$ algorithm for the k-reachability problem. A similar idea is used in computing the *shortest route* between two points inside a simple polygon and detailed description will be given in chapter 6.

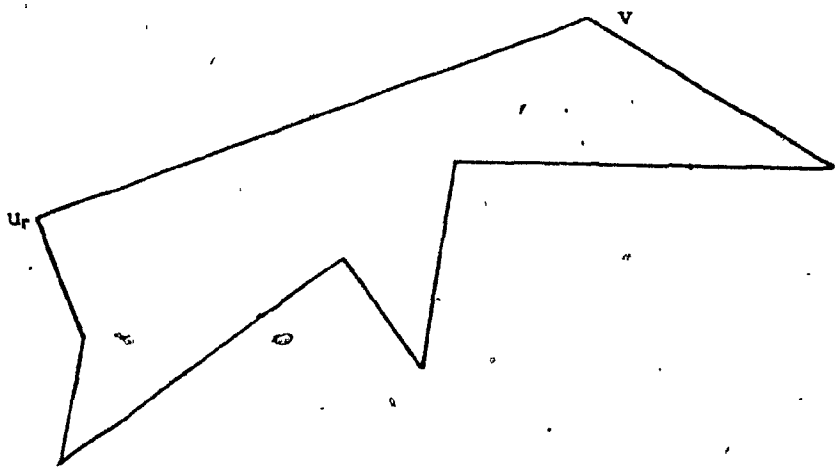
Another problem which uses the algorithm for computing the weak visibility polygon from an edge in the design of an efficient solution is that of checking the *separability* of polygons. Two simple polygons P and Q are said to be *movably separable* with a *single translation* if P can be moved an arbitrary distance away from Q by a single translation, without collision. In [T&S], Toussaint et al. presented a quadratic time algorithm for checking the separability of two polygons. The algorithm is based on computing the directions of movability for every

vertex of P with respect to the polygon Q, and then computing the intersection of these directions to obtain the directions of movability for the polygon P. The linear time algorithm for computing the visibility polygon from a point [E&A] was used to compute the directions of movability for each vertex of P, which led to an overall $O(|P|.|Q|)$ running time. In a later paper [S&T], they presented a more efficient algorithm for checking the separability which runs in $O((|P|+|Q|)\log|Q|)$ time. Let $q_i q_j$, $i < j$, be an edge of the convex hull of Q which is not an edge of the polygon. The chain $CN(q_i, q_j)$ defines a *convex deficiency* of Q with the *lid* $q_i q_j$. The algorithm is based on preprocessing deficiencies of Q by computing the *weak visibility polygon* in each deficiency from the corresponding lid, which can be performed in $O(|Q|\log|Q|)$ time. Using a suitable point location method [Kir or L&Y], the algorithm searches the weak visibility polygons and computes the directions of movability for the vertices of P with respect to Q in $O(|P|\log|Q|)$ time. It then proceeds to compute the directions of movability for the polygon P as in the previous algorithm.

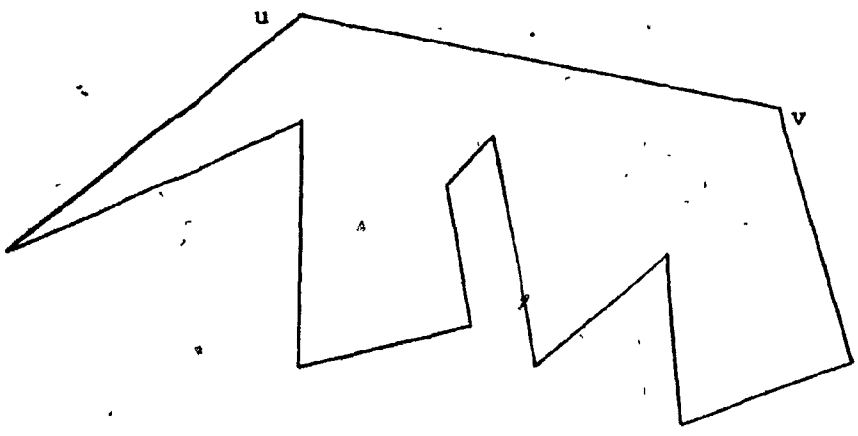
Figure 3.1



Completely Visible Polygon from the edge uv



Strongly Visible Polygon from the edge uv



Weakly Visible Polygon from the edge uv

Figure 3.2

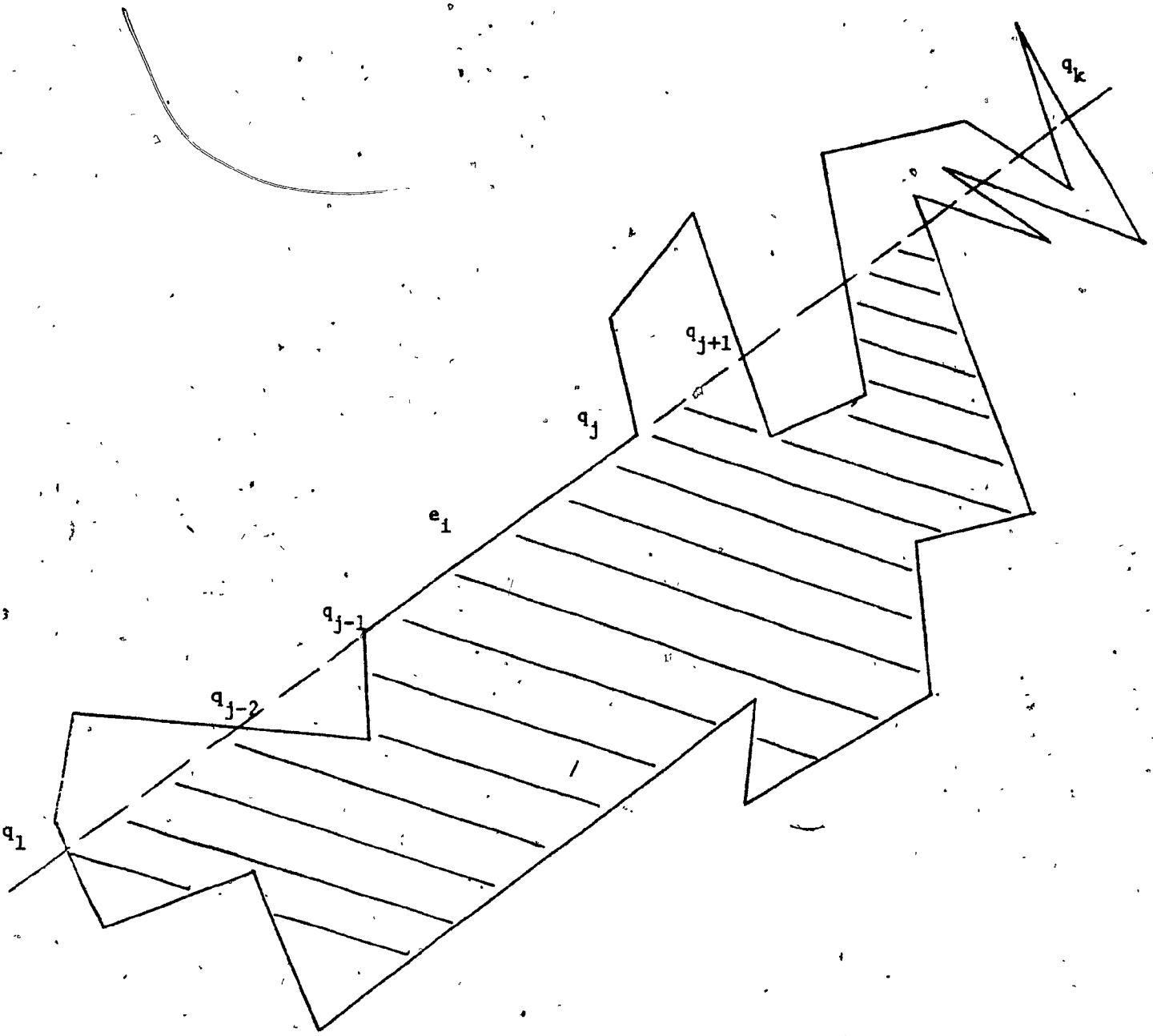


Figure 3.3

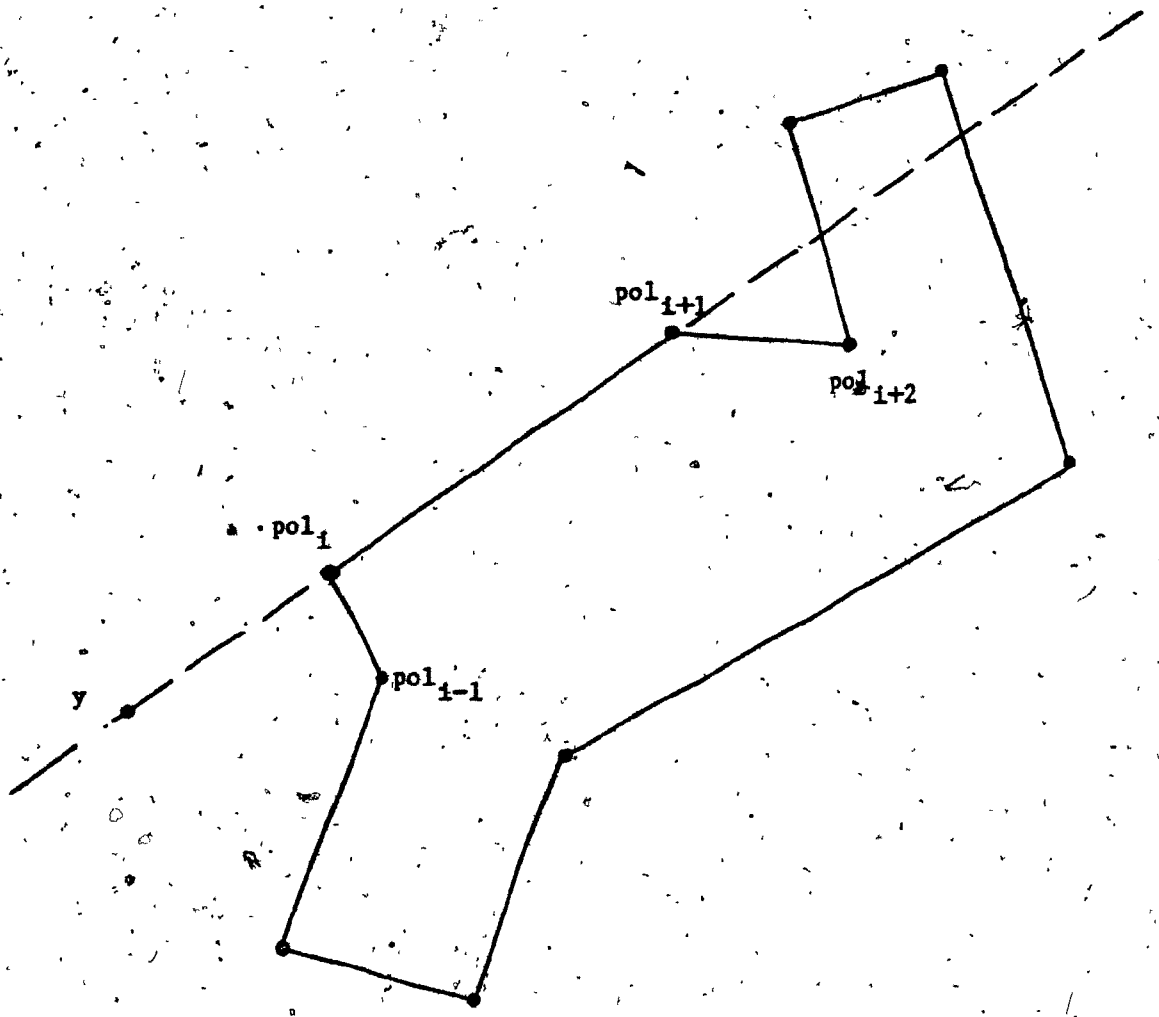


Figure 3.4

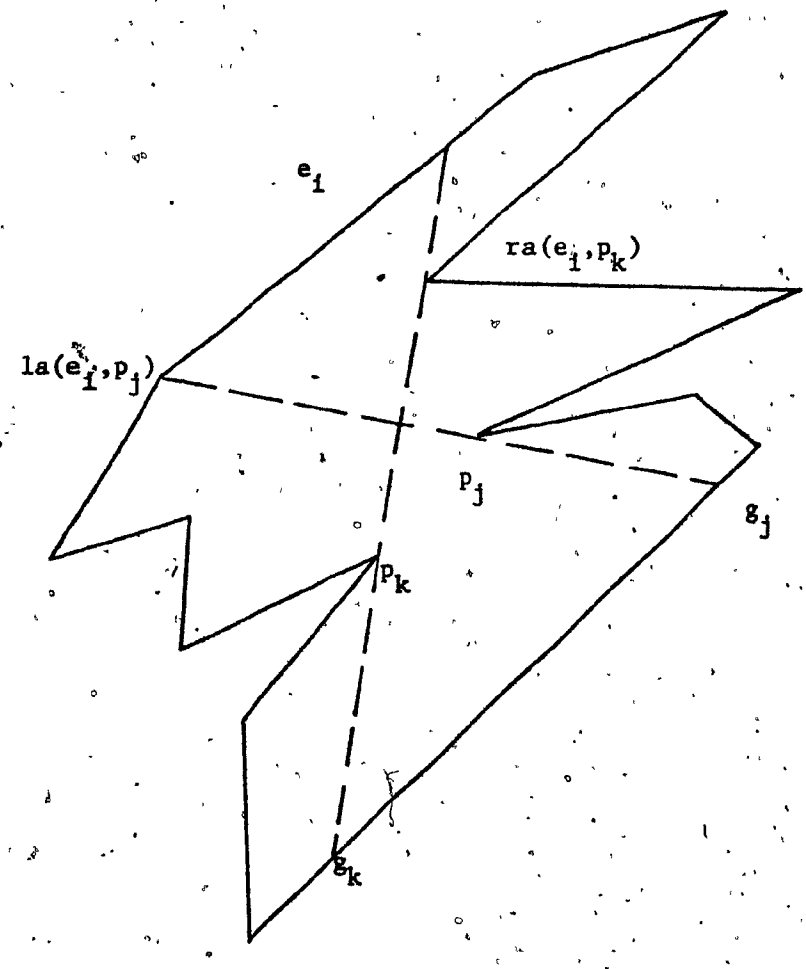


Figure 3.5

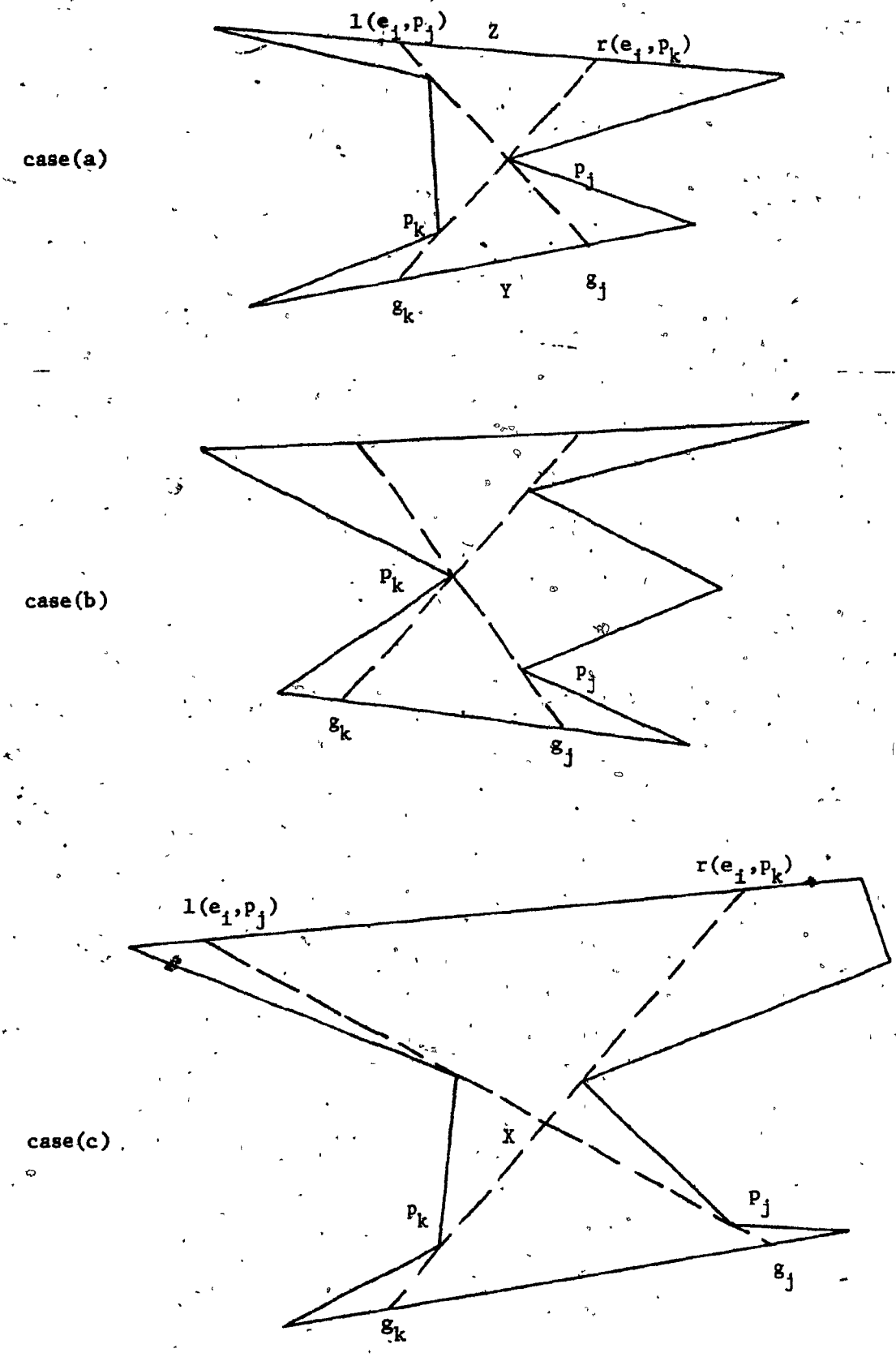

case(a)
$l(e_i,p_j)$
z
$r(e_i,p_k)$
p_j
p_k
g_k
y
g_j
case(b)
p_k
p_j
g_k
g_j
case(c)
$r(e_i,p_k)$
$l(e_i,p_j)$
x
p_k
p_j
g_j
g_k

Figure 3.6

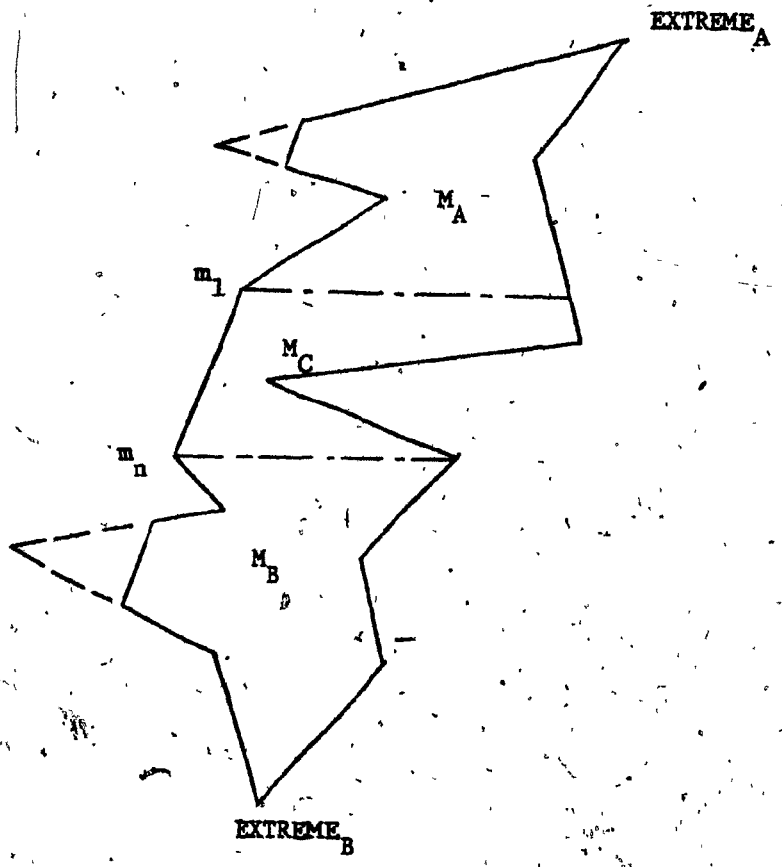
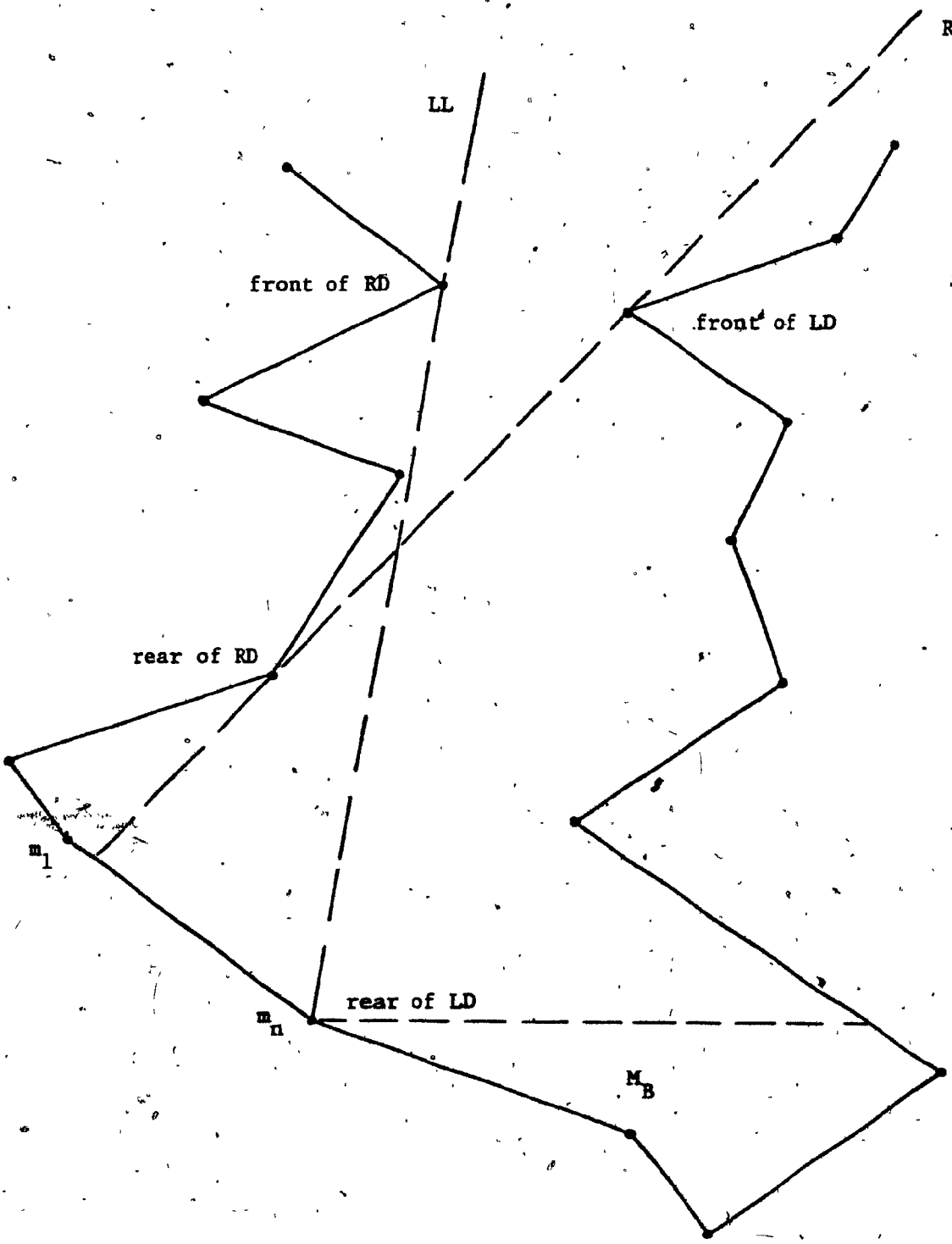


Figure 3.7



procedure MON-SCAN-1 (M)

input:
A simple polygon M that is monotone with respect to the y-axis.

output:
Right and left intercepts of the vertices of M that are weakly visible from e_n.

Initialization Steps

1. Compute the connected subset M^* whcih is in standard form with repsect to the edge e_n;

2. Locate the two vertices with the maximum and minimum y-coordinates, $EXTREME_A$ and $EXTREME_B$ respectively;

3. Decompose M^* into the two subpolygons:
   M_1 whose vertices have y-coordinates larger than that of m_n,
   and M_2 whose vertices have y-coordinates less than that of m_1;

Steps for processing the subpolygon M_1

1. Insert the vertex m_1 into the rear of the deque RD;
   Insert the vertex m_n into the front of the deque LD;

2. cur-vertex ← the vertex of M_1 with the smallest y-coordinate larger than that of m_n;

3. *Repeat*

   case 1: cur-vertex is a vertex of the chain CN(m_1, $EXTREME_A$)

      case 1a: cur-vertex lies to the left of the directed line RL
      terminate processing the subpolygon M_1;

      case 1b: cur-vertex lies to the right of the directed line RL
      and to the left of the directed line LL

         delete vertices from the front of the deque RD until the cur-vertex together with the vertices in the deque RD form a convex chain;

         delete vertices from the rear of the deque LD until the vertices of LD lie to the left of the diretced line joining cur-vertex to the rear of LD;

         store the front vertex of RD and the rear vertex of LD as the right and left anchor points of cur-vertex, respectively;

         insert cur-vertex into the front of the deque RD;

case 2: cur-vertex is a vertex of the chain CN($EXTREME_A$, m_n)

case 2a: cur-vertex is a vertex in M_C

delete vertices from the front of the deque LD until cur-vertex together with the vertices in LD form a convex chain;

store the front vertex of LD as the left anchor point of cur-vertex;

insert cur-vertex into the front of LD;

case 2b: cur-vertex is a vertex in M_A and lies to the right of the directed line LL
terminate processing the subpolygon M_1;

case 2c: cur-vertex is a vertex in M_A and lies to the left of the directed line LL
and to the righ: of the directed line RL

delete vertices from the front of the deque LD until the cur-vertex together with the vertices in the deque LD form a convex chain;

delete vertices from the rear of the deque RD until the vertices of RD lie to the right of the diretced line joining cur-vertex to the rear of RD;

store the front vertex of LD and the rear vertex of RD as the left and right anchor points of cur-vertex, respectively;

insert cur-vertex into the front of the deque LD;

cur-vertex ← vertex of M_1 with the smallest y-coordinate larger than that of cur-vertex;

Until cur-vertex = $EXTREME_A$;

Steps for processing the subpolygon M_2

Details of these steps are similar to those of the steps for processing M_1, and are thus omitted.

end MON-SCAN-1

Figure 3-8

Figure 3.9

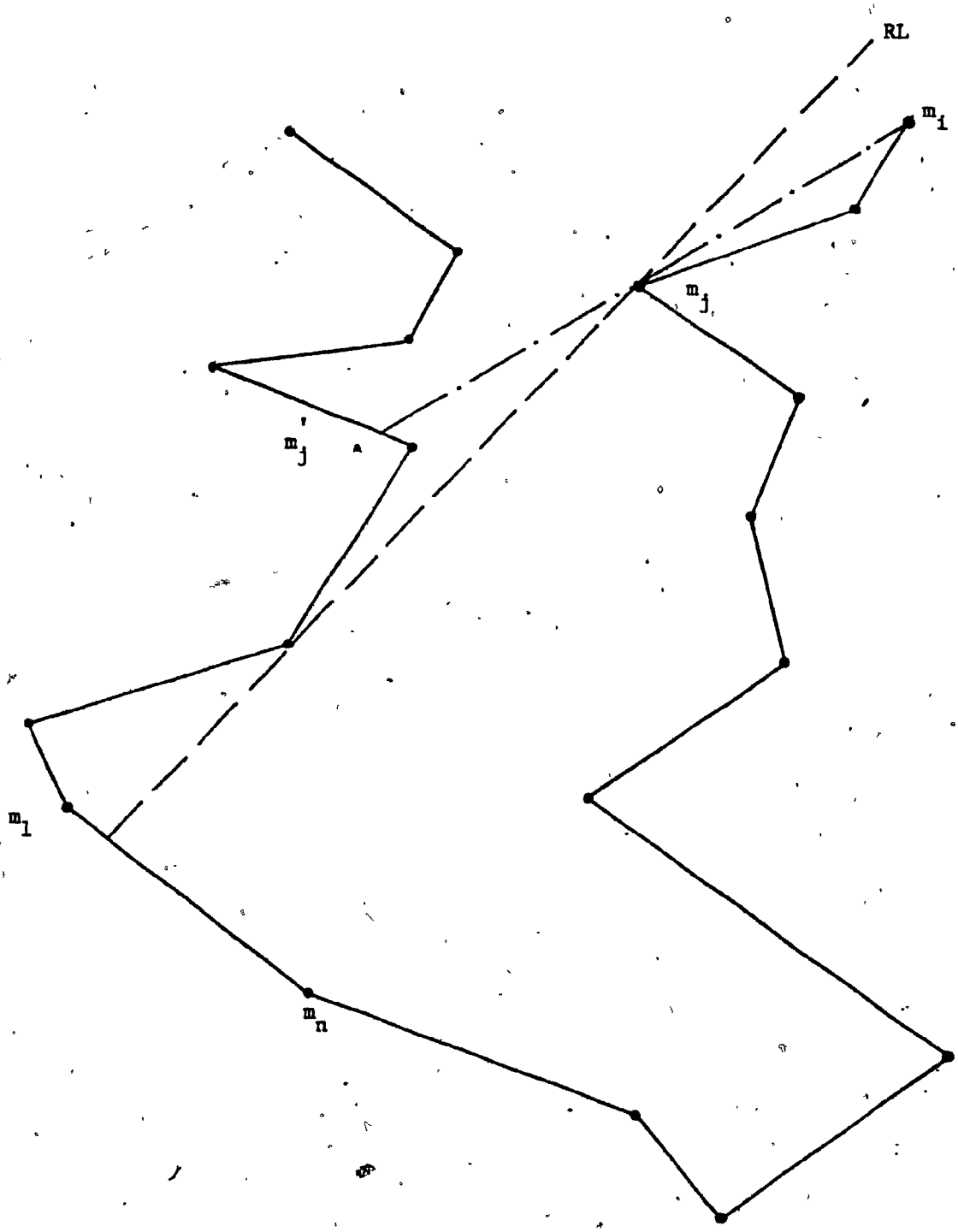


Figure 3.10

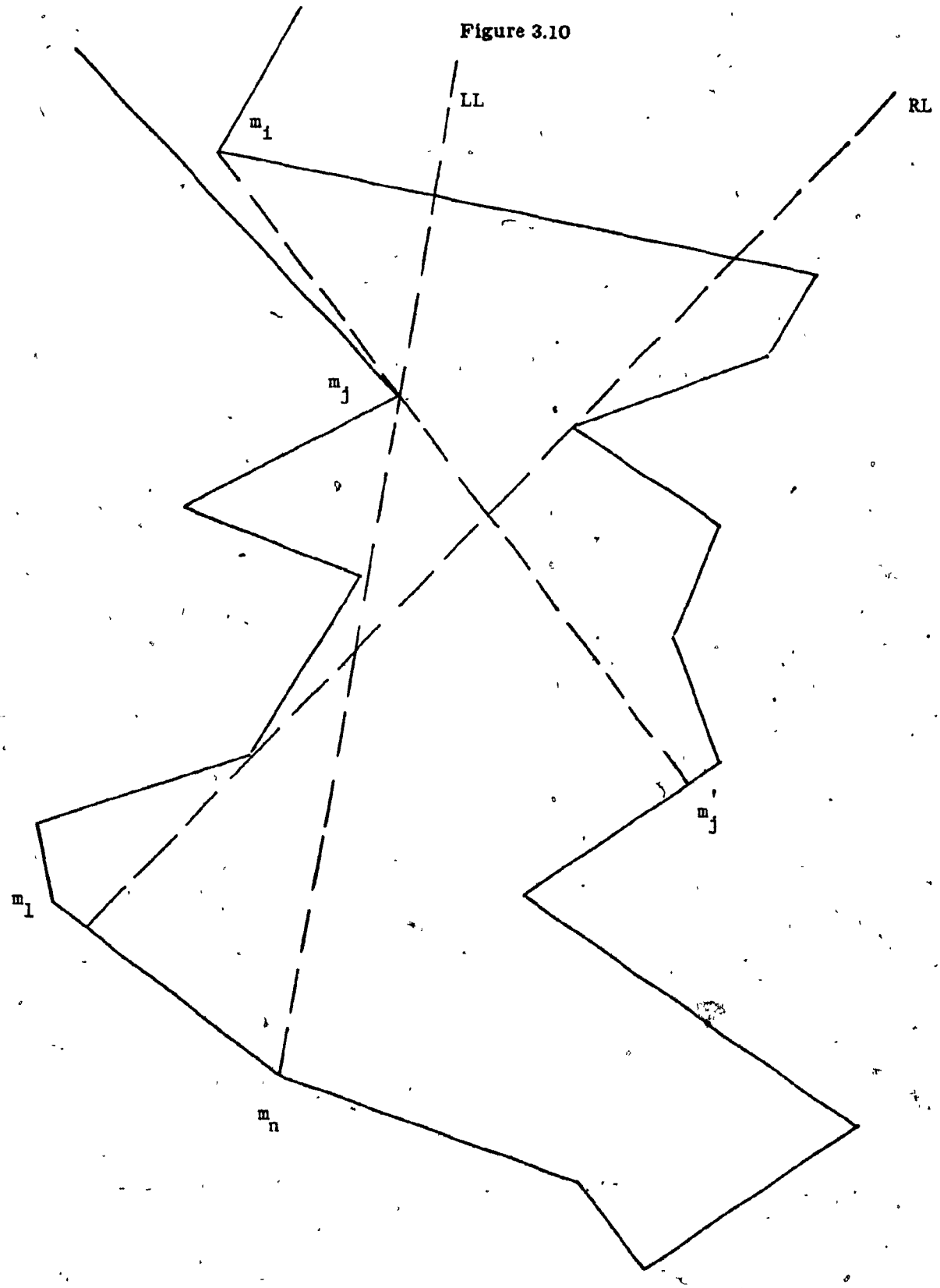


Figure 3.11

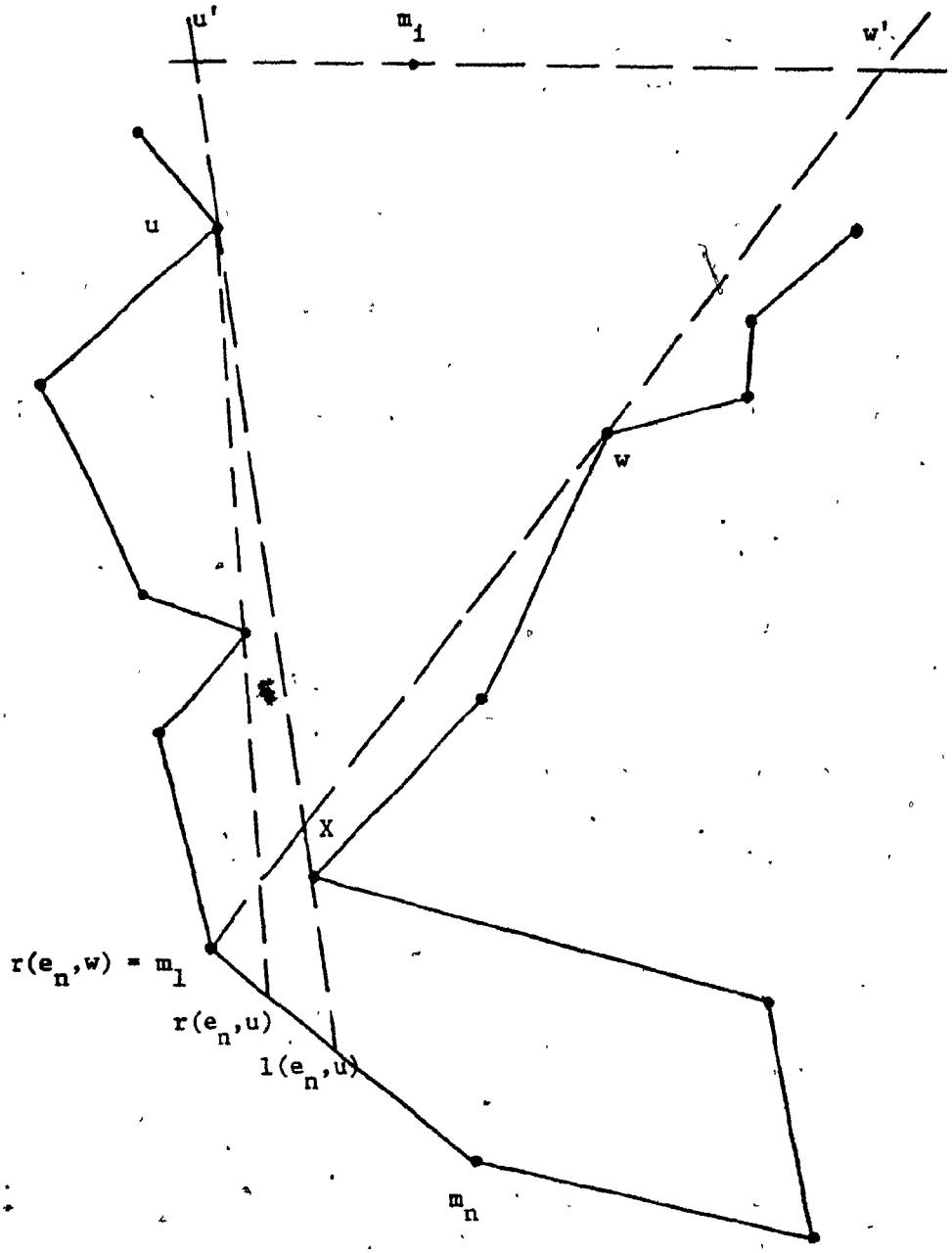


Figure 3.12

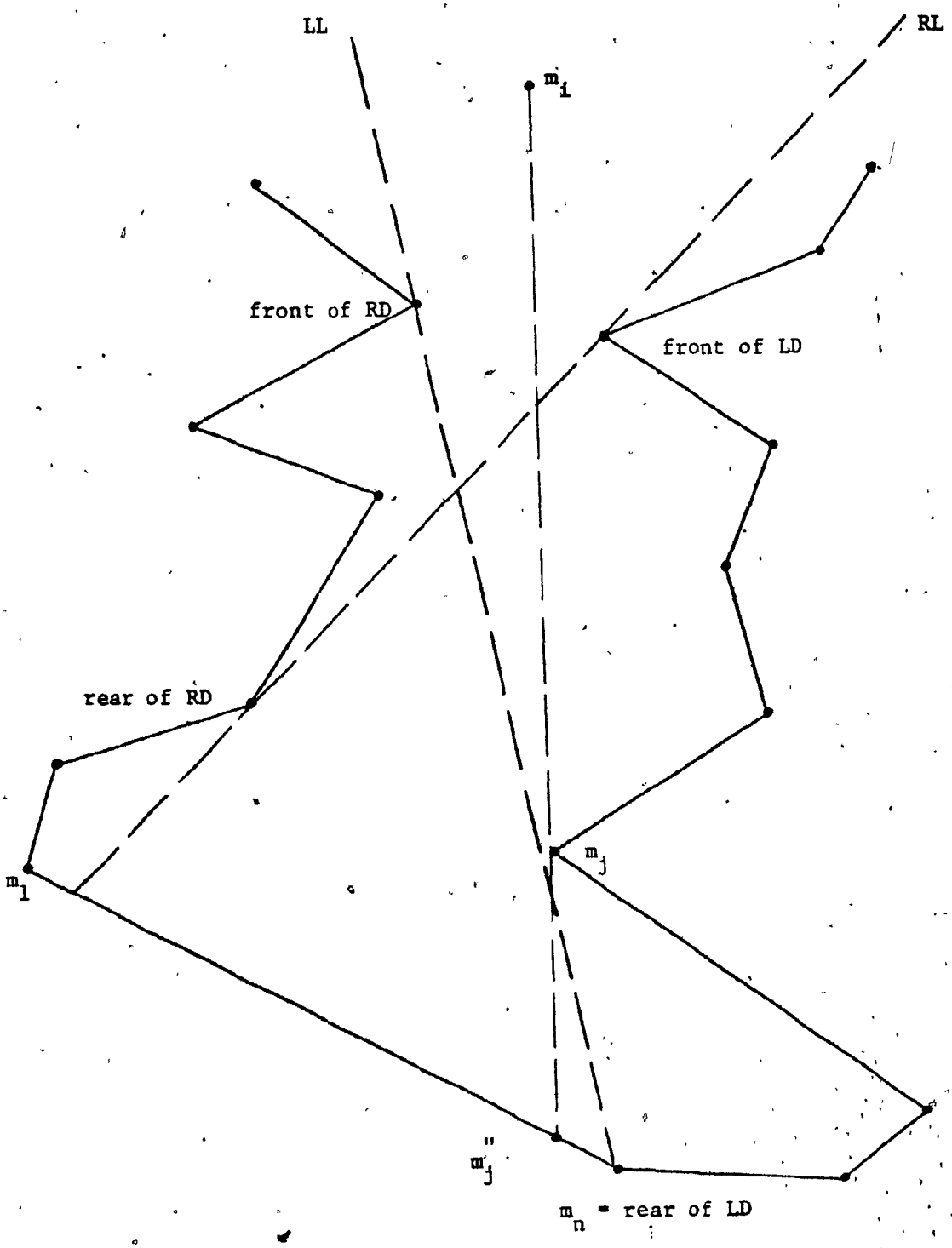
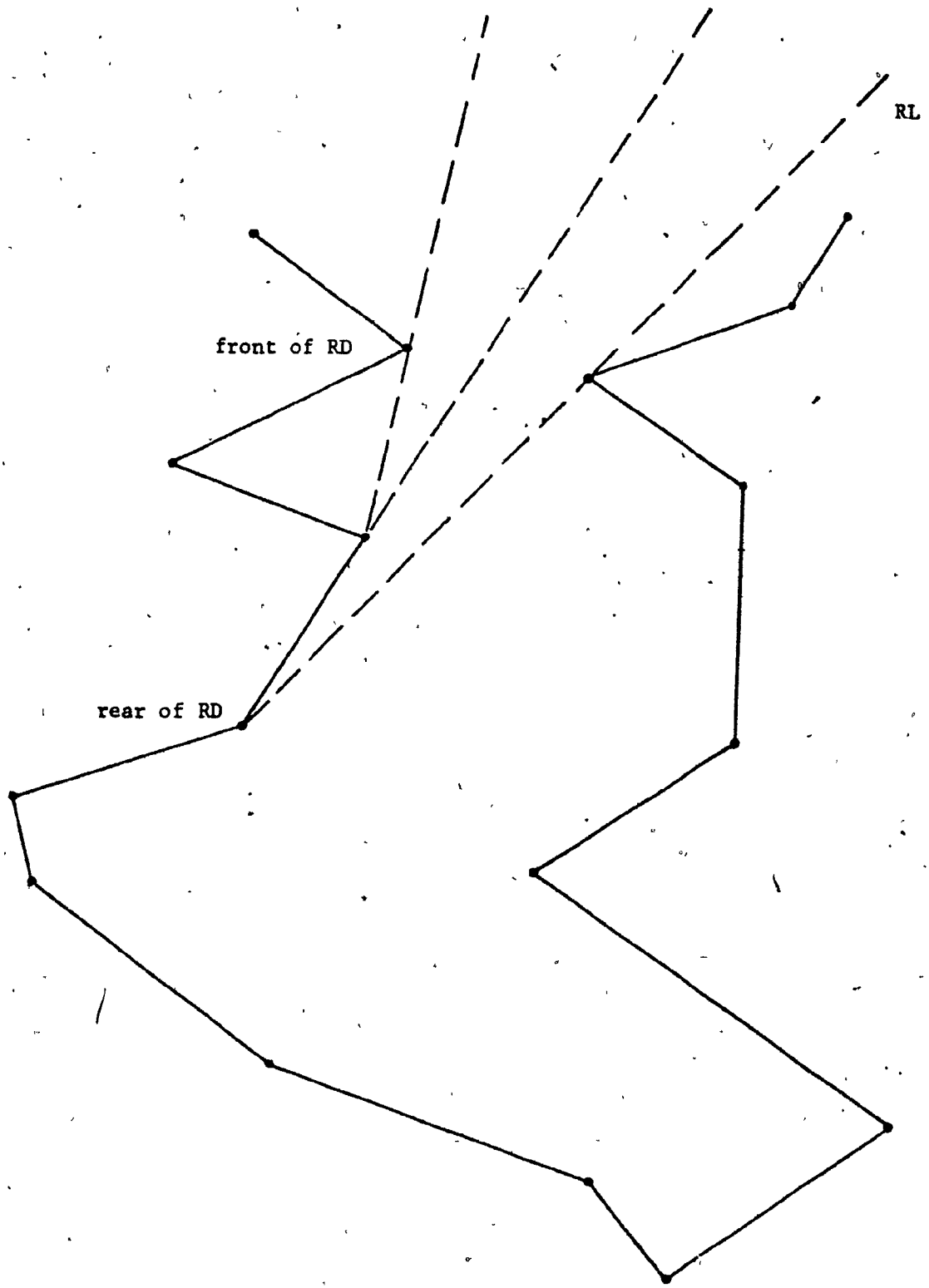


Figure 3.13



procedure MON-SCAN-2 (M)

input:
A simple polygon M that is monotone with respect to the y-axis.

output:
Right and left intercepts of the vertices of M that are weakly visible from e_n.

Initialization Steps

1. Compute the connected subset M^* whcih is in standard form with repsect to the edge e_n;

2. Locate the two vertices with the maximum and minimum y-coordinates, $EXTREME_A$ and $EXTREME_B$ respectively;

3. Decompose M^* into the two subpolygons:
   M_1 whose vertices have y-coordinates larger than that of m_n,
   and M_2 whose vertices have y-coordinates less than that of m_1;

Steps for processing the subpolygon M_1

1. Insert the vertex m_1 into the tree RTree;
   Insert the vertex m_n into the tree LTree;

2. cur-vertex ← the vertex of M_1 with the smallest y-coordinate larger than that of m_n;

3. *Repeat*

   case 1: cur-vertex is a vertex of the chain CN(m_1, $EXTREME_A$)

      case 1a: cur-vertex lies to the left of the directed line RL
      terminate processing the subpolygon M_1.

      case 1b: cur-vertex lies to the right of the directed line RL
      and to the left of the directed line LL

         search the tree RTree for the right anchor point of cur-vertex;
         RTree ← part of RTree whose vertices have y-coordiantes $\leq$ y-coordinate of the right anchor point;

         search the tree LTree for the left anchor point of cur-vertex;
         LTree ← part of LTree whose vertices have y-coordiantes $\geq$ y-coordinate of the left anchor point;

         store the vertex RTreeMax of Rtree and the vertex LTreeMin of LTree as the right and left anchor points of cur-vertex, respectively;

         insert cur-vertex into the tree RTree;

case 2: cur-vertex is a vertex of the chain CN($EXTREME_A$, m_n)

case 2a: cur-vertex is a vertex in M_C

search the tree LTree for the left anchor point of cur-vertex;
LTree ← part of LTree whose vertices have y-coordiantes ≤ y-coordinate of the left anchor point;

store the vertex LTreeMax of LTree as the left anchor point of cur-vertex;

insert cur-vertex into the tree LTree;

case 2b: cur-vertex is a vertex in M_A and lies to the right of the directed line LL
terminate processing the subpolygon M_1;

case 2c: cur-vertex is a vertex in M_A and lies to the left of the directed line LL
and to the right of the directed line RL

search the tree RTree for the right anchor point of cur-vertex;
RTree ← part of RTree whose vertices have y-coordiantes ≥ y-coordinate of the right anchor point;

search the tree LTree for the left anchor point of cur-vertex;
LTree ← part of LTree whose vertices have y-coordiantes ≤ y-coordinate of the left anchor point;

store the vertex RTreeMin of Rtree and the vertex LTreeMax of LTree as the right and left anchor points of cur-vertex, respectively;

insert cur-vertex into the tree LTree;

cur-vertex ← vertex of M_1 with the smallest y-coordinate larger than that of cur-vertex;

Until cur-vertex = $EXTREME_A$;

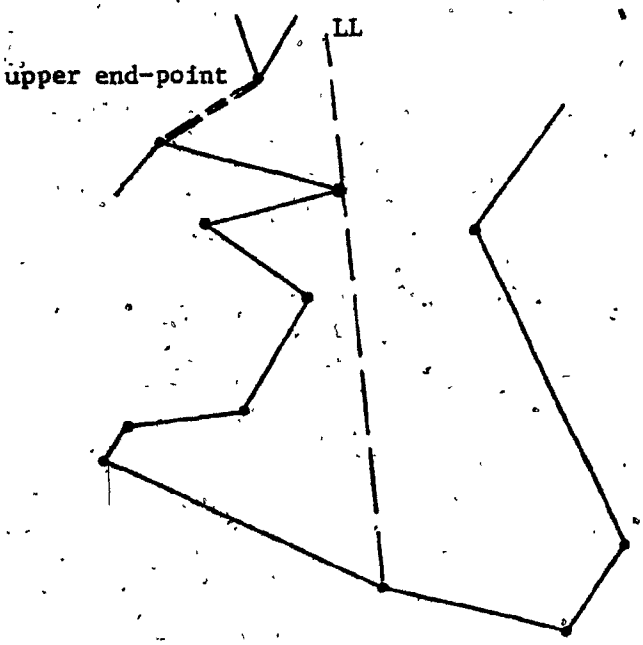
Steps for processing the subpolygon M_2

Details of these steps are similar to those of the steps for processing M_1, and are thus omitted.

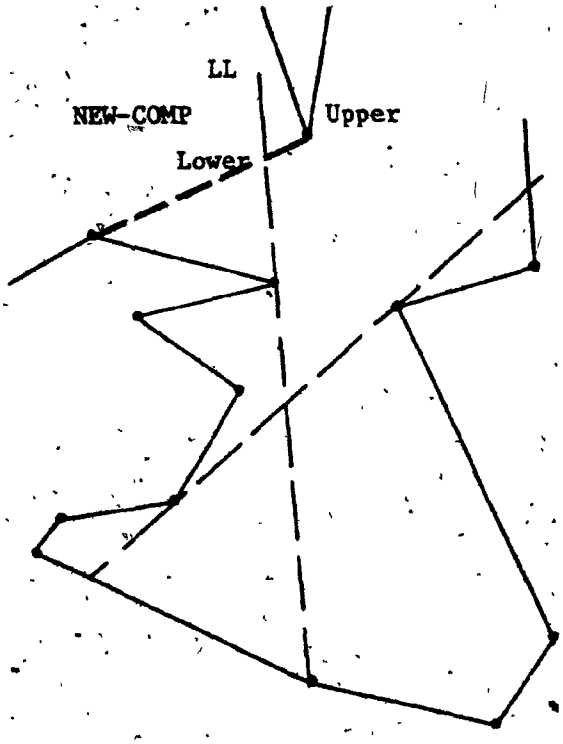
end MON-SCAN-2

Figure 3-14

Figure 3.15



case(a)



case(b)

procedure SCAN-M1 (M,RTree,LTree,Extreme)

input:
1. A simple polygon M that is monotone with respect to the y-axis.
2. Two balanced trees, RTree and LTree, which contains prospective right and left anchor points respectively.
3. The vertex with the largest y-coordinate in M.

output:
Right and left intercepts of the vertices of P that are weakly visible from e_n.

Method:

cur-vertex ← the vertex of M with the smallest y-coordinate;

Repeat

case 1: cur-vertex is a vertex in the chain CN(p_1,Extreme)

if cur-vertex is an upper end-point of a decomposition chord *then*

compute the two extreme points of the chord that are weakly visible from e_n;
Lower ← lower end-point of the decomposition chord or its intersection with LL;
Upper ← upper end-point of the decomposition chord or its intersection with RL;

search the tree RTree for the right anchor point of Upper;
$RTree_1$ ← part of RTree whose vertices have y-coordiantes $\leq$ y-coordinate of the right anchor point;
$RTree_2$ ← part of RTree whose vertices have y-coordiantes $\geq$ y-coordinate of the right anchor point;

search the tree LTree for the left anchor point of Upper;
$LTree_1$ ← part of LTree whose vertices have y-coordiantes $\geq$ y-coordinate of the left anchor point;
$LTree_2$ ← part of LTree whose vertices have y-coordiantes $\leq$ y-coordinate of the left anchor point;

Fetch the montone component which contains the found chord, NEW-COMP;

Locate the vertex with the largest y-coordinate in NEW-COMP, NewExtreme;

NEW–COMP$_1$ ← part of NEW-COMP whose vertices have y-coordinates larger than that of Lower;

SCAN-M1(NEW–COMP$_1$,$RTree_2$,$LTree_2$,NewExtreme);

case 1a: Upper $\neq$ cur-vertex (cur-vertex is not weakly visible from e_n)
terminate processing the subpolygon M;

case 1b: Upper = cur-vertex (cur-vertex is weakly visible from e_n)

RTree ← $RTree_1$;
LTree ← $LTree_1$;

store the vertex RTreeMax of Rtree and the vertex LTreeMin of LTree as the right and left anchor points of cur-vertex, respectively;

insert cur-vertex into the tree RTree;

else

case 1c: cur-vertex lies to the left of the directed line RL
terminate processing the subpolygon M;

case 1d: cur-vertex lies to the right of the directed line RL
and to the left of the directed line LL

search the tree RTree for the right anchor point of cur-vertex;
RTree ← part of RTree whose vertices have y-coordiantes $\leq$ y-coordinate of the right anchor point;

search the tree LTree for the left anchor point of cur-vertex;
LTree ← part of LTree whose vertices have y-coordiantes $\geq$ y-coordinate of the left anchor point;

store the vertex RTreeMax of Rtree and the vertex LTreeMin of LTree as the right and left anchor points of cur-vertex, respectively;

insert cur-vertex into the tree RTree;

end if

case 2: cur-vertex is a vertex in the chain CN(Extreme,p_n)

if cur-vertex is an upper end-point of a decomposition chord *then*

if cur-vertex is a vertex in M_C *then*
Lower ← lower end-point of the decomposition chord;
Upper ← cur-vertex;

search the tree LTree for the left anchor point of Upper;
$LTree_1$ ← part of LTree whose vertices have y-coordiantes $\leq$ y-coordinate of the left anchor point;
$LTree_2$ ← part of LTree whose vertices have y-coordiantes $\geq$ y-coordinate of the left anchor point;

Fetch the montone component which contains the found chord, NEW-COMP;

Locate the vertex with the largest y-coordinate in NEW-COMP, NewExtreme;

$NEW\text{–}COMP_1$ ← part of NEW-COMP whose vertices have y-coordinates larger than that of Lower;

SCAN-M1($NEW\text{–}COMP_1$,RTree,$LTree_2$,NewExtreme);

case 2a:

LTree ← $LTree_1$;

store the vertex LTreeMax of LTree as the left anchor point of cur-vertex;

insert cur-vertex into the tree LTree;

else

compute the two extreme points of the chord that are weakly visible from e_n;
Lower ← lower end-point of the decomposition chord or its intersection with RL;
Upper ← upper end-point of the decomposition chord or its intersection with LL;

search the tree RTree for the right anchor point of Upper;
$RTree_1$ ← part of RTree whose vertices have y-coordiantes $\geq$ y-coordinate of the right anchor point;
$RTree_2$ ← part of RTree whose vertices have y-coordiantes $\leq$ y-coordinate of the right anchor point;

search the tree LTree for the left anchor point of Upper;
$LTree_1$ ← part of LTree whose vertices have y-coordiantes $\leq$ y-coordinate of the left anchor point;
$LTree_2$ ← part of LTree whose vertices have y-coordiantes $\geq$ y-coordinate of the left ahchor point;

Fetch the montone component which contains the found chord, NEW-COMP;

Locate the vertex with the largest y-coordinate in NEW-COMP, NewExtreme;

NEW-COMP$_1$ ← part of NEW-COMP whose vertices have y-coordinates larger than that of Lower;

SCAN-M1(NEW-COMP$_1$,$RTree_2$,$LTree_2$,NewExtreme);

case 2b: Upper $\neq$ cur-vertex (cur-vertex is not weakly visible from e_n)
terminate processing the subpolygon M;

case 2c: Upper = cur-vertex (cur-vertex is weakly visible from e_n)

RTree ← $RTree_1$;
LTree ← $LTree_1$;

store the vertex LTreeMax of Ltree and the vertex RTreeMin of RTree as the left and right anchor points of cur-vertex, respectively;

insert cur-vertex into the tree LTree;

end if

else
case 2d: cur-vertex is a vertex in M_C

search the tree LTree for the left anchor point of cur-vertex;
LTree ← part of LTree whose vertices have y-coordiantes $\leq$ y-coordinate of the left anchor point;

store the vertex LTreeMax of LTree as the left anchor point of cur-vertex;

insert cur-vertex into the tree LTree;

case 2e: cur-vertex is a vertex in M_A and lies to the right of the directed line LL
terminate processing the subpolygon M_1;

case 2f: cur-vertex is a vertex in M_A and lies to the left of the directed line LL
and to the right of the directed line RL

search the tree RTree for the right anchor point of cur-vertex;
RTree ← part of RTree whose vertices have y-coordiantes $\geq$ y-coordinate of the right anchor point;

search the tree LTree for the left anchor point of cur-vertex;
LTree ← part of LTree whose vertices have y-coordiantes $\leq$ y-coordinate of the left anchor point;

store the vertex RTreeMin of Rtree and the vertex LTreeMax of LTree as the right and left anchor points of cur-vertex, respectively;

insert cur-vertex into the tree LTree;

end if

cur-vertex ← vertex of M with the smallest y-coorninate larger than that of cur-vertex;

Until cur-vertex == Extreme

end SCAN-M1

Figure 3-16

Figure 3.17

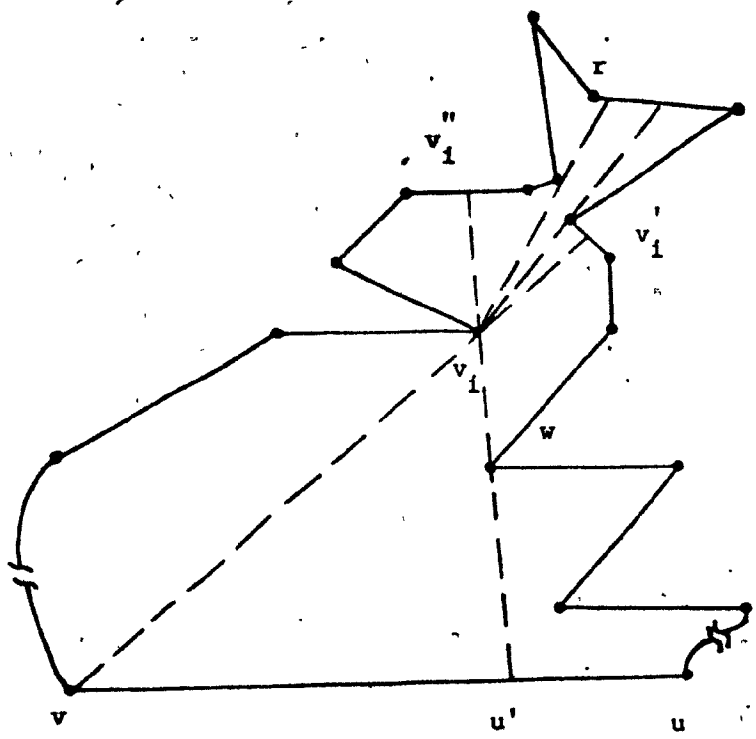
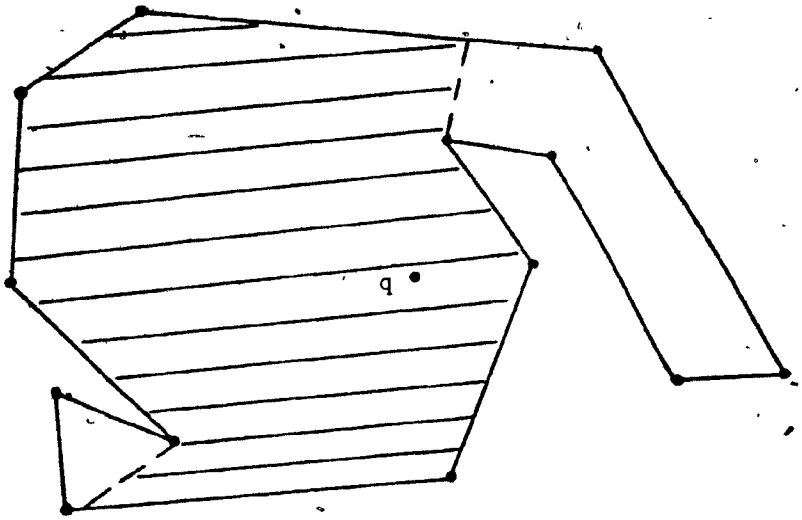
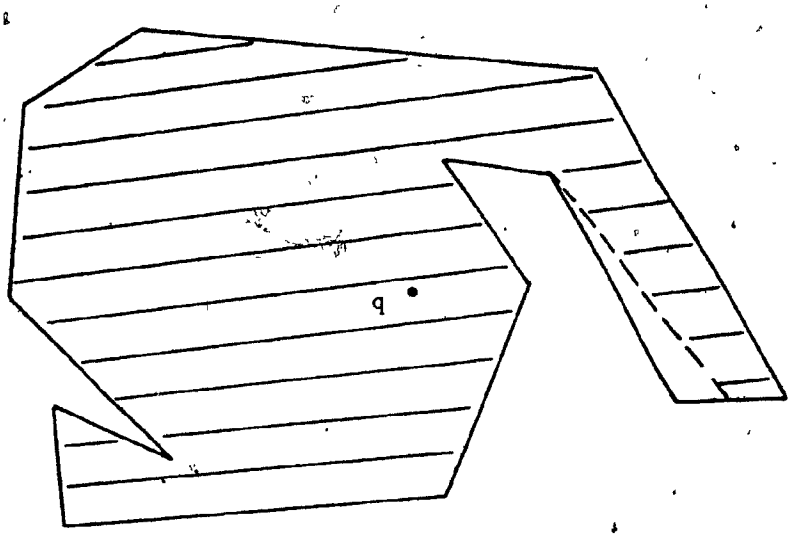


Figure 3.18



0-reachable region of q



1-reachable region of q

4. Recognizing Visibility Graphs of Simple Polygons

4.1. Introduction

Embedding a graph in the plane is a mapping of its vertices into distinct points and its edges into curves connecting the corresponding mapped vertices. Usually the mapping must satisfy some constraints. Examples are *planar embedding* where the mappings of edges must meet only at common mapped vertices, and *straight-line embedding* where edges must be mapped to straight line segments.

In this chapter we study the problem of recognizing *visibility graphs*, i.e., determining whether a graph can be embedded in the plane such that

(a) *a given hamiltonian cycle of the graph forms the boundary of a simple polygon,*

(b) *edges are mapped into straight line segments,*

(c) *the mapped vertices are visible if and only if they correspond to adjacent vertices in the graph.*

In section 4.2 we show that *maximal outerplanar graphs* are visibility graphs, and give an algorithm to draw the corresponding simple polygon. In section 4.3 we describe an algorithm to recognize visibility graphs of a specialized class of simple polygons, the class of *convex fans*. Section 4.4 is devoted to discussing the problem of recognizing visibility graphs of arbitrary simple polygons.

4.2. Embedding Maximal Outerplanar Graphs as Visibility Graphs

A graph which admits a *planar embedding* such that all the vertices lie on the exterior face is said to be an *outerplanar* graph. A *maximal outerplanar* graph, a *mop* for short, is an outerplanar graph such that the addition of a single edge results in a non-outerplanar graph (since the existence of a *cut-vertex* will result in the graph being non-hamiltonian, graphs to be dealt with through the rest of this section are assumed to be *2-connected*). The collection of

exterior edges (i.e. edges that lie on the exterior face) defines a *hamiltonian cycle* in the graph. Beyer et. al. [B&J&M] showed that a *2-connected mop* has a unique hamiltonian cycle defined by the collection of exterior edges and presented a linear time algorithm to report such cycle. In this section we show that every *2-connected mop* is the visibility graph of a simple polygon and give an algorithm to draw such a polygon in the plane.

Given a *mop* G=(V,E) of order n labelled such that the sequence $(v_1, v_2, \ldots, v_n, v_1)$ forms a hamiltonian cycle, the algorithm locates and embeds one *interior face* of G at a time. The removal of an interior face decomposes the graph into at most two components of connected faces, where two faces are said to be connected if they share an edge. The algorithm then proceeds to embed each component separately in a recursive fashion.

The key properties for locating interior faces efficiently are given in the following lemmas.

Lemma 4.1 For every exterior edge of G, there exists exactly *one* vertex in G that is adjacent to both end-points

Proof Consider the edge of the hamiltonian cycle connecting v_n to v_1. Let v_k be the vertex with the largest label that is adjacent to v_1 and let $v_{k'}$, $k' \neq k$, be the vertex with the smallest label that is adjacent to v_n (Refer to Figure 4.1 for illustration). It is easy to see that at least one extra edge can be added without violating the outerplanarity of G, contradicting the assumption that the given graph G is maximal. Therefore k must be equal to k'. It follows directly from *Jordan curve theorem* and the outerplanarity of G that only one such vertex can exist. Thus the lemma follows.

Q.E.D.

Lemma 4.2 For every interior edge of G, there exists exactly *two* vertices in G that are adjacent to both end-points.

Proof In a planar embedding of G, the given hamiltonian cycle is mapped into a closed curve which partitions the plane into two disjoint regions. A mapping of an interior edge of G, say

(v_i, v_j), decomposes the bounded region into two bounded regions R_1 and R_2 as shown in Figure 4.2. Induced subgraphs on the vertices on the boundary of each one of the regions R_i, i= 1, 2, form smaller maximal outerplanar graphs with the edge (v_i, v_j) being an exterior edge. From lemma 4.1, there exists only *one* vertex in each induced subgraph that is adjacent to both v_i and v_j and thus the lemma follows.

Q.E.D.

We now describe the algorithm EMBED-A-MOP for embedding a mop G on the plane such that the given hamiltonian cycle forms the boundary of a simple polygon whose visibility graph is the mop G.

procedure EMBED-A-MOP

procedure EMBED-A-FACE (Bottom, LeftPoint, RightPoint)
local variable Left, Right, NewBottom;
local variable NewPoint, NewLeftPoint, NewRightPoint;

method:

Define
Left ← the vertical line passing through the vertex LeftPoint;
Right ← the vertical line passing through the vertex RightPoint;
(X,Y) ← coordinates of the middle point of the line segment Bottom;

if there exists an unmarked vertex adjacent to both LeftPoint and RightPoint *then*
NewPoint ← the unmarked vertex adjacent to both LeftPoint and RightPoint;
if LeftPoint $\neq v_1$ *or* RightPoint $\neq v_n$ *then*
Erase the line segment joining the vertex LeftPoint and the vertex RightPoint;
else
return

Map the vertex NewPoint as the point (X,Y+ϵ), where ϵ has a small positive value;

Mark the vertex NewPoint;
Draw a line segment to join the vertex LeftPoint and the vertex NewPoint;
Draw a line segment to join the vertex NewPoint and the vertex RightPoint;

Delete
LeftPoint and RightPoint from the list of adjacent vertices of NewPoint;
NewPoint from the list of adjacent vertices of LeftPoint;
NewPoint from the list of adjacent vertices of RightPoint;

Split the list of adjacent vertices of NewPoint such that one list contains the vertices with labels less than that of NewPoint and the remaining vertices are contained in the other list.

NewBottom ← the line segment joining the vertex NewPoint and the intersection point of the half-line "Left" with the directed line joining the vertex RightPoint to the vertex NewPoint;

EMBED-A-FACE (NewBottom, LeftPoint, NewPoint);

NewBottom ← the line segment joining the vertex NewPoint and the intersection point of the half-line "Right" with the directed line joining the vertex LeftPoint to the vertex NewPoint;

EMBED-A-FACE (NewBottom, NewPoint, RightPoint);

end EMBED-A-FACE

for each vertex in G *do*
sort labels of the adjacent vertices in increasing order;

Map the vertices v_1 and v_n as the two points $(x_1,0)$ and $(x_2,0)$ on the x-axis such that $x_1 < x_2$;

Mark the vertices v_1 and v_n;

Define B as the line segment joining the points $(x_1,0)$ and $(x_2,0)$.

EMBED-A-FACE (B, v_1, v_n);

end EMBED-A-MOP

An example to demonstrate the performance of the procedure is shown in Figure 4.3.

Using the following result (stated previously as lemma 3.1),

> **Lemma 4.3**[E&A] Let POL be a simple polygon and e_i be an edge in its boundary such that the vertices pol_{i-1} and pol_{i+2} lie on the same side of the line passing through e_i. If y is a point in the exterior of POL which is collinear with the the edge e_i, then the line segment joining y to a point in the interior of POL intersects the chain CN(pol_{i+1}, pol_i).

we prove the correctness of the algorithm in the following theorem. We also give the complexity of the algorithm

Theorem 4.4 Given a *mop* G of order n and the associated hamiltonian cycle. The procedure EMBED-A-MOP embeds the graph G in the plane as the visibility graph of a simple polygon in $O(n\log n)$ time.

Proof At each call of the procedure EMBED-A-FACE, the following conditions are satisfied:

a) the embedded (marked) vertices form a simple polygon, denoted by IntPol, whose visibility graph is the subgraph of G induced on the embedded vertices.

b) the chain $CN(v_1, v_n)$ of the polygon IntPol is monotone with respect to the x-axis.

The conditions a)-b) are clearly satisfied when the procedure EMBED-A-FACE is called for the first time. Now, assume that conditions a)-b) have always been satisfied prior to the call of the procedure EMBED-A-FACE with the vertices v_i and v_j as the LeftPoint and RightPoint respectively.

If no unmarked vertex that is adjacent to both v_i and v_j exists, the procedure *returns* immediately and conditions a)-b) remain satisfied. Otherwise, the procedure embeds the vertex that is adjacent to both v_i and v_j, say v_k, in the region that is the intersection of the half-planes to the left of the line Right, to the right of the line Left and above the line passing through Bottom. Extend the line segment connecting v_k to v_i until it intersects the boundary of IntPol in a point, say Z', and extend the line segment connecting v_k to v_j until it intersects the boundary of IntPol in a point, say Z" (as shown in Figure 4.4). From lemma 4.2, exactly one of the embedded vertices is adjacent to both v_i and v_j, and this vertex lies on the line through the line segment Bottom. Therefore, no embedded vertex can lie in the quadrilateral $(v_i, v_j, Z", Z')$. The region bounded by the chain $CN(Z', v_i)$ and the line segment (v_i, Z') together with the vertex v_k satisfy the conditions of lemma 4.3. Also the region bounded by the chain $CN(v_j, Z")$ and the line segment $(Z", v_j)$ together with the vertex v_k satisfy the conditions of lemma 4.3. Therefore, v_k is not visible from any of the previously embedded vertices except v_i and v_j, and thus condition a) is satisfied after embedding v_k. It is easy to see that condition b) is also satisfied, which completes the proof of correctness for each step and the algorithm.

At every call of the procedure EMBED-A-FACE, a vertex is mapped into a point in the plane that is not changed during the following calls. To find an unmarked vertex that is adjacent to both LeftPoint and RightPoint, if one exists, we only have to check the vertices

with the smallest and largest labels in the adjacency lists of LeftPoint and RightPoint. A process which can be performed in constant time. The time required to process the NewPoint is proportional to logarithm of the number of its adjacent vertices. Since the number of edges in the graph G is $O(n)$, it follows that the total running time of the algorithm is $O(n \log n)$.

Q.E.D.

4.3. Recognizing the Visibility Graph of a Convex Fan

In this section we use the decomposition strategy to check the feasibility of embedding a graph, with a given hamiltonian cycle, in the plane as the visibility graph of a specialized class of simple polygons, called *convex fans*. The algorithm is based on using a subset of edges of the given graph, called *maximal diagonals*, to partition the graph into disjoint subgraphs which are independently embedded in non-overlapping regions of the plane. Each component is then processed in a similar fashion (i.e., its maximal diagonals are located and used to decompose the component further into smaller components which can be independently embedded in the plane). The process is continued until all the vertices are embedded or the graph is found not to be a visibility graph of a convex fan.

First we introduce some additional terminology. A simple polygon is said to a *convex fan* if there exists a *convex vertex* (i.e., a vertex with an interior angle less than π) that is visible from all the points in the interior of the polygon. Such a vertex will be called a *kernel point* of the polygon. It is easy to see that the vertices of the convex fan appear in sorted angular order around the kernel point as the boundary of the polygon is traversed. Let P be a convex fan with the vertices labeled such that p_1 is a point kernel. Two chords *interlace* if their end points alternate on BD(P) when the boundary of the polygon is traversed. A chord connecting the vertices p_i and p_j, $2 \leq i < j \leq n$ is said to be a *maximal diagonal* in the $CN(p_2, p_n)$, if there is no chord connecting p_l and p_k such that $2 \leq l \leq i < j \leq k \leq n$, where either $l \neq i$ or $k \neq j$. p_i (p_j), where $i < j$, is called the *left* (*right*) end point of the maximal diagonal connecting p_i and p_j (Refer to Figure 4.6 for an illustration). Since the chain $CN(p_{i+1}, p_{j-1})$ together with the

kernel point p_1 form a convex fan, we can similarly define *maximal diagonals* in the chain $CN(p_{i+1}, p_{j-1})$

In the following lemma we introduce a key property of maximal diagonals in convex fans.

Lemma 4.5 Two maximal diagonals in a convex fan do not interlace.

Proof (Refer to Figure 4.6) Assume that $CHORD_{p_i p_j}$ and $CHORD_{p_l p_k}$ are two interlacing maximal diagonals in a convex fan P such that $i < l$. Since the chord $CHORD_{p_i p_j}$ partitions the polygon P into two components one contains p_l and the other contains p_k. The two chords must intersect in a point inside P, say X. It is easy to see that the quadrilateral (p_1, p_i, X, p_k) is convex and lies completely inside P. Therefore, p_i and p_k are visible contradicting the fact that both $CHORD_{p_i p_j}$ and $CHORD_{p_l p_k}$ are maximal diagonals.

Q.E.D.

Corollary 4.5.1 If $CHORD_{p_i p_j}$ is a maximal diagonal in the chain $CN(p_2, p_n)$, then a vertex in $CN(p_{i+1}, p_j)$ is *not* visible from vertices in the chain $CN(p_2, p_{i-1})$ and a vertex in $CN(p_i, p_{j-1})$ is *not* visible from vertices in the chain $CN(p_{j+1}, p_n)$

Using the above property we can hierarchically describe a convex fan by recursively identifying the maximal diagonals. For the convex fan P (shown in Figure 4.7) whose vertices labeled such that p_1 is its kernel point, the top of the hierarchy (level 0) represents the whole polygon. Nodes in the first level represent the maximal diagonals in the chain $CN(p_2, p_n)$. In the following levels, *sons* of the node corresponding to the maximal diagonal connecting p_i to p_j represent the maximal diagonals in the chain $CN(p_{i+1}, p_{j-1})$.

In the following lemmas we study the visibility between the end-points of a maximal diagonal and the vertices stored in its subhierarchy.

Lemma 4.6 Let $CHORD_{p_i p_j}$ be a maximal diagonal in the convex fan P and $Q = (q_{k_1}, q_{k_2}, \ldots, q_{k'})$ be the sequence of end-points of the maximal diagonals in the chain $CN(p_{i+1}, p_{j-1})$. p_i is visible from a consecutive subsequence of Q

Proof (Refer to Figure 4.8) From the definition of a maximal diagonal and from lemma 4.5, we can easily observe that the vertices of Q form a convex chain since the vertices of the chain $CN(p_2,p_n)$ are sorted in angular order around p_1. The vertex p_i lies outside the region defined by the intersection of the half-planes to the right of the directed line connecting p_1 to q_{k_1} and to the left of the directed line connecting p_1 to $q_{k'}$, shown hashed in Figure 4.8, and is thus visible from consecutive subsequence of Q.

Q.E.D.

Lemma 4.7 Let $Q = (q_{k_1}, q_{k_2}, \ldots, q_{k'})$ be the sequence of end-points of the maximal diagonals in the chain $CN(p_{s+1}, p_{t-1})$, and $CHORD_{p_s p_t}$ be a maximal diagonal stored as their ancestor in the hierarchical description of the polygon. If k_m is the the largest label of a vertex in Q that is visible from p_i, then vertices in Q with larger labels are *not* visible from vertices in the chain $CN(p_i, p_{k_m-1})$.

Proof (Refer to Figure 4.9.) Extend the line segment joining p_i to p_{k_m} until it intersects BD(P) in a point, say X. The arm $ARM_{p_{k_m}X}$, which completely contains the vertices of Q with label larger than k_m, together with the vertex p_i satisfy the conditions of lemma 4.3. Therefore p_i is *not* visible from the vertices of Q with labels larger than k_m. Vertices of the chain $CN(p_{i+1}, p_{k_m-1})$ and vertices in the arm $ARM_{p_{k_m}X}$ lie on the same side of a straight line. It follows from lemma 4.3 that they are also *not* visible.

Q.E.D.

Lemma 4.8 Let $Q = (q_{k_1}, q_{k_2}, \ldots q_{k'})$ be the sequence of end-points of the maximal diagonals in the chain $CN(p_{s+1}, p_{t-1})$. Let $LEP(p_s, p_t)$ be the sequence of *left vertices* of maximal diagonals stored in the hierarchical description of the polygon on the path from the top node to the node that corresponds to the maximal diagonal connecting p_s to p_t, and i (j) be the *smallest* label between the end-points in $LEP(p_s, p_t)$ that is visible from q_{k_i} (q_{k_j}). If k_i is greater than k_j, then i is less than or equal to j.

Proof Similar to lemma 4.7.

Lemma 4.9 Let $Q = (q_{k_1}, q_{k_2}, \ldots q_{k'})$ be the sequence of end-points of the maximal diagonals in the chain $CN(p_{s+1}, p_{t-1})$. Let $LEP(p_s, p_t)$ be the sequence of *left vertices* of maximal diagonals stored in the hierarchical description of the polygon on the path from the top node to the node that corresponds to the maximal diagonal connecting p_s to p_t. If q_{k_m} is visible from lep_{i_1} and lep_{i_2} and *not* visible from the vertices between them in the sequence $LEP(p_s, p_t)$, then vertices of Q with labels smaller than k_m are *not* visible from vertices in $CN(lep_{i_1+1}, lep_{i_2-1})$.

Proof (Refer to Figure 4.10 for illustration.) Consider the convex fan $P' = \{p_1, \ldots, p_s, q_{k_1}, q_{k_2}, \ldots, q_{k'}, p_t, \ldots, p_n\}$. Extend the line segment joining q_{k_m} to lep_{i_2} until it intersects BD(P') in a point, say X. The vertices in $LEP(p_s, p_t)$ with labels between i_1 and i_2 lie completely in the arm $ARM_{X\, lep_{i_2}}$. Since vertices in the chain $CN(lep_{i_1}, lep_{i_2})$ and vertices in Q with labels smaller than k_m lie on the same side of straight line. It follows from lemma 4.3 that they are *not* visible.

Q.E.D.

Right end-points of maximal diagonals satisfy properties similar to those shown in lemmas 4.6-9. The proofs are omitted.

Algorithm RECOGNIZE-CONVEX-FAN

Let G=(V,E) be a graph of order n with the vertices labeled such that the sequence $(v_1, v_2, \ldots, v_n, v_1)$ forms a hamiltonian cycle, and the vertex v_1 has degree n-1. Note that such a vertex must exist for the graph to be the visibility graph of a convex fan, and will be mapped as its kernel point. We now describe an algorithm for checking whether G is the visibility graph of a convex fan and producing an embedding of such polygon, if one exists. The algorithm proceeds as follows:

Step 1 Define G' to be the subgraph of G induced on the set of vertices $V-\{v_1\}$. The

algorithm computes the maximal diagonals of G' with respect to the hamiltonian path $v_2, v_3, \ldots, v_n$. If the maximal diagonals do not interlace, the algorithm partitions G' by deleting the end-points of the maximal diagonals and proceeds to process each component in a recursive fashion. Otherwise, the graph cannot be embedded as the visibility graph of a convex fan and the algorithm terminates unsuccessfully. A detailed description of this step is given in Figure 4.11.

Step 2 Define a graph G"=(V",E") such that V" is the set of vertices V-$\{v_1\}$ and E" is the set of maximal diagonals reported in step 1 plus the pair (v_2, v_n), if it is not already included. The algorithm uses the procedure BUILD-HIER-DESC, described in chapter 2, to compute a hierarchical description of G". Such a hierarchical description can be built since the edges of G" form a nested set of diagonals. The added edge (v_2, v_n) is stored at the top of the hierarchy and will be handled as a special case in the next step. It will be mapped as a single point corresponding to the node v_1.

Step 3 First the algorithm embeds the vertex v_1 at an arbitrary point in the plane, say O. Starting at the root of the hierarchy, it embeds vertices of the maximal diagonals stored as its sons in the hierarchy as a convex chain, as shown in Figure 4.12, and then proceeds to attempt embedding the corresponding subgraphs separately. To embed the sequence of end-points Q = $(v_{k_1}, v_{k_2}, \ldots)$ of the maximal diagonals stored in the hierarchy as the sons of the maximal diagonal connecting v_i to v_j, the algorithm first checks that it satisfies the properties in lemmas 4.6-9, and then embeds the vertices as a convex chain such that every two connected vertices are visible. Details of this step is given in the procedure EMBED-A-CHAIN, shown in Figure 4.13.

We now give an overall description of the algorithm

procedure RECOGNIZE-CONVEX-FANS (G)

input: A graph G=(V,E) of order n and a permutation π of the vertices such that the sequence $(v_1, v_2, \ldots, v_n, v_1)$ forms a hamiltonian cycle.

output: An embedding of G in the plane such that G is the visibility graph of a convex fan.

global variables:

LEP(C) = $(lep_1, lep_2, \ldots, lep_{m_1})$ left end-points of the maximal diagonals stored in the hierarchy as C and its ancestors arranged in order of increasing distance from the top. For each end point p, we associate a pointer *p-fep* to the furthest entry in the list such that the two points are connected by an edge in the graph.

REP(C) = $(rep_1, rep_2, \ldots, rep_{m_2})$ of right end-points of the maximal diagonals stored in the hierarchy as C and its ancestors arranged in order of increasing distance from the top. For each end point p, we associate a pointer *p-fep* to the furthest entry in the list such that the two points are connected by an edge in the graph.

method:

1. Compute the maximal diagonals of the graph G using the procedure REPORT-MAX-DIAG.
2. Build a hierarchical description of G based on the maximal diagonals reported in the first step using the procedure BUILD-HIER-DESC.
3. Map the top node of the hierarchy into an arbitrary point in the plane, O.

Embed the vertices of the maximal diagonals stored as its sons as a convex chain with the point O on its concave side.

```
for each son "Current" of the top node in the hierarchy do
    LEP(Current) ← A;
    REP(Current) ← A;
    EMBED-A-CHAIN (Current);
end for
```

end RECOGNIZE-CONVEX-FANS

We now prove the correctness of the procedure RECOGNIZE-CONVEX-FANS.

Theorem 4.10 Given a graph G=(V,E) with the vertices labeled such that the sequence $(v_1, v_2, \ldots, v_n, v_1)$ forms a hamiltonian cycle. The procedure RECOGNIZE-CONVEX-FANS *terminates successfully* if and only if G is the visibility graph of a convex fan.

Proof The procedure terminates *unsuccessfully* when one of the properties in lemmas 4.5-9 is not satisfied, and thus G with the given labeling can not be the visibility graph of a convex fan.

For the converse, one can easily see that the edges reported by the procedure REPORT-MAX-DIAG, after a successful completion, form a set of non-interlacing diagonals. Correctness of the procedure EMBED-A-CHAIN is proved by showing that the embedded vertices satisfy the following condition at each call of the procedure:

the embedded vertices form a convex fan with the kernel point O, denoted by IntFan, whose visibility graph is the subgraph of G induced on the embedded vertices.

At the first call of the procedure EMBED-A-CHAIN only the node v_1 is processed and the condition is satisfied. Now, assume that the condition has been satisfied prior to considering the node of the hierarchy corresponding to the maximal diagonal connecting v_l and v_r.

The procedure embeds the end-points of Q as a convex chains in the region that is the intersection of the half-planes to the left of the half-line connecting O to r, the right of the half-line connecting O to l and the right of the half-line connecting r to l. The fact that their intersection is *not* empty follows from the fact that the embedded vertices form a convex fan with O as the kernel point (Refer to Figure 4.14). After q_{k_i} is embedded such that connectivities of the end-points $q_{k'}$ ..., q_{k_i} in the graph are satisfied, the procedure embeds the end-point $q_{k_{i-1}}$ in the half-plane to the left of the directed line joining O to q_{k_i}. Such a mapping does not contradict the fact that the end-points of Q and of LEP(l,r) satisfy the condition of lemma 4.8-9. Exact mapping of $q_{k_{i-1}}$ is then selected to satisfy its connectivity in the graph.

The loops in the procedure EMBED-A-CHAIN, which check that lemmas 4.8-9 are satisfied, can be implemented in quadratic running time. We can conclude that the procedure, and also the algorithm RECOGNIZE-CONVEX-FANS, runs in $O(n^3)$ time.

Q.E.D.

4.4. Discussion of the General Problem

In the previous section we studied the problem of embedding a graph G, with a given hamiltonian cycle, as the visibility graph of a convex fan and presented a polynomial algorithm for reporting such an embedding, if one exists. This problem generalizes naturally in two ways.

One generalization is Given a graph, with a known hamiltonian cycle, check whether it can be embedded in the plane as the visibility graph of a simple polygon. Meisters [Mei] proved that a simple polygon has at least two *ears*, an *ear* is a convex vertex with the extra property that its two neighbours on the boundary of the polygon are visible It follows directly from Meisters' result that every visibility graph of a simple polygon has at least one induced subgraph which is the visibility graph of a convex fan Therefore a necessary condition for embedding a graph G of order n, with a given hamiltonian cycle, as the visibility graph of a simple polygon is that there exists a labelling of of the vertices $v_1, v_2, \ldots, v_n$ and a corresponding sequence of graphs $G_1, G_2, \ldots, G_n$ such that:

(1) $G_1 \leftarrow G$

(2) If Vis=$(v_{k_1}, v_{k_2}, \ldots, v_{k_m})$ is the sequence of nodes in G_i adjacent to v_i arranged in the same order as they appear in the given hamiltonian cycle, then (v_i, Vis, v_i) is a hamiltonian cycle in G_i.

(3) The induced subgraph on v_i and Vis is the visibility graph of a convex fan.

(4) The graph G_{i+1} is obtained from G_i by deleting v_i together with all the incident edges.

Finding the sufficient condition(s) for a graph with a known hamiltonian cycle to be the visibility of a simple polygon, equivalently designing an embedding algorithm, is an open problem Unlike the procedure RECOGNIZE-CONVEX-FANS, where a decomposition criterion has been devised for the case of a convex fan, an algorithm for this generalization must be capable of resolving the interaction between two already embedded simple polygons to form a new polygon.

A second generalization is: Given a graph check whether it can be embedded in the plane as the visibility graph of a convex fan One may attempt to find a subset of the vertices V' whose deletion, together with the kernel point, decomposes the graph into $|V'|+1$ components, check the feasibility of embedding the nodes of V', and then proceed to process each component separately. Unfortunately, there may *not* exist such subset of vertices in the

visibility graph of a convex fan. In this case, the end-points of any edge in the graph may be selected as the two neighbours of the kernel point on the boundary of the polygon. Using the nine vertices graph shown in Figure 4.15-a as a building block, where at least two edges can be selected, a graph of order n with $2^{n/8}$ choices can be constructed. An example of a graph of order seventeen which has at least four such choices is given in Figure 4.15-b. Therefore an algorithm for solving the problem based on attempting all possible choices requires $\Omega(2^n)$. However it is still an open problem to find the complexity of the problem itself.

Figure 4.1

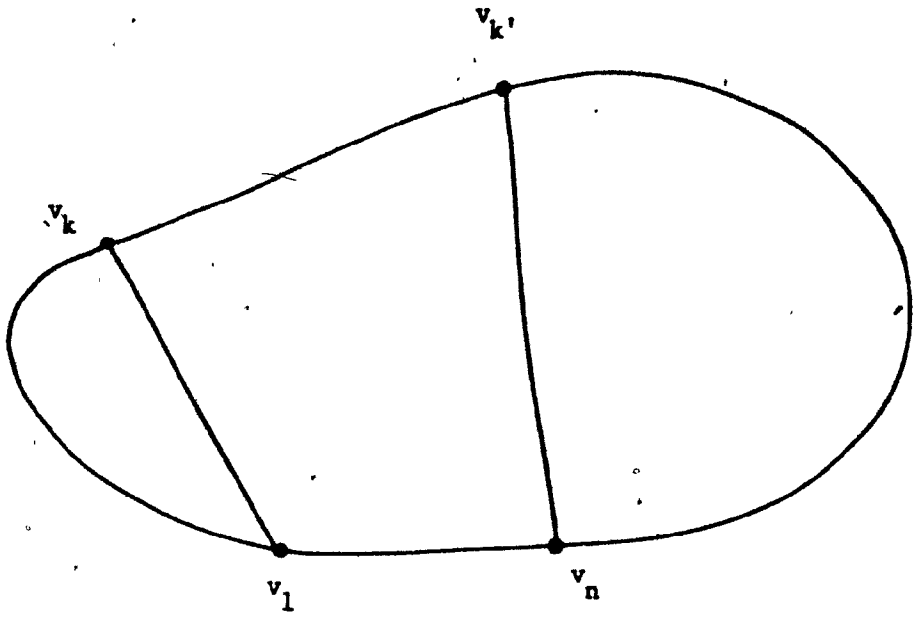


Figure 4.2

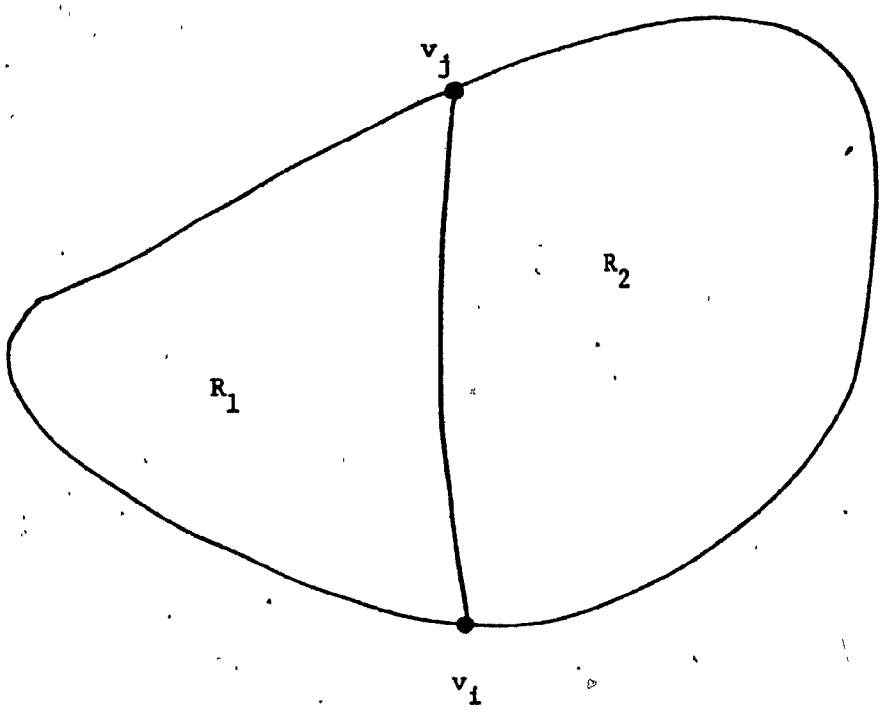
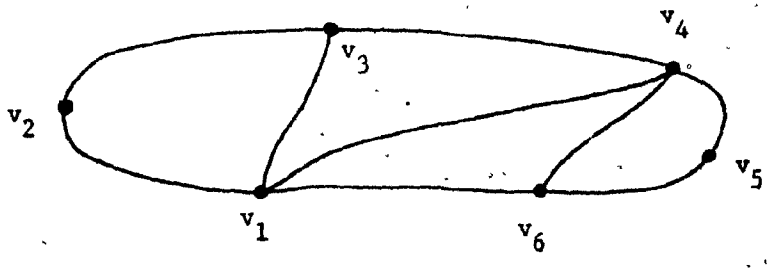
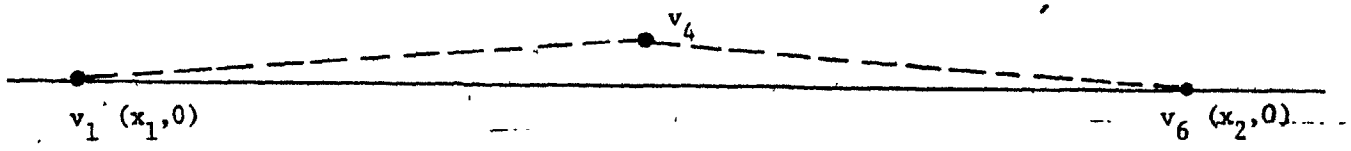


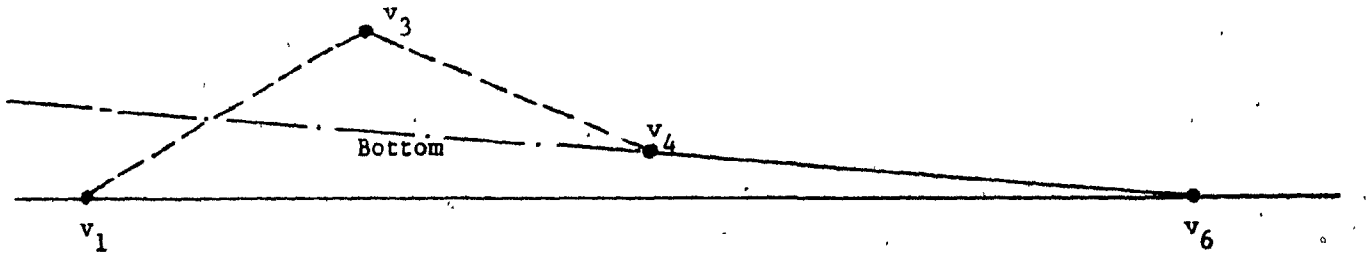
Figure 4.3

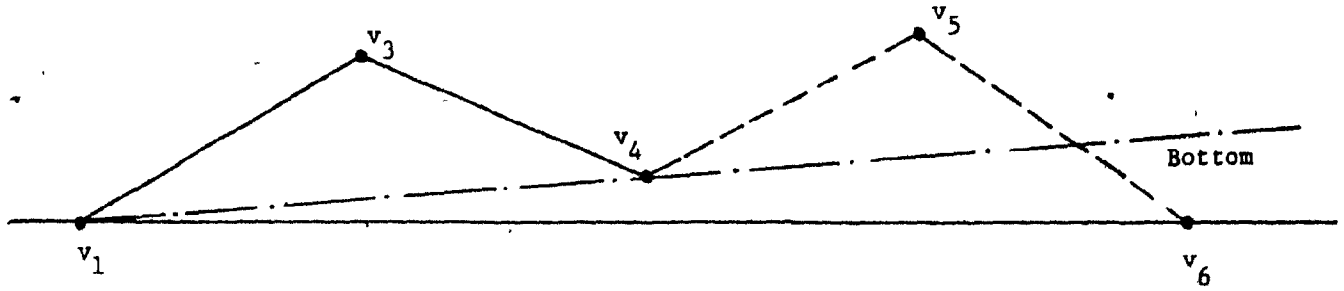


A mop G of order six

Embedding Steps:







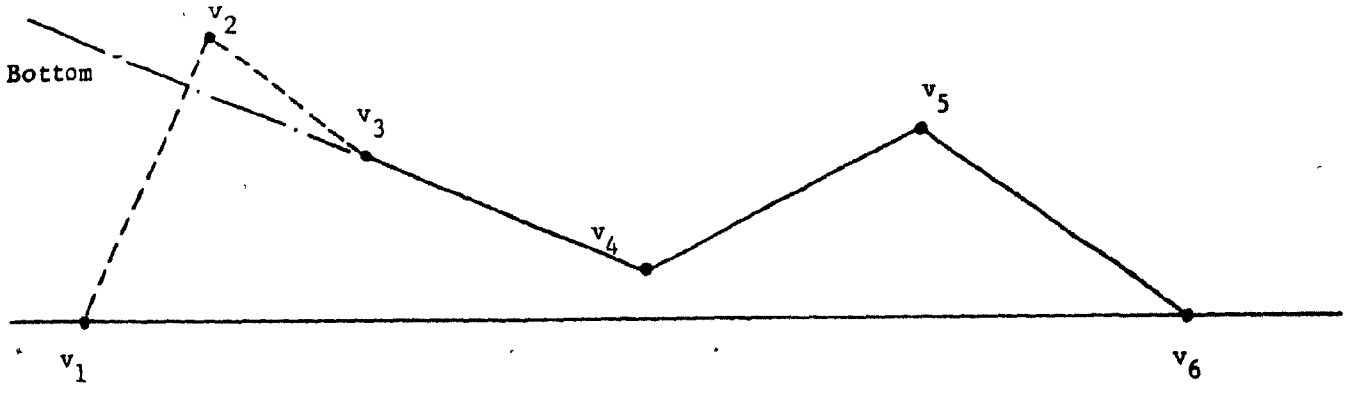


Figure 4.4

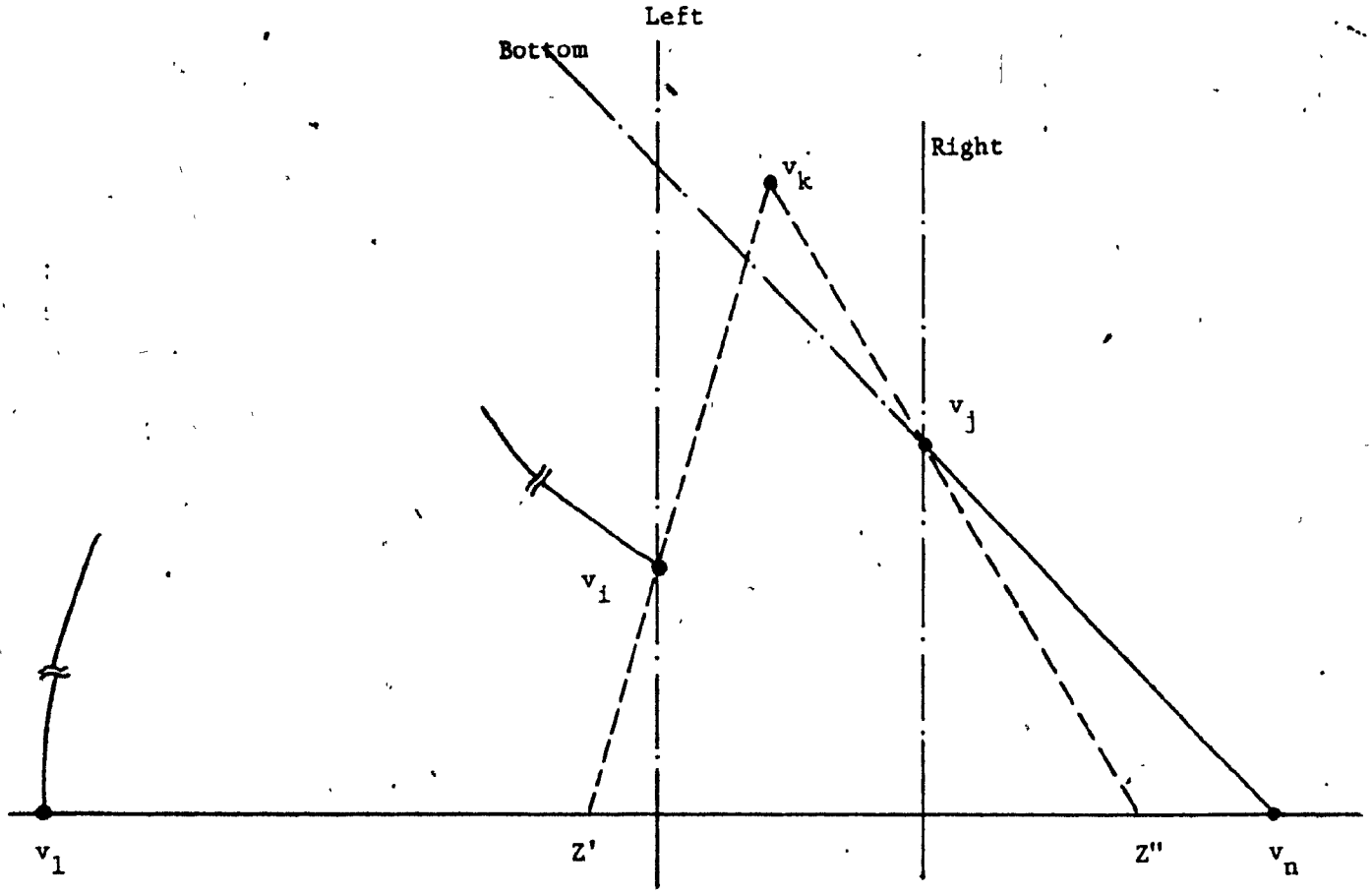


Figure 4.5

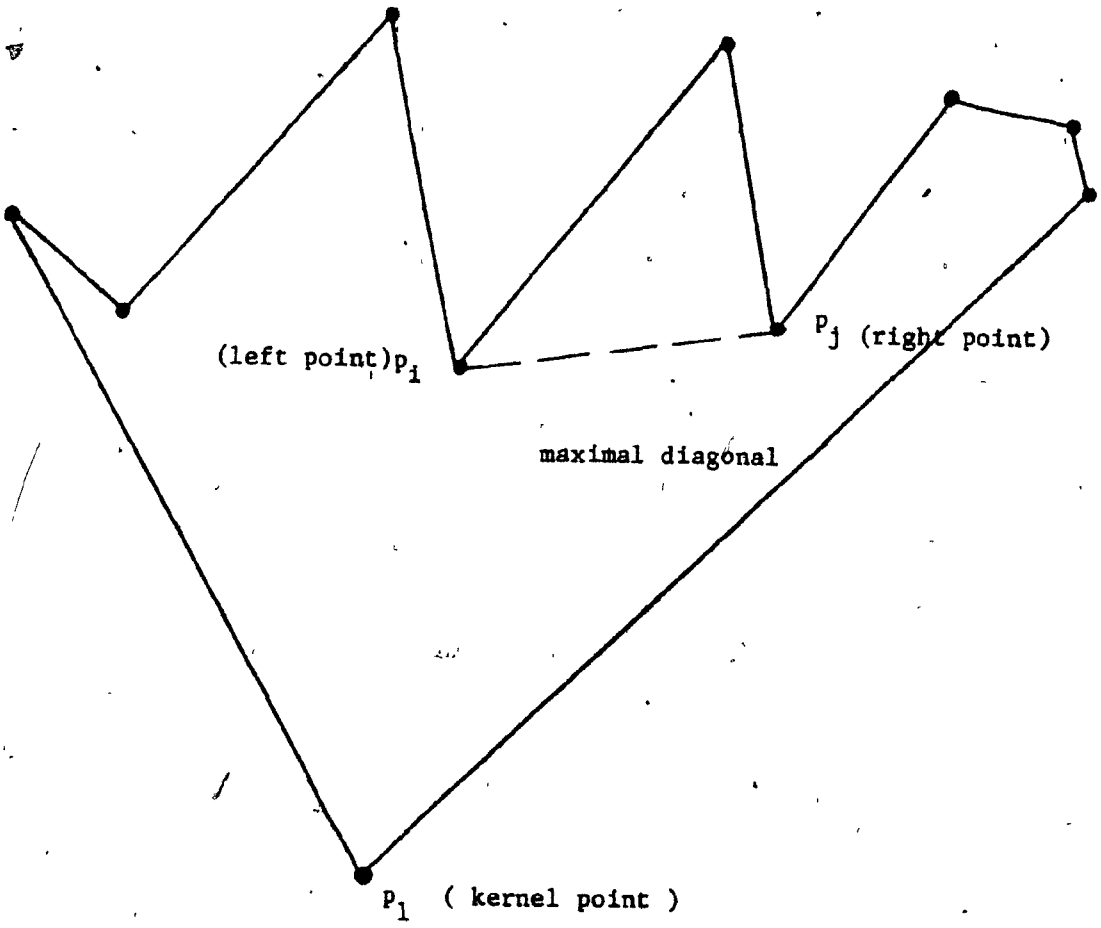

(left point)p_i
p_j (right point)
maximal diagonal
p_1 (kernel point)

Figure 4.6

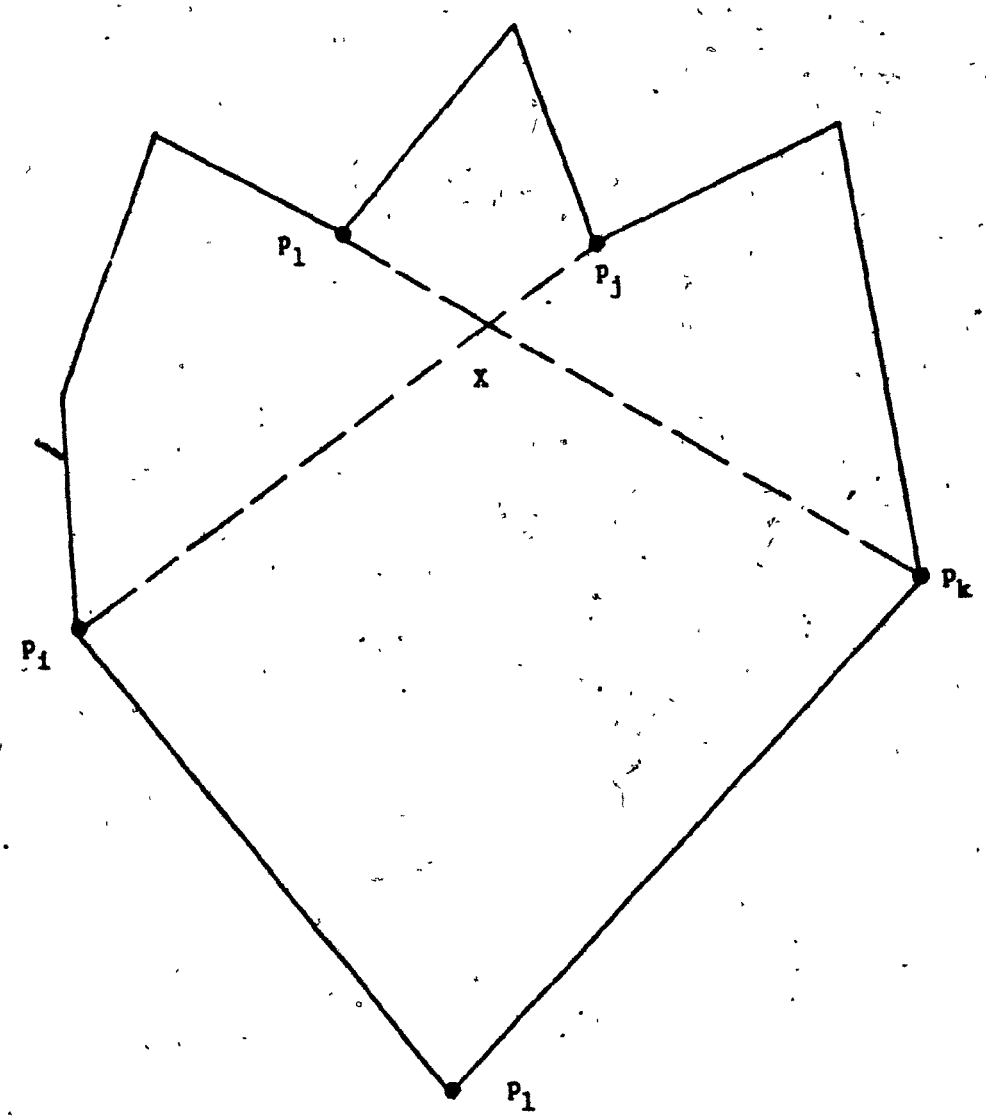
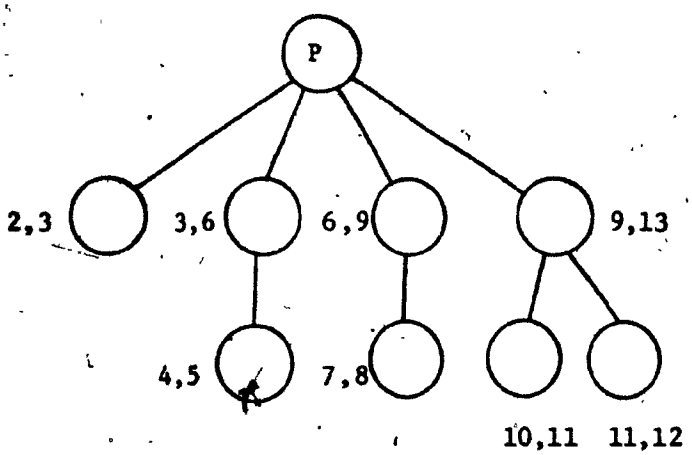


Figure 4.7


P
2,3
3,6
6,9
9,13
4,5
7,8
10,11
11,12

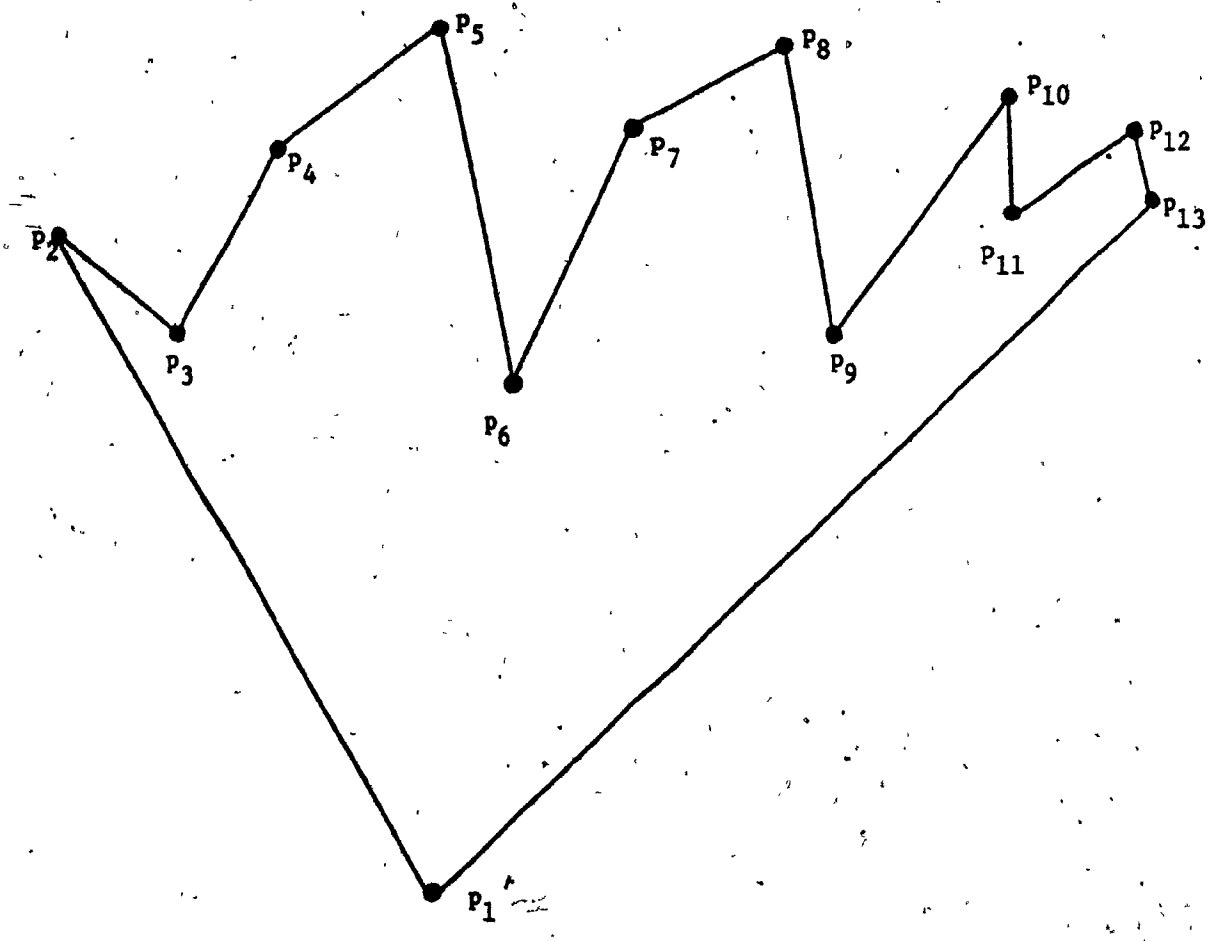

P1
P2
P3
P4
P5
P6
P7
P8
P9
P10
P11
P12
P13

Figure 4.8

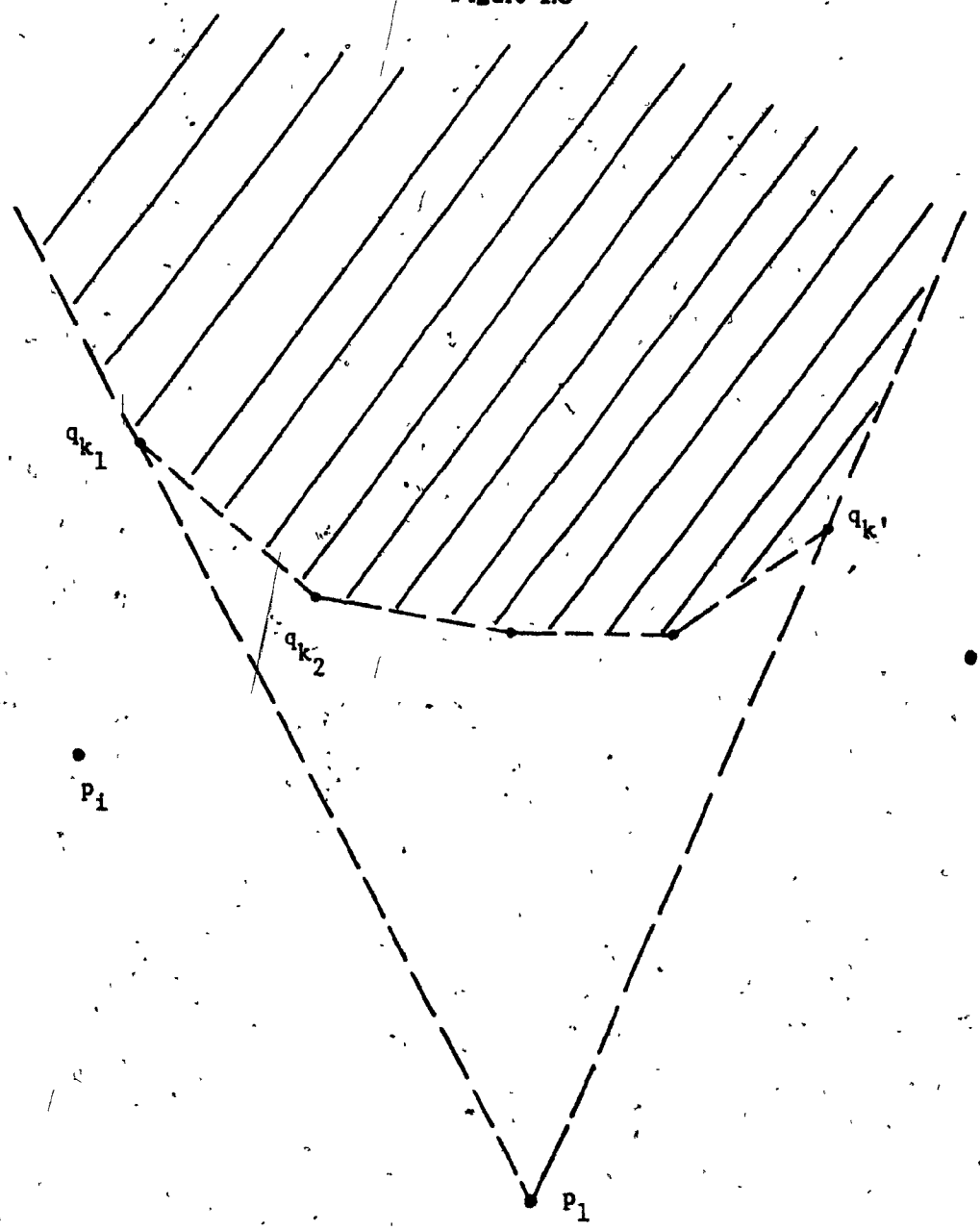


Figure 4.9

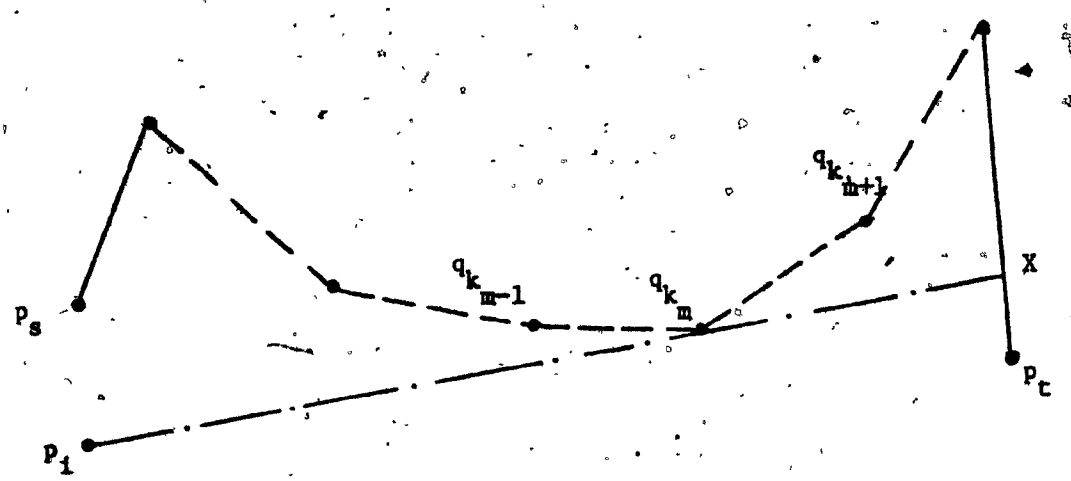
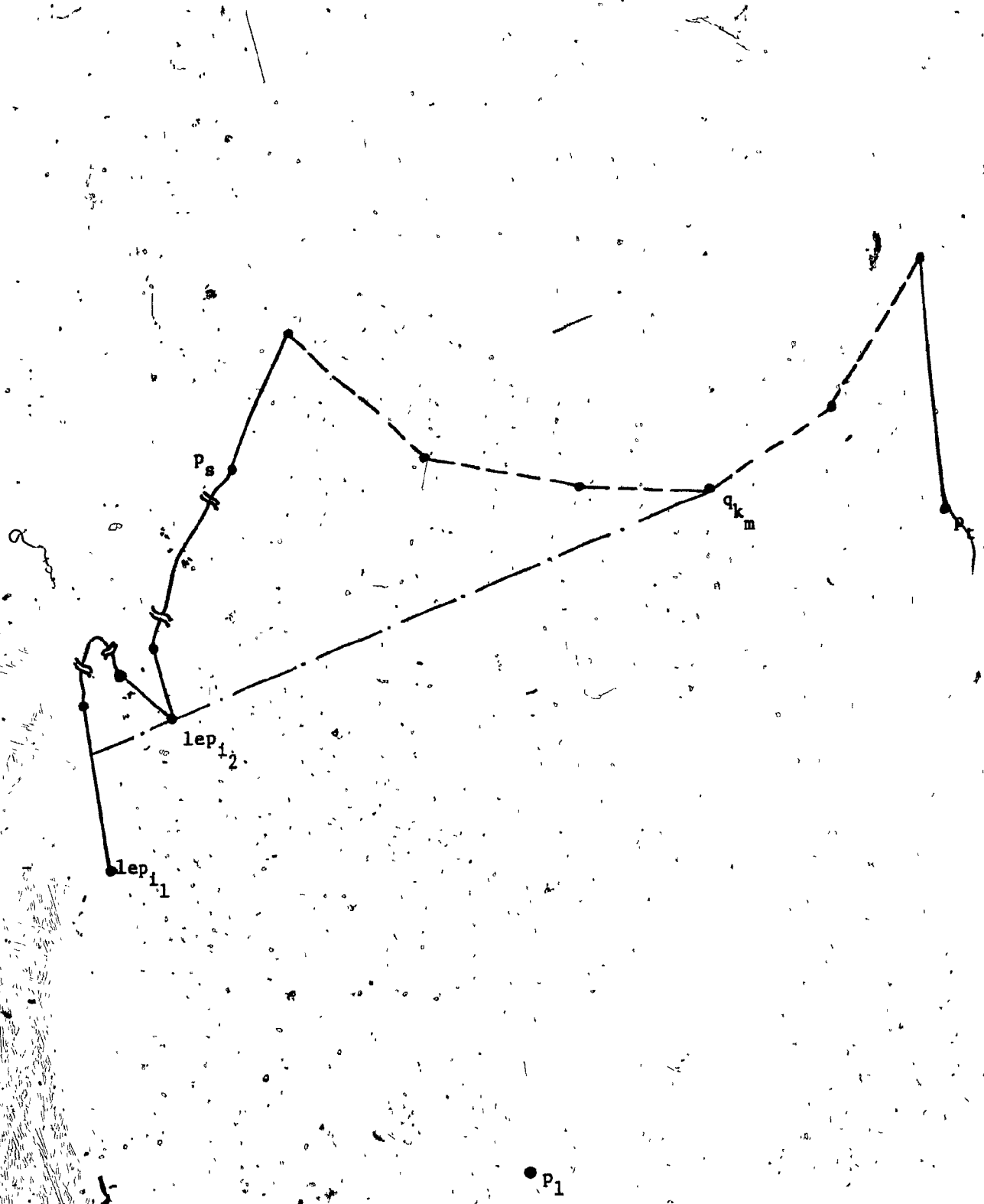


Figure 4.10



procedure REPORT-MAX-DIAG(G)

input: A graph G=(V,E) of order n with the vertices labeled such that the sequence (v_1, v_2, ..., v_n, v_1) is a hamiltonian cycle of G.

output: A list of the maximal diagonals of the graph G.

method:

procedure REPORT-AND-CHECK (G,LeftLabel,RightLabel);

local variables

The largest label k of a vertex in the sequence ($v_{LeftLabel}$, ..., $v_{RightLabel}$) connected to $v_{LeftLabel}$;

The induced subgraph G_1 on vertices of the sequence ($v_{LeftLabel+1}$, ..., v_k);

The induced subgraph G_2 on vertices of the sequence (v_k, ..., $v_{RightLabel}$);

if no vertex in G_1 except v_k is connected to a vertex in G_2 *then*

if G_1 is not empty *then*
REPORT-AND-CHECK (G_1, LeftLabel+1, k-1);
if G_2 is not empty *then*
REPORT-AND-CHECK (G_2, k, RightLabel);
else
terminate unsuccessfully "the graph cannot be embedded as the visibility graph of a convex fan"

end REPORT-AND-CHECK

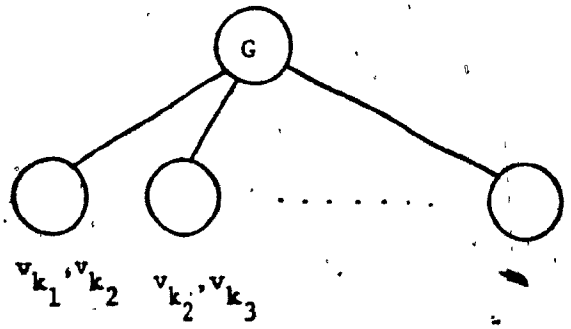
for i = 2 *to* n *do*
arrange the adjacent vertices with labels greater than i in increasing order of their labels;

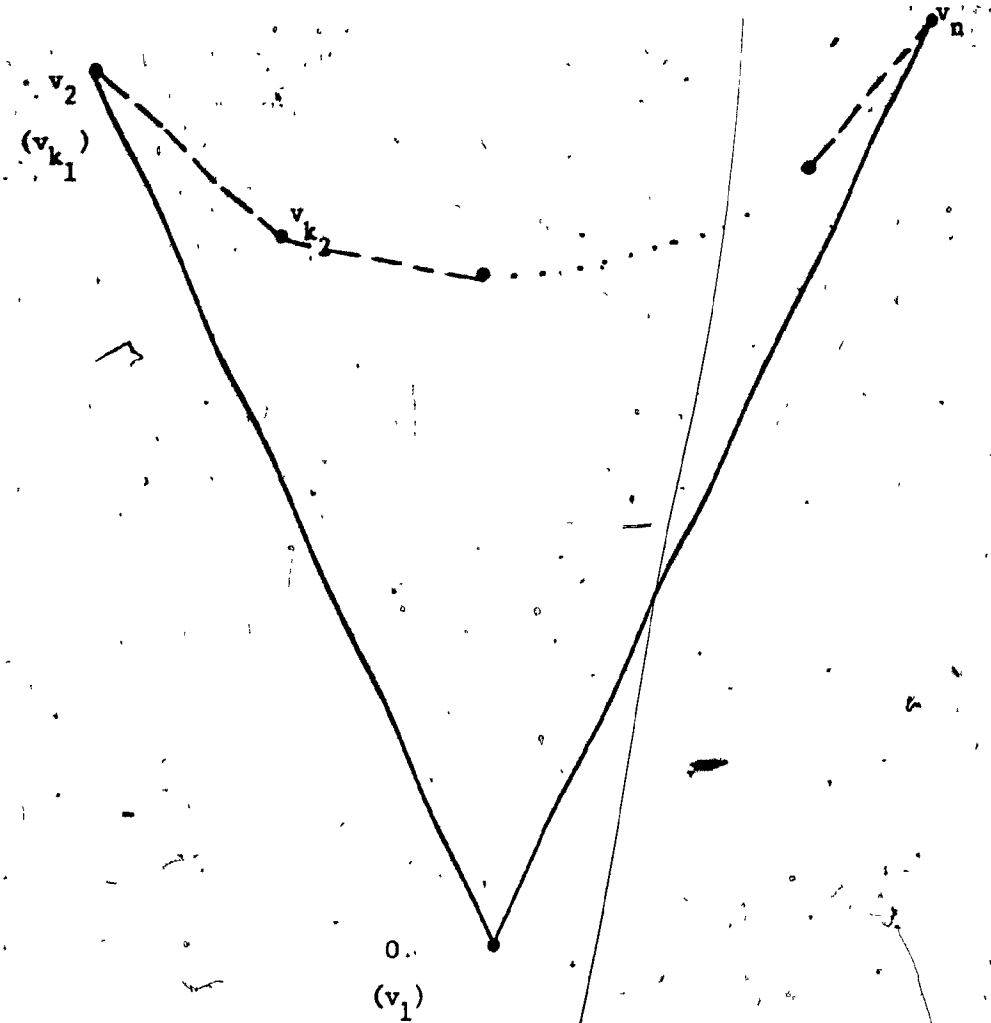
REPORT-AND-CHECK (G,2,n);

end REPORT-MAX-DIAG

Figure 4.11

Figure 4.12





procedure EMBED-A-CHAIN (C)

input: A pointer C to a node in the hierarchy.
output: An embedding of the sequence of end-points of the maximal diagonals stored as the sons of the node C in the hierarchy.
local variables.
The sequence Q = (q_{k_1}, q_{k_2},..., $q_{k'}$) of end-points of the maximal diagonals stored as the sons of the node C in the hierarchy.
The left end-point l of the maximal diagonal stored in the node C.
The right end-point r of the maximal diagonal stored in the node C.

method:
add l to the list LEP
add r to the list REP

for each end-point p in LEP(C) *do*
if p and end-points of Q do not satisfy lemmas 4.6-7 *then*
terminate unsuccessfully "the graph cannot be embedded as the visibility graph of a convex fan"
end for

for each end-point p in REP(C) *do*
if p and end-points of Q do not satisfy lemma 4.6-7 *then*
terminate unsuccessfully "the graph cannot be embedded as the visibility graph of a convex fan"
end for

for each vertex q_{k_i} in Q *do*
$m \leftarrow$ smallest index between vertices in LEP(C) such that lep_m and q_{k_i} are visible
if Q and lep_1, lep_2,.., lep_m do not satisfy lemma 4.8
or Q and lep_m ,...., lep_{m_1} do not satisfy lemma 4.9 *then*
terminate unsuccessfully "the graph cannot be embedded as the visibility graph of a convex fan"
end for

for each vertex q_{k_i} in Q *do*
$m \leftarrow$ smallest index between vertices in REP(C) such that rep_m and q_{k_i} are visible
if Q and rep_1, rep_2,..., rep_m do not satisfy lemma 4.8
or Q and rep_m , .., rep_{m_2} do not satisfy lemma 4.9 *then*
terminate unsuccessfully "the graph cannot be embedded as the visibility graph of a convex fan"
end for

Starting at $q_{k'}$, embed vertices of the sequence Q as a convex chain such that:
point O lies on its concave side.
a vertex is visible from previously mapped point if and only if they are connected in G.

for each son CSON of the node C in the hierarchy *do*
EMBED-A-CHAIN (CSON)
end for
remove l from the list LEP
remove r from the list REP

end EMBED-A-CHAIN

Figure 4.13

Figure 4.14

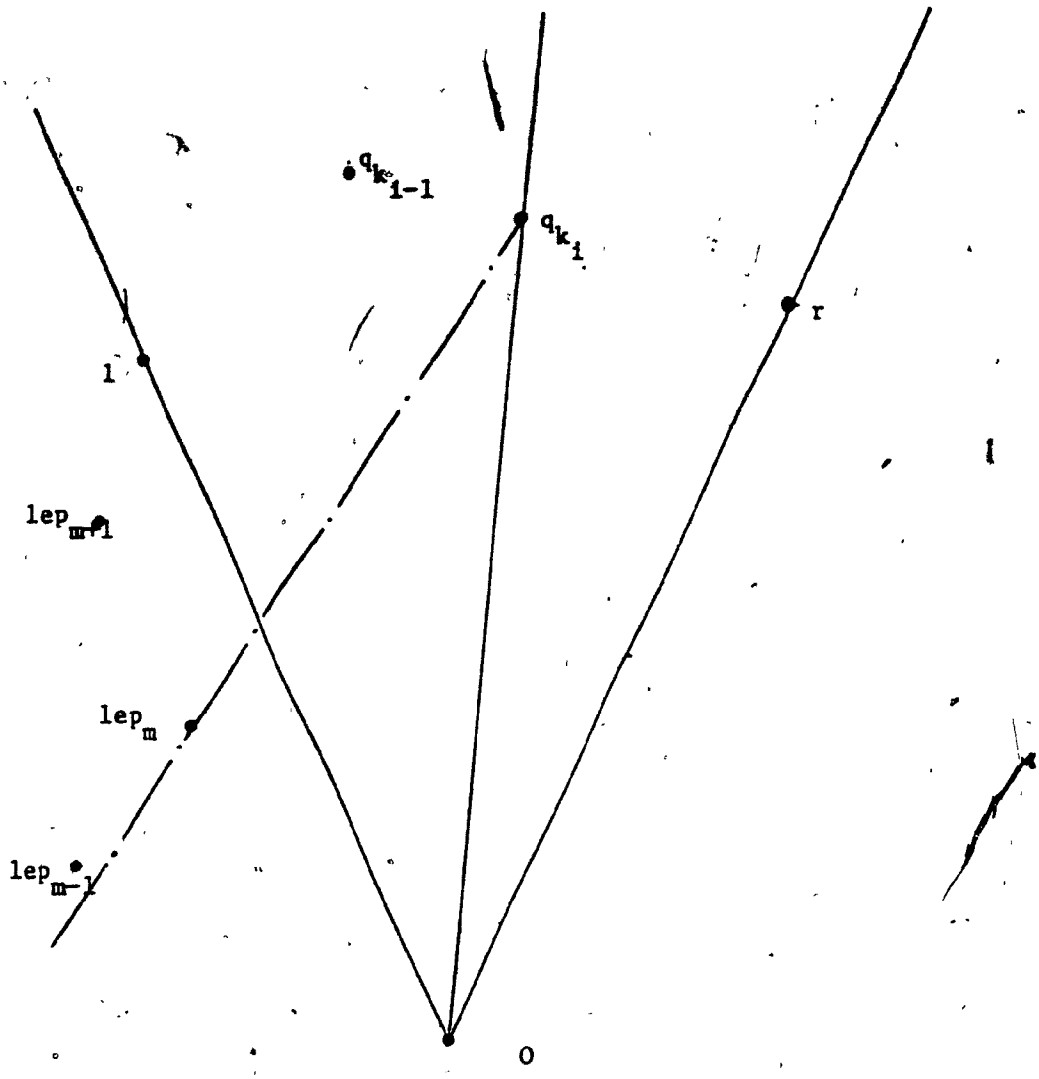
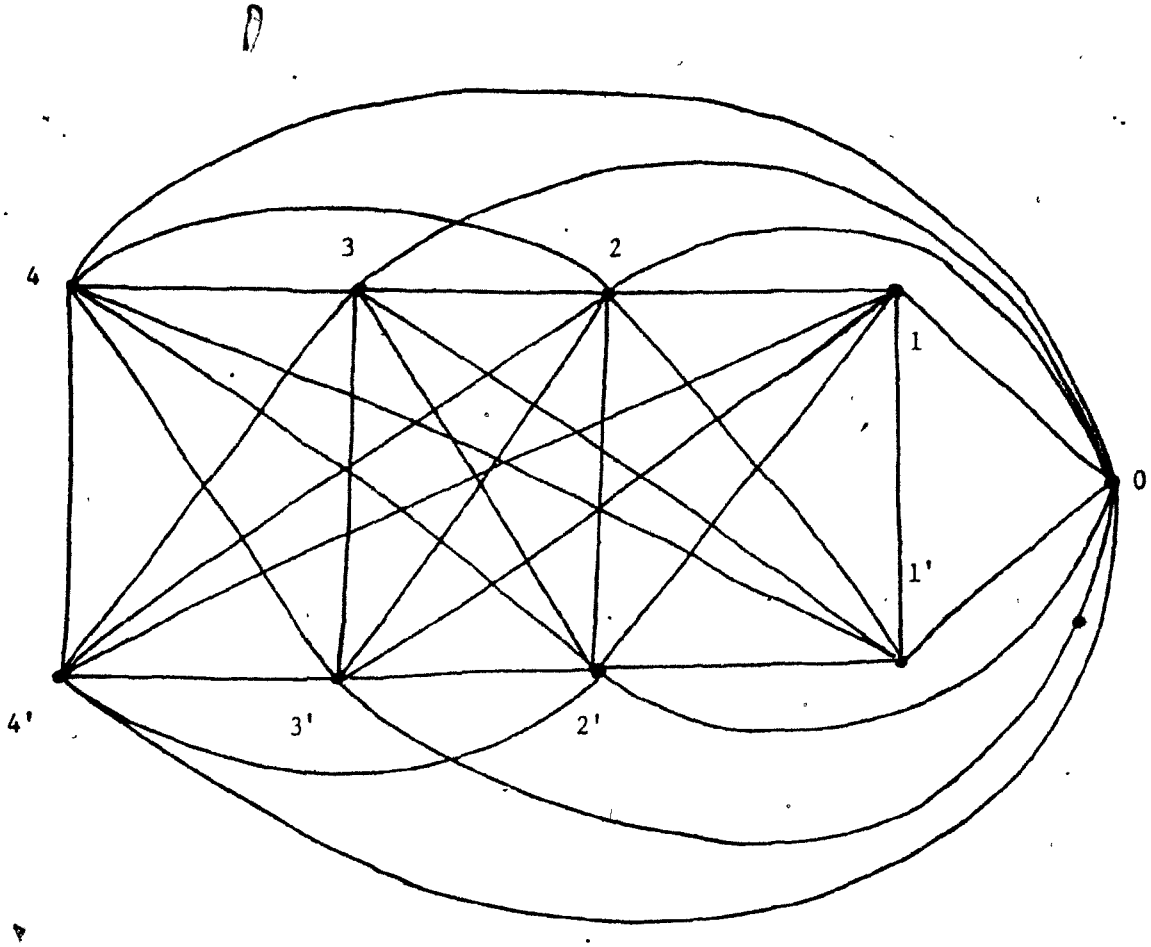
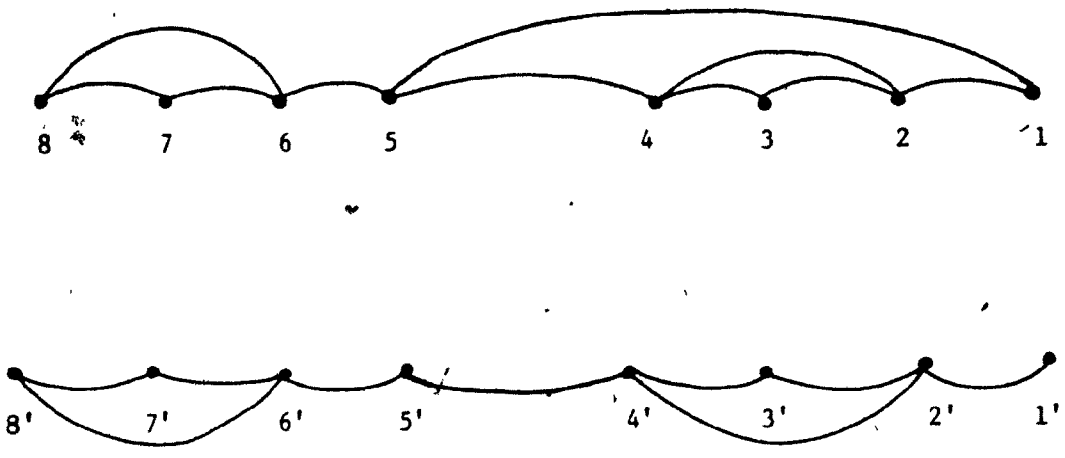


Figure 4.15-a



At least two choices 1,1' and 3,1'

Figure 4.15-b



N.B.

Every vertex in the top row is connected to all the vertices in the bottom row, and the vertex 0 is connected to the other sixteen vertices. However, those edges are not drawn.

At least four choices for the first two maximal diagonals

1,1' ; 5,5'
1,1' ; 5,7'
3,1 ; 5,5'
3,1' ; 5,7'

5. Routing inside Simple Polygons

5.1. Introduction

The *layout of a VLSI circuit* is the specification of the positions of its components and the paths of wires on the chip. Due to the overwhelming sizes of such designs, a bottom-up approach for solving the layout problem is usually adopted. At each level we are given predesigned components of the circuit. Each component has terminals located along its boundary, and each terminal is associated with a single network We have to place the components (the placement problem) and paths of wires to interconnect the different networks (the routing problem) on the chip, and form a component for the next level up

Szymanski [Szy] proved that the routing problem is NP-complete, and with Yannakakis [S&Y] showed that the problem remains NP-complete even in the special case of *channel routing* and each network contains only *two* terminals. However, polynomial time optimal algorithms have been presented for restricted versions of the routing problem. Pinter [Pin] presented polynomial algorithm for the restricted version of routing inside a T-shaped area, and with Leiserson [L&Pin] described polynomial algorithm for the *river routing* problem, a special case of the channel routing problem where networks are mapped into non-intersecting wires in the channel. Other polynomial optimal algorithms for the river routing problem have been described in [D&K&S&S&U,S&D,Tom]. On the other hand, many provably good polynomial approximation algorithms were designed for the channel routing problem (examples are [B&B&L,R&B&M,R&F]), and used as a part of a complete system for solving the routing problem

In this chapter we use the hierarchical description of simple polygons, based on a decomposition into rectangles, to design an efficient algorithm for solving a restricted version of the routing problem, where the terminals are located on the boundary of a closed region and wires are to be mapped inside the closed region. The hierarchical description is used to provide an ordering for processing the rectangles and to efficiently report the networks to be routed in

each rectangle, which are then processed using one of the existing heuristics.

5.2. Routing in a General Rectilinear Channel

Given a *rectilinear* simple polygon R (i.e., the edges are either vertical or horizontal) with n vertices, and a set of m networks $Net_1, Net_2, \ldots, Net_m$ connecting pairs of terminals (a_i, b_i) $i = 1, 2, \ldots, m$. Each terminal is described by its cartesian coordinates and the label of the edge of the polygon R which contains it. It is required to find a mapping of the networks into the interior of R as a collection of *horizontal* and/or *vertical* line segments where direction-changes occur only at end-points of the line segments. Horizontal (vertical) line segments should be separated by a minimum distance imposed by the fabrication process. However horizontal and vertical line segments of different networks can intersect at any interior point. An algorithm to compute such a mapping proceeds in four distinct steps.

Step 1 The objective of this step is to compute a set of non-intersecting *chords* which decomposes the interior of the rectilinear polygon R into a set of rectangles. We use the plane sweep paradigm (refer to [S&H] and [N&P] for a detailed description) to report the required chords. The algorithm sorts the vertices of the polygon in decreasing order with respect to their y-coordinates and then sweeps a horizontal line in the vertical direction from top to bottom. At each position of the sweeping line, the vertical edges intersecting it are kept in sorted order with respect to the x-axis. When the sweeping line encounters a horizontal edge, the algorithm reports the necessary chord(s), if any, required to decompose the interior of R into rectangles. After end-points of the chords are computed, the algorithm updates the list of vertical edges intersecting the sweeping plane.

Step 2 The algorithm uses the procedure BUILD-HIER-DESC, described in chapter 2, to compute the hierarchical description of R based on the chords computed in step 1.

Step 3 Starting at an arbitrary point on the boundary of R, the algorithm traverses the

boundary and computes the distances of the terminals and end points of the decomposition chords from the starting point When either a terminal or an end-point of a chord, say q, is encountered the algorithm increments the traversed distance and assigns the new value to the encountered point as a label, denoted by d_q This process maps the networks and the decomposition chords into a set of intervals on the *real line* which will be used in step 4 to report efficiently the networks to be routed in each rectangle Using the distance from the starting point as the search key, the terminals and end points are stored in a data structure *SBD* capable of performing *concatenable queue* operations [A&H&U] efficiently A 2-3 tree is an appropriate choice

Step 4 Each *leaf node* in the hierarchical representation of the polygon R, denoted by HIER-DESC(R), represents a rectangle such that exactly one of its boundaries is one of the chords reported in step 1 During the process of routing networks in such a rectangle, terminal positions of the networks that cross the only chord are *not* fixed

The algorithm selects a leaf node (i e , a node with degree one), extracts the networks to be routed in the corresponding rectangle from the data structure *SBD* , and attempts to embed the selected networks It then deletes the leaf node and the incident edge which results in a hierarchical representation of a smaller polygon, and updates the data structure *SBD* to delete the networks which have been completely embedded. The process is then repeated until all the networks are completely routed

Let v be a leaf node of the hierarchy and (q_1, q_2) be the chord corresponding to its incident edge such that d_{q_1} is less than d_{q_2} To report the networks to be routed inside the selected rectangle, the algorithm searches the data structure *SBD* and deletes the entries with distances in the interval between d_{q_1} and d_{q_2}. Using a 2-3 tree, such an operation can be performed in time proportional to $\log(|SBD|)$. Let $N = \{Net_1, Net_2, \ldots, Net_{m_1}\}$ be the set of networks to be routed completely in the selected rectangle, and let $N' = \{Net'_1, Net'_2, \ldots, Net'_{m_2}\}$ be the set of networks with only one terminal on the boundary of

the selected rectangle. Using the *crossing placer* [Riv, pp. 479], we compute a fixed terminal location on the chord (q_1, q_2) for each network in the set N', and then use a provably good approximate algorithm [R&F] for routing networks of $N \bigcup N'$ inside the rectangular channel. Finally, the newly fixed crossing points of the N' networks are inserted into the data structure *SBD*.

This algorithm provides an interactive tool, for a human designer, which is capable of efficiently performing the tedious and time consuming task of detailed routing in a rectilinear channel. Decomposing the routing area on the chip into rectilinear channels and selecting the networks to be routed inside each channel depend on inter-modules communications and is usually performed by the circuit designer.

6. Shortest Route inside Simple Polygons

6.1. Introduction

In this chapter we study the use of the decomposition strategy in reporting the shortest route between two points, s and t, inside a simple polygon P.

One method for solving the problem is to compute the required path one line-segment at-a-time. First we compute the visibility polygon of s, VP(P,s). If $t \in$ VP(P,s), then the shortest path is the line-segment joining s to t. Otherwise, we find the hidden region which contains the point t together with the corresponding *window* (i.e., the chord which separates the hidden region from the visibility polygon of s), and then compute the visibility polygon of the point $window_1$. Refer to Figure 6.1 for illustration. The later step is then repeated until the point t lies in the visible region. Since the visibility polygon of a point can be computed in linear time [E&A, Lee]. The shortest route can be computed in time $O(kn)$, where k is the number of line-segments in the path. However, k can be of $O(n)$ which results in an $O(n^2)$ algorithm.

In section 6.2 we present a linear time, and thus optimal, algorithm for computing the shortest route inside a monotone polygon. For the case of a simple polygon with n vertices, Chazelle [Cha2] and Lee and Preparata [L&P2] presented algorithms for computing the shortest route between two points. After computing a triangulation of the polygon in $O(n\log n)$ time, both algorithms checks *every* triangulation edge to report the edges which intersect the shortest route and then proceed to compute the shortest route. In section 6.3 we describe a different algorithm for computing the shortest route between two interior points. After computing the hierarchical description in $O(n\log n)$ time, the algorithm searches it to report the sequence of monotone components which contains the shortest route. Compared to the previous algorithms [Cha2,L&P2], the number of edges checked when searching the hierarchical description is much smaller than the number of edges in the triangulation of the polygon. The algorithm then uses the method developed in section 6.2 to compute the shortest route in time proportional to the number of vertices in the reported sequence of components. Applications of the shortest route

algorithms to an optimization problem and to a separability problem are discussed in section 6.4.

6.2. An Algorithm for Computing the Shortest Route inside a Monotone Polygon

Given a simple polygon M, of n vertices, that is monotone with respect to the y axis and two interior points s and t, it is required to compute the shortest route between them which lie completely inside M. The algorithm proceeds as follows:

The algorithm partitions the polygon M into two components M_1 and M_2, such that vertices of M_1 lie above s and vertices of M_2 lie below s, and computes the two corresponding extreme points $EXTREME_A$ and $EXTREME_B$. It then selects the component which contains the point t. Through the following description, we will assume that M_1 has been selected. The case of $t \in M_2$ is similar and its description is omitted. The algorithm then scans the vertices in order of their y coordinates maintaining a *planted tree* SPTree (i.e., a rooted tree in which the relative order of the subtrees of each node is part of its structure). The root of SPTree represents the point s, each node represents a scanned vertex of the polygon and its path to the root represents the shortest route from the point s to the corresponding vertex. The *sons* of each node in SPTree are stored in a *doubly linked list* in clockwise angular order around their *father* node. In addition the algorithm keeps a pointer T to the last reported *turning point* on the route from s to t, a pointer to the node in the subtree rooted at T which corresponds to the last considered vertex of the chain CN($EXTREME_B$, $EXTREME_A$), denoted by LastLeft, a pointer to the node in the subtree rooted at T which corresponds to the last considered vertex of the chain CN($EXTREME_A$, $EXTREME_B$), denoted by LastRight, a pointer to the node in the subtree rooted at T with the most clockwise angle and corresponds to a vertex of the chain CN($EXTREME_B$, $EXTREME_A$), denoted by LEFT, and a pointer to the node in the subtree rooted at T with the most counterclockwise angle and corresponds to a vertex of the chain CN($EXTREME_A$, $EXTREME_B$), denoted by RIGHT (an illustrative example is given in Figure 6.2).

When a new vertex in the chain CN($EXTREME_B$, $EXTREME_A$) is encountered, the algorithm checks its position relative to the directed line joining LastLeft to its *father*. If it lies to the right of the directed line, then the new vertex is added as the son of LastLeft and the pointer LastLeft is updated. Otherwise, the algorithm checks its position relative to the directed line joining RIGHT to T. If it lies to the right of the directed line, then the algorithm searches the path from LastLeft to T in the tree SPTree for the appropriate position to insert the new vertex and updates the pointers LEFT and LastLeft. Otherwise, it searches the path from RIGHT to LastRight in the tree SPTree for the appropriate position and updates the pointers T, LEFT and LastLeft. A vertex in the chain CN($EXTREME_A$, $EXTREME_B$) is processed in a similar fashion with the roles of LEFT and RIGHT, and LastLeft and LastRight interchanged. The scanning process continues until the point t is encountered. A complete description of this step, procedure GenSPTree, is shown in Figure 6.3.

The shortest route can now be reported by climbing the tree back to the root.

Analysis of the Algorithm

We first prove the correctness of the algorithm, and then discuss its time performance in theorem 6.4.

Lemma 6.1 Let Q = q_1, q_2,... be a simple chain that is monotone with respect to the y axis, as shown in Figure 6.4. The shortest route from q_1 to q_i which lies completely on the right (left) hand side of Q is the sequence of convex hull edges which lie on the right (left) hand side of Q, denoted by RCH(1,i) (LCH(1,i)).

Proof Straightforward.

Lemma 6.2 During the scanning of the monotone subpolygon M_1, the shortest route from the point T to each vertex of CN($EXTREME_B$, LastLeft) with y coordinates larger than that of the point T consists of vertices which belong to that chain.

Proof By construction, RIGHT is a pointer to the most counter-clockwise of the processed vertices of CN(LastRight, $EXTREME_B$). Therefore, the processed vertices of CN(LastRight, $EXTREME_B$) lie to the left of the directed line joining LEFT to the point T. The lemma then follows directly from this construction and lemma 6.1.

Q.E.D.

Lemma 6.3 Prior to the execution of each iteration in the procedure GenSPTree, the paths from the root of the tree SPTree to the other nodes represent the shortest routes from the point s to the vertices of the polygon considered, so far.

Proof Initially the tree SPTree contains one node, which corresponds to the point s, and the statement of the lemma is correct. Now assume that the lemma is satisfied prior to processing the vertex $m_i \in$ CN($EXTREME_B$, $EXTREME_A$), we will show that the procedure GenSPTree inserts a node in the tree SPTree such that the path from its root to the new node represents the shortest route from s to the vertex m_i inside the polygon M and correctly updates the pointers.

If m_i lies to the right of the directed line connecting RIGHT to T, then the shortest route from m_i to T consists of vertices of the chain CN($EXTREME_B$, $EXTREME_A$) as shown in lemma 6.2. If follows from the monotonicity of M and lemma 6.1, that shortest route from LastLeft to T forms a convex chain which lies completely below the vertex m_i. Therefore it suffices to search the convex chain forming the shortest route from LastLeft to T for its point of support corresponding to m_i as the appropriate father node of a new node corresponding to m_i. It is easy to see that the shortest route from m_i to T is also a convex chain and that no new turning point on the route to t need be created. Therefore the procedure does not change the pointer to T, last reported turning point in the route from s to t.

If m_i lies to the left of the directed line connecting RIGHT to T, then it suffices to search the convex chain forming the shortest route from RIGHT to LastRight for its point of support corresponding to m_i, say w, as the appropriate father node of a new node corresponding to m_i.

Again it is clear that the shortest route from m_i to T is also a convex chain and that the shortest route from s to t must pass through the vertex w. Therefore the procedure changes T to point to the vertex w before proceeding to process the next vertex.

The case of $m_i \in \text{CN}(EXTREME_A, EXTREME_B)$ is similar and its analysis is omitted.

Q.E.D.

Theorem 6.4 Procedure GenSPTree computes the shortest route from s to t correctly in linear running time.

Proof Correctness of the reported shortest route has been in lemmas 6.1-6.3. We analyze the performance of the procedure by computing the maximum number of times it processes each vertex of the polygon M.

Each vertex is added to the SPTree once. When GenSPTree processes a new vertex it either climbs SPTree from a leaf node (pointed to by LastLeft or LastRight) towards the current turning point (pointed to by T) or descends from T towards a leaf node until a node, say w, is found to be the appropriate father node of the new vertex. In the first case, nodes in the subtree rooted at w will not be examined by the procedure in the future iterations. In the second case only nodes in the subtree routed at w, which have not been examined in the current iteration, will be examined in future iterations. Therefore we conclude that the procedure runs in $O(n)$ time since each vertex is processed at most three times.

Q.E.D.

6.3. An Algorithm for Computing the Shortest Route inside a Simple Polygon

Given a simple polygon P of n vertices and two interior points s and t, we describe the use of a hierarchical description of P based on a decomposition into monotone components to develop an algorithm for computing the shortest internal path between two points in $O(n\log n)$ time. The algorithm proceeds as follows:

Step 1 Preprocess the polygon P by decomposing it into a set of components that are monotone with respect to a vertical line [L&P1], arranging the monotone chains for point location [L&P1], and computing the hierarchical description HIER_DESC(P) based on this decomposition

Step 2 For each of the points s and t the algorithm searches the monotone chains for the monotone component which contains it, a process which can be performed in $O(\log^2 n)$, and marks the component which contains each point. The algorithm then searches the hierarchy for the sequence of components which contains the shortest path from s to t and stores it in a *deque* data structure *COMPS*. Initially the *deque* is empty. Starting at the top of HIER_DESC(P), the algorithm checks whether the two marked components are the same as the component corresponding to the set difference between the current node and its sons, the current component, or lie in the sub-hierarchies of the current node. If the two components are the same as the current component, then the algorithm inserts it in *COMPS* and proceeds to step 3. If the two marked components lie in the same sub-hierarchy, then the algorithm descends to the corresponding son and repeats the process. Otherwise, it inserts the current component into *COMPS* at the appropriate end (i.e., such that the current component and the component at that end share a chord), descends to the corresponding son(s) and repeat the search process.

Step 3 Let $COMPS_1, COMPS_2, \ldots, COMPS_k$ be the monotone components reported in the previous step, where s lies on an edge of $COMPS_1$ and t lies on an edge of $COMPS_k$, and let *GATES* be the set of chords separating the components in *COMPS* such that $GATE_i$ separates the components $COMPS_i$ and $COMPS_{i+1}$.

The algorithm partitions $COMPS_1$ into two components M_1 and M_2, such that vertices of M_1 lie above s and vertices of M_2 lie below s, and computes the two corresponding extreme points $EXTREME_A$ and $EXTREME_B$. It then selects the part which contains $GATE_1$ as an edge, and scan its vertices in order of their y coordinates maintaining the shortest routes from s to the different vertices in a *planted tree* SPTree and the pointers T, LEFT, RIGHT, LastLeft

and LastRight as described in the previous section

When a new vertex, say u, is the end-point of a *GATE*, the algorithm first processes it as a vertex of the current component and inserts it in SPTree in the appropriate place. It then partitions the new component into NM_1 and NM_2 such that vertices of NM_1 lie above u and vertices of NM_2 lie below u and selects the sub-component which contains the gate to the following component as an edge. There are two cases to consider (refer to Figure 6.5 for an illustration)

(1) if the selected sub-component together with the current component form a monotone polygon, then the algorithm proceeds in scanning the vertices as previously described

(2) if the selected sub-component together with the current component do not form a monotone polygon, then the algorithm changes the pointer T to point to the vertex u and also updates the appropriate pointers It then scans the vertices of the selected sub-component as previously described

The scanning process continues until the point t is processed At this time, the shortest path can be reported by climbing the tree back to the root

We now state the main result of this chapter in theorem 6.5 , The proof follows the same steps of theorem 6.4, and is thus omitted.

Theorem 6.5 After an $O(n\log n)$ time preprocessing of the simple polygon P, the shortest route between two points inside a simple polygon can be computed in linear running time.

6.4. Applications

In this section we discuss the use of the shortest route procedures in developing efficient algorithms for an optimization problem and for other geometric problems.

A) A Control Problem of Bellman

Let x_i be the quantity of some item on hand at time i. Due to attrition, the amount on hand at time $i+1$ will be cx_i unless augmented or diminished by an amount y_i. The objective is to minimize $\sum y_i^2$ subject to the constraints that:

$$a_i \leq x_i \leq b_i \quad i = 0,1,\ldots,n \text{ with } a_0 = b_0,\ a_n = b_n$$

Dantzig [Dan] presented a geometric interpretation of the problem as follows (Refer to Figure 6.6):

map each lower constraint into the point $(c^{-2i},\ a_i\ c^{-i})$.

map each upper constraint into the point $(c^{-2i},\ b_i\ c^{-i})$.

place a "string" between the end points, $(c^0,\ x_0)$ and $(c^{-2n},\ x_n\ c^{-n})$, threading through the constraints points and "draw" tight.

Dantzig then proved that the values of x_i $i = 0,1,\ldots,n$ *uniquely* define the optimal solution and presented an $O(n^2)$ algorithm for computing those values. Since the tight position of the string represents the shortest route between the two end points which is threaded through the constraints points, we can use the procedure GenSPTree to compute it $O(n)$ time.

B) Movable Separability of Simple Polygons

In chapter 3 we described the use of the algorithm for computing the weak visibility polygon from an edge in developing an efficient solution for the *separability* problem. We now describe the use of the algorithm for computing the shortest route in developing a simpler

solution for the same problem. Toussaint [Tou2] showed that directions of monotonicity of the minimum-perimeter polygon enclosing one of the polygons and lying completely outside the other define all possible directions for separating the two polygons with a single translation.

To compute such a polygon, the following subproblem is encountered (refer to Figure 6.7(a)-(b) for illustration):

> Given two simple polygons, P and Q, such that Q lies completely in the interior of P. Compute the minimum-perimeter polygon which encloses Q, lies completely inside P and has the edge (p_i, p_{i+1}) on its boundary. (Note that some vertices may appear twice on the boundary of the resulting polygon.)

A solution which uses the shortest internal route algorithm proceeds as follows:

1) Find the vertex of Q, say q_k, furthest from the line passing through the edge (p_i, p_{i+1}), denoted by L
2) Find the closest intersection point of the half-line starting at q_k and perpendicular to L with the boundary of P, say X.
3) In the polygon $(q_k, X, \ldots, p_i, p_{i+1}, \ldots, X', q_k', \ldots, q_{k+1})$, compute the shortest internal route from q_k to p_i and from p_{i+1} to q_k'. The boundary of the required polygon is the union of the two routes together with the edge (p_i, p_{i+1}).

It is easy to see that the minimum-perimeter polygon is computed in $O((|P|+|Q|) \log (|P|+|Q|))$.

Figure 6.1

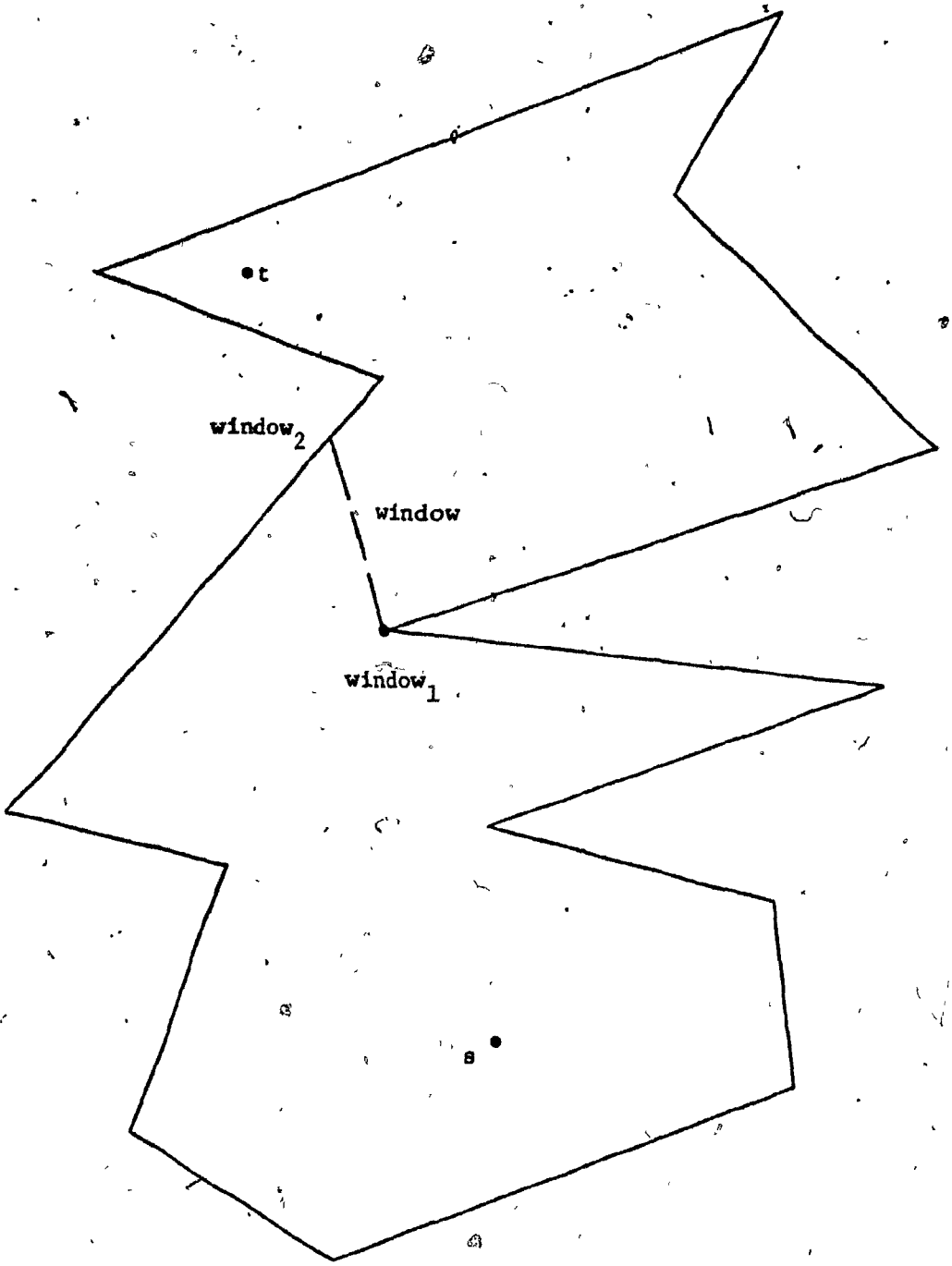
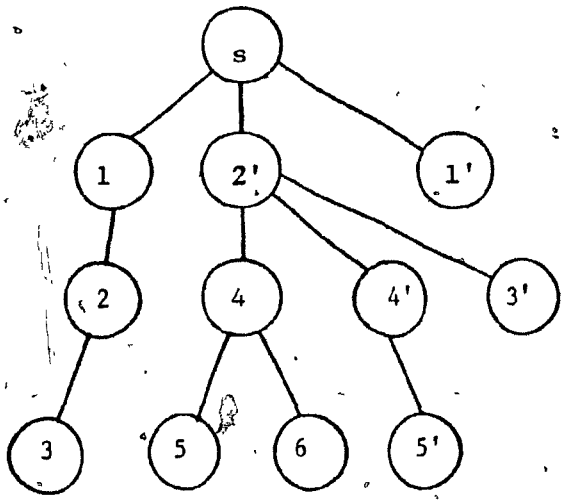
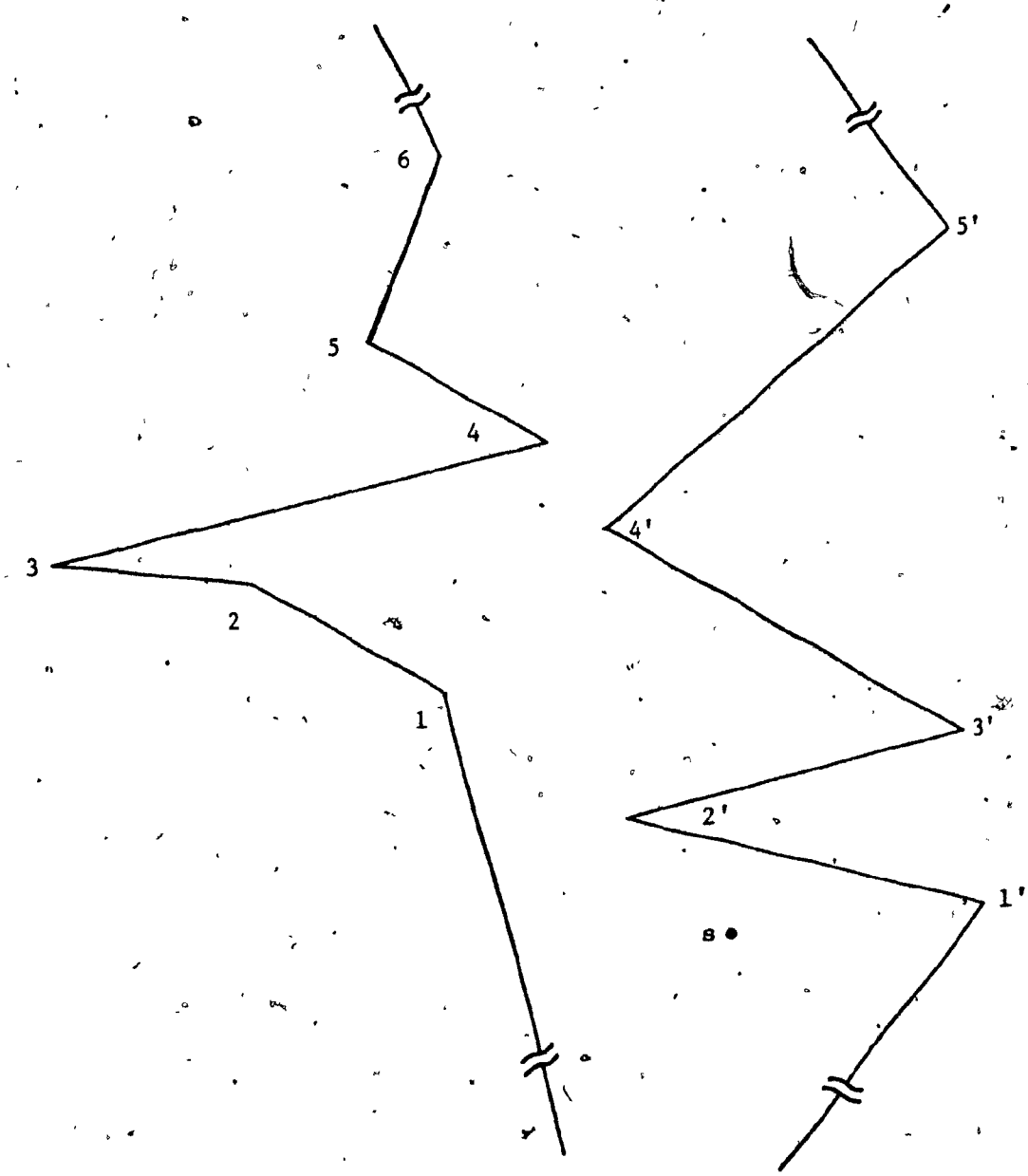


Figure 6.2



2': T
5': LastRight
4': RIGHT
6 : LastLeft
4 : LEFT



procedure GenSPTree (M,s,t)

input:
A polygon M that is monotone with respect to the y-axis, and two points, s and t, in M.

output:
A tree, SPTree, that contains the shortest paths from s to t and to the vertices of M.

method

Locate the two vertices with the maximum and minimum y-coordinates, $EXTREME_A$ and $EXTREME_B$ respectively;

Decompose M into the two subpolygons:
M_1 whose vertices have y-coordinates larger than that of s;
and M_2 whose vertices have y-coordinates less than that of s;

if $t \in M_1$ *then*

create a tree, SPTree, with s stored as its root;

cur-vertex ← the vertex of M_1 or the point t
with the smallest y-coordinate larger than that of s;

Repeat

case 1: cur-vertex is a vertex of the chain CN($EXTREME_B$,$EXTREME_A$)

case 1a: cur-vertex lies to the right of the line connecting LastLeft to its *father*

insert cur-vertex as son of LastLeft;

case 1b: cur-vertex lies to the right of the line connecting RIGHT to T

search the path from LastLeft to T in the tree SPTree for a node such that cur-vertex lies to the right of the line connecting the node to its *father*;
insert cur-vertex as son of the found node;

case 1c: cur-vertex lies to the left of the line connecting RIGHT to T

search the path from RIGHT to LastRight in the tree SPTree for a node such that cur-vertex lies to the left of the line connecting the node to its *father*;
insert cur-vertex as son of the found node;

case 2: cur-vertex is a vertex of the chain CN($EXTREME_A$, $EXTREME_B$)

case 2a: cur-vertex lies to the left of the line connecting LastRight to its *father*

insert cur-vertex as son of LastRight;

case 2b: cur-vertex lies to the left of the line connecting LEFT to T

search the path from LastRight to T in the tree SPTree for a node such that cur-vertex lies to the left of the line connecting the node to its *father*;
insert cur-vertex as son of the found node;

case 2c: cur-vertex lies to the right of the line connecting LEFT to T

search the path from LEFT to LastLeft in the tree SPTree for a node such that cur-vertex lies to the right of the line connecting the node to its *father*;
insert cur-vertex as son of the found node;

cur-vertex ← vertex of M_1 or the point t with the next y-coordinate;

Until cur-vertex = $EXTREME_A$

else { $t \in M_2$ }

Details of these steps are similar to those of the steps for processing M_1, and are thus omitted.

end GenSPTree

Figure 6-3

Figure 6.4

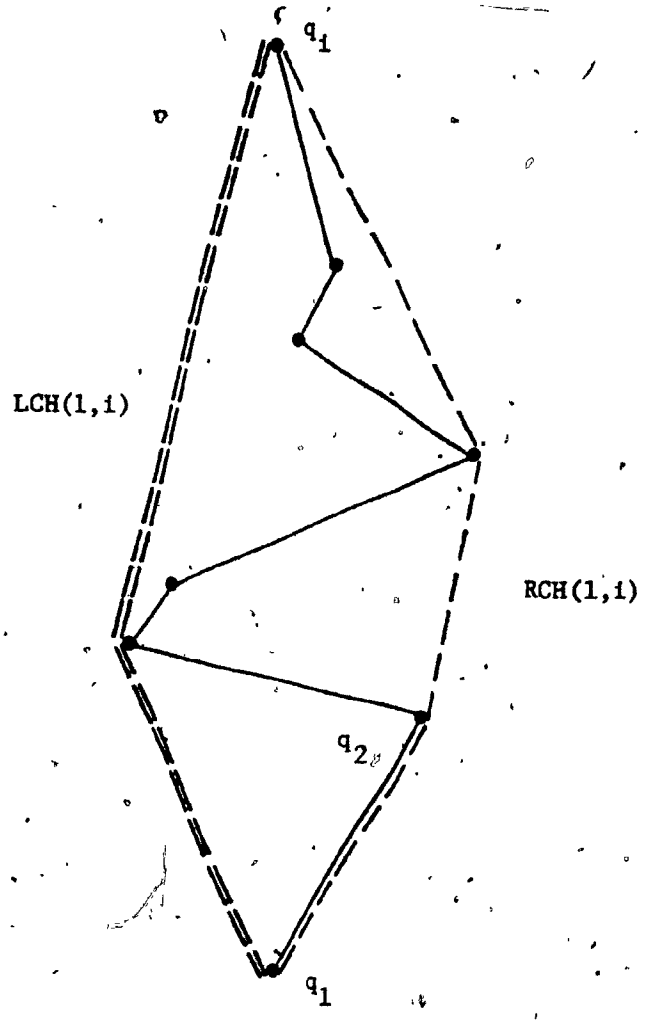
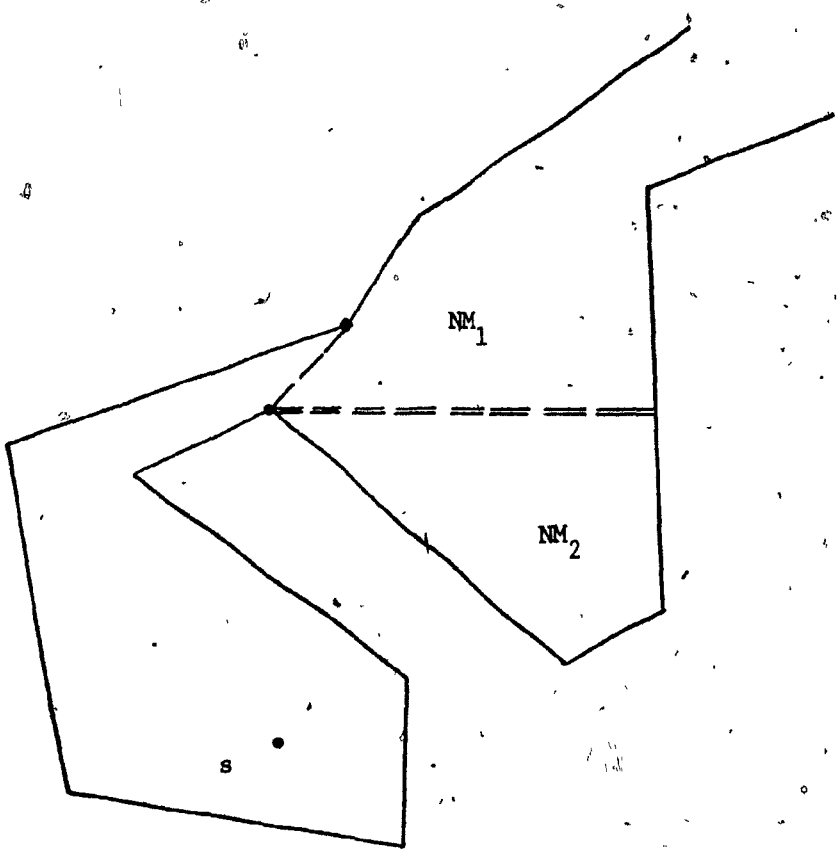
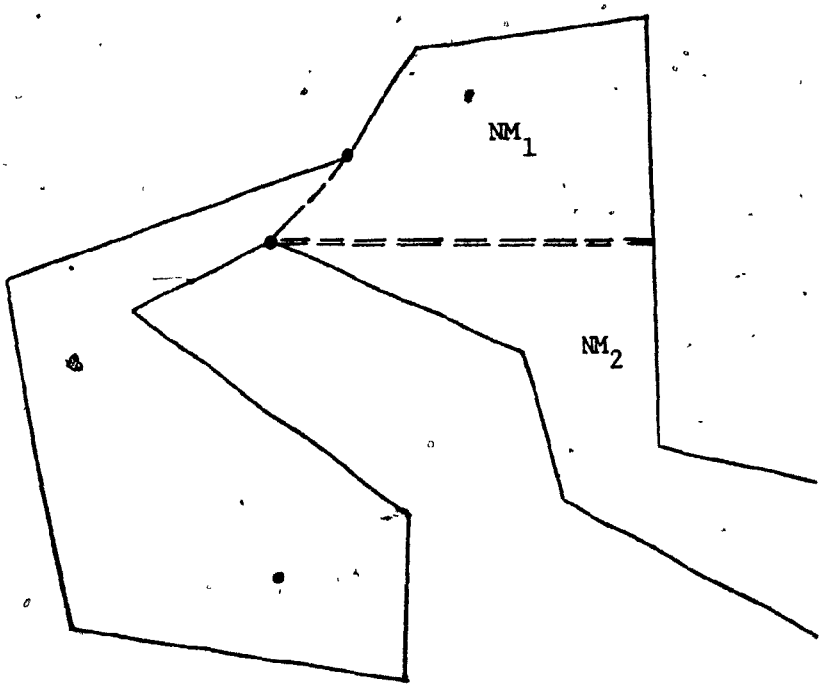


Figure 6.5



case(1)



case(2)

Figure 6.6

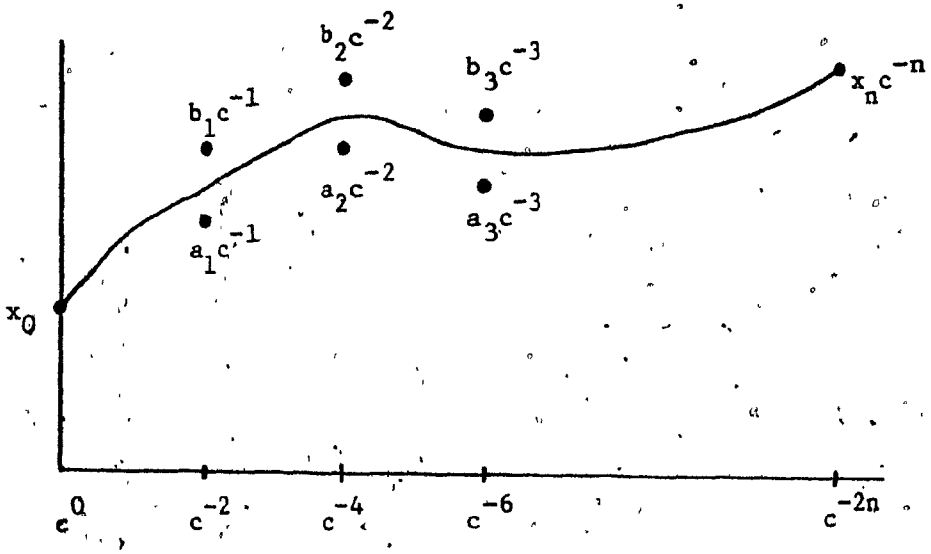
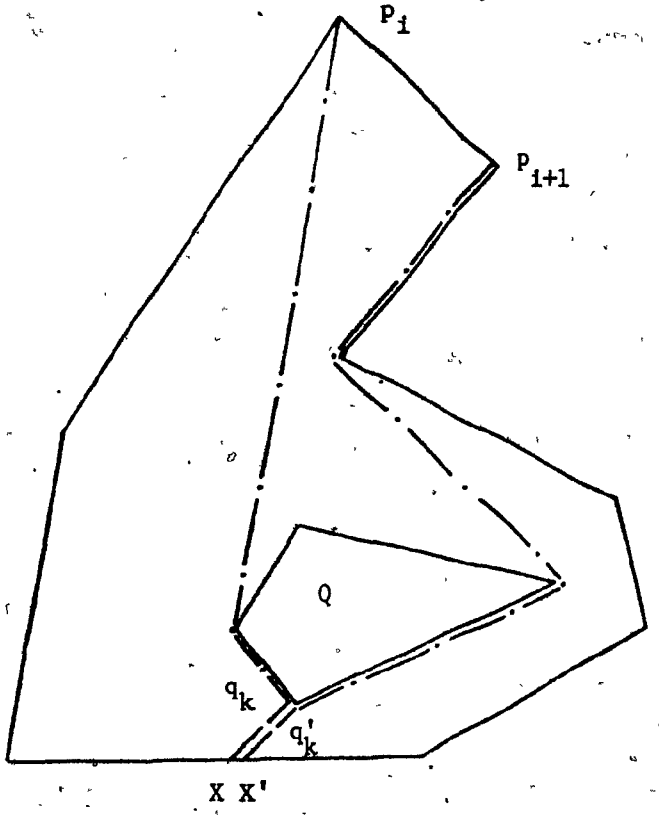
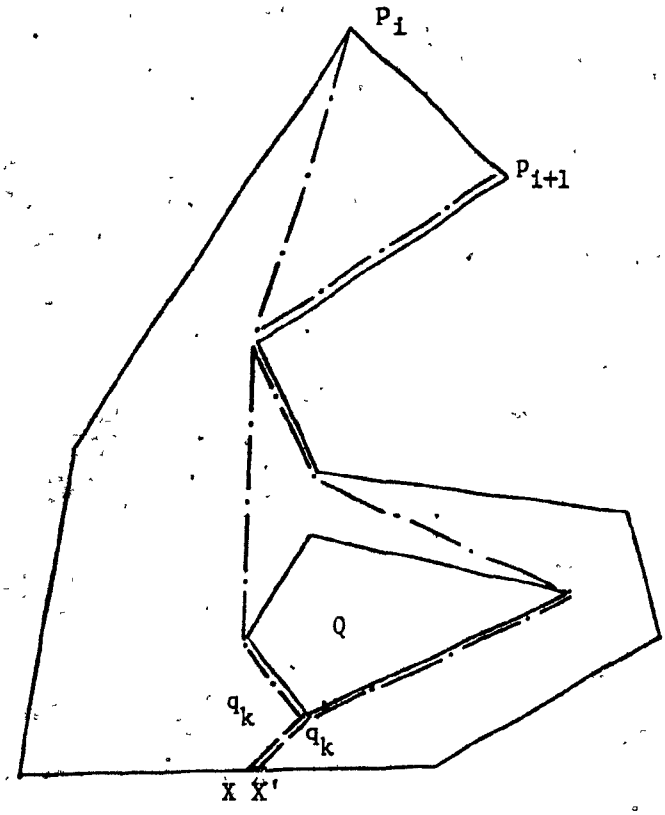


Figure 6.7





Bibliography

[A&H&U] A. V. Aho, J. E. Hopcroft and J. Ullman, *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, 1974.

[Asa] T. Asano, "Efficient algorithms for finding the visibility polygon for a polygonal region with holes," personal communication, 1984.

[Asa2] T. Asano, "An efficient algorithm for computing the reachability polygon from a point: Rectilinear Case," personal communication 1984.

[A&E] D. Avis and H. ElGindy, "A combinatorial approach to polygon similarity," *IEEE Trans. on Information Theory*, Vol. IT-29 (1983), pp. 148-150.

[A&T] D. Avis and G. T. Toussaint, "An optimal algorithm for determining the visibility of a polygon from an edge," *IEEE Trans. on Computers*, Vol.C-30 (1981), pp. 910-914.

[B&J&M] T. Beyer, W. Jones, and S. Mitchel, "Linear algorithms for isomorphism of maximal outerplanar graphs," *Journal of the Association for Computing Machinery*, Vol. 26, No. 4, October 1979, pp. 603-610.

[B&B&L] B. S. Baker, S. N. Bhatt, and F. T. Leighton, "An Approximation algorithm for Manhattan Routing," *Proceedings of the 15th Annual ACM Symposium on Theory of Computing*, 1983, pp. 477-486.

[Cha] B. Chazelle, *Computational Geometry and Convexity*. Ph.D. Dissertation, Technical Report No. CMU-CS-80-150, Dpeartment of Computer Science, Carnegie-Mellon University, July 1980.

[Cha2] B. Chazelle, "A theorem on polygon cutting with applications," *23th Annual IEEE Symposium on Foundations of Computer Science*, 1982, pp. 339-349.

[C&L] W.H. Chu and D.T. Lee, "The hidden line elimination problem revisited," *Proceedings of the International Conference on Advanced Automation*, Taipei, Taiwan, December 1983, pp. 469-480.

[Dan] G.B. Dantzig, "A control problem of Bellman," *Management Science*, Vol. 17, No. 9, May 1971, pp. 542-546.

[D&B] L.S. Davis and M.L. Benedikt, "Computational models of space: Isovists and Isovist fields," *Computer Graphics and Images Processing*, Vol.11, pp. 49-72.

[D&K&S&S&U] D. Dolev, K. Karplus, A. Siegel, A. Strong, and J.D Ullman, "Optimal algorithms for structured assembly," *VLSI Design*, March/April 1982, pp. 38-42.

[Don] B.R. Donald, "Hypothesizing channels through free-space in solving the findpath problem," Massachusetts Institute of Technology, Artificial Intelligence Laboratory, A.I.Memo No. 736, June 1983.

[E&A] H. ElGindy and D. Avis, "A linear algorithm for computing the visibility polygon from a

point," *Journal of Algorithms*, Vol.2 (1981), pp. 186-197.

[E&A&T] H. ElGindy, D. Avis, and G. T. Toussaint, "Applications of a two-dimensional hidden-line algorithm to other geometric problems," *Computing*, Vol.31 (1983), pp. 191-202.

[F&P] H-Y. F. Feng and T. Pavlidis, "Decomposition of polygons into simpler components: Feature generation for syntactic pattern recognition," *IEEE Trans. on Computers*, Vol. C-24 (1975), pp. 636-650.

[Fu] K.S. Fu, *Syntactic Methods in Pattern Recognition*. Academic Press, 1974.

[G&J&P&T] M.R. Garey, D.S. Johnson, F.P. Preparata, and R.E. Tarjan, "Triangulating a simple polygon," *Information Processing Letters*, Vol.7 (1978), pp. 175-179.

[G&S] S.K. Ghosh and R.K. Shyamasundar, "A linear time algorithm for computing the visibility polygon from an edge," *Proc of Seventh International Conference on Pattern Recognition*, 1984, pp 471-474.

[Har] F. Harary, *Graph Theory*. Readings, Addison-Wesley, 1969.

[High] D.W. Hightower, "A solution to line-routing problems on the continuous plane," *Proc. of Design Automation Workshop*, 1969, pp. 1-24.

[Joh] D. S. Johnson, "NP-Completeness Column: An ongoing guide," Reading, *Journal of Algorithms*, Vol.3 (1982), No.4, pp. 381-395.

[Kei] J.M. Keil, *Decomposing Polygons into Simpler Components*. Ph.D. Dissertation, Technical Report No. 163/83, Department of Computer Science, University of Toronto, 1983.

[Kir] D. Kirkpatrick, "Optimal search in planar subdivisions," *SIAM J. Computing*, Vol.12 (February 1983), pp. 28-35.

[Lee] D.T. Lee, "Visibility of a simple polygon," *Computer Vision, Graphics and Image Processing*, Vol.22 (1983), pp. 207-221.

[L&L] D.T. Lee and A. Lin, "Computing visibility polygon from an edge," personal communication, 1984.

[L&P1] D.T Lee and Franco P. Preparata, "Location of a point in a planar subdivision and its applications," *SIAM J. Computing*, Vol.6 (1977), pp. 594-606.

[L&P2] D.T. Lee and Franco P. Preparata, "Euclidean shortest paths in the presence of rectilinear barriers," *Network*, Vol.14 (1984), pp. 393-410.

[L&Y] D.T. Lee and C.C. Yang, "Location of multiple points in planar subdivision," *Information Processing Letters*, Vol.9 (November 1979), pp. 190-193.

[L&Pin] C.E. Leiserson and R.Y. Pinter, "Optimal placement for river routing," in *VLSI Systems and Computations*, eds. H.T. Kung, R. Sproull, and G. Steele, Computer Science Press, 1981, pp. 126-142.

[L&Pap] W. Lipski Jr. and C.H. Papadimitriou, "A fast algorithm for testing for safety and detecting deadlocks in locked transaction systems," *J. of Algorithms*, Vol.2 (1981), pp.

211-226.

[L&W] T. Lozano-Perez ad M. Wesley, "An algorithm for plannig collision-free paths among polyhedral obstacles," *Commuications of the ACM*, Vol.22 (1979), pp. 560-570.

[Mar] K. Maruyama, "A study of visual shape perception," Technical Report No. UIUCDCS-R-72-533, University of Illinois, Urbana, 1972.

[M&A] D. McCallum and D. Avis, "A linear algorithm for finding the convex hull of a simple polygon," *Information Processing Letters*, Vol.9 (1979), pp. 201-207.

[M&C] C. Mead and L. Conway, *Introduction to VLSI Systems*. Addison-Wesley, 1980

[Mei] G.H. Meisters, "Polygons have ears," *American Mathematics Monthly*, June-July 1975, pp. 648-651.

[N&S] W.M. Newman and R.F. Sproull, *Principles of Interactive Computer Graphics*. McGraw-Hill, New York, 1979.

[N&P] J. Nievergelt and F.F. Preparata, "Plane-sweep algorithms for intersecting geometric figures," *Communications of the ACM*, Vol.25, No.10, 1982, pp. 739-747.

[ORou] J. O'Rourke, "The signature of a curve and its applications to pattern recognition," Technical Report JHU-EECS-82.9, Johns Hopkins Uiversity, 1982.

[O&C&O&N] J O'Rourke, C. Chien, T. Olson, and D. Naddor, "A new linear algorithm for intersecting convex polygons," *Computer Graphics and Image Processing*, Vol. 19 (1982), pp. 384-391.

[Pav] T. Pavlidis, *Structural Pattern Recognition*. Springer-Verlag, New Yotk, 1977.

[Pin] R.Y. Pinter, "Optimal routing in rectilinear channels," in *VLSI Systems and Computations*, eds. H.T Kung, R. Sproull, and G. Steele, Computer Science Press, 1981, pp. 160-177.

[P&S] F. P. Preparata and K. J. Supowit, "Testing a simple polygon for monotonicity," *Information Processing Letters*, Vol.12 (1981), pp. 161-164.

[Rap] D. Rapapport, "A linear algorithm for eliminating hidden lines from a polygonal cylinder," Technical Report No. SOCS-83-26, School of Computer Science, McGill University, December 1983.

[Riv] R.L. Rivest, "The PI (Placement and Interconnect) system," *19th Design Automation Conference*, June 1982, pp. 475-481.

[R&B&M] R.L. Rivest, A. Baratz, and G.L. Miller, "Provably good channel routing algorithms," *Proceedings CMU Conference on VLSI Systems and Computations*, 1981, pp. 153-159.

[R&F] R.L. Rivest and C M Fiduccia, "A greedy channel router," *Computer-aided Design*, Vol.15, No.3, May 1983, pp. 135-140.

[S&T] J.-R. Sack and G.T. Toussaint, "Translating polygons in the plane," Technical Report

SCS-TR-47, School of Computer Science, Carleton University, March 1984.

[Sha] M.I. Shamos, *Computational Geometry.* Ph.D. Dissertation, Yale University, 1978.

[S&H] M.I. Shamos and D. Hoey, "Geometric intersection problems," *Seventh Annual IEEE Symposium on Foundations of Computer Science*, October 1976, pp. 208-215.

[Shap&H] L.G Shapiro and R.M. Haralick, "Decomposition of two-dimensional shapes by graph-theoretic clustering," *IEEE Trans. on Pattern Recognition and Machine Intelligence*. Vol. PAMI-1 (1979), pp. 10-20.

[S&D] A. Siegel and D. Dolev, "The separation for general single-layer wiring barriers," in *VLSI Systems and Computations*, eds. H.T. Kung, R. Sproull, and G. Steele, Computer Science Press, 1981, pp 143-152.

[Szy] T. Szymanski, "Dogleg channel Routing is NP-Complete," Unpublished Manuscript, 1981.

[S&Y] T. Szymanski and M. Yannakakis, Personal Communication to [Joh], 1982.

[Tom] M. Tompa, "An optimal solution to the wire-routing problem," *Proceedings of the 12th Annual ACM Symposium on Theory of Computing*, 1980, pp. 161-176.

[Tou] G.T. Toussaint, "Pattern recognition and geometrical complexity," *Proc. 5th Int. Conf. on Pattern Recognition*, 1980, pp 1324-1347.

[Tou2] G.T. Toussaint, "Shortest path solves translation separability of polygons," personal communication 1985

[T&E] G.T. Toussaint and H ElGindy, "Separation of two monotone polygons in linear time," *Robotica*, Vol.2 (1984), pp. 215-220.

[T&S] G.T. Toussaint and J.-R. Sack, "Some new results on moving polygons in the plane," *Proc. Robotic Intelligence and Productivity Conference*, Detroit (1983), pp. 158-163.

[Y&Pap&K] M. Yannakakis, C.H. Papadimitriou and H.T. Kung, "Locking policies: Safety and freedom from deadlock," *Proceeding of the Foundations of Computer Science*, 1979, pp. 286-297.

[Yao] F.F. Yao, "On the priority approach to hidden surface algorithms," *Proc. 21st Annual Symposium on Foundations of Computer Science*, October 1980, pp.301-307.