Rectilinear Computational Geometry

by

Jörg-Rüdiger Sack

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

School of Computer Science
McGill University
Montréal, Quebec
May, 1984

Abstract

In this thesis it is demonstrated that the structure of rectilinear polygons can be exploited to solve a variety of geometric problems efficiently. These problems include.

(1) recognizing polygonal properties, such as star-shapedness, monotonicity, and edge-visibility,

(2) removing hidden lines,

(3) constructing the rectilinear convex hull,

(4) decomposing rectilinear polygons into simpler components, and

(5) placing guards in rectilinear polygons.

A new tool for computational geometry is introduced which extracts information about the winding properties of rectilinear polygons. Employing this tool as a preprocessing step, efficient and conceptually clear algorithms for the above problems have been designed.

Résumé

Dans cette thèse nous démontrons que la structure des polygones rectilignes permet de résoudre toute une gamme de problèmes géométriques associés à ces polygones. En particulier, nous étudierons:

(1) la vérification de propriétés telles la configuration en étoile, la monotonicité et la visibilité d'une arête

(2) la suppression des lignes cachées

(3) la construction d'une région convexe et rectiligne qui contient le polygone choisi

(4) la décomposition en corps simples

(5) l'insertion de "surveillants" a l'intérieur d'un polygone.

Nous présentons un outil géométrique qui accumule des renseignements au sujet des propriétés "circulaires" des polygones rectilignes. Plusieurs algorithmes clairs et efficaces ont été développés en utilisant cet outil comme étape initiale.

Contents

Acknowledgements

I am indebted to my thesis adviser Godfried T. Toussaint, for his encouragement when I needed it, for his interest in my ideas, for many discussions related to this thesis and for getting me started in computational geometry in the first place.

I am also grateful to the Computational Geometry Group at McGill University, in particular to David Avis for constructive criticism and discussions about algorithm design. I would particularly like to thank Thomas Strothotte for valuable comments and time spent during the final phase of the document preparation.

This work has been financially supported by Social Sciences and Humanities Research Council of Canada, McDonald Stewart Biomedical Image Processing Laboratory, World University Service of Canada, Natural Sciences and Engineering Research Council of Canada (from G.T. Toussaint's grant No. A-9293), McGill University and the German Academic Exchange Service (DAAD).

Meinen Eltern Gewidmet

Chapter 1
Introduction

1.1. Computational Geometry, some Terminology and Methodology

Geometry is a subject that has inspired Man for thousands of years. It has played an integral role in the origin of mathematics and other related sciences. Geometrical problems arise in such areas as computer graphics, pattern recognition, image analysis, robotics and VLSI. Consequently, geometrical algorithms have been developed in these fields. As a discipline of its own computational geometry did not become recognized until after the dissertation of M. Shamos in 1978 [Sh77]. Shamos showed that the available knowledge in classical geometry does not always provide us with the right "ammunition" to solve certain problems efficiently. The aim of computational geometry is to design efficient and possibly optimal algorithms for solving geometrical problems.

The efficiency of a solution for a given geometrical problem is often dependent on the specific nature of the objects involved. Knowledge of the type of objects is frequently available when a geometrical problem arises in a particular environment. A first distinction is whether the objects are given as sets of points or as polygons. A more detailed analysis might reveal structural knowledge to determine into which class of polygons, if any, the objects fall. Often algorithms for structured polygons are easier to write, thus easier to prove correct, and exhibit lower space/time complexities. We illustrate these points by first considering a problem which is harder for a set of points than for a polygon. We then demonstrate that it might be easier to design algorithms for certain restricted classes of polygons rather than for arbitrary *simple*, i.e. not self-

intersecting polygons.

Consider the task of determining the convex hull of an arbitrary set of points versus that of a simple polygon connecting these points. A polygon is *convex* if all its internal angles are less than 180°. The *convex hull* of a set points is defined as the minimum area convex polygon enclosing all points, (see Figure 1.1). The vertices of the convex hull are points of the set. The convex hull of a simple polygon is defined analogously. The convex hull of a set of n points can be determined in O(n log n) time and this is optimal [Sh77, Ya79, Gr72]. Kirkpatrick and Seidel [KiS83] have recently shown that the convex hull of a set of n points can be determined in O(n log h) time, where h is the number of vertices on the convex hull. However, linear time algorithms for computing the convex hull of a simple polygon exist [McA79, Le83a, GrY83, BhE81]. To obtain these bounds, the implicit order of the vertices along the boundary of the polygon has been exploited.

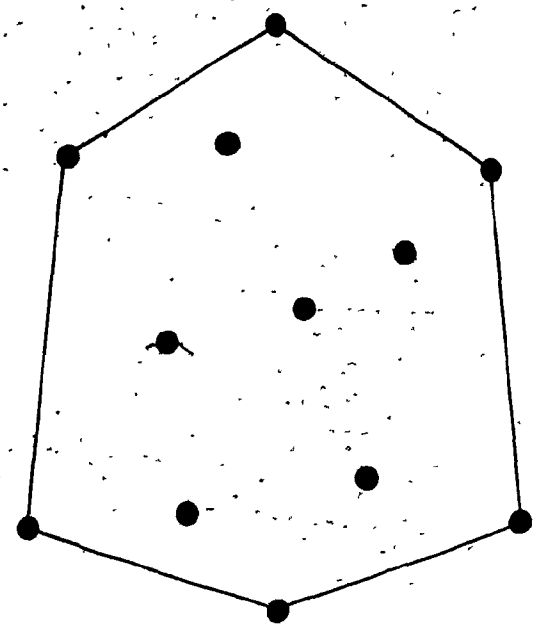


Figure 1.1

The convex hull of a set of points.

If not otherwise mentioned, we will assume all objects considered in this thesis are simple polygons. Furthermore, we specify a *polygon* P by its vertices $(p_1,...,p_n)$ listed in clockwise order. Each vertex is specified by its Cartesian coordinates. An edge e_i of P is the line-segment joining p_{i-1} to p_i (we identify p_0 with p_n and use modulo operations on all index operations). Polygons are assumed to be in standard form, i.e. no three consecutive vertices are collinear and e_1 is the edge with maximum y-value.

The Jordan Curve Theorem for polygons states that a simple polygon partitions the plane into two disjoint regions, the *interior* and the *exterior*, which are separated by the boundary of the polygon. The *inside* of the polygon is the interior together with its boundary. Similarly we define the *outside*.

Before proceeding we define several structural properties of polygons, illustrated in Figure 1.2.

A *polygonal chain* $C_{i,j}$ is a sequence of consecutive vertices $p_i,...,p_j$ of P. A polygonal chain is *monotone* with respect to a line ℓ if the projections of p_k, k=i,...,j on ℓ are ordered in exactly the same way as the vertices in $C_{i,j}$.

A polygon is *monotone* if there exists a line ℓ s.t. the boundary of P can be partitioned into two chains $C_{i,j}$ and $C_{j,i}$ each of which is monotone with respect to ℓ.

A point x in P *sees* a point y in P, if the open line segment joining x and y lies inside P. In case that a vertex p_i sees a vertex p_j, the line segment p_ip_j, if inserted into P, is called (internal) *diagonal* of P. In case that the open line segment p_ip_j lies completely in the exterior of P, we refer to p_ip_j as an *external diagonal* of P.

A polygon P is *star-shaped* if there exists at least one point x inside P such that x can see the entire polygon. The set of all points x from which the

entire polygon is visible is called the *kernel*.

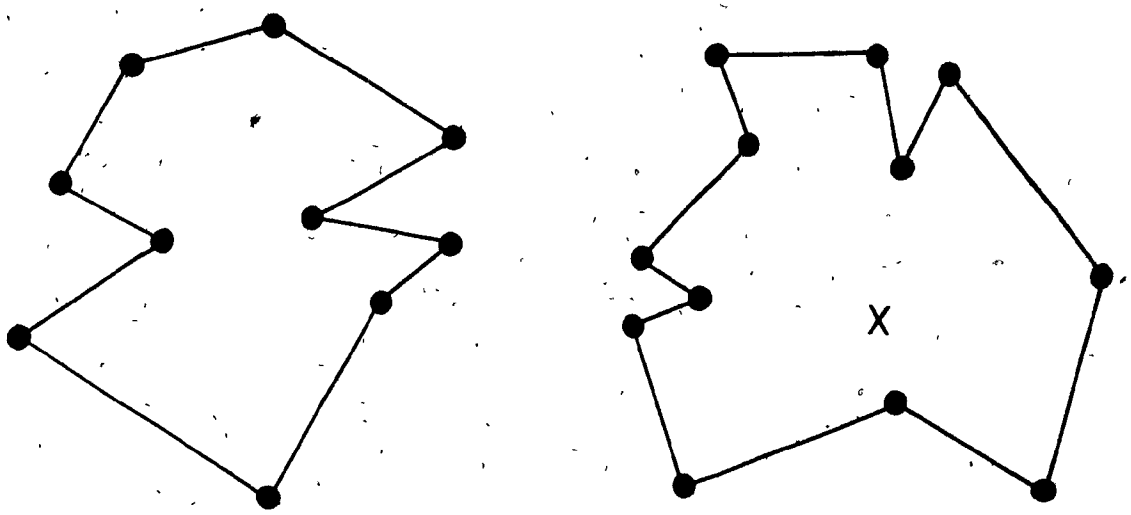
A polygon is *weakly visible* from an edge pq of P if for every point x in P there exists a point y on pq such that y sees x. A polygon is *edge-visible* if there exists an edge from which the polygon is weakly visible.

A polygon P is *rectilinearly convex* if every pair of points in P having the same y-coordinate or x-coordinate is visible. It should be clear that every convex polygon is rectilinearly convex but that the converse is not necessarily true.

We are given the task of solving a certain geometrical problem involving polygons. If the polygons have specific structural properties, it may often be possible to exploit these to obtain a clearer algorithm design or to gain efficiency. We now illustrate this by considering the *triangulation problem*; where the task is to insert diagonals into a given n-vertex polygon P such that P is partitioned into triangles with non-intersecting interiors. In linear-time one can triangulate star-shaped, monotone, edge-visible and convex polygons, (see the algorithms described in [GJPT78, ScL80, ToA82, To83]). In contrast to this, any simple polygon can be triangulated in O(n log n) time [GJPT78, Ch82, Chl83]. However, this has not yet been shown to be optimal, i.e. no non-trivial lower bound on this problem exists. Other examples of problems for which fast algorithms exist for restricted polygon classes include point inclusion in convex or star-shaped polygons [Sh77], medial axis of a convex polygon [Pr77], and intersections between convex or star-shaped polygons [Ch80, MF82].

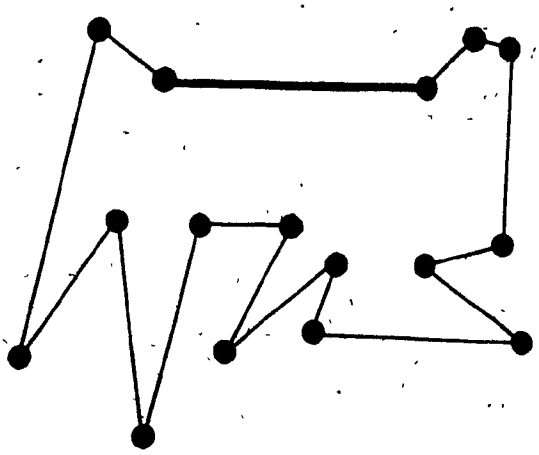
Note that all complexity results stated in this thesis are worst-case analyses with the underlying machine model of a real RAM, as described in [AHU74]. Unless otherwise stated, all complexities stated in this thesis are functions in the number of vertices of the polygons under consideration.

Imagine now an environment in which little or no knowledge about the specific

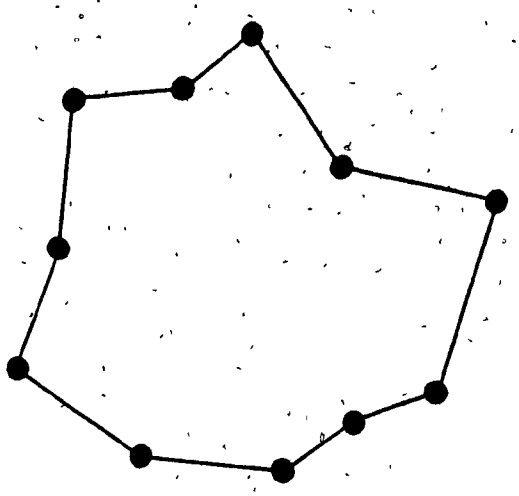


(a)
A monotone polygon

(b)
A star-shaped polygon



(c)
An edge-visible polygon



(d)
A rectilinearly convex polygon

Figure 1.2

Various types of polygons.

structure of the input to the geometric task is available. Even in under these circumstances, we may still be able to use some of the efficient algorithms which work correctly only on restricted classes of polygons. To achieve this, the usual strategy is to decompose the polygon into simple components, solve the problem on each component using a specialized algorithm, and then combine the partial solutions [GJPT78, ScL80, MF82, AT81]. This strategy is a primary motivation for the development of efficient *decomposition techniques*.

Polygons are typically decomposed into convex, star-shaped, monotone, or edge-visible polygons. We say that two polygons are *non-overlapping* if their interiors are non-intersecting. A decomposition is called a *partitioning* if the object is decomposed into non-overlapping pieces. If overlapping pieces are allowed, the decomposition is called a *covering*. Algorithms to partition arbitrary simple polygons into convex polygons [FeP75, ChD79, Ch82], star-shaped polygons [AT81], monotone polygons [GJPT78], triangles [GJPT78, Ch183] and trapezoids [AA83] have appeared in the literature. Given an n-vertex simple polygon, these decomposition techniques typically exhibit a worst-case run-time of O(n log n). For a survey of polygon decomposition techniques see [To80, KS84].

Decomposing a polygonal object into simpler component parts can be done with or without introducing additional vertices which are called *Steiner points*. Whether or not Steiner points are allowed often makes a big difference in the solution of a problem Clearly some problems are only solvable if Steiner points are allowed.

These decomposition techniques have also received attention in such areas as pattern recognition and image analysis [Sk70, FeP75]. In pattern recognition, one extracts information from an object in order to identify or classify it. A description of an object can be an enumeration of its simple components, and a characterization of the relation among them [FeP75, Pa77].

From the practical as well as theoretical point of view, the class of rectilinear

polygons merits attention. *Rectilinear polygons* are defined as polygons whose edges are parallel to either of two given orthogonal directions. In order to obtain shorter algorithm descriptions, we assume that in any rectilinear polygon considered, no two horizontal (vertical) edges have the same y-coordinate (x-coordinate). In image processing, the boundaries of objects are stored on a grid which usually implies that digitized images are rectilinear polygons. Other typical domains where rectilinear polygons naturally occur are VLSI design and processing of satellite data. We see rectilinearity as another source of structure in polygons, which originates from the specific hardware being used. We will call the computational geometry which deals with rectilinear polygons, *rectilinear computational geometry* (some authors [Wo84] prefer isothetic instead of rectilinear).

1.2. Thesis Results Related to the Literature

In this thesis, we show how to exploit the rectilinear structure for the design of efficient algorithms. Most of the results obtained in this thesis are directly or indirectly tied to visibility problems in rectilinear polygons In the first part of the thesis we introduce a new tool in rectilinear computational geometry, called the *labeling scheme*. We found this tool useful for almost all aspects of rectilinear geometry discussed in this thesis. In Section 3.1 we give a construction for generating rectilinear polygons algorithmically. Besides a theoretical aspect, also discussed in Section 3.1, this has the practical benefit of being able to generate n-vertex rectilinear polygons for any choice of n.

As mentioned above, an important aspect in the design of efficient algorithms is the presence or absence of structure in polygons. In Section 3.2 we therefore turn our attention to the design of efficient algorithms for testing rectilinear polygons for structural properties. We can test rectilinear polygons for star-shapedness, monotonicity, rectilinear convexity and edge-visibility in linear time. Previously the best algorithm for testing a polygon for edge-visibility had been quadratic in the

number of vertices [AT81a]. Linear-time algorithms for testing a polygon for monotonicity [PrS81] or star-shapedness [LeP79] had been known previously. The rectilinear structure allows us to develop more efficient algorithms. Efficiency is of particular importance in rectilinear geometry since the rectilinear images produced by today's scanning devices typically contain large quantities of points.

In Chapter 4 we present efficient linear-time hidden-line elimination algorithms for several models of visibility. Prior to the work described in this thesis, the best algorithm for hidden-line elimination in polygons was linear and utilized three stacks [ElA81]. The hidden-line algorithm presented here exhibits the same worst-case time complexity but uses only one stack and is conceptually simpler [Sa83]. Recently, Lee [Le83] independently obtained a similar algorithm. The newly developed tool allows for a unified, succinct approach to solving these rectilinear computational geometry problems. In Chapter 5 we show further applications of the labeling-scheme to a shortest path problem, and a convex hull problem relevant to rectilinear geometry, the determination of the rectilinear convex hull.

The last chapter of this thesis is devoted to decomposition problems in rectilinear computational geometry. As an application of a particular partitioning problem, we discuss the guard-placement problem for rectilinear polygons. The problem description in its more general form is due to Klee [Ho76] in 1973. It can be stated as follows. What is the minimum number of guards always sufficient to see the inside of an n-vertex polygon? Guards are placed on fixed locations. The first solution is due to V. Chvátal who showed that $\left\lfloor \frac{n}{3} \right\rfloor$ guards are always sufficient (and sometimes necessary) [Ch75]. A simpler proof based on triangulation was later given by Fisk [Fi78]. Kahn, Klawe and Kleitman [KKK83] showed that in the case of rectilinear polygons, $\left\lfloor \frac{n}{4} \right\rfloor$ guards suffice. This bound is also tight in the sense that for some rectilinear polygons,

$\left\lfloor \frac{n}{4} \right\rfloor$ guards are also necessary. As will be illustrated in Chapter 6, the main step in their proof is to show that rectilinear polygons can be partitioned into convex quadrilaterals. A partitioning of a polygon P into convex quadrilaterals will be called a (convex) *quadrilaterization* of P. Their primary concern was to show the existence of a convex quadrilaterization for rectilinear polygons, whereas our interest is in developing efficient algorithms to actually perform such a task.

The basic building units in designing our algorithms are a special class of edge-visible rectilinearly convex polygons, which we will call *pyramids*. First we demonstrate how to quadrilaterize pyramids Next we present an algorithm to quadrilaterize rectilinear star-shaped polygons by first partitioning them into pyramids, then quadrilaterizing these and subsequently merging the resulting solutions. We then show how to extend these results to rectilinear polygons which are monotone in some direction. We employ the same basic methodology of partitioning the polygon into pyramids and merging the resulting pyramid quadrilaterizations. Both algorithms are optimal, i.e. the run-time is linear in the number of vertices. Subsequently we show how to partition an arbitrary rectilinear polygon into monotone pieces. Notice however, that not every decomposition of a rectilinear polygon into monotone pieces will produce polygons that are rectilinear As not every arbitrary monotone polygon is quadrilaterizable, the decomposition step has to ensure that only those monotone components are created that do admit a quadrilaterization. The run-time of partitioning an n-vertex rectilinear polygon into monotone polygons admitting quadrilaterization turns out to be O(n log n). Thus the entire task to quadrilaterize an n-vertex rectilinear polygon can be performed in O(n log n) time.

Often when dealing with a decomposition, one is interested in minimizing either its weight or the number of pieces created by the decomposition. The *weight* of a decomposition may be defined as the sum of the edge-lengths (usually Euclidean) of all

diagonals. Klincsek [Kl80] describes an $O(n^3)$ algorithm for the minimum-weight triangulation of a simple n-vertex polygon. In a joint work with M. Keil, we showed how to solve the minimum-weight quadrilaterization problem in $O(n^4)$ time [KS84].

The algorithms presented in this thesis were implemented in part by Shigeo Inoue as a Master's Project under G. T. Toussaint at McGill University, Montreal, and in part by James Dean, Kai Ng and Mark Waldvogel as course projects in CS-95-590 at Carleton University, Ottawa under the author's supervision. This thesis presents and analyses the algorithms from a theoretical point of view. A discussion of implementation considerations may be found in [In82] and [DNW84].

1.3. Summary of Results

The main results of the thesis can be briefly summarized as follows:

(1) *A new tool, the labeling scheme, for rectilinear polygons is introduced and analyzed.* The technique captures information about the winding properties of rectilinear polygons, making processing in subsequent algorithms simpler.

(2) *A linear-time algorithm for detecting whether a given rectilinear polygon is edge-visible is presented.* This improves the $O(n^2)$ algorithm for arbitrary polygons suggested in [AT81a]. A variety of classes of rectilinear polygons are characterized.

(3) *Improvement in the efficiency of algorithm for detecting monotonicity and rectilinear convexity of rectilinear polygons.*

(4) New, efficient algorithms, based on the labeling scheme, *for hidden-line elimination for rectilinear polygons have been designed.* Based on these algorithms, an algorithm for the construction of the convex hull of a rectilinear polygon and an algorithm for a shortest rectilinear path problem has been designed

(5) *Algorithms for quadrilateralization of rectilinear polygons have been designed.*

The algorithms described for star-shaped rectilinear polygons and for monotone rectilinear polygons require linear time, which is optimal. The algorithm for simple rectilinear polygons requires O(n log n) time. These quadrilateralization algorithms were the first known algorithms for placing $\left\lfloor \frac{n}{4} \right\rfloor$ guards in rectilinear polygons.

Chapter 2
A New Tool for Computational Geometry

In this chapter we introduce a new tool for designing efficient algorithms in rectilinear computational geometry. The key idea is to extract information about the "winding properties" of a given rectilinear polygon. We can see this tool as a preprocessing step to subsequent algorithms. This preprocessing step serves a dual purpose. In the presence of structural information, certain algorithms may gain in efficiency. Secondly, a variety of cases and subcases are often tested inside the body of algorithms, which may lead to complex program structures as well as to involved proofs of correctness, if not even to faulty algorithms. Having knowledge about the winding properties of the polygon, the number of such cases and subcases may be reduced, leading to clear and conceptually simple solutions. To illustrate this point, we will present a hidden-line algorithm which has the simplicity of a one stack algorithm, similar to Sklansky's convex hull algorithm [Sk70] and whose run-time is linear. For a discussion of Sklansky's convex hull algorithm, see [ToA82]. The gain in efficiency by using this preprocessing step is even greater if several algorithms subsequently utilize the structural information so extracted.

2.1. The Labeling-Scheme

Informally, the new tool is a specific way of labeling the edges of a given rectilinear polygon P, where the labels are dependent on the "turns" exhibited during a clockwise traversal of P. Before giving a formal definition, we will introduce some terminology. Let α_i denote the clockwise outer angle from edge e_i to edge e_{i+1} at a vertex p_i. The *exterior angle* θ_i at p_i is defined as $\alpha_i - \pi$. We speak of a *right-turn* at a vertex p_i if

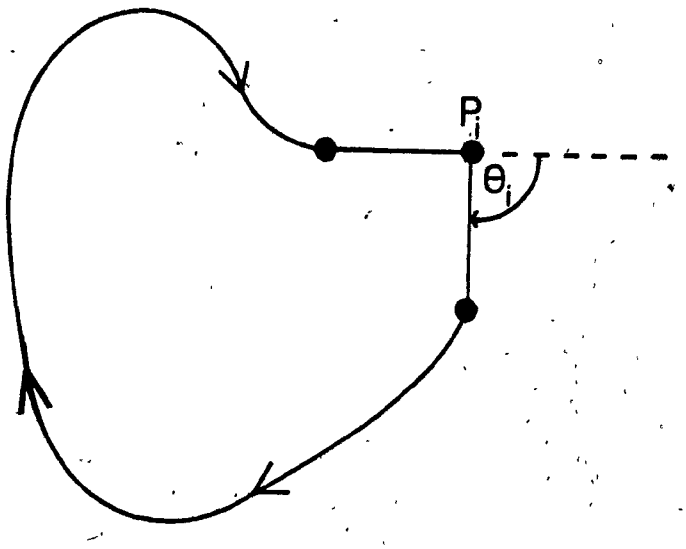


Figure 2.1

Exterior angle θ_i at p_i.

$\theta_i > 0$ and of a *left-turn* otherwise (see Figure 2.1).

We now generalize the notion of left-turn and right-turn defined at vertices, to that of turns of polygonal chains, $C_{i,j}=\{p_i,p_{i+1},\ldots,p_j\}$. The *angular turn* $t'_{i,j}$ of a polygonal chain $C_{i,j}$, for i<j is defined as $\sum_{k=i}^{j-1}\theta_k$. In case that i>j, $t'_{i,j}$ is defined as $2\pi - t'_{j,i}$. If $t'_{i,j}>0$ we say that the chain $C_{i,j}$ performs a *right-turn* and $C_{i,j}$ performs a *left-turn* in case $t'_{i,j}<0$.

Exterior angles at vertices of rectilinear polygons take one of two possible values: $\frac{\pi}{2}$ or $-\frac{\pi}{2}$. Therefore in rectilinear computational geometry turns between edges are integer multiples of $\frac{\pi}{2}$. This implies that we can assign to each edge e_i an integer label ℓ_i as follows:

$\ell_1 := 0$ (* the edge with maximum y-value is arbitrarily labeled 0 *)

$\ell_i := \frac{2}{\pi} t'_{1,i}$, for $1<i\leq n$

An alternative definition of the labeling-scheme is given below:

$\ell_1 := 0$

$\ell_{i+1} := \ell_i + 1$ if p_{i-1}, p_i, p_{i+1} is a right-turn

$\ell_{i+1} := \ell_i - 1$ otherwise.

This equivalent definition suggests immediately an algorithm to determine the labels of a rectilinear polygon. We define the *(integer) turn* $t_{i,j}$ between e_i and e_j as the angular turn $\frac{2}{\pi}t'_{i,j}$. We will speak of an *even turn* between two edges e_i and e_j if the corresponding integer $t_{i,j}$ is even, otherwise the *turn is odd*. The *net-turn* between two edges e_i and e_j is defined as $t_{i,j}$ modulo 4. We use the term i-edge to denote all edges labeled i. For implementation considerations, it is interesting to note that no computations involving angles are needed in order to determine the labels.

We recall now that the vertices of polygon P are given in clockwise order. An edge e of a polygon P is called a *top-edge, bottom-edge, left-edge,* or *right-edge,* if the interior of P is below, above, to the right, or to the left of a line collinear with e, respectively. This defines four *types* of edges. We say that two edges are *oriented* in the same way if they are of the same type.

2.2. Some Properties of the Labeling-Scheme

Before studying the properties of the labeling-scheme, it is convenient to state the following *Theorem of Turning Tangents* known in differential geometry [Kli78, Ca76]. It is often referred to by its German name *Umlaufsatz*. A complete clockwise traversal of a simple polygon describes a 2π revolution. In the labeling-scheme, this is reflected by the fact that for a complete turn: $t_{i,j}+t_{j,i}=4$. An immediate consequence, also observed by O'Rourke [OR83], is that the number of reflex vertices in any n-vertex rectilinear polygon is exactly $\frac{n}{2} - 2$. We can similarly observe that the number of vertices in any rectilinear polygon is even.

Property 2.1 – *Labels of horizontal edges are even, while those of vertical edges are odd.*

Proof: We note that in a rectilinear polygon, horizontal and vertical edges alternate. We restrict ourselves to proving the result for the set of horizontal edges $\{e_1, e_3, ..., e_{n-1}\}$. We will show by induction on the index j=1,3,...,n−1 that for all horizontal edges e_j the turn $t_{1,j}$ between e_1 and e_j is even. Let j=3 then by inspection verify that $t_{1,3}$ is either 0 or 2. Therefore let $3 < j \leq n-2$. We can write $t_{1,j+2}$ as $t_{1,j} + t_{j,j+2}$. By induction we assume that $t_{1,j}$ is even. Between e_j and e_{j+2} are exactly two exterior angles contributing each 1 or −1 to the labeling thus $t_{j,j+2}$ is also even. As the label ℓ_j is equal to $t_{1,j}$ the correctness of Property 2.1 follows. ■

Property 2.2 – *The net-turn between parallel edges oriented in the same direction is zero.*

Proof: Between parallel edges oriented in the same way a (clockwise) traversal of a polygon performs a 2π net-turn. As each $\frac{\pi}{2}$-turn contributes adding +1 to the labeling, the result follows. ■

2.3. Rectilinear Cuts

We define a *horizontal cut* h through a rectilinear polygon P, as a horizontal line which intersects all those vertical edges of P having the same y-coordinate as h. We assume that no horizontal cut has the same y-value as any horizontal edge in P. By $(e_{i_1}, ..., e_{i_k})$, we denote the sorted list of all k edges intersected by h, where e_{i_1} is the leftmost such edge and e_{i_k} the rightmost edge. The corresponding label-list, denoted by $(\ell_{i_1}, ..., \ell_{i_k})$, is called a *horizontal cut-sequence* of length k. In a similarly way we can define *vertical cut-sequences*. A *rectilinear cut* is a cut which is either horizontal or vertical. Similarly a *rectilinear line* is defined as a line that is either vertical or horizontal. The main use of the cut concept is the following important Property 2.3

and Theorem 2.1.

Let h be a rectilinear line intersecting a rectilinear polygon P and let e_i, e_j be two edges with labels ℓ_i, ℓ_j respectively, adjacent in the corresponding rectilinear cut. Let q_iq_j be the open line-segment of h connecting e_i and e_j. The function sign(x) returns -1 if its argument x is negative and $+1$ if x is positive.

Property 2.3.

(a) q_iq_j is inside P iff $\ell_j-\ell_i=+2*sign(j-i)$, and

(b) q_iq_j is outside P iff $\ell_j-\ell_i=-2*sign(j-i)$

Proof: We assume w.l.o.g. that the line h intersecting P is horizontal.

(a): An implication of the Jordan Curve Theorem is that edges which are adjacent in any rectilinear cut have opposite orientation. To prove (a) we introduce Steiner points $q_1,...,q_k$ at the points of intersection between h and $e_1,...,e_j$ respectively. Let q_iq_j be an internal line-segment connecting q_i and q_j. W.l.o.g. let e_i be to the left of e_j, then e_i is a left edge and e_j is a right edge. (Refer to Figure 2.2) The line-segment q_iq_j partitions P into two polygons $Q_1=\{q_i,p_i,...,p_j,q_j\}$ and $Q_2=\{q_j,p_j,...,p_{i-1},q_i\}$. In Q_1 and Q_2 both exterior angles at q_i and q_j are identical, thus $t_{i,j}=t_{j,i}=2$. We conclude that (i) if $i\leq j$ then $t_{i,j}=\ell_j-\ell_i=2$ and (ii) if $i>j$ then $t_{j,i}=\ell_i-\ell_j=2$, thus $\ell_j-\ell_i=-2$. (b): If q_iq_j is an external line-segment joining q_i and q_j. Conversely, to (a) the orientations of the edges e_i and e_j are reversed and thus are the two cases. Therefore if $i\leq j$ then $t_{i,j}=\ell_j-\ell_i=-2$. If $i>j$ then $t_{j,i}=\ell_i-\ell_j=-2$ implying that $\ell_j-\ell_i=2$ ■

Theorem 2.1: *Let $(\ell_{i_1},...,\ell_{i_k})$ be a horizontal cut-sequence of length k, then*

(i) $|\ell_{i_j}-\ell_{i_{j+1}}|=2$, and

(ii) $\ell_{i_1}=3$, $\ell_{i_k}=1$,

Proof: (i) Follows from Property 2.3.

(ii) Let $(\ell_{i_1},...,\ell_{i_k})$ be a horizontal cut-sequence of length k determined by a horizontal

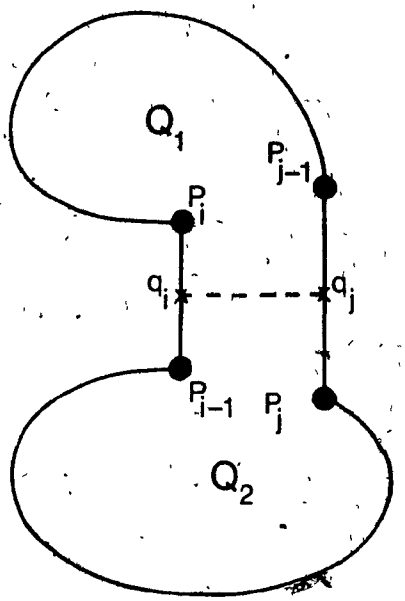


Figure 2.2

Partitioning of a polygon by an internal line segment.

cut h. We now prove that the label $\ell_{1,k}$ of the rightmost edge intersected by h is equal to 1. Since edge e_2 is adjacent to the y-max edge e_1, its label is 1. Let e_j be the rightmost edge intersected by h whose corresponding label is ℓ_j. By the Jordan Curve Theorem e_j and e_2 have the same orientation thus by Property 2.2. $\ell_j \bmod 4 = 0$. If the turn $t_{2,j}$ from e_2 to e_j is zero then the result follows; otherwise $C_{2,j}$ describes at least one full revolution between e_2 and e_j. This however contradicts either the simplicity of P or the fact that e_j is the rightmost edge intersected by h, see Figure 2.3 a,b, respectively. In a similar way we can show that the leftmost edge in any horizontal cut-sequence is labeled 3. ■

We now turn our attention to partitionings of the plane induced by inserting interior or external line-segments into a polygon P such that points on the boundary of P are connected. Any such interior line-segment partitions the polygon P into two regions. In the case that d is an external line-segment of P, a different situation arises.

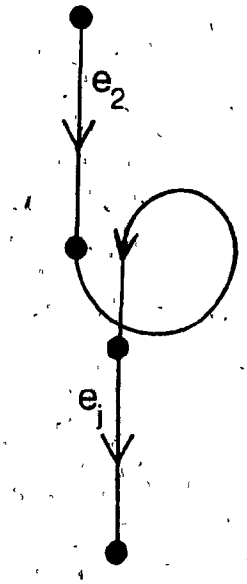


Figure 2.3(a)

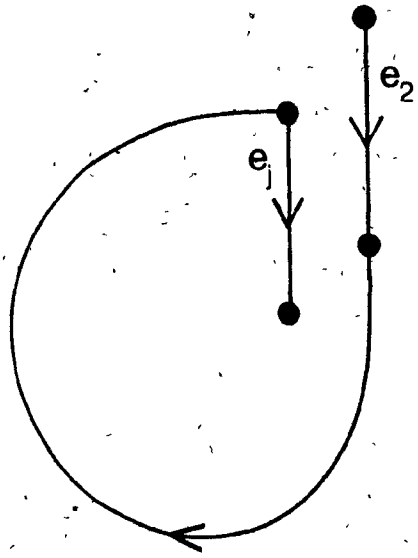


Figure 2.3(b)

Figure 2.3

If edge e_2 is visible and the turn from e_2 to edge e_j is a full revolution, then either (a) the simplicity is violated or (b) e_j is not visible.

We can describe this as d inducing a partitioning of a plane having a polygonal hole P. One such component is bounded while the other is unbounded. The following Theorem characterizes which of the two regions is bounded and which is unbounded. This is illustrated in Figure 2.4.

Theorem 2.2. *Let $q_i q_j$ be an external line-segment connecting the edges e_i and e_j of P.*

(i) In the case that $\ell_i - \ell_j = 2$, the bounded region induced by d is the polygon $\{q_i, p_i, p_{i+1}, \ldots, p_{j-1}, q_j\}$.

(ii) In the case that $\ell_i - \ell_j = -2$, the bounded region is described by the polygon $\{q_i, p_{i-1}, p_{i-2}, \ldots, p_{j+1}, q_j\}$.

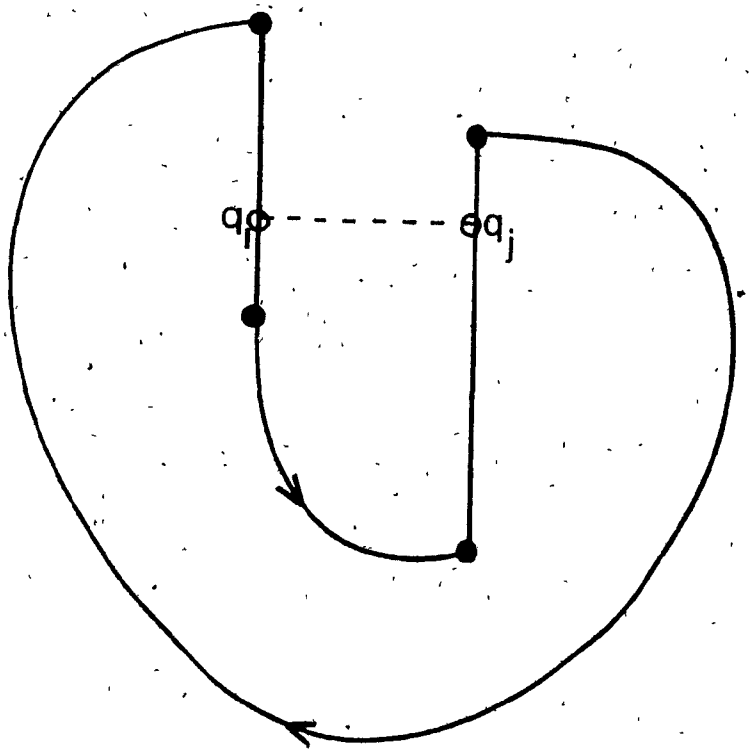


Figure 2.4

A bounded region and an unbounded region are created by inserting the external diagonal $q_i q_j$.

Proof: As a consequence of the Jordan Curve Theorem, the bounded region is obtained by connecting q_i to q_j by a polygonal chain of P being oriented oppositely to the clockwise orientation of P. To find this chain, we apply the previously derived Property 2.3 and obtain that as $q_i q_j$ is an external line-segment, $\ell_j - \ell_i = -2*\text{sign}(j-i)$. Therefore, if $\ell_i - \ell_j = 2$ then $\text{sign}(j-i)=1$, implying that $i<j$. The bounded region is therefore $\{q_j, p_{j-1}, p_{j-2}, \ldots, p_{i+1}, p_i, q_i\}$. The converse holds for the case (ii) that $j<i$ and thus the bounded polygon is $\{q_i, p_{i-1}, p_{i-2}, \ldots, p_{j+1}, p_j, q_j\}$. ■

We sketch a consequence resulting from Theorem 2.2. Take any two points p, q located in the exterior of P. By the Jordan Curve Theorem, p and q can be joined by a simply connected polygonal path, path(p,q), which does not intersect the boundary of P properly. If now exactly one of the points falls within a bounded region as constructed above, then path(p,q) has to intersect the external line-segment separating the regions. This result is used in Chapter 5, when we consider a shortest path problem.

2.4. Generalized Cuts

We will now generalize the notion of horizontal cut-sequences by defining *generalized cut-sequences*. Let ℓ denote an oriented line intersecting a rectilinear polygon P. Let $(e_{i_1},\ldots,e_{i_k})$ denote the sorted list of edges intersected by ℓ in the order in which they are intersected by ℓ. Edge e_{i_1} is the first and e_{i_k} the last such edge. The corresponding label-sequence is called a *(generalized) cut-sequence* and is denoted by $(\ell_{i_1},\ldots,\ell_{i_k})$.

Theorem 2.3. *Let $(\ell_{i_1},\ldots,\ell_{i_k})$ be a generalized cut-sequence; then*

$$|\ell_{i_{j+1}} - \ell_{i_j}| \leq 3, \text{ for } j=1, k-1.$$

Proof: Let ℓ be a line whose induced cut-sequence with respect to P is $(\ell_{i_1},\ldots,\ell_{i_k})$. For ease of notation let e_i, e_j be two edges which are adjacent in a generalized cut. Furthermore let the intersection point of e_i with ℓ be denoted by q_i and that of e_j be denoted by q_j. W.l.o.g. assume that e_i is horizontal. We consider two polygons defined as $Q_1:=(q_i,p_i,\ldots,q_{j-1},q_j)$ and $Q_2:=(q_j,p_j,\ldots,p_{i-1},q_i)$. If e_j is horizontal then the discussion is analogous to that of Theorem 2.1. Otherwise, e_j is vertical. The sum of the interior angles at q_i and q_j in either Q_1 or Q_2 is $\frac{\pi}{2}$ or $\frac{3}{2}\pi$ In case that i<j we consider the polygon Q_1. A clockwise traversal of Q_1 from q_i to q_j performs a $2\pi-(\alpha_i-\alpha_j)$ revolution, which is by the above

(i) The point q_i is above e_j As the interior of P is above e_j the segment q_iq_j of ℓ is

inside P. In this case if $j<i$ then $\ell_j-\ell_i=-3$ and if $i<j$ then $\ell_j-\ell_i=1$.

(ii) The point q_i is below e_j. As the exterior of P is below e_j, conversely to case (i), we get for $j<i$, $\ell_j-\ell_i=1$ while for $j>i$, $\ell_j-\ell_i=-3$. Similarly to the proof of the corresponding lemma for horizontal cut-sequences, the simplicity of P forbids other cases.

(b) The case that e_j is a top-edge is similar to (a) and is thus omitted. ■

Corollary 2.2. *Under the assumptions of Theorem 2.3,*

(a) Let q_iq_j be inside P. $C_{i,j}$ is a right-turn if and only if $i<j$.

(b) Let q_iq_j be outside P. $C_{i,j}$ is a left-turn if and only if $i<j$.

Theorem 2.4. *The label difference $\ell_i-\ell_j$ is positive if and only if $\{q_j, p_{j-1}, p_{j-2}, \ldots, p_{i+1}, p_i, q_i\}$ is a bounded polygonal region. Similarly, the label difference $\ell_i-\ell_j$ is negative if and only if $\{q_i, p_{i-1}, p_{i-2}, \ldots, p_{j+1}, p_j, q_j\}$ is a bounded polygonal region.*

Proof: The result is a straight forward generalization of the corresponding Theorem 2.2 for rectilinear cuts. We therefore omit its proof. ■

In this chapter we have introduced the labeling-scheme for rectilinear polygons. Several properties which we will use throughout the remainder of this thesis have been derived. The next chapter will now investigate structural properties of the rectilinear polygons.

Chapter 3
Structural Properties of Rectilinear Polygons

In this chapter we will design several efficient algorithms for testing a rectilinear polygon for structural properties. The algorithms are based on the labeling-scheme as introduced in the previous chapter.

3.1. Construction of Rectilinear Polygons

We will characterize a rectilinear polygon P by its label-sequence. The *label-sequence* is defined as the concatenation of all labels encountered in a clockwise traversal of P starting with label ℓ_1. It is important to note that a given label-sequence does not uniquely characterize a rectilinear polygon. However two simple rectilinear polygons with the same label-sequences are somewhat "similar". We therefore say that two simple rectilinear polygons are *similar* if their corresponding label-sequences are identical

Given an arbitrary sequence of integers, a rectilinear polygon, whose label-sequence is s, may not necessarily exist. We are therefore interested in characterizing those sequences of integers that are label-sequence for some rectilinear polygon. This characterization will be given using a formal language-like approach We will use this characterization to generate one representative for each class of similar rectilinear polygons. Furthermore a necessary condition for simplicity of a rectilinear polygon is given, which can be determined in linear time.

In order to describe the label-sequence formally, we now introduce some notation which is analogous to the characterization of languages in formal language theory, see [HoU79]. We will establish a correspondence between rectilinear polygons and strings generated by a grammar. Structural properties of rectilinear polygons can then be

detected by parsing the corresponding strings. We define a grammar G=(N, T, P, S), where N={A, S} is the set of nonterminal symbols, T={integers numbers} is the set of terminal symbols, and S is the start-symbol of the grammar. We use i as a variable to denote integers. The production set P of G is defined as follows: we have used the notion *grammar* in a more general sense, by allowing an infinite number of terminal symbols as well as productions.

(a) S --> 0 1 A

(b) i A --> i i+1 A

(c) i A --> i i−1 A

(d) 3 A --> 3

We give an example. The label-sequence for the rectilinear polygon of Figure 3.1 is 0 1 0 1 2 3. It is generated by applying the productions (a) (c) (b) (b) (b) (d) in sequence.

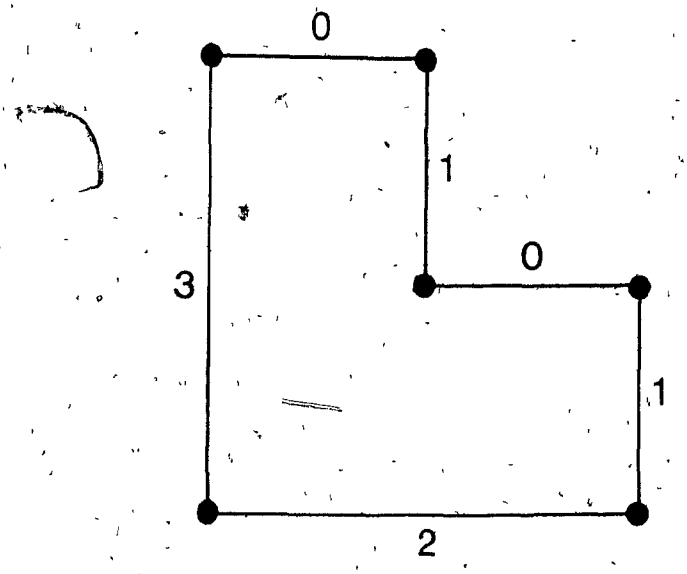


Figure 3.1

Label-sequence 0 1 0 1 2 3 generated by applying productions (a) (c) (b) (b) (b) (d).

Property 3.1. *All label-sequences for simple rectilinear polygons can be generated by the above grammar G*

Proof: We show that for a given simple rectilinear polygon P, its label-sequence can be generated by the grammar G. The label-sequence for any rectangle is 0 1 2 3. This sequence can be generated by applying production rules (a), (b), (b) and (d) in that order. Furthermore, by Theorem 2.1, we know that label-sequences of simple rectilinear polygons start with 0,1 and end with 3. Since the production (a) and (d) are applied the initial and final productions, respectively, any word created by the grammar also has this property. Both these productions are applied exactly once The difference between any two consecutive labels in the label-sequence is either 1 or -1 depending on whether the turn at the vertex adjacent to both edges is a right-turn or left-turn, respectively. In the label-sequence this corresponds to applying productions (b) or (c), respectively. ■

Property 3.2. *For each word s generated by G there exists a simple rectilinear polygon whose label-sequence is s.*

Proof: Let s be a word generated by the grammar G. Let $|s|$ denote the length of s. We show by induction on $|s|$ how to generate the rectilinear polygon whose label-sequence is s. We first observe that the length of s is equal to the number of vertices in the polygon whose label-sequence is s. The shortest word generated by G is 0 1 2 3. Any rectangle has this label-sequence and thus the existence follows for this case Now we assume that for any word s' and even number $n>4$, for which $|s'| \leq n-2$, we can construct a rectilinear polygon whose label-sequence is s'. We have to show that for any word s with $|s|=n$, we can construct a rectilinear polygon P, s.t. the label-sequence of P is s. In a derivation of s, say at steps $j-1$ and j, productions (b) and (c) are applied either in the order (b) followed by (c), or (c) followed by (b) If we now apply all productions used for s except those at steps j, $j+1$ then a shorter word s' of length $n-2$ is generated by G. We demonstrate how P' is used to construct the desired

polygon P. W.l.o.g. assume that the edge e_j in P' inserted at step j is a right edge. Otherwise a similar construction can be given. Let d denote the minimum perpendicular distance between any pair of parallel edges in P'. Now construct a point p^* located on e_j whose distance from p_j is $\frac{d}{2}$. Furthermore let R be a rectangle satisfying the following properties:

(i) width=d, height=$\frac{d}{2}$,

(ii) the edges of the rectangle are parallel to the polygonal edges,

(iii) the line segment $p^* p_{j+1}$ is a symmetry axis of R.

We label the four corner points of R in a clockwise fashion by r, s, t, u, starting with the top, rightmost corner point as r. The construction is illustrated in Figure 3.2.

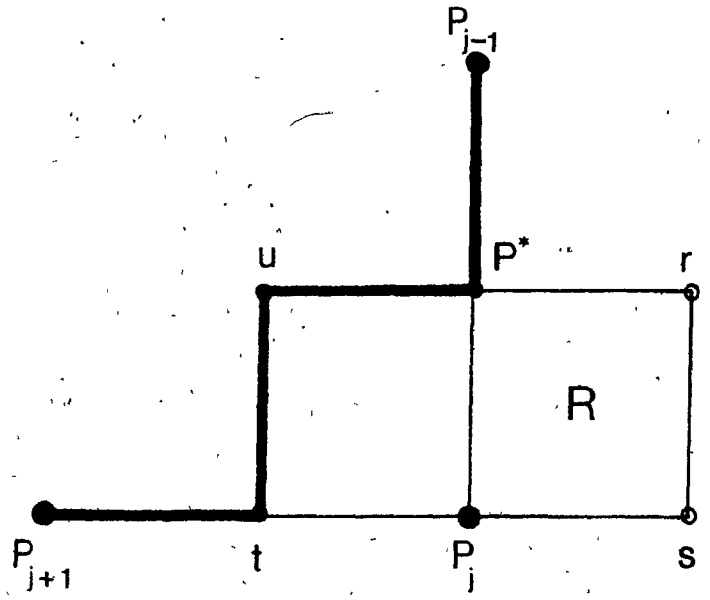


Figure 3.2

An example of the creation of new edges. P_{j-1}, P_j, P_{j+1} is replaced here by P_{j-1}, P^*, u, t, P_{j+1}.

As the distance between any two parallel edges is at least d so is the length of any edge and no edge of P, except e_j intersects the interior of R. Edge e_j and the boundary st of R are collinear. We only discuss the case that the turn at p_j in P' is a right-turn. A similar argument can be made otherwise. Next, we incorporate the two production rules (b), (c) we previously eliminated. Two cases arise depending on the order in which, in the derivation of s at steps j−1 and j, the productions (b), (c) are applied.

(i) Production (b) is followed by (c). In this case we replace vertex p_j by p^*, u, t. This has the effect that the chain p_{j-1}, p_j, p_{j+1} is replaced by p_{j-1}, p^*, u, t, p_{j+1}.

(ii) Production (c) is followed by (b). In this case we replace vertex p_j by p^*, r, s. This has the effect that the chain p_{j-1}, p_j, p_{j+1} is replaced by p_{j-1}, p^*, r, s, p_{j+1}.

The construction of P is completed by making all other vertices of P' also vertices of P. The label-sequences s and s' differ exactly at the two positions determining the two new vertices. Therefore the total number of vertices in P is n. As the rectangle R possesses the intersection properties discussed above, P is a simple rectilinear polygon. Lastly, the right-turn at p_j is maintained, as a right-turn at either t or s, depending on whether case (i) or (ii) applies. We have thus constructed a simple rectilinear polygon P with n vertices whose label-sequence is s. ■

3.2. Algorithms for Detection of Structure in Rectilinear Polygons

In this section we will sketch further applications of the labeling-scheme in relation to detecting structural properties of rectilinear polygons. We will show in Chapter 6, how rectilinear polygons with specific properties can be convexly quadrilaterized in linear time. Classes of rectilinear polygons which can be quadrilaterized in linear time

include monotone, star-shaped and edge-visible polygons. For arbitrary simple, rectilinear, n-vertex polygons, such a partitioning into convex quadrilaterals takes O(n log n) time. In order to apply any of these linear time algorithms, we must first establish that the given rectilinear polygon has any one of these structural properties. If the test took O(n log n) time then clearly little would be gained. We will therefore present efficient linear-time algorithms for testing a rectilinear polygon for structural properties. These algorithms detect star-shapedness, rectilinear convexity, monotonicity and edge-visibility. We start by discussing a necessary condition for determining whether a given rectilinear polygon is simple or not.

3.2.1. A Necessary Condition for Simplicity

A seemingly difficult open problem in computational geometry is to find a linear-time algorithm or an Ω(n log n) lower bound proof on the problem of deciding whether a given polygon is simple. Shamos showed that for a given n-vertex polygon, O(n log n) is upper bound for this problem [Sh77]. His solution is obtained by considering each polygonal edge as a line-segment by itself and subsequently testing whether any two line-segments so obtained intersect. Ω(n log n) is lower bound for the problem of testing whether any two of n line-segment intersect. However the edges of a polygon are not simply a collection of line-segments. Therefore this lower bound does not hold for the problem of testing whether a given polygon is simple or not. Rather than developing another O(n log n) algorithm for testing a rectilinear polygon for simplicity, we present a necessary condition for simplicity. This condition can be verified in linear time, since from Property 3.1 it follows that if a rectilinear polygon P is simple, its label sequence can be generated by G.

We give an example for a rectilinear polygon which is not simple and show that its label-sequence cannot be generated by G. The label-sequence s for the rectilinear polygon shown in Figure 3.3 is s = 0 1 2 3 4 5 6 7. As this sequence ends with 7, it

cannot possibly be generated by G.

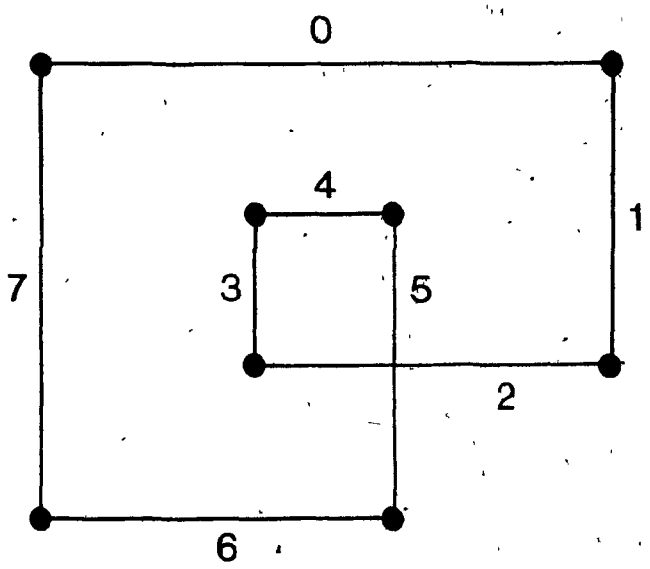


Figure 3.3

The polygon has label sequence 0 1 2 3 4 5 6 7, and thus is not simple.

The fact that the label-sequence of an arbitrary rectilinear polygon P is generated by G; does not necessarily imply that P is simple. Figure 3.4 illustrates such a situation. We will always detect, in linear time, that a rectilinear polygon is not simple, if a traversal of P does not perform a 2π revolution and thus its label sequence cannot be generated. For example a traversal of the polygon in Figure 3.3 performs a 4π revolution. It remains an open problem whether simplicity of a polygon which performs a 2π-turn, can be tested for in linear time.

We will next discuss algorithms to test polygons for star-shapedness, rectilinear convexity, monotonicity and edge-visibility.

3.2.2. Star-Shapedness Test

Recall that the kernel of a simple polygon P is the set of all points in P which can each see the entire polygon. Each edge e_i of P splits the plane into two half-planes, a

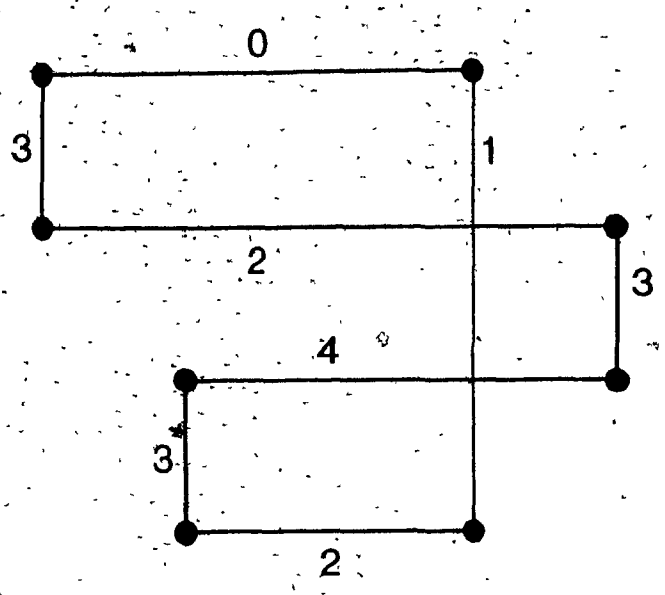


Figure 3.4

Although the label sequence is legal, the polygon is not simple.

positive and a negative one. The positive half plane contains all points on the plane located to the right of the line through e_i and oriented in the same way as e_i. The kernel of a polygon P can be constructed by intersecting all positive half- planes defined by edges of P. Although not optimal, this procedure demonstrates that the kernel can be constructed by intersecting convex regions and is thus itself convex. Lee and Preparata [LeP76] describe an algorithm which uses the order of the edges as given by the polygon to construct the kernel in linear time. Their procedure can easily be simplified for rectilinear polygons as we will demonstrate below, see also [ToE81].

Consider the four extremal chains $C_{y_{max}x_{max}}$, $C_{x_{max}y_{min}}$, $C_{y_{min}x_{min}}$, $C_{x_{min}y_{max}}$ of a rectilinear polygon P. These extremal chains are either disjoint or meet at an edge. We call two extremal chains *adjacent chains* if they share a common edge. We say that a chain $C_{i,j}$ is *empty* if j=i+1, otherwise $C_{i,j}$ is non-empty. We define a *stair* as a rectilinear polygonal chain monotone in both the x-direction and y-direction. In this

section we only consider stairs which are extremal chains of a given rectilinear polygon.

Recall that an edge e_i is *completely visible* from a point p if each point on e_i is visible from p. Similarly a polygonal chain $C_{i,j}$ is said to be *completely visible* from a point p if all edges on $C_{i,j}$ are completely visible from p. We continue by examining properties of stairs:

Property 3.3. *It is necessary and sufficient for a stair $C_{i,j}$ to be completely visible from a point p that the endpoints p_i, p_{i+1} of the first edge and p_{j-1}, p_j of the last edge are visible from p.*

Proof: The necessity is obvious. To prove the sufficiency we assume that there exists a point p which sees the endpoints of the first and last edge of $C_{i,j}$. We commence by recalling a result by Avis and Toussaint [AT81a], stating that an edge is completely visible from a point if and only if both its endpoints are visible from this point. We will prove the result for $C_{i,j}=C_{y_{max}x_{max}}$. The other stairs can be handled similarly. W.l.o.g we let $C_{i,j}$ be a stair of alternating top and right-edges. In this case, $C_{i,j}$ is a chain of vertices with monotonically decreasing y-values and increasing x-values. As p sees the endpoint of the lowest edge of the stair, p must be located to the right of the rightmost reflex vertex in $C_{i,j}$. Similarly, p is located above of the highest reflex vertex in $C_{i,j}$. We assume now that a point q located on some edge e_k, $i \leq k \leq j$, is not visible from p, then there exists some edge e_m intersecting the line-segment pq. The intersection point of e_m with pq must be between p and q. Two cases arise (a) $m>k$ and (b) $m<k$. (a) In case that $m>k$ the chain $C_{k,m}$ is not monotonically increasing in the y-direction. In case (b), $m<k$, the chain $C_{m,k}$ is not monotonically decreasing in the x-direction. In either case, P is not a stair which is a contradiction. ■

Property 3.4. *The extremal edges of rectilinear star-shaped polygons are connected by stairs.*

Proof: The proof is straight forward and thus omited. ■

In Figure 3.5 we give an example of a rectilinear star-shaped polygon.

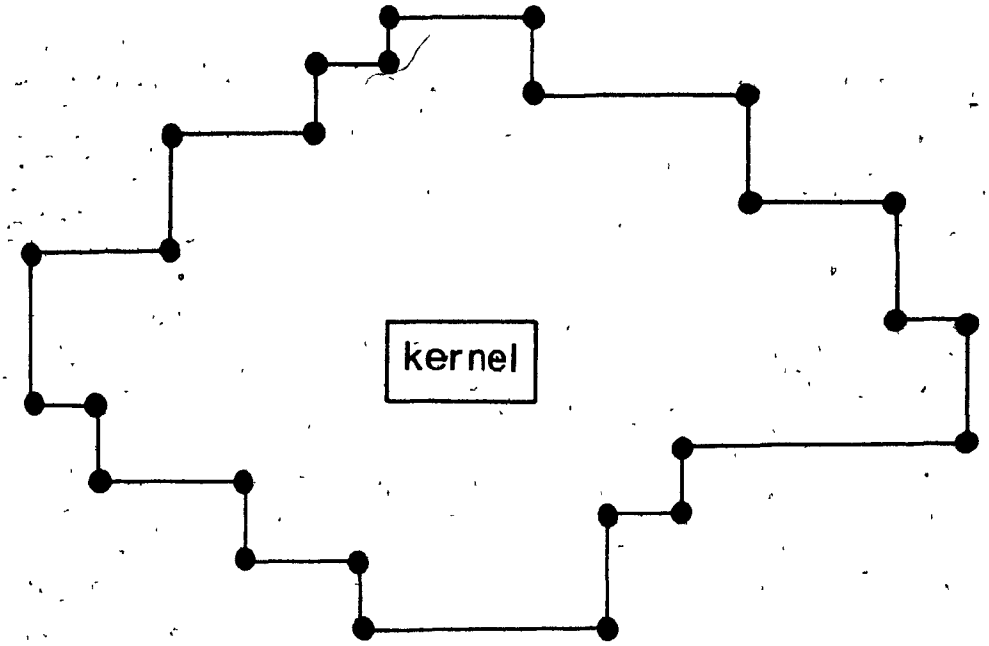


Figure 3.5

A rectilinear star-shaped polygon.

The structural properties of rectilinear polygons are now related to label-sequences. We show that label-sequences provide a concise notation for structural properties. As strings defining label-sequences can be parsed efficiently, we will express structural properties using label-sequences. We use the *Kleene star* to denote zero or more repetitions of labels, i.e. $(ab)^*$=abababababab..., (see [Hol'79]). The symbol "+" is used to denote one or more repetitions of the letters, i.e. $(ab)^+ := (ab)(ab)^*$.

Fact: A rectilinear polygon P with label-sequence s is star-shaped iff (i) $s=(01)^+(21)^*(23)^+(43)^*$, and (ii) the y_{max}-edge sees the y_{min}-edge and the x_{max}-edge sees the x_{min}-edge.

Proof: (i) Edges on $C_{y_{max}x_{max}}$ have alternating labels 0 or 1, starting with 0 as the label of $e_{y_{max}}$ and ending with 1 as label of $e_{x_{max}}$. The sequence $(21)^*$ characterizes the possibly empty stair $C_{x_{max}y_{min}}$. (In case that the stair is empty its description is the empty string, denoted by ϵ). $(23)^+$ describes $C_{y_{min}x_{min}}$ and similarly, $(43)^*$ describes the possibly empty stair $C_{x_{min}y_{max}}$.

(ii) Let $e_{max}=t_\ell t_r$, $e_{x_{max}}=r_t r_b$, $e_{y_{min}}=b_r b_\ell$ and $e_{x_{min}}=l_b l_t$. Furthermore let [a,b]:=[min{y-value of ℓ_t, y-value of r_t}, max{y-value of ℓ_b, y-value of r_b}] and [c,d]:= [max{x-value of t_ℓ, x-value of b_ℓ}, min{x-value of t_r, x-value of b_r}]. Let p#q denote the point with horizontal coordinate equal to that of p and vertical coordinate equal to that of q, analogously q#p is defined, see Figure 3.6 for illustration Now consider the rectangle defined as R:= c#a, c#b, d#b, d#a. Each point p inside R sees both endpoints of each of the four extremal edge. Thus by Property 3.3 p sees each stair completely, implying that p is inside the kernel. Any point q outside R will not be able to see at least one of the endpoints of an extremal edge, therefore q cannot be inside the kernel. Thus we conclude that the kernel is precisely the rectangle R as defined above. ■

This suggests an algorithm to test whether a rectilinear polygon P is star-shaped, see also [ToE81]. Once we have verified that P is star-shaped, the kernel of P is readily constructed using only the extremal edges. It follows that the kernel of a rectilinear polygon is a rectangle. The steps are given below.

Algorithm 3.1: Test for Star-Shapedness

Input: A rectilinear polygon P.

Output: 'Yes', if P is star-shaped; 'No', otherwise.

```
Construct the label-sequence s for P;
if s is of the form (01)+ (21)* (23)+ (43)*
   then
      begin
         construct the rectangular shaped kernel R;
         if R is non-empty
            then
               report 'Yes', exit
      end
report 'No'.
```

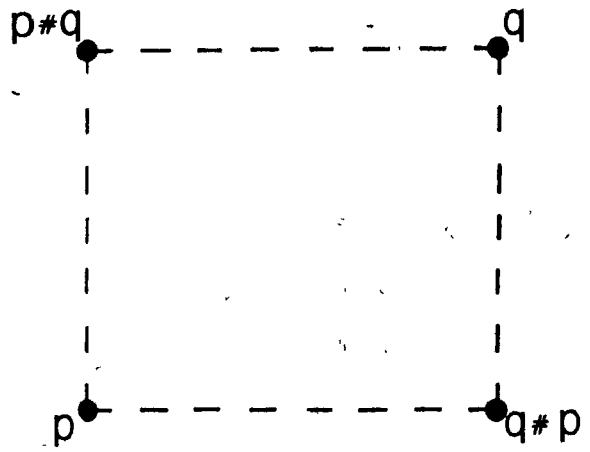


Figure 3.6

The point with vertical coordinate equal to that of q and horizontal coordinate equal to that of p is denoted by p#q.

3.2.3. Monotonicity Test

We will prove a theorem enabling us to design an algorithm for testing whether or not a simple rectilinear polygon is monotone. The existing algorithm for testing a simple polygon for monotonicity is due to Preparata and Supowit [PrS81]. Their algorithm exhibits the same worst case complexity as the algorithm proposed here. Due to the rectilinear structure, our algorithm is conceptually simple, and efficient. Both algorithms are capable not only of detecting whether a polygon is monotone, but also of reporting all directions of monotonicity for P. Let ℓ be any line and its orientation be described by the counterclockwise angle ω formed by ℓ and the x axis (see Figure 3.7).

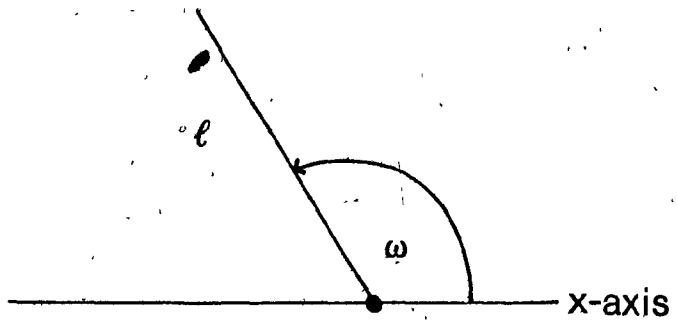


Figure 3.7

The counter clockwise angle is formed by line ℓ and the x-axis.

Our algorithm will follow immediately once we have characterized monotone rectilinear polygons. To obtain this characterization let us first define several terms. We say that a polygonal chain, $C_{ik} = p_i,...,p_j, .., p_k$ *changes* the monotonicity with respect to a given line ℓ, at an edge e_j, if the order of the projections of

$p_{j-1}, p_j, p_{j+1}, p_{j+2}$ onto ℓ differ from the polygonal order. An example is given in Figure 3.8.

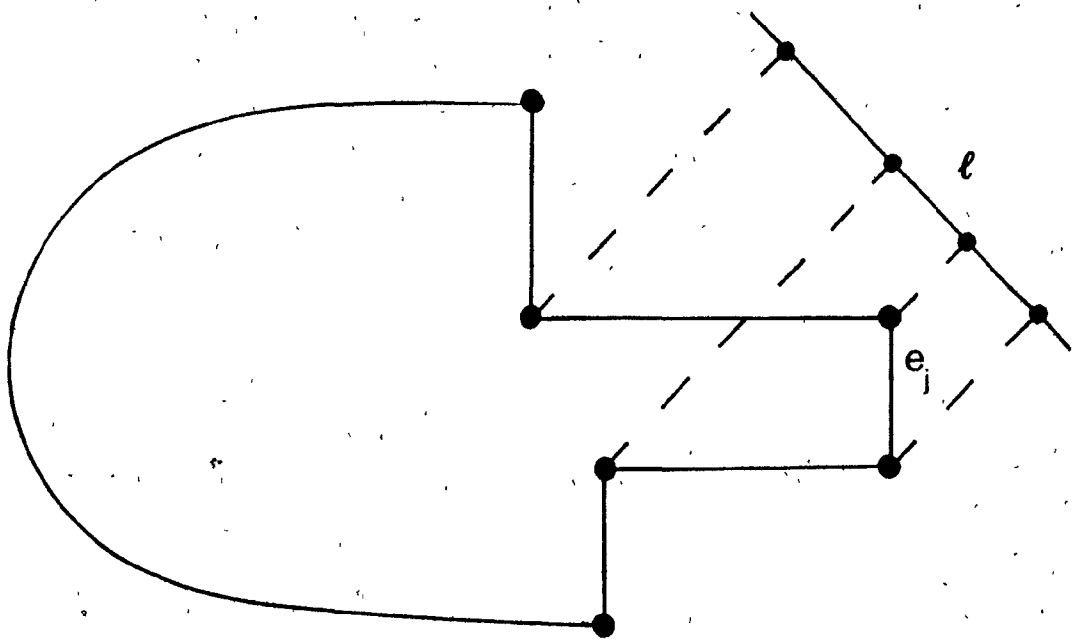


Figure 3.8

The monotonicity with respect to ℓ changes at edge e_j.

Property 3.5. *If a rectilinear polygon P is monotone with respect to any one line ℓ; for which ω is in $(0,\frac{\pi}{2})$, $\{\omega$ in $(\frac{\pi}{2},\pi)\}$, then P is monotone with respect to all lines ℓ with ω in $[0,\frac{\pi}{2}]$, $\{\omega$ in $[\frac{\pi}{2},\pi]\}$.*

Proof: Let a rectilinear polygon P be monotone with respect to some line ℓ, for which ω is in $(\frac{\pi}{2},\pi)$ Furthermore let y_{max}, y_{min} denote indices of the edges with maximum and minimum y-value, respectively. The monotonicity with respect to ℓ is clearly changed at the edges $e_{y_{max}}$ and $e_{y_{min}}$ W.l.o.g. let $e_{y_{max}} = p_n p_1$ and $e_{y_{min}} = p_k p_{k+1}$. If P is monotone with respect to ℓ, then both chains must be monotone with respect to ℓ. The chains can not contain any edges changing the monotonicity. Therefore, as P is

monotone with respect to ℓ, all horizontal edges on the left chain are bottom edges and all horizontal edges on the right chain are top-edges (see Figure 3.9). But polygons whose extremal chains have this property are monotone with respect to any line ℓ with ω in $[\frac{\pi}{2},\pi]$. The proof for the case when P is monotone with respect to a line ℓ with ω in $(0,\frac{\pi}{2})$ is similar and can be obtained by rotation of the polygon by $\frac{\pi}{2}$. ■

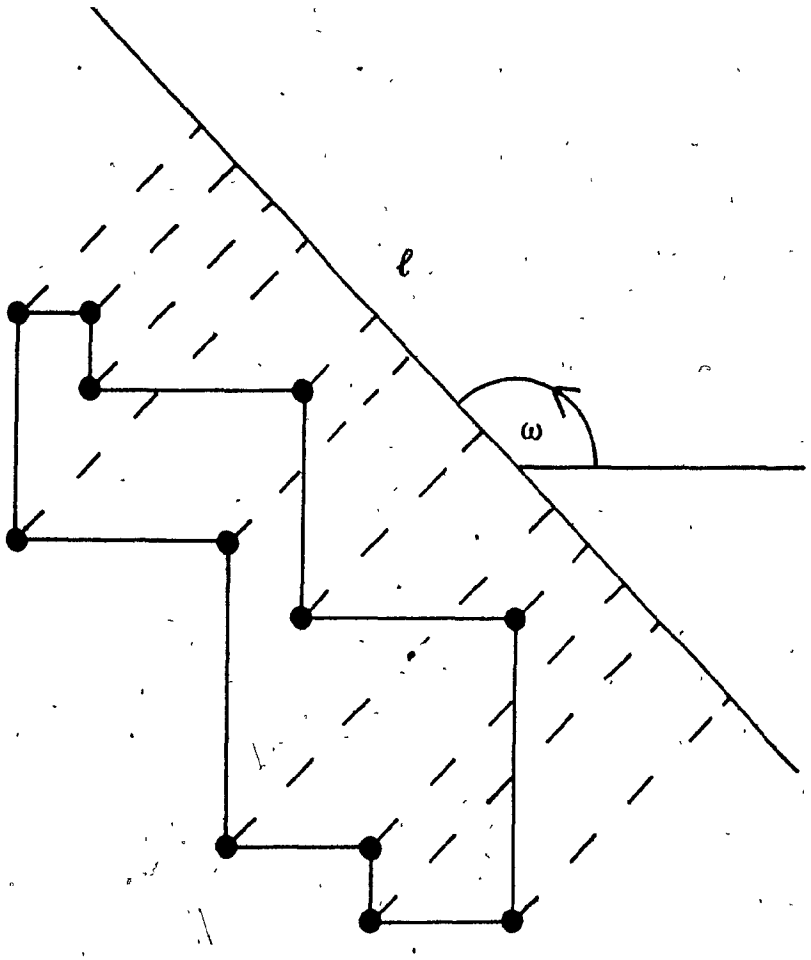


Figure 3.9

A polygon monotone with respect to all lines ℓ, whose counter clockwise angle formed with the x-axis is in the interval $[\frac{\pi}{2},\pi]$.

We call the x-axis and the y-axis the two *major axes* in rectilinear geometry and refer to the directions defined by them, as the *major directions*. From Property 3.5, it follows that a rectilinear polygon P is monotone, then it is monotone with respect to one of the two major axes. We will state this result in Corollary 3.1 for subsequent use. Furthermore, if P is monotone to any line not parallel to one of the two major axes, then P is monotone to all lines ℓ for which either ω in $[0, \frac{\pi}{2}]$ or ω is in $[\frac{\pi}{2}, \pi]$.

Corollary 3.1. *Any monotone rectilinear polygon is monotone with respect to at least one of the two major axes.*

We can therefore assume w.l.o.g. that every monotone rectilinear polygon is monotone with respect to the y-axis. Any rectilinear polygon monotone in the x-direction can be rotated in O(n) time to be monotone in the y-direction.

Let all vertical edges be directed from top to bottom, although this is strictly speaking a directed graph, we maintain the notion polygon. Given a rectilinear polygon P and a horizontal edge e we can denote the number of incoming (outgoing) edges into (out off) e by In(e), (Out(e)), respectively. As P is a simple polygon In(e) + Out(e) = 2, for all horizontal edges e in P. A bottom-edge e is called a *bottom-peak*, if Out(e) = 2 and is called a *bottom-valley* if In(e) = 2. A top-edge is called a *top-peak*, if In(e) = 2, and is called a *top-valley* is Out(e) = 2, see Figure 3.10 for illustrations. A *peak* is either a top-peak or a bottom peak. A *valley* is either a top-valley or a bottom valley.

Remark 3.1. *An edge $e_i = p_{i-1}p_i$ in a rectilinear polygon is a valley if the turns at p_{i-1} and at p_i are both right-turns. An edge e_i is a peak if the turns at p_{i-1} and p_i are both left-turns. Thus peaks and valleys can be detected by examining the labels of both edges adjacent to e_i. We define left peaks, right peaks and valleys, similarly. Top and bottom peaks are called horizontal peaks, left and right peaks are called vertical peaks.*

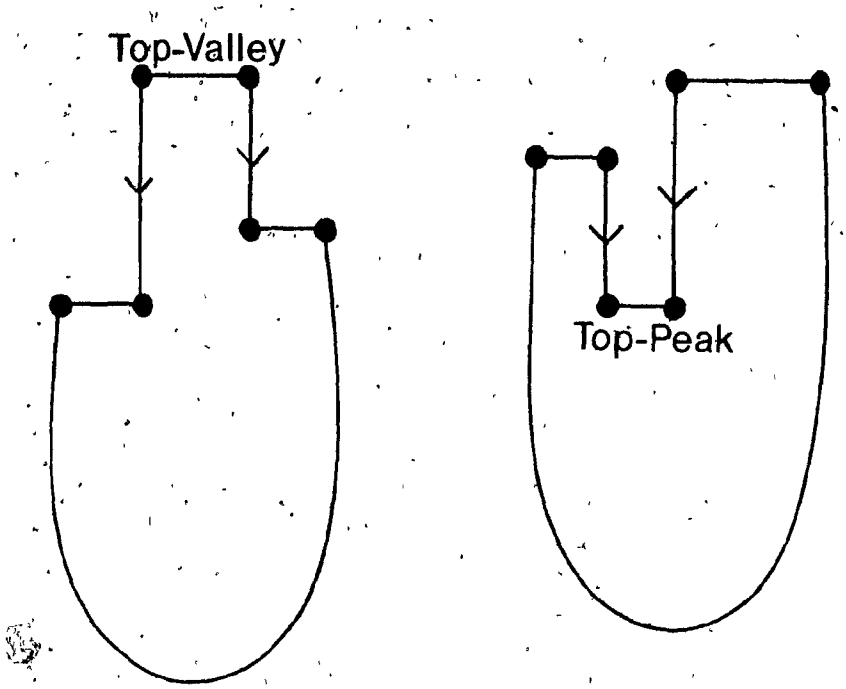


Figure 3.10

Top-valleys and top-peaks.

Property 3.6: *Let P be a rectilinear polygon, and T_v, T_p, be the number of top valleys and top peaks, respectively. Let B_p, B_v, be the number of bottom peaks and bottom valleys, respectively, then*

$$T_p = T_v - 1, B_p = B_v - 1.$$

Proof: We will show by induction on the number, n, of vertices in P that $T_p = T_v - 1$, the other part $B_p = B_v - 1$ is similar The class of rectilinear polygons with four vertices is the rectangle. For this polygon obviously $T_p = 0$ and $T_v = 1$, therefore $T_p = T_v - 1$.

Now we assume that P has more than four vertices. Then there exists at least one reflex vertex p_i and we can w.l o.g assume that p_i is as depicted in Figure 3.11, i.e. p_i is adjacent to both a left and a bottom edge. If the reflex vertex is not of the type as

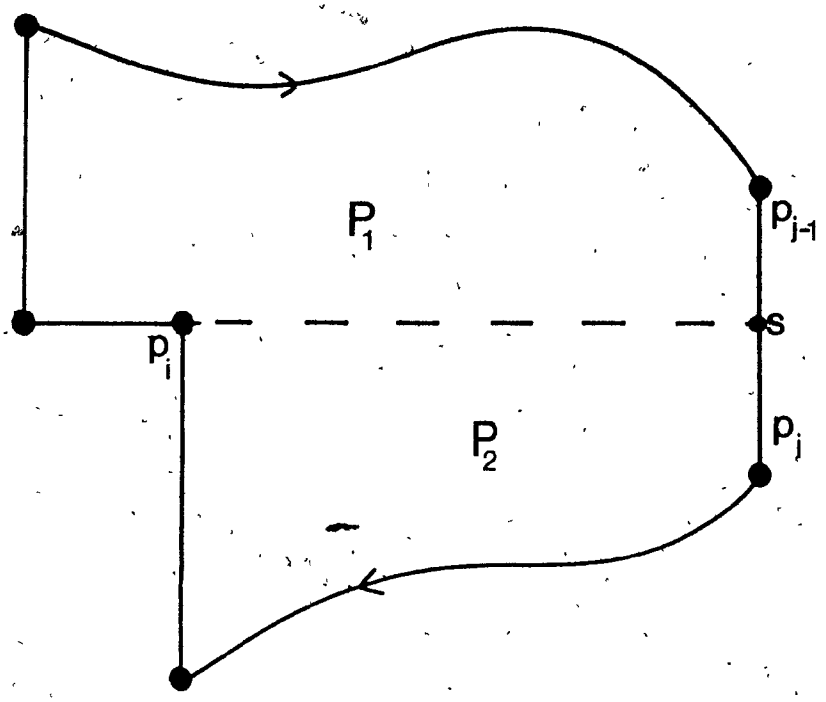


Figure 3.11

Partitioning a rectilinear polygon P into P_1, P_2 to show that the number of top-valleys minus one equals the number of top-peaks.

in the Figure 3.11 then a similar argument can be given. The horizontal extension of $\vec{p}_i$ towards the interior of P intersects P, the first such intersection point will be called s, say located at edge e_j. We define two polygons $P_1 = (s, p_{i+1}, p_{i+2}, \ldots, p_{j-1})$, and $P_2 := (s, p_j, p_{j+1}, \ldots, p_i)$ Let T_{v_i} denote the number of top-valleys is polygons P_i, i=1,2. Similarly T_{p_i}, B_{v_i} and B_{p_i} are defined. The newly created edge p_is is a top edge in P_2 having two outgoing edges, therefore $T_v = T_{v_1} + T_{v_2} - 1$, and $T_p = T_{p_1} + T_{p_2}$. Now we apply the induction hypothesis to conclude that $T_p = T_{p_1} + T_{p_2} = (T_{v_1} - 1) + (T_{v_2} - 1) = T_v - 1$. ■

Corollary 3.2. *The number of horizontal peaks and the number of horizontal valleys differ by exactly two. The total number of horizontal and vertical peaks combined differs from the total number of such valleys by exactly four.*

Theorem 3.1. *Let P be a rectilinear polygon.*

(i) P does not contain any horizontal peaks if and only if P is monotone with respect to the y-direction.

(ii) P does not contain any vertical peaks if and only if P is monotone with respect to the x-direction.

Proof: (i) We first notice that the monotonicity in the y-direction changes when a horizontal peak or horizontal valley is present. The extremal edges in the y-direction are both horizontal valleys. The y_{max}-edge is a top-edge and the y_{min}-edge a bottom-edge. Therefore by using Property 3.6, the two extremal chains connecting these edges contain a horizontal peak if and only if they contain a horizontal valley. As only horizontal peaks or valleys can change the monotonicity in the y-direction the result follows.

(ii) Similarly we can show that P is monotone in the x-direction if and only if P contains no vertical peaks. Alternatively, we can prove the result by rotating P by $\frac{\pi}{2}$ and applying (i) ■

Corollary 3.3. *A rectilinear polygon is monotone in both major directions if and only if it contains no peaks.*

We continue our investigation of monotone rectilinear polygons by characterizing rectilinearly convex polygons. Recall that a polygon P is called *rectilinearly convex* if every two points in P having the same horizontal or vertical coordinate are visible from each other. In the traditional sense of convexity the class of convex rectilinear polygons is restricted to containing only rectangles The notion rectilinearly convex is more appropriate in the context of rectilinear computational geometry.

Theorem 3.2. *A rectilinear polygon is rectilinearly convex if and only if it is monotone in both major directions.*

Proof: "=>" Proof by contradiction. We assume that P is not monotone in one of the major directions Thus P contains a peak, say the edge $e_i = p_{i-1}p_i$. W.l.o.g. we assume that e_i is a bottom peak Now we take any pair of points in the interiors of e_{i-1} and e_{i+1} having the same y-coordinate. As these two points are not (internally) visible the polygon P is not rectilinearly convex.

"<=" If a polygon is monotone with respect to some line ℓ then any two points with identical projections onto ℓ are visible. Thus as P is monotone in both major directions, any two points with the same x-coordinate or y-coordinate are visible and the result follows. ■

Theorem 3.3. *A rectilinear polygon P is rectilinearly convex if and only if*

(a) P is monotone with respect to all lines ℓ with ω in $[0, \frac{\pi}{2}]$ or

(b) P is monotone with respect to all lines ℓ with ω in $[\frac{\pi}{2}, \pi]$ or

(c) P is a monotone to only the two major directions and no other direction.

Proof: "<=" Follows from Theorem 3.2 and Property 3.5

"=>"By Theorem 3.2 and Corollary 3.3, a rectilinearly convex polygon P contains no peaks. By Property 3.6, P contains exactly two horizontal and two vertical valleys. These valleys are the four extremal edges in both major directions. These valleys are either adjacent edges in P, or they are connected by non-empty stairs. We distinguish between three cases: (a) The chains $C_{x_{min}y_{max}}$ and $C_{x_{max}y_{min}}$ are empty. In this case, all horizontal edges on $C_{y_{max}x_{max}}$ are top-edges. All horizontal-edges on $C_{y_{min}x_{min}}$ are bottom edges. The resulting polygon is monotone with respect to any direction in $[\frac{\pi}{2}, \pi]$,

(b) The chains $C_{y_{max}x_{max}}$ and $C_{y_{min}x_{min}}$ are empty. We get, analogously to (i), that the polygon is monotone with respect to all directions in $[0, \frac{\pi}{2}]$.

(c) In any other case, there exist two adjacent extremal chains $C_{i,j}$, $C_{j,k}$ which are non-empty. W.l.o.g. assume that j=1. The monotonicity changes at e_1 for any direction of monotonicity. Take any direction d in $(0, \frac{\pi}{2})$ then $C_{1,k}$ is not monotone with respect to that direction. Similarly for any direction in $(\frac{\pi}{2}, \pi)$ $C_{k,1}$ is not monotone. As P is rectilinearly convex we conclude that P is monotone in the two major directions and in no other direction. See Figure 3.12 as illustration. ■

Corollary 3.4. *Rectilinear star-shaped polygons are rectilinearly convex.*

As a consequence of Theorems 3.1 to 3.3 we obtain a linear-time algorithm for testing whether a rectilinear polygon is monotone. It will also report all directions of monotonicity. In O(n)-time we can test whether P contains any peaks. If P does not contain any horizontal peaks, then P is monotone in the y-direction. If P does not contain any vertical peaks, then P is monotone in the x-direction. If P is monotone in the x-direction and P is monotone in the y-direction then P is rectilinearly convex. In that case the algorithm performs a constant time test to determine whether P is monotone to all lines for which ω is in $[0, \frac{\pi}{2}]$ (or $[\frac{\pi}{2}, \pi]$) or P is monotone only to both major directions.

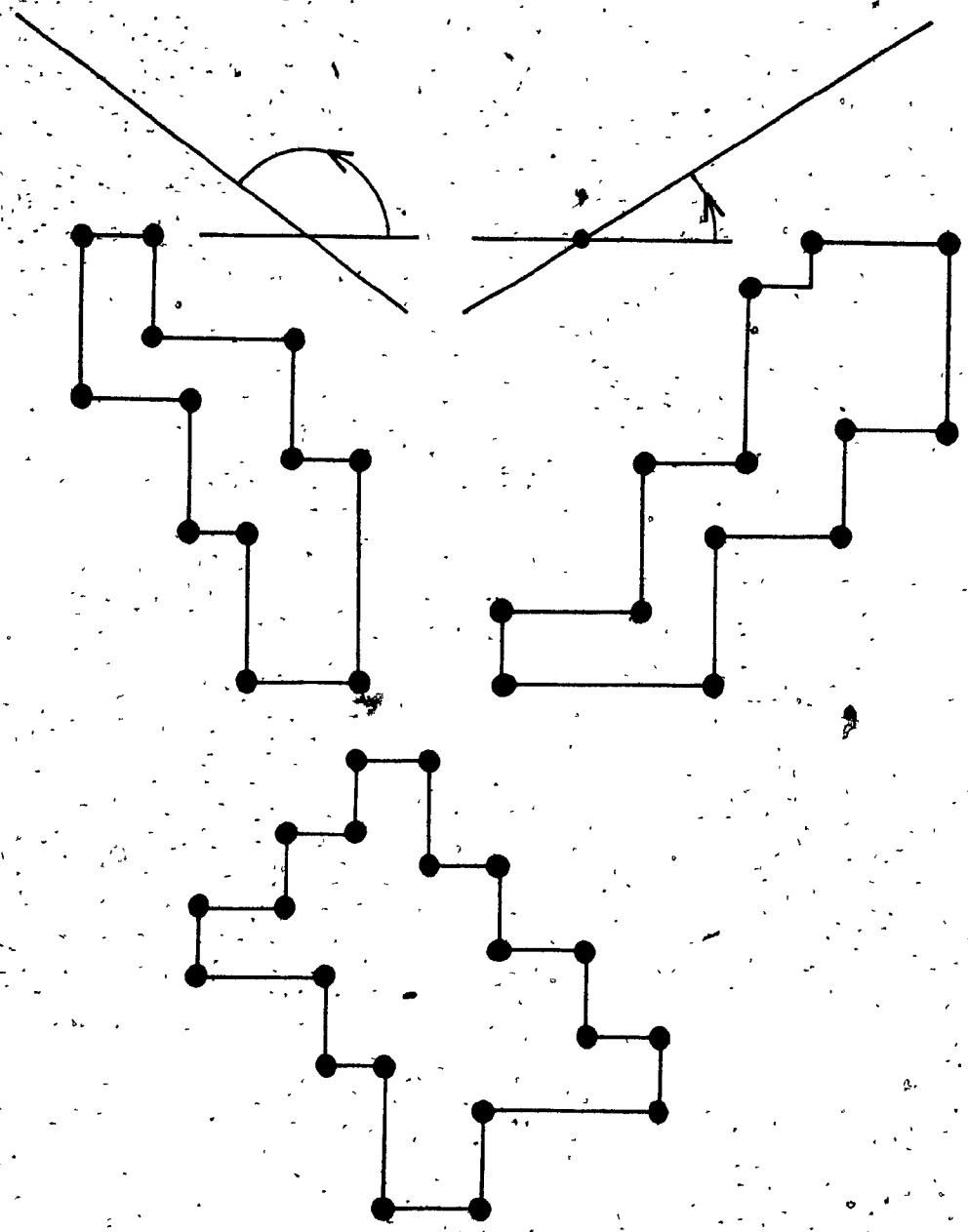


Figure 3.12

Monotonicity

Algorithm 3.2: Test for Monotonicity

Input: A rectilinear polygon P.

Output: All directions with respect to which P is monotone.

```
i := 1;
monotone_in_y := true;
monotone_in_x := true;
while i < = n and (monotone_in_x or monotone_in_y) do
    begin
        if e_i is a horizontal peak
            then
                monotone_in_y .= false
        else
            if e_i is a vertical peak
                then
                    monotone_in_x := false;
        i := i + 1
    end (* while *);
if monotone_in_y and monotone_in_x
    then
        begin
            if y_max-edge is adjacent to x_min-edge
                and y_min-edge is adjacent to x_max-edge
                then
                    "P is monotone to all lines ℓ with ω in [π/2,π]";
            if y_max-edge is adjacent to x_max-edge
                and y_min-edge is adjacent to x_min-edge
                then
                    "P is monotone to all lines ℓ with ω in [0,π/2]"
        end
else
    begin
        if monotone_in_y
            then "P is monotone in y";
        if monotone_in_x
            then "P is monotone in x"
    end.
```

Clearly the run-time of this algorithm is linear in the number of edges. We therefore have shown that:

Theorem 3.4. *All directions of monotonicity for a given rectilinear polygon can be found in linear time.*

Similarly to star-shaped polygons, we can also express monotone polygons by their label-sequences:

(i) P is rectilinearly convex iff s is in $(0,1)^+(2,1)^*(2,3)^+(4,3)^*$.

(ii) P is monotone with respect to all lines between $\frac{\pi}{2}$ and π iff s is in $(0,1)^+(2,3)^+$.

(iii) P is monotone with respect to all lines between 0 and $\frac{\pi}{2}$ iff s is in $(0,1)(2,1)^*(2,3)(4,3)^*$.

(iv) P is monotone with respect to the y-direction iff $((0,1)(2,1)^*)^+((2,3)(4,3)^*)^+$.

The monotonicity in the x-direction can be expressed analogously.

3.2.4. Edge-Visibility Test

Recall that a polygon P is weakly visible from a specified edge e_i if for each point p in P there exists a point q on e_i that sees p. Avis and Toussaint presented a linear, and thus optimal, algorithm for determining whether or not a given polygon is weakly visible from a specified edge (see [AT81a]). An interesting problem which is directly related has been posed by Toussaint [To80]: Given a simple polygon, does there exist an edge from which the polygon is edge-visible? In case that such an edge exists the polygon is said to be *(weakly) edge-visible.*

A straight forward approach for testing a polygon for edge-visibility is to apply the algorithm of Avis and Toussaint to each edge in turn, yielding an $O(n^2)$ algorithm. It is an open problem whether a better upper bound can be achieved. Here we present a linear-time algorithm to test a given rectilinear polygon for edge-visibility. To achieve this we will show that any edge-visible rectilinear polygon admits a partitioning into rectilinear polygons called histograms and pyramids, whose structures we will next define.

Edelsbrunner *et al.* use the term *vertical histogram* to denote a rectilinear polygon that has one horizontal edge, called the base, equal in length to the sum of lengths of all the other horizontal edges [EOW83]. Analogously, horizontal histograms can be defined. A *histogram* is a rectilinear polygon which is a horizontal or vertical histogram. Pyramids have been defined in [SaT81] as a rectilinear, edge-visible, star-shaped polygons with two (possibly empty) stairs. Here we give an alternate shorter definition. A *pyramid* is a rectilinearly convex histogram. It is readily verified that the two definitions are equivalent.

Property 3.7. *Any histogram is weakly visible from its base.*

Proof: Follows immediately from the definition of histogram. ■

We now give a characterization of edge-visible rectilinear polygons.

Property 3.8. *A rectilinear polygon P is edge-visible if and only if there exists an edge e_i such that P can be partitioned into two pyramids, one of which is completely visible from p_i, the other completely visible from p_{i-1}, and into a histogram with base e_i.*

Proof: "<=" Using the assumption, P can be partitioned into three components: two pyramids and a histogram. By assumption the two pyramids are completely visible from e_i. By Property 3.7 the histogram is weakly visible from e_i. Therefore the entire polygon P is weakly visible from e_i.

"=>" We assume that the edge from which e_i is weakly visible is a horizontal top-edge. Extend e_i to the left and right towards the interior until P is intersected at points u,r, respectively. This is illustrated in Figure 3.13. Consider now the resulting polygon $P_r=\{p_i,...,r\}$ of points in P which are not below p_i and not to the left of p_i. In

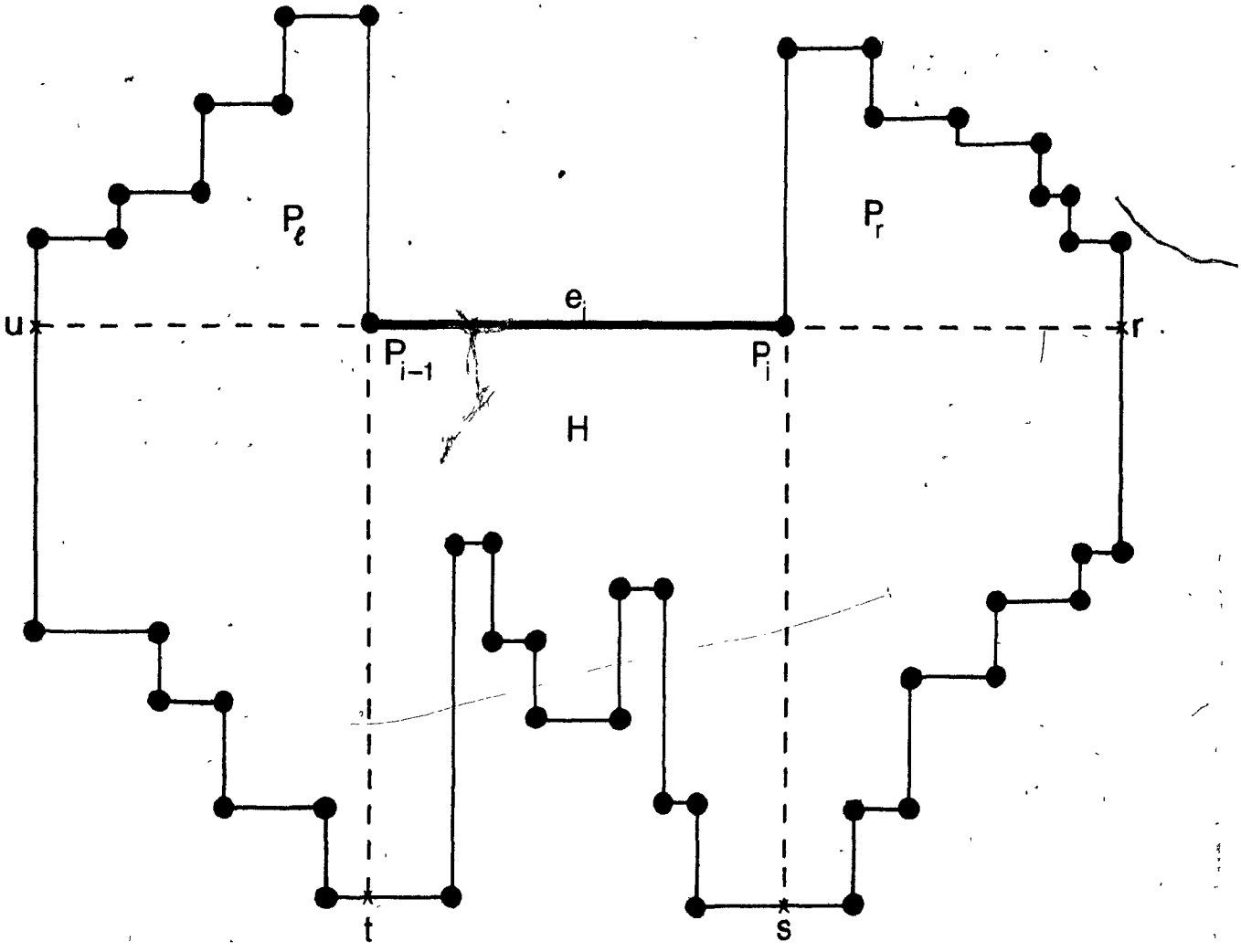


Figure 3.13

An edge-visible rectilinear polygon

the trivial case $P_r=\{p_i\}$, otherwise we show that P_r is a pyramid with one empty stair. For this observe that no point in the interior of P_r is visible from any point except from p_i. As P is weakly visible from e_i, P_r must be completely visible from p_i. P_r is therefore a rectilinear star-shaped polygon. We know from previous sections that rectilinear star-shaped polygons consist of four stairs each of which may be empty. The vertex p_i has minimum y-coordinate and minimum x-coordinate within P_r. As it is located inside the kernel of P_r, three of the four stairs are empty and thus $p_{i+1},...,r$ is a single stair. We summarize the intermediate result that P_r is completely visible from p_i and the chain $p_{i+1},_r$ is a single stair. By an analogous argument follows that u,...,p_{i-2} is a stair completely visible from p_{i-1}. Now consider the remaining unexamined polygonal chain $C_{r,u}$. By Theorem 2.3, all horizontal edges on $C_{r,u}$ are bottom-edges. Any vertical peak, however, contains a top-edge and a bottom-edge. Consequently $C_{r,u}$ cannot possibly contain any vertical peaks. By the characterization of monotone polygons given in Theorem 3.1, the chain $C_{r,u}$ is therefore monotone in the x-direction. We now extend the edge e_{i+1} towards the interior of P and denote the intersection point with the boundary of P by s, see Figure 3.13. In a similar fashion create the Steiner point t by extending p_{i-1}. This defines two (possibly empty) chains, one from r to s, called $C_{r,s}$ and the other from s to u, called C_{b_r},u. By Theorem 2.3 all vertical edges in $C_{r,s}$ are right-edges. The chain $C_{r,s}$ consists of right-edges and bottom-edges and is therefore monotone in both major directions and is thus a stair. Combining the above results yields that $\{p_i,p_{i+1},...,s\}$ is a pyramid consisting of two stairs, both completely visible from p_i. Similarly it can be shown that $\{t,...,p_{i-2}\}$ is a pyramid completely visible from p_{i-1} It remains to be shown that the polygon H:=$\{p_{i-1},p_i,s,\ ,t\}$ is a histogram By Theorem 2.3, $C_{s,t}$ contains no top-edges and thus does not contains any vertical peak. This implies that H is monotone is the x-direction. In this case the length of the edge e_i is equal to the length of the other extremal chain $C_{s,t}$, implying that H is a histogram. The histogram and the two

pyramids cover P and do not intersect each other properly. Therefore P is partitioned into two pyramids visible from p_i and p_{i-1}, respectively and a histogram whose base is e_i. ■

Corollary 3.5. *Edge-visible rectilinear polygons are monotone.*

We now present a linear-time algorithm for testing whether a rectilinear polygon is edge-visible. The algorithm is based on the characterization of edge-visible polygons as given above. We saw that rectilinear edge-visible polygons are monotone in a direction parallel to the edge from which they are weakly visible. Let us assume that the edge of visibility, if it exists at all, is a horizontal edge In case that this edge is vertical, rotate P by $\frac{\pi}{2}$. Let ℓ be the index of the edge with minimum x-coordinate and r the index of the edge with maximum x-coordinate. The two resulting upper and lower monotone chains are $C_{\ell,r}$ and $C_{r,\ell}$, respectively. Notice that a rectilinear polygon can be reconstructed if only a list of its vertical edges is given.

Algorithm 3.3: Test for Edge-Visibility

Input: A rectilinear polygon P monotone in the x-direction. Should P be monotone in the y-direction, P must first be rotated by $\frac{\pi}{2}$.

Output: All edges from which P is weakly visible; no edge is printed in case P is not edge-visible.

```
create a list L of all vertical edges in the upper
   and lower chains C_{ℓ,r} and C_{r,ℓ} sorted by x-coordinate;
scan L from left to right until the first right edge,
   say e_j, is encountered,
if {during the scan no edge from the upper chain
    has been encountered}
    then
       candidate_{up1} := p_ℓ
else
   if e_j is on the upper chain
      then
         candidate_{up1} := p_j
if {during the scan no edge from the lower chain
    has been encountered}
```

```
    then
       candidate_low1 := p_{l-1}
  else
    if e_j is on the lower chain
       then
          candidate_low1 := p_{j-1};
  call the rectilinear polygon constructed
     of all edges scanned so far P_ℓ;
  scan L from right to left until the first left edge,
     say e_j is encountered;
  if {during the scan no edge from the upper chain
     has been encountered}
     then
        candidate_up2 := p_{r-1}
  else
     if e_j is on the upper chain
        then
           candidate_up2 := p_{j-1}
  if {during the scan no edge from the lower chain}
     has been encountered}
     then
        candidate_low2 := p_r
  else
     if e_j is on the lower chain
        then
           candidate_low2 := p_j;
  P_r := the rectilinear polygon constructed of all edges scanned;

  for k:=up, low do
     begin
        if candidate_k1, candidate_k2 are endpoints of the same edge, say e_i
           then
              if candidate_k1 is in kernel(P_ℓ)
                 and candidate_k2 is in kernel(P_r)
                 then P is edge-visible from e_i;
     end.
```

Theorem 3.5. *In linear time, a rectilinear polygon can be tested for edge-visibility.*

Proof: Initially the rectilinear n-vertex polygon P is tested for monotonicity in the x-direction or y-direction. By Theorem 3.4 this requires at most O(n) time. Assume that P is monotone in the x-direction. Both the upper chain as well as the lower chain of vertical edges of P are sorted by their x-coordinates Merging these chains to obtain L requires therefore O(n) time. To find the candidate vertices for an edge e_i from which P is edge-visible, this list is scanned at most twice. Subsequently the two pyramids P_ℓ

and P_r located to the left and to the right of e_i are tested for complete visibility from the respective endpoints of e_i. This test is performed by first determining the kernel of the pyramids (see Chapter 3.2.2). Subsequently the endpoints can be tested for point inclusion in the kernel. Both operations require at most linear time. The algorithm therefore locates both pyramids and also checks whether they are visible from the respective endpoints of e_i, which is performed in O(n) time. By the initial test P is monotone in the x-direction and therefore all vertices which are not located in either of the pyramids, are vertices of the histogram H. This completes the proof. ■

3.2.5. Related Concepts

In [OR82a] O'Rourke introduced the related concept of the signature of a curve Γ as a function associating with each point p of Γ the length of Γ to the left of or on the line tangent to Γ at this point. He shows that for arbitrary simple closed curves the signature does not uniquely describe the curve. However, for the case of rectilinear curves the curve is uniquely determined by the values of the signature. This again shows that rectilinear polygons are more structured than arbitrary simple polygons. It might be fruitful to test both the signature and the label-sequence of an object as combined features in a pattern recognition system.

Recently Chazelle et al. [Chl83] defined the concept of the sinuosity s of a simple polygon. We will illustrate the concepts introduced by them using the terminology developed in this thesis. They decompose polygonal chains into subchains according to the following rule. Traverse the polygon and start a new chain every time a full 2π-turn (-2π-turn) is completed. The *sinuosity* of a polygon is defined as the number of such chains obtained. We now state their main result:

Theorem 3.6. *It is possible to triangulate a simple n-sided polygon in O(n log s) time and O(n) space* [Chl83].

Proof: The proof can be found in [Chl83].■

This result is nice, since by using their technique very efficient structure dependent triangulations can be obtained. The worst-case performance of their algorithm is nevertheless O(n log n) for certain polygons and thus the problem of whether an arbitrary simple n-vertex polygon can be triangulated in linear time remains open. It is interesting to note that these two concepts are related. In fact we can derive the sinuosity from the label-sequence, since the labeling-sequence contains the sinuosity information

The computational geometry literature contains many algorithms which perform a given task efficiently provided the input polygons have certain structural properties. In the context of this thesis, quadrilaterization algorithms discussed in Chapter 6 serve as example. The main part of this chapter therefore dealt with the design of efficient, linear-time algorithms for testing a given rectilinear polygon for such structural properties, as star-shapedness, monotonicity or edge-visibility.

Chapter 4
Hidden-Line Removal in Rectilinear Polygons

4.1. Literature

Problem description: For a given scene and a given position called the *viewpoint* and a viewing direction of an observer, one wants to eliminate all edges (faces) or parts of edges (faces) which the observer cannot see.

This hidden-line (surface) elimination problem is of fundamental importance in the area of computer graphics where a frequently performed task is to render a realistic picture of an object displayed on a graphics device. The practical relevance of the problem has stimulated the development of a large number of hidden-line and hidden-surface elimination algorithms. For survey articles on hidden-line and hidden-surface algorithms see [SSS74, OWW82].

From a computational geometry point-of-view, the precision of any computation and thus that of the solution, should not depend on the resolution of the graphics output device. We therefore restrict our attention to *object-space* algorithms, i.e. algorithms which perform all computations in the same coordinate-system in which objects are stored. We briefly survey the existing results according to the following criteria:

(a) What is the dimensionality of the objects and the object-space?

(b) How many objects are in the given scene?

(c) What is the specific nature of the underlying objects?

We will first sketch results obtained for 2-dimensional projections of 3-dimensional scenes. Ottmann et al. [OWW82] showed how a plane-sweep techniques can be

employed to eliminate all hidden surfaces of a given 3-dimensional scene of arbitrary polyhedra consisting of n edges. If k denotes the number of edge intersections in the specific 2-dimensional projection, then their algorithm exhibits an O((n+k) log n) time-complexity using O(n log n) space. Other object-space algorithms restrict the class of input polygons considered. Yao defines *elementary objects* as the Cartesian product of a simple polygon with an interval along the z-axis. The idea of Yao's [Ya80] algorithm is to assign priority numbers to each one of n given faces and subsequently defining an ordering among them. The final step in her solution to the hidden-surface problem is simply to display the faces in this order, thereby achieving the desired overlay effect. Yao proves that such an ordering can be found in O(n log n)-time, given the restriction on the class of polygons considered

In addition to restricting the class of polygons, one can also restrict the number of elements in the scene to consider the hidden-line problem for single objects. Rappaport [Ra82] considers single *monotone slabs* as input polygons. A monotone slab is the Cartesian product of a monotone polygon along the z-axis. Using this restriction, he obtains a linear-time algorithm for eliminating hidden-surfaces in a monotone slab. ElGindy and Avis [ElA81] presented a hidden-line elimination algorithm for single simple polygons in the plane Their algorithm runs in linear time, but utilizes three stacks and is rather complicated. In this thesis we present a simple and efficient hidden-line elimination algorithm for rectilinear polygons. Our algorithm utilizes one stack and exhibits a linear run-time [Sa83]. It is based on the labeling-scheme as introduced in Chapter 2. Recently, Lee independently obtained a similar algorithm [Le83]. Rappaport and Toussaint restricted the problem further to finding a simple hidden-line algorithm for star-shaped polygons [RaT83].

4.2. Models of Visibility

Following the terminology of Yao [Ya80], we characterize visibility of objects by

adopting either a perspective model or a parallel model.

4.2.1. Perspective Model

In the *perspective model,* the viewpoint is located anywhere on the plane. A solution to the hidden-line problem consists of finding all those points of P that are visible from the viewpoint. As the viewpoint might be located either inside or outside the polygon, we distinguish between two kinds of visibility: internal and external visibility.

Internal visibility: We say that a line-segment *lies inside* P if the interior of the line-segment lies in the interior of P. Two points are said to be *internally visible* if the line-segment joining them lies inside P. The hidden-line problem from a point v inside P is to determine all those points of P that are internally visible from v.

External visibility: We say that a line-segment *lies outside* P if the interior of the line-segment lies in the exterior of P. Two points are said to be *externally visible* if the line-segment joining them lies outside P. The hidden-line problem from a point v outside P is to determine all those points of P that are externally visible from v.

It is known that the internal and external hidden-line problems are related in that usually any algorithm capable of solving one hidden-line problem is, in principle, capable of solving the other (only minor changes to the algorithms are necessary).

4.2.2. Parallel Model

In the *parallel model,* we assume rays to emanate from a source located at infinity. All rays form a fixed, clockwise angle θ with the x-axis The interior of the ray ∞p from ∞ to a point p, is defined as the open half-line from ∞ to p, excluding the point p. A point p is said to be *visible from* ∞ if the ray ∞p lies outside P. The hidden-line problem in the parallel model is to determine all those points of P that are visible from ∞. Alternatively we refer to this problem as the hidden-line problem form a point at

infinity.

We will say that a point is *visible* if it is clear from the context whether internal visibility, external visibility, or visibility from ∞ is meant. Similarly an edge e_i is said to be visible if there exists a point on e_i that is visible. An edge e_i is completely visible if each point on e_i is visible.

As is commonly done, we will give the solution for the hidden-line problem as a polygonal chain, referred to as the *visibility chain VC*. VC is obtained by connecting all segments on the boundary of P that contain exclusively visible points. In VC the polygonal order is preserved.

4.2.3. Overview of Algorithms

It has been shown that hidden-line elimination in the parallel model and perspective model are equivalent in the sense that a solution obtained for one model can be transformed into a solution for the other model [NeS79]. However this is not the case in rectilinear geometry, as the transformations applied do not preserve the rectilinearity. For solving the hidden-line problems in rectilinear geometry, we give efficient and conceptually simple algorithms in this chapter.

First, we show how to solve the hidden-line problem for rectilinear polygons in the parallel model. The main points will be illustrated for the case that all rays are parallel to the x-axis. We extend these results to points at infinity not necessarily parallel to one of the axes. Then we show how to solve the hidden-line problem in the perspective model for the case that the viewpoint is located inside P. We will place a rectilinear coordinate system at the viewpoint and let the axes of the coordinate system be parallel to the edges of the rectilinear polygon. Hereby an arbitrary rectilinear polygon can be split into four chains. A solution for the hidden-line problem in the perspective model can thus be obtained by determining the visibility for each chain.

4.3. Algorithm for Hidden-line Elimination: Parallel Model

4.3.1. Rays Parallel to one of the Major Axes

Throughout this section, we will assume that all rays are parallel to one of the major axes. We assume w.l.o.g. that this axis is the x-axis. Phrased in a different way, we say that the point of visibility is located at infinity, ∞, on the positive x-axis. This case is not only attractive for illustrating the main points nicely, but also it is of relevance for rectilinear geometry, as will be demonstrated below. The following Lemma 4.1 follows directly from the definition of visibility from ∞.

Lemma 4.1. *The visibility chain is rectilinear if the rays of visibility are parallel to one of the axes.*

Remark 4.1. *The converse of Lemma 4.1 is not necessarily true as illustrated in Figure 4.1. The polygonal chain from the y_{max}-edge to the x_{max}-edge is monotone with respect to both the x and y-direction and is thus identical to the visibility chain.*

We have defined visibility of a point with respect to a polygon P. This notion is now generalized to *visibility* of a point with respect to a *polygonal chain*. A point p is visible from ∞, with respect to a polygonal chain $C'_{1,t}$, if the interior of the half-open line-segment from ∞ to p does not intersect any edge in $C'_{1,t}$.

We now define when a *point is behind* a polygonal chain. A point p is behind a polygonal chain C if the directed open half-line ∞p intersects C. We extend this definition to when an *edge is behind* a chain C. An edge e_t is behind a chain C if for all points p on e_t, p is behind C. If for a point p the directed open half-line from ∞ through p first intersects p before intersecting C, we say that p is *in front* of C. An edge e_t is in *front of* a chain C, if for all points p on e_t, p is in front of C. The definitions can be canonically extended to edge-segments. In the following Lemma 4.2, we state a

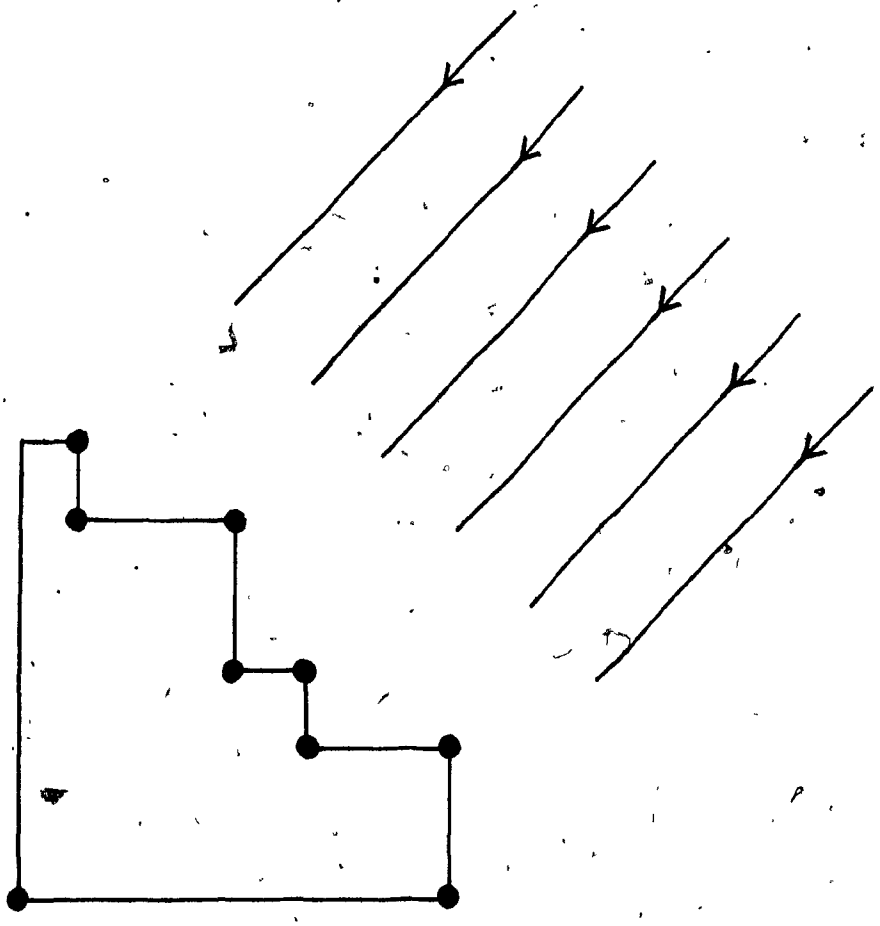


Figure 4.1

The visibility chain is rectilinear although the rays are not parallel to one of the axes.

necessary condition for an edge to be visible.

Lemma 4.2. *If a vertical edge is visible from a point located on the positive x-axis, then its label is 1.*

Proof: The result follows directly from Theorem 2.1 in Chapter 2. ■

The analogous results derived for points of visibility located at infinity on the y-axis or at $-\infty$ on the x-axis, are summarized in Figure 4.2. On the basis of Lemma 4.2 we can design the skeleton of an algorithm for computing the visibility chain from ∞.

Direction of visibility	Labels of visible edges
$+\infty$ on x-axis	1
$+\infty$ on y-axis	2
$-\infty$ on x-axis	3
$+\infty$ on y-axis	4,0

Figure 4.2

Relationship between direction of visibility and labels of visible edges.

Algorithm 4.1: Hidden-Line Problem in the Parallel Model

Input: A rectilinear polygon P.

Output: The visibility chain as seen from a point at infinity.

```
VC:= empty; (* initialize the visibility chain *)
for each 1-edge e_i in P do
   if e_i is not behind VC
      then
         begin
            if any portion of VC is obscured by e_i
              then remove that portion from VC;
            augment VC by the portion of e_i that is visible
              with respect to C_{1,i}
         end.
```

Algorithm 4.1 examines the 1-edges of P according to their polygonal order. Elaboration is required on how to implement the step testing whether an edge e_i is behind the visibility chain so far constructed. Figure 4.3 illustrates that a test based exclusively on the x-coordinates and y-coordinates of TOP and p_{i-1}, p_i is not sufficient. The test of whether e_i is behind VC can, however, be performed by a binary search on VC. This is possible because at any time during the execution of the algorithm, the visibility chain VC is monotone. The proof for the monotonicity of VC is similar to that of Lemma 4.4 stated below and we can omit it here. We conclude that the worst case run-time of Algorithm 4.1, if so implemented, is O(n + k log k), where k is the number of 1-edges in P. The algorithm, performs well if k is small compared to n. However, in case that k=O(n) the worst-case time complexity is O(n log n).

To find a worst-case linear-time algorithm we utilize the winding information provided by the labeling scheme. For this purpose, we introduce the concept of an antagonist for an edge. Consider a point p visible from the viewpoint v and located on edge e_i. Let a>i denote the smallest edge-index s.t. the edge e_a crosses the line vp. We call e_a *antagonist* for e_i. We note that this definition of antagonist is independent of

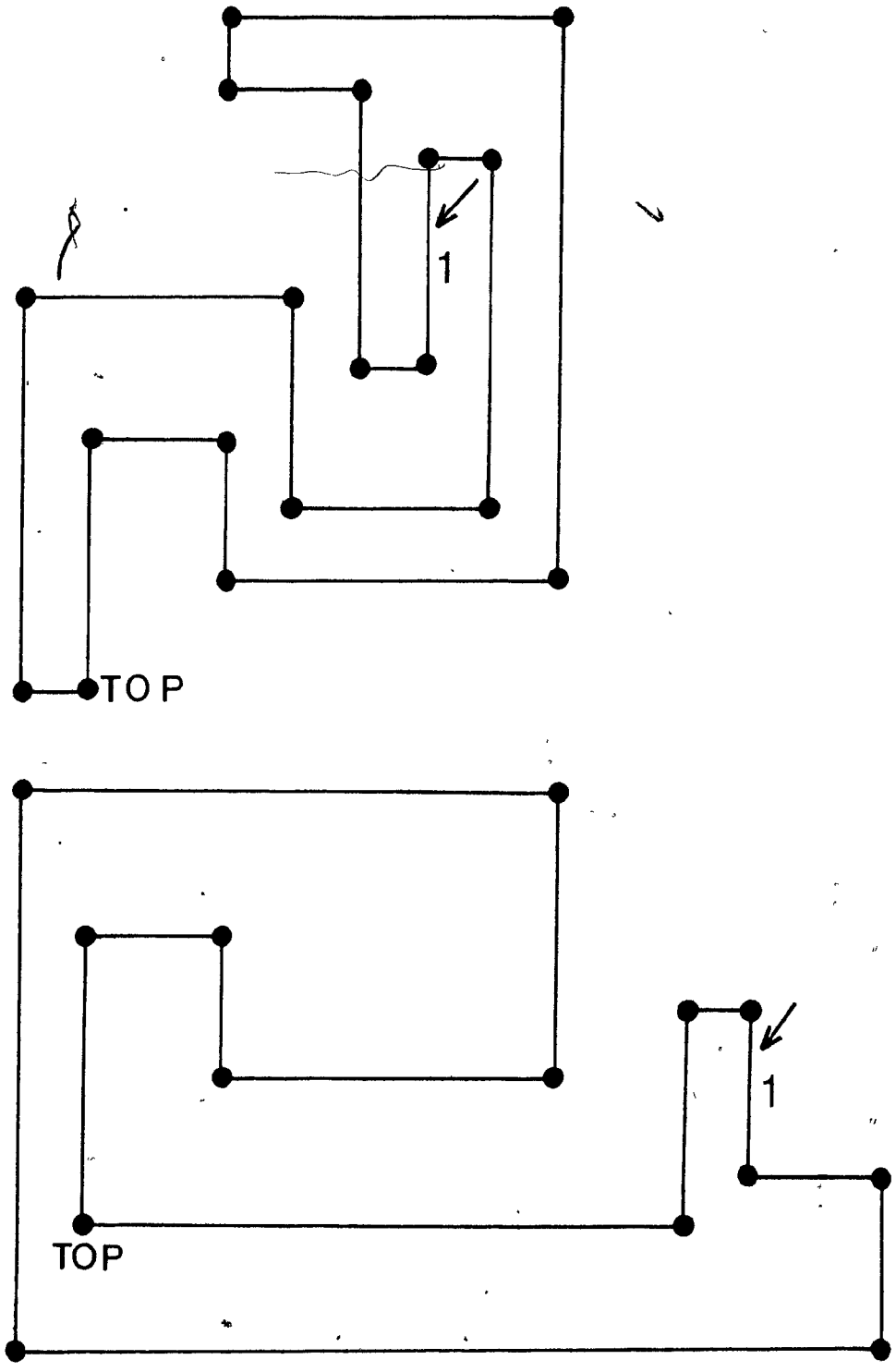


Figure 4.3

Testing the top_of_stack called TOP and the 1-edge e_i does not give enough information to determine whether or not e_i is hidden.

the model of visibility. Furthermore, one edge may have several antagonists and several edges may share the same antagonist (see Figure 4.4). For a point located on a 1-edge, we can define a unique antagonist (if it exists at all). The antagonist of a point q located on a 1-edge e_i is the antagonist of e_i with smallest index that has the same y-coordinate as q. Lemma 4.3 characterizes antagonists.

Lemma 4.3. *Antagonists for 1-edges are −1-edges.*

Proof: With the properties of the labeling scheme derived in Lemma 4.2 and Theorem 2.1. the result follows immediately. ■

Using Lemmas 4.2 and 4.3, we are able to design a linear-time algorithm, Algorithm 4.2, for solving the hidden-line problem from a viewpoint located at ∞ on the x-axis. The algorithm efficiently skips all edges not labeled either 1 or −1. For ease of description, we assume that each edge e_i of P is already labeled with ℓ_i. We maintain a stack, S, containing the visibility chain of all edges so far considered. The points in S are stored in order of decreasing y-coordinates, where the lowest point is at the top of the stack, referred to as *TOP*. We say that a point p is to the ***right of*** a point q, if the x-coordinate of p is greater than the x-coordinate of q. Similarly, we use the expressions to the *left* of, *higher* than, and *lower* than. For two given points p, q, we have used p#q to denote the point whose x-coordinate is that of p and whose y-coordinate is that of q (see Figure 3.6). To determine the visibility chain from ∞ we only need to examine the chain $C_{1,m}$ connecting p_1 to the vertex p_m, with minimum y-coordinate.

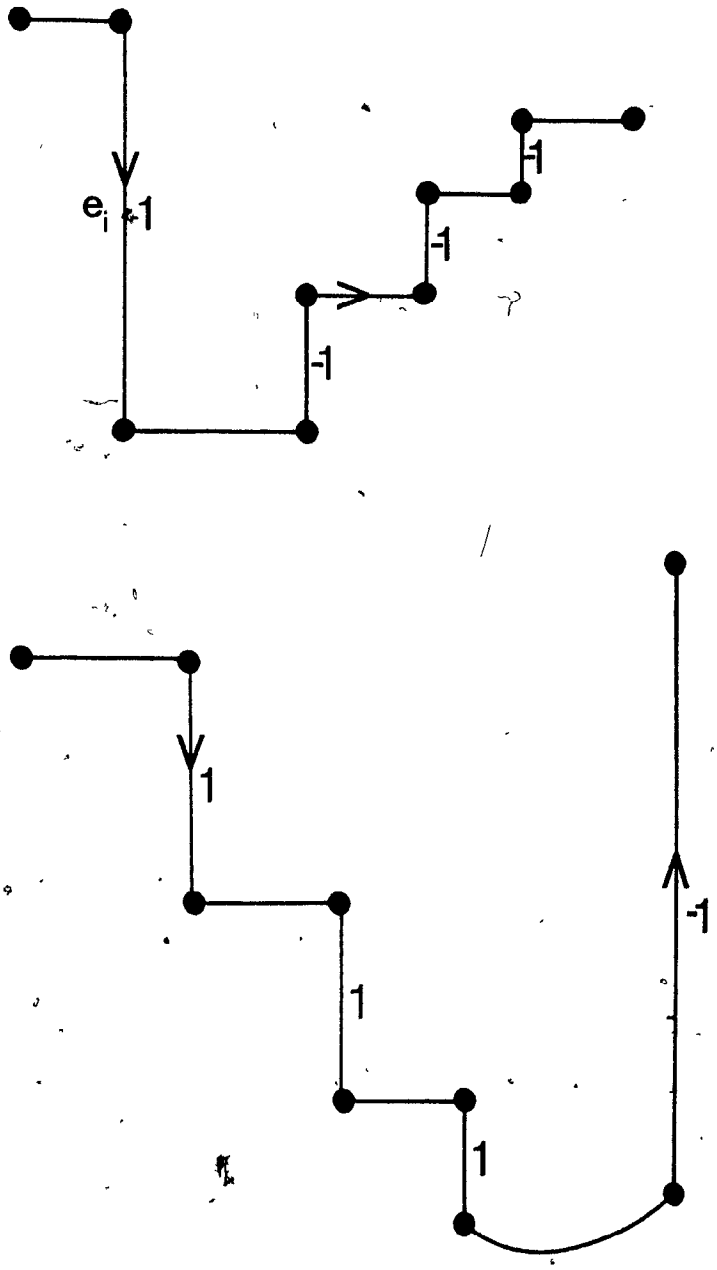


Figure 4.4

Antagonists

Algorithm 4.2: Hidden-Line Problem in Parallel Model

Input: The polygonal chain $C_{y_{max}}, C_{y_{min}} = C_{1,m}$ of a rectilinear polygon P.

Output: The visibility chain as seen from a point at infinity.

```
(* stack initialization *)
S:= empty,
TOP = p_1;
for i := 2 to m step 2 do
    if ℓ_i = 1 and p_i is below TOP and p_{i-1} is not below TOP
      then
        begin
          (* The visible part of e_i is pushed *)
          if TOP<> p_i#TOP then PUSH(p_i#TOP);
          PUSH(p_i)
        end
    else
      if ℓ_i = -1 and p_{i-1} is not higher than TOP
        then (* e_i may be antagonist *)
          begin
            while p_i is to the right of TOP and above TOP do
              begin
                (* shrink the visibility chain to the level of p_i *)
                LASTTOP:= TOP;
                POP;
              end
              if p_i is below TOP
                then PUSH(LASTTOP#p_i)
          end.
```

An example illustrating Algorithm 4.2 is given in Figure 4.5. To prove the correctness of Algorithm 4.2, we introduce some additional notation. VC(i) is defined as the visibility chain constructed by the algorithm *after* edge e_i has been processed. We also use *at time i* to denote the execution of the loop-body for edge e_i. For ease of notation, let TOP(i) contain the value of TOP after edge e_i has been processed. In an implementation of Algorithm 4.2 TOP is a single pointer variable pointing the top_of_stack.

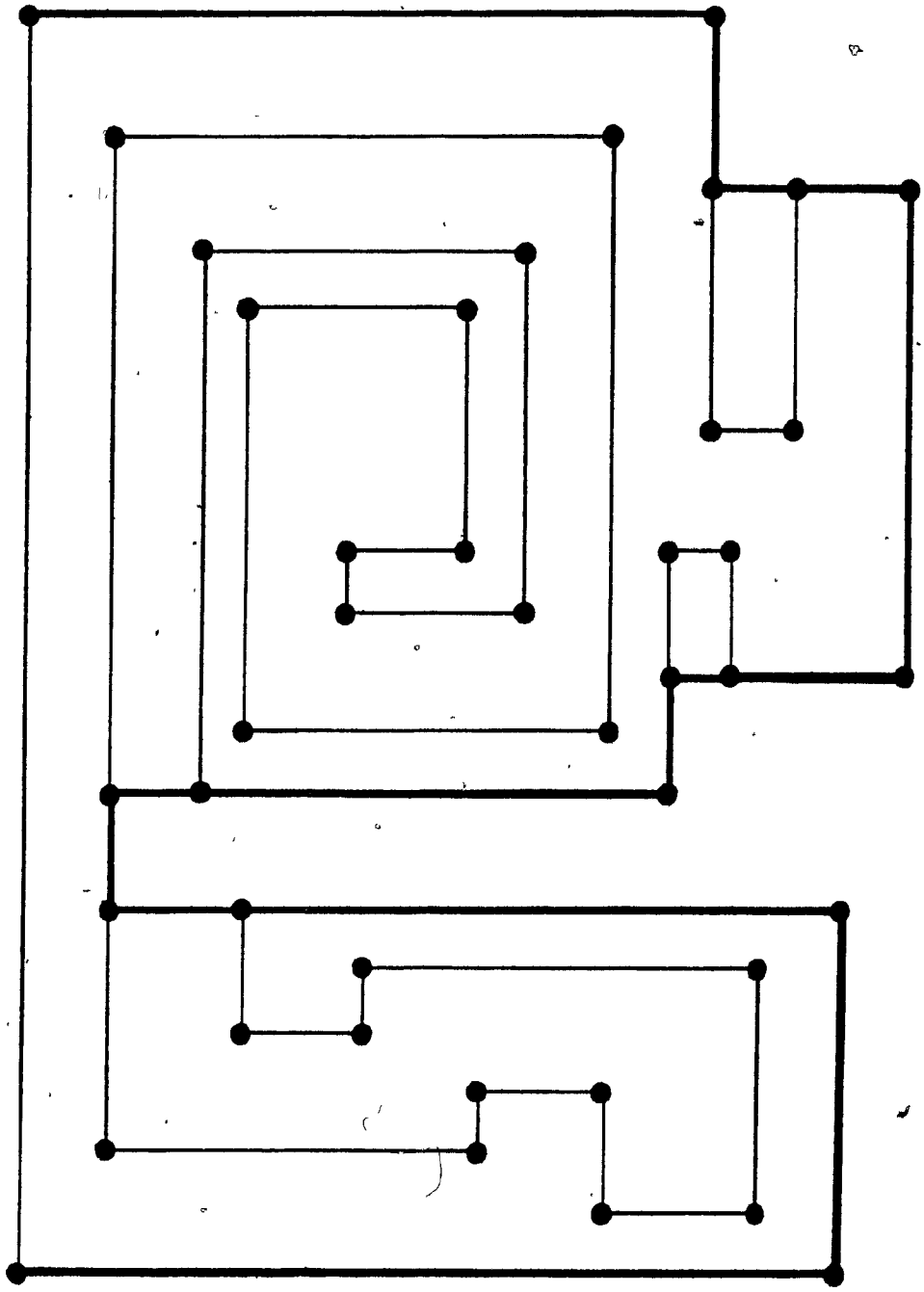


Figure 4.5

Bold lines represent the solution to the hidden-line problem from a view-point located at $(+\infty, 0)$.

Lemma 4.4. *The chain VC(i) is monotonic in the y-direction.*

Proof: Initially, VC(2) contains the single edge e_2 and is thus monotone in the y-direction. Subchains of monotone chains are monotone. Thus if, at any time i, the stack content is popped, clearly VC(i) remains monotone. We assume that for i>3, VC(i−2) is monotone. We now examine the case that the visibility chain is augmented at time i. In that case, the first operation, PUSH(p_i,#TOP), is executed. This guarantees that VC is contiguous. As p_i is below TOP(i−2) the second operation, PUSH(I_i'), ensure that VC(i) also remains monotone ■

Corollary 4.1. *The visibility chain VC(i) constructed by Algorithm 4.2 is monotone and rectilinear.*

Corollary 4.1 and Lemma 4.1 illustrate that if the viewpoint is located on one of the axis, then all chains created are rectilinear. This visibility problem is therefore of particular relevance for rectilinear computational geometry. In Chapter 5, when discussing rectilinear hull problems, we will refer back to this point.

Lemma 4.5. *Let p_j be the vertex with lowest y-value on the chain $C_{1,j}$ then TOP(j)=p_j*

Proof: We claim that p_j is visible with respect to $C_{1,j}$. Refer to Figure 4.6. As p_j is the lowest vertex in $C_{1,j}$ it is always possible to complete $C_{1,j}$ to form a simple polygon $\{p_1,..,p_j,q,r\}$ by adding two more edges p_jq and qr without intersecting any edge so far considered. By Property 2.3, the edge qr is a 3-edge and p_jq is a 2-edge, implying that e_j is a 1-edge. Furthermore p_j is lower than TOP(j−2) and thus p_j is pushed at time j, whereby the result follows. ■

We say that an edge-segment is on VC(i), if both its endpoints are on VC(i). As an edge-segment may be the entire edge, the definition encompasses edges A point q is said to be on VC(i) if there exists an edge-segment on VC(i) containing the point. An edge-segment is removed from VC(i) if both its endpoints are popped from the stack. A

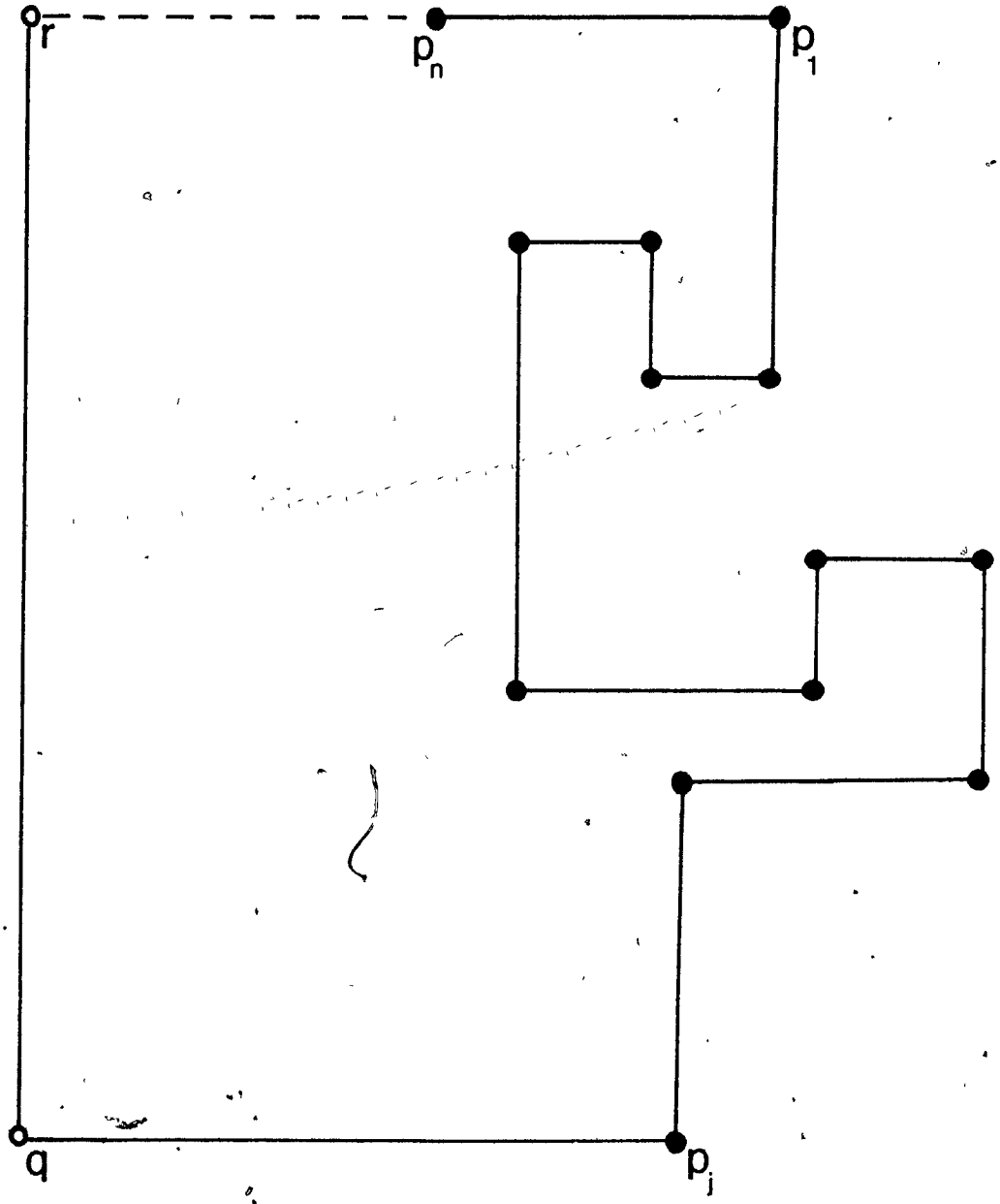


Figure 4.6

It is always possible to 'complete' a polygon from the lowest vertex p_j considered so far.

point is removed from VC(i) if the edge-segment containing it is removed from VC(i).

Lemma 4.6. *Let a point q be on VC(k). Then point q is popped iff q possesses an antagonist.*

Proof: Let q be located on an edge e_i. As q is on VC(k), point q is pushed onto the visibility chain at time i. Thus, by Lemma 4.5, TOP(i)=p_i.

"<=" If e_i has an antagonist, e_k, with y-value as high as q then by definition of antagonist, e_k is the first edge on $C_{i+1,m}$ intersecting the half-line $q\infty$. We want to show by induction on the length of the chain $C_{i,k}$ that q is popped when e_k is encountered. Trivially, if k=i+2 then q is removed from the stack at time k. Otherwise, since e_k is antagonist for q, q is not removed from the visibility chain before e_k is encountered. This implies:
(a) TOP(j) is not above q for all j with i<j<k, and (b) no 1-edge e_j with endpoint p_j above q is pushed between times i and k. We show that for any 1-edge e_m with endpoint p_{m-1} below TOP(i) there exists an antagonist e_a, such that i<a≤k. This is verified by observing that for each such 1-edge e_m on $C_{i+1,k}$ the turn form e_m to e_k is $-\pi$ and thus there exists an edge intersecting $p_{m-1}\infty$. The smallest such index defines the antagonist e_a for p_{m-1}. If a<k then by induction on the chain $C_{m,a}$ all 1-edges e_m are completely removed. Subsequently at time k, e_k will force q to be removed.

"=>" Follows directly from the description of the algorithm and the definition of antagonist. ■

Lemma 4.6 characterized under what conditions a point is popped from the stack. We now characterize the conditions under which a point is pushed onto the stack.

Lemma 4.7. *Denote by e_s=qr an edge-segment of an edge e_i with endpoints q,r. The lower endpoint r of e_s is not pushed iff (a) e_s is behind VC(i−2), or (b) q is below TOP(i−2).*

Proof: Recall from the algorithm description that a segment e_s=qr of an edge e_i is pushed if and only if r is below TOP(i−2) and q is not below TOP(i−2).

"<=" If either (a) or (b) is true then by the algorithm r is not pushed.

"=>" We assume now that e_s is not pushed. If q is below TOP(i−2) then (b) is true and the proof is completed. Therefore let q be above TOP(i−2). Since r is not pushed, r must be above TOP(i−2). We must show that e_s is behind VC(i−2). We assume the contrary, i.e. e_s is not behind VC(i−2). Since TOP(i−2) is a point located on a 1-edge, say e_t, by Property 2.2. the turn from e_t to e_s is 0. Edge e_s is above TOP(i−2) and thus an antagonist e_a, t<a<i, exists which intersects the half-line TOP(i−2) ∞. However, by Lemma 4.6 this edge e_k would have forced TOP(i−2) to be popped before e_s was encountered, which is a contradiction to the choice of TOP(i−2). ■

Lemma 4.8. *If a segment e_s=qr of an edge e_i is visible then its lower endpoint r is pushed on the stack.*

Proof: We prove this Lemma by contradiction, showing that if the lower endpoint r of a segment e_s of an edge e_i is not pushed then e_s is not visible. From Lemma 4.7 follows that if an edge is not pushed then it is (a) either behind VC(i−2) or (b) q is below TOP(i−2). If an edge-segment is behind a chain then by definition of behind, it is not visible. Consider therefore case (b) when q is below TOP(i−2). Let TOP(i−2) be contained in some edge e_t. As q is not pushed the endpoint p_{i-1} of edge e_i is not on the visibility chain and thus i>t+2. As q is below TOP(i−2) there exists at least one 1-edge on the chain connecting e_t and e_i. The lowest endpoint of this edge is pushed when encountered and must have been removed from the stack before e_{i-2} was examined, thus there exists an antagonist e_k, k<i, s.t. TOP(k)=TOP(i−2). The edge e_i lies inside the polygon defined by {TOP(i−2), p_t,..,p_k} and is thus not visible. Figure 4.7. illustrates this construction. ■

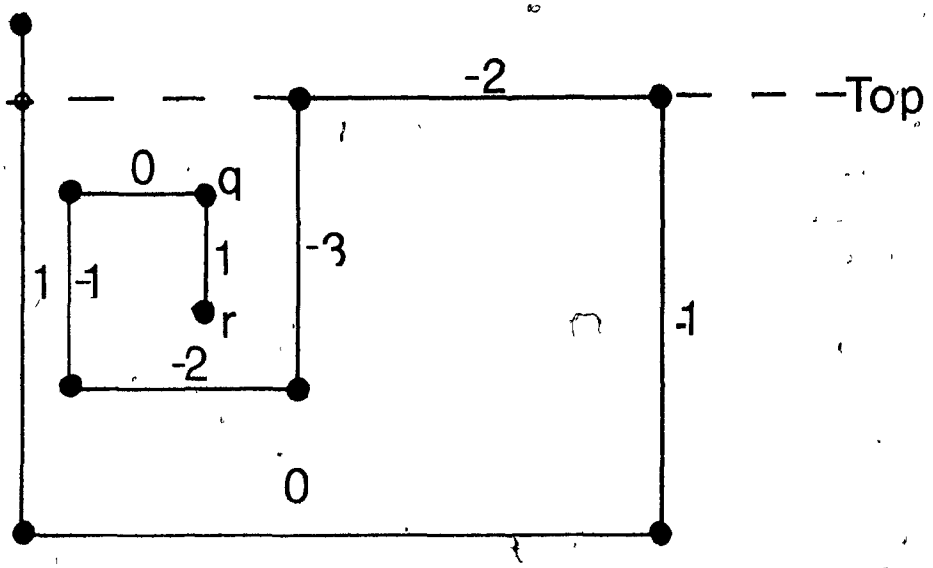


Figure 4.7

Illustrates that an edge e_i = qr is not visible if its upper end point q is below the top_of_stack.

Lemma 4.9. *If a segment of an edge is completely visible then both its endpoints are on the visibility chain.*

Proof: By Lemma 1.8 if an edge-segment is visible then its lower endpoint is pushed onto the stack. Thus it remains to be shown that in case the edge-segment e_s=q_r is completely visible then also the upper endpoint q is stacked. We assume this is not true. If q is behind VC(i−2) then clearly q is not visible violating the property that e_s is completely visible. Therefore q is in front of VC(i−2). This case is similar to Lemma 4.7 and we therefore omit its discussion here. ■

Lemma 4.10. *If the upper endpoint of an edge-segment is popped then the entire edge-segment is not visible.*

Proof: By Lemma 4.6 a point q is popped from the stack if there exists an antagonist for q. As P is simply connected and the turn from the segment to its antagonist is $-\pi$ the entire edge-segment is not visible. ■

Lemma 4.11. *If an edge-segment e_s, is on the visibility chain VC(m), then e_s is completely visible.*

Proof: We give a proof by contradiction assuming that a segment e_s=qr of a 1-edge e_i is not completely visible but located on VC(m). If the segment e_s is not visible then by Theorem 2.1. there exists a z on e_s and a cut-sequence (e_{k_1}, ..., e_{k_j}, e_s, ...), j>0 as high as z. Recall that the edges in the cut-sequence are listed in order of increasing distances from the point of visibility and that e_{k_1} is the edge closest to the viewpoint. If there exists a −1-edge e_{k_ℓ} such that k_ℓ>i and 0<ℓ≤j, then there exists a −1-edge e_{k_ℓ} closer to the viewpoint then e_i which is encountered after e_i. Thus z posses an antagonist and by Lemma 4.6 e_s is not on VC(m). If there exists no such −1-edge then by Theorem 2.1. for for all indices k_ℓ, with 0<ℓ≤j, and k_ℓ>i the corresponding edge labels ℓ_{k_ℓ} are positive. The edge e_{k_1} closest to the viewpoint is obviously visible along a ray with vertical coordinate as high as z. The point on e_{k_1} with vertical coordinate equal to that of z is according to Lemma 4.8 pushed onto VC(k_1). By Lemma 4.10 this point is never popped. Consequently the edge-segment qz is behind VC(i−2). By Lemma 4.7 z is not pushed on the stack which implies that it e_s is not on VC(m). ■

Theorem 4.1. *An edge-segment e_s is on VC(m) iff e_s is completely visible.*

Proof: "=>" If an edge e_s is on VC(m) then by Lemma 4.11 it is completely visible. "<=" If e_s is completely visible then by Lemma 4.8 it is pushed on the stack S and by Lemma 1.10 no portion of e_s is ever popped from S. ■

Theorem 4.2. *In this parallel model of visibility, Algorithm 4.2 solves the hidden-line problem for rectilinear polygons in linear time.*

Proof: The time-complexity of the algorithm is linear since each edge is examined exactly once. The correctness follows from the Theorem 4.1. ■

We have shown how to design and analyze a simple and efficient hidden-line algorithm for removing hidden lines in rectilinear polygons. The concept of a visible label and of an antagonist was introduced. Depending on the model of visibility, these sets differ. Therefore the main step for developing other hidden-line algorithms based on the labeling-scheme is to determine the set of all labels for visible edges and their antagonists. We will focus on this, since the resulting algorithms have a similar basic program structure.

4.3.2. Parallel Rays of Arbitrary Orientation

We now turn our attention to rays that are not parallel to on of the axes but still parallel to each other. Let S_1 denote the unit circle in the x-y coordinate-system. A unit vector can be specified by its clockwise angle formed with the (positive) x-axis. We denote the half-open interval of unit vectors, $0 \leq \alpha < \frac{\pi}{2}$, by I_1, similarly $I_2 = [\frac{\pi}{2}, \pi)$, $I_3 = [\pi, \frac{3\pi}{2})$, and $I_4 = [\frac{3\pi}{2}, 2\pi)$. The angle α_r of an (oriented) ray r is defined as the angle α of the unit vector that is parallel to r and has the same orientation as r. To obtain a concise notation, we say that the point of visibility, at ∞, is in interval I_j if the angle of the corresponding unit vector falls in the interval I_j. The direction of visibility is defined as the orientations of the rays.

A rectilinear polygon P can be split into its four extremal chains defined as

CII_1 from the x_{max}-edge to the y_{min}-edge,

CII_2 from the y_{min}-edge to the x_{min}-edge,

CII_3 from the x_{min}-edge to the y_{max}-edge,

CII_4 from the y_{max}-edge to the x_{max}-edge

Let V_k denote the set of labels for visible edges located on chain, CII_k, and A_k denote the corresponding set of antagonistic labels. An edge is *potentially visible* if its label is in V_k. Depending one the location of the viewpoint v, different edges are visible.

Lemma 4.12. *For a given direction of visibility and a chain CH_k, the number of labels in V_k is at most two.*

Proof: Consider for any extremal chain CH_k the subchain of all those edges that are visible from the viewpoint v. The resulting visibility chain is monotone in the direction perpendicular to the orientation of the rays of visibility. In Chapter 3.2 we have seen that such a monotone chain contains only edges labeled with at most three distinct labels and these labels are consecutive integers. Since no parallel rays can see both a left edge and a right edge, or both a top edge and a bottom edge, from any given viewpoint at most two distinctly labeled edges are visible. Thus from each viewpoint $|V_k| \leq 2$ and in case that $|V_k|=2$, $V_k=\{i,i+1\}$, for some i. ■

We observe that for a given direction of visibility either at least one of the endpoints of a chain CH_j is visible or none of the edges on CH_j are visible. More precisely, for v in I_k, none of the edges of CH_j are visible if $|j-k|=2$; whereas both extremal edges of CH_j are visible, in case j=k. For all other locations of v exactly one of the endpoints of CH_j is visible. For example for v in I_1, both extremal edges of CH_1 are potentially visible, exactly one of the extremal edges of CH_2 and CH_4, but none of the edges of CH_3 are.

Lemma 4.13. *If the set of visible edge-labels V_k is $\{i, i+1\}$ then the set of antagonistic labels is A_k is $\{i-1, i-2\}$*

Proof: Let the potentially visible i-edges be horizontal. Since in a rectilinear polygon horizontal and vertical edges alternate, it follows that (i+1)-edges are vertical. As, by assumption, both i-edges and (i+1)-edges are potentially visible the point of visibility is located either in I_2 or I_4. If the view-point is located in I_4 then i-edges are top-edges and antagonists are either vertical (i−1)-edges or horizontal (i−2)-edges, corresponding to turns of $-\frac{\pi}{2}$ and $-\pi$ respectively. See Figure 18 for illustration. These edges are

also antagonists for (i+1)-edges. In case the view-point is located in I_2 the antagonists are (i−1)-edges or (i−2)-edges. The case that i-edges are vertical can be handled in a similar way. ■

We have determined the sets of visible labels and antagonist are known for each of the four extremal chain. An algorithm similar to Algorithm 4.2 can be developed for solving this hidden-line problem We omit its description.

We summarize the results obtained for extremal chains in Figure 4.9.

4.4. Algorithm for Hidden-line Elimination: Perspective Model

Throughout this section, the model of visibility is the perspective model as described earlier. We will describe an algorithm to solve the hidden-line problem of a rectilinear polygon from a view-point located *inside* P. Let h=$(x-x_1)+y_1$, $x \geq x_1$ be an infinite horizontal half-line originating at p with the same orientation as the positive x-axis. As p is inside P, h intersects P at at least one point. The point being closest to p is referred to as the *east edge* for p. In an analogous way the *north, south* and *west* edges are defined. We will refer to these edges as e_{east}, e_{north}, e_{south}, and e_{west}-edges, respectively.

We define four *canonical chains* connecting the edges north to east, east to south, south to west, and west to north, respectively. In each chain we maintain the order given by P, e.g. the chain connecting e_{north} and e_{east} is $C_{north,east}$ if north < east. Otherwise it is $C_{east,north}$ For a given canonical chain $C_{i,j}$ we say that e_i, e_j are the defining edges.

Lemma 4.14. *If the e_{east} is labeled i then*

(i) $\ell_{south} = i+1$ *if south>east and* $i-3$ *otherwise,*

(ii) $\ell_{west} = i+2$ *if west>east and* $i-2$ *otherwise,*

(iii) $\ell_{north} = i+3$ *if north>east and* $i-1$ *otherwise.*

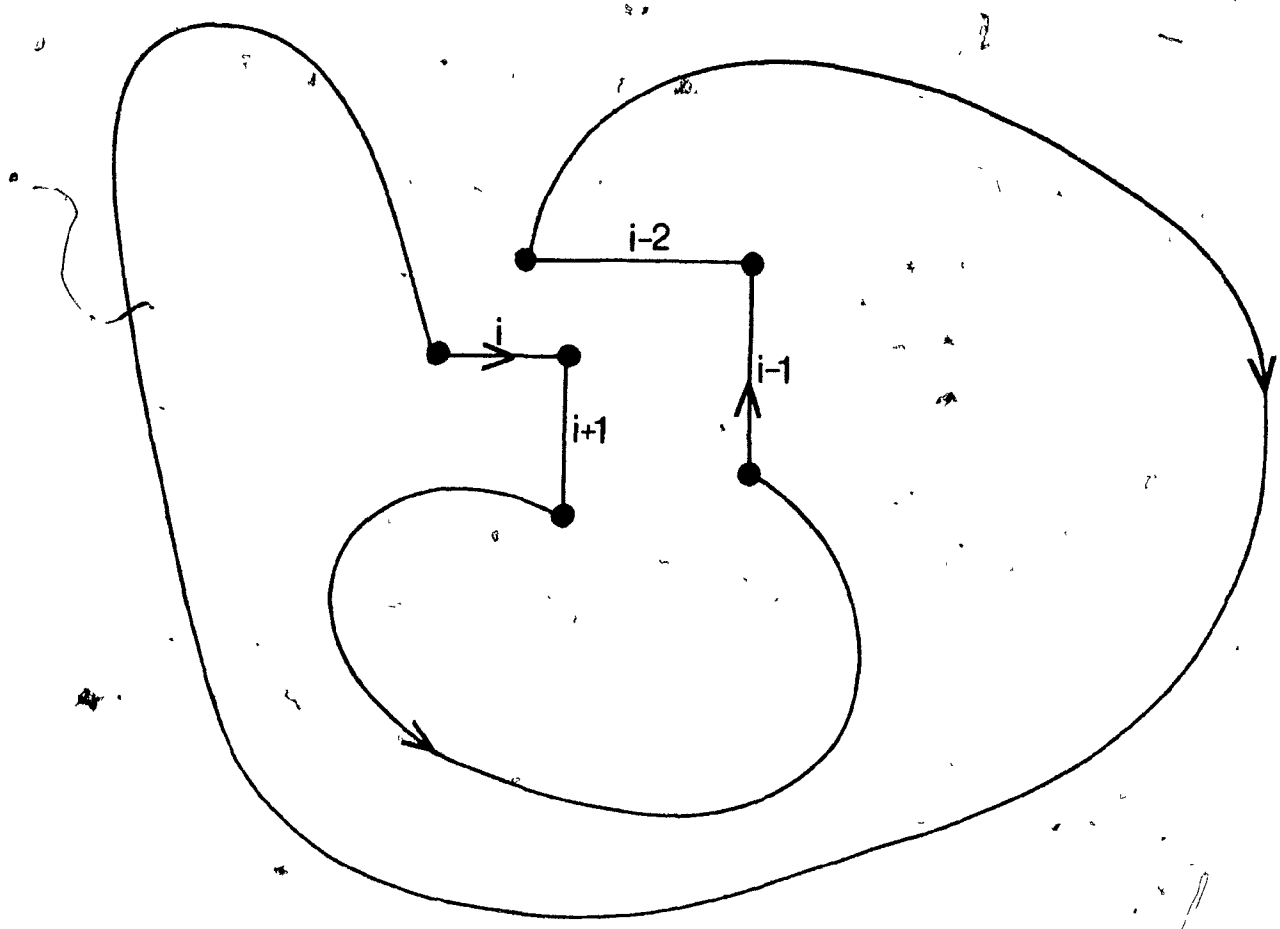


Figure 4.8

i−1, i+2 edges are antagonists for i, i+1 edges.

Interval	Labels	Chains
I_1: $[0, \frac{\pi}{2})$	1, 2	CH_1, CH_2, CH_4
I_2: $[\frac{\pi}{2}, \pi)$	2, 3	CH_1, CH_2, CH_3
I_3: $[\pi, \frac{3\pi}{2})$	3, 4	CH_2, CH_3
	-1, 0	CH_4
I_4: $[\frac{3\pi}{2}, 2\pi)$	0, 1	CH_1, CH_3
	4, 5	CH_4

Figure 4.9

Labels of visible edges for different viewing directions.

Proof: The turn from e_{east} to e_{north} is either $\frac{3\pi}{2}$ or $-\frac{\pi}{2}$ depending on whether, in a traversal of P, e_{east} is encountered before e_{north}, or after e_{north}, respectively. Thus the corresponding labels take on the values i+3 or i−1, respectively. An analogous argument holds for the other pairs of edges ■

Lemma 4.15. *If an edge located on a canonical chain $C_{i,j}$ is visible then its label is determined by the labels ℓ_i or ℓ_j.*

Proof: The line of visibility originating at a point p, performs a turn between 0 and $\frac{\pi}{2}$ while scanning the canonical chain $C_{i,j}$ for visibility of its edges. Let $C_{i,j}$ be a canonical chain with defining edges e_i, e_j.

(a) i<j: Let e_k be an edge on chain $C_{i,j}$, i<k≤j The turn $t_{k,j}$, from e_k to e_j, is either 0, in case e_k and e_j are parallel, or equal to 1, otherwise. In particular, for k=j, $\ell_j = \ell_i + 1$. Therefore the labels of visible edges on a canonical chain are equal to the labels of its defining edges

(b) i>j: Depending on whether k is encountered before, or after e_1, the value of ℓ_k differs, since at e_1 the labeling is restarted with $\ell_1=0$. Using the notation of Chapter 2.1, if $1\leq k\leq j$ then $\ell_j=t_{1,k}+t_{k,j} = \ell_k + \{0 \text{ or } 1\}$. This implies that $\ell_k=\ell_j$ or $\ell_k=\ell_j-1$. Otherwise, i.e $i\leq k\leq n$, $\ell_k = t_{1,i}+t_{i,k}=\ell_i+\{0 \text{ or } 1\}$ and thus $\ell_k=\ell_i'$ or $\ell_k=\ell_i+1$ The labels of the defining edges ℓ_i and ℓ_j are related by $\ell_i=\ell_j+3$.
Thus in both cases the labels of the defining edges determine the label of visible edges. ■

We have noticed in the above proof to Lemma 4.15 that the label of any visible edge located on a chain $C_{i,j}$ takes any one value in $\{\ell_i, \ell_{i+1}, \ell_{j-1}, \ell_j\}$ For at least three of the four canonical chains this reduces to $\{\ell_i,\ell_j\}$. In case there exists a chain for which i>j, ℓ_j and ℓ_i are related by $\ell_j=\ell_i-3$. Four labels would have to be checked for, when designing a visibility algorithm. However, if we define $\ell_k^* = \ell_k+4$ for $1\leq k\leq j$ and $\ell_k^*=\ell_k$, for $i\leq k<=n$, then the set of visible labels to consider is reduced to $\{\ell_i, \ell_j^*\}$. This allows for a unique treatment of all four chains when presenting the algorithm.

Recall that an antagonist for an edge e_i is an edge with smallest index k s. t. the oriented half-line from a point p, on e_i, to ∞ intersects e_k. It is an important difference between visibility from the inside and visibility from the outside that the antagonistic labels are greater than the labels of edges which they obscure, in case of interior visibility. Antagonistic labels are smaller than visible labels in case of exterior visibility To relate antagonistic labels to visible labels, we state the following result:

Lemma 4.16. *Let e_a be antagonist for e_i then $1\leq \ell_a^*-\ell_i^*\leq 9$*

Proof: The line-segment of ℓ connecting e_a and e_i is inside P, since we are considering visibility from inside P. By Theorem 2.2 the label difference between ℓ_a and ℓ_i is greater then 0 and less than or equal to 3. ■

Lemma 4.17. *Corresponding to each pair of visible labels* (ℓ_i, ℓ_j^*) *is the pair of antagonists* (ℓ_i+2, ℓ_j^*+2).

Proof: Let $C_{i,j}$ be a canonical chain. Antagonists for ℓ_i-edges are ℓ_i+2-edges and ℓ_i+3-edges, since the corresponding turns are π and $\frac{3\pi}{2}$. Antagonists for ℓ_j^*-edges are ℓ_j^*+1-edges and ℓ_i+2-edges, since the corresponding turns are $\frac{\pi}{2}$ and π. Since, by Lemma 1.11 and the definition of ℓ_j^*, the difference $\ell_j^*-\ell_i$ is equal to 1. The result follows. ■

We will now describe the algorithm for hidden-line elimination in the perspective model. Let V be defined as the set of visible labels $V := \{\ell_i, \ell_j^*\}$. Let A be defined as the set of antagonistic labels $A := \{\ell_i+2, \ell_j^*+2\}$. Assume that the point of visibility, p, is located at (x_p, y_p). Let ℓ be an infinite half-line originating at p. Let MOVE_ℓ(q) be a procedure which moves the half-line ℓ to intersect q. If two lines or line-segments ℓ_1 and ℓ_2 intersect at a point we denote this intersection point by $\ell_1 @ \ell_2$. Furthermore let line(u,t) denote the line-segment joining u and t. Note that the line ℓ is oriented, and therefore we can speak of the "left" of ℓ, and the "right" of ℓ.

Algorithm 4.3: Hidden-Line Problem in Perspective Model

Input: A rectilinear polygon P and a point of visibility p, inside P.

Output: The visibility polygon from p

```
find canonical chains C_{i,j};
for each canonical chain C_{i,j} do
   begin
      L_i := {infinite half-line originating at p and intersecting e_i perpendicularly}
      L_j := {infinite half-line originating at p and intersecting e_j perpendicularly}
      ℓ = L_i
      PUSH(ℓ#e_i)
      for k := i to j do
         if C_{i,k} intersects L_i an odd number of times
            and e_k does not lie completely outside the quadrant formed by L_i and L_j
            then
              begin
                 if ℓ_k* ∈ V and e_k intersects ℓ
                   then
                      begin
                        if TOP <> ℓ@e_k
                           then PUSH(ℓ@e_k)
                        if e_k intersects L_j
                           then PUSH(L_j@e_k)
                        else
                           PUSH(p_k)
                      end
                 else
                   if ℓ_k* ∈ Λ
                      and e_k intersects line(p,TOP)
                      then
                        begin
                           while line(p,TOP) is intersected by e_k do
                             begin
                               PREVIOUS_TOP:= TOP
                               POP;
                             end
                           MOVE_ℓ(p_k)
                           if TOP is to the left of
                             then
                               PUSH(Line(TOP,PREVIOUS_TOP)@ℓ)
                           else
                               MOVE_ℓ(TOP)
                        end
              end
   end
join the four visibility chains constructed.
```

Theorem 4.3. *Algorithm 4.3 solves the hidden-line problem in the perspective model in linear time*

Proof: For a given view-point, v, the algorithm splits a rectilinear polygon P into the four canonical chains with respect to v. This operation can be performed in linear time. The labels of both edges defining a canonical chain were related in Lemma 4.1.1. Using Lemma 4.1.5 we can construct the set of all visible labels for each canonical chain. Using Lemmas 4.1.6 and 4.1.7, we can determine the antagonistic label set can be determined. Therefore, in linear time, the sets V, of visible labels and A, of antagonists are constructed. In the main loop of the algorithm each edge is examined exactly once. Since for each edge at most one PUSH-operation and one POP-operation is executed, the entire procedure requires linear time. With the development of visible labels and antagonists the proof of correctness is similar to Theorem 4.2. It is omitted here. ■

We have shown how the labeling-scheme can be used in the design of efficient algorithms for eliminating hidden-lines in rectilinear polygons. For some applications, it might be necessary to generate many different views of the same object. Algorithm 4.3 can be adapted to perform such a task efficiently. As an initial step, performed only once, we will label the edges of P using the labeling-scheme. This step will keep, for each distinct label i, a linked list of all edges labeled i. Subsequently, for each view-point, the algorithm determines the facing edges of the four canonical chains. This operation is performed in linear time. If we allow O(n log n) preprocessing, this task can be performed in O(log n) time. (This point-location problem has been studied by many researchers in the area of computational geometry, see for example Kirkpatrick [Ki83], Lee and Preparata [LeP79], or Preparata [Pr81]). Since we keep linked-list for all distinct labels, we have determined, in O(log n) time, all those edges that are either potentially visible or antagonistic. Thus often the total number of edges to be examined is reduced. This results in substantial savings in particular if the polygon contains many spirals.

Chapter 5
Further Applications of the Labeling Scheme

5.1. Construction of the Rectilinear Convex Hull

One of the most extensively studied problems in computational geometry is the construction of the convex hull of a simple polygon or of a set of points We recall that the *convex hull* of a set of points is defined as the minimum area convex set containing the original set. Lower bounds of Ω(n log n) for the convex hull of a set of n points have been derived for various models of computation [Sh77, Av79, Em80, Ya79]

Since it costs O(n log n) time to construct a simple polygon from a set of points, polygons are more structured than sets of points This has been exploited to develop linear-time convex hull algorithms for simple polygons [McA79, Le83a, BhE81, GrY83]. The history of geometric algorithms reveals the great difficulty of finding simple, elegant, but correct solutions for obtaining linear-time convex hull algorithms for simple polygons The simplest algorithm is due to Sklansky [Sk70]. It has been pointed out that his algorithm can fail to correctly determine the convex hull for arbitrary simple polygons Toussaint and Avis [ToA82] proved, however, that Sklansky's algorithm will correctly determine the convex hull, if the input polygon is weakly externally visible. Since visibility chains are weakly externally visible, any algorithm capable of computing the hidden-line problem can be used, in conjunction with Sklansky's algorithm, to solve the convex hull problem [EAT83] In Chapter 4 several efficient algorithm for hidden-line elimination in rectilinear polygon have been presented. Thus, in combining Algorithm 4 2 with Sklansky's algorithm, a conceptually simple and efficient convex hull algorithm for rectilinear polygons is obtained. We now discuss a variant of the convex hull problem and modify this idea to solve this variant. An alternate solution was

presented in [MP82a].

In the context of rectilinear geometry we are interested in determining the *rectilinear convex hull* of a rectilinear polygon, defined as the minimum area rectilinearly convex polygon enclosing P. See Figure 5.1 for illustration. Instead of rectilinear convex hull we will frequently use the term *rectilinear hull*. Similarly the y-hull (or x-hull) of a rectilinear polygon is the minimum area rectilinear polygon monotone in the y-direction (or x-direction) and enclosing P. The solution to the hidden-line problem from a view-point exterior to P is called the *visibility hull* of a polygon. It has been observed that the convex hull of a simple polygon is the union of visibility hulls over all directions of visibility [ToS84]. For the rectilinear hull this reduces to determining the union of the x-hull and the y-hull. The y-hull of a rectilinear polygon P can be determined as follows.

Algorithm 5.1: Algorithm for Computing the y-Hull

Input: A rectilinear polygon P

Output: The y-hull of P

use Algorithm 4.1 to
    compute C_r, the visibility hull from $+\infty$ on the x-axis,
    compute C_ℓ, the visibility hull from $-\infty$ on the x-axis,
join the chains at the extreme edges in the y-direction.

Similarly the x-hull can be determined. The x-hull of a rectilinear polygon P is denoted by x-hull(P), similarly is the y-hull of P denoted by y-hull(P). We can now state the algorithm for computing the rectilinear hull of a rectilinear polygon.

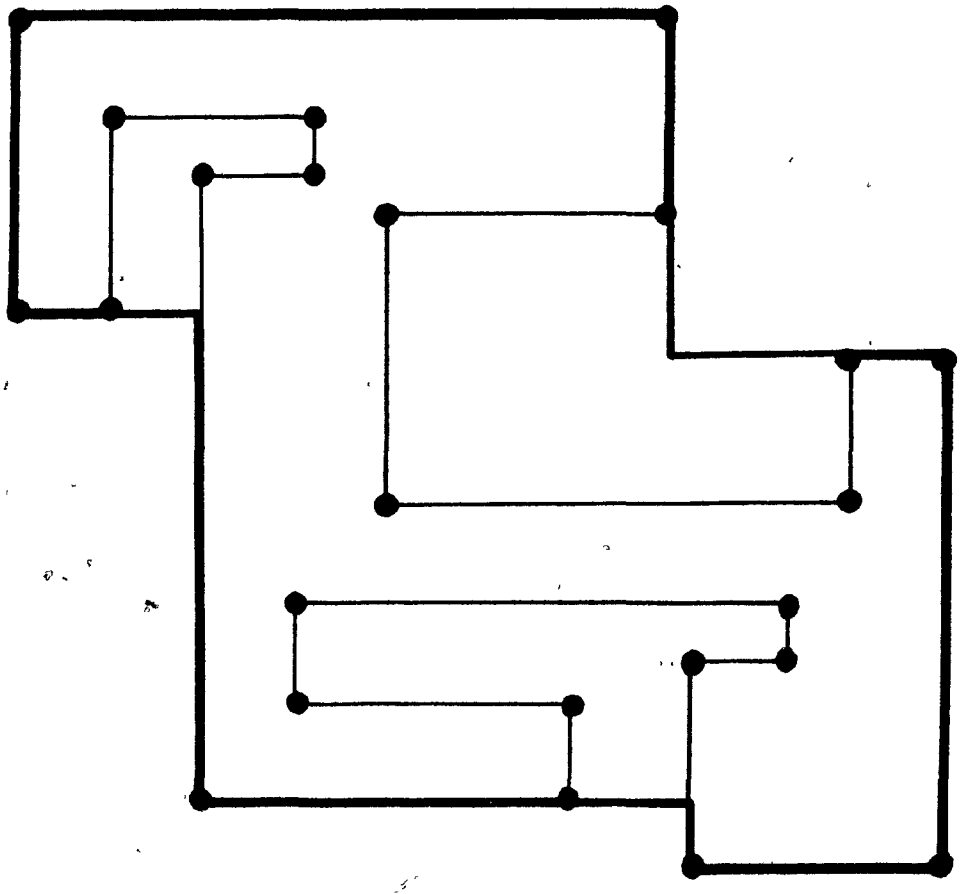


Figure 5.1

The rectilinear convex hull of a rectilinear polygon.

Algorithm 5.2: Algorithm for Computing the Rectilinear Hull

Input: A rectilinear polygon P

Output: The rectilinear hull of P

P_y.=y-hull(P);
rectilinear hull of P:=x-hull(P_y)

Lemma 5.1. *Algorithm 5.2 correctly determines the rectilinear hull of a rectilinear polygon, in linear time.*

Proof: By Theorem 4.2 computing the x-hull and y-hull takes linear time, thus the run-time of Algorithm 5.2 is linear. We will now show the correctness of this procedure. We show that the union of y-hull(P) and x-hull(P) can be computed by determining x-hull(y-hull(P)) or alternatively y-hull(x-hull(P)). In order to prove that x-hull(y-hull(P)) = union(x-hull(P), y-hull(P)) we show the containment in both directions. We first show that union(x-hull, y-hull(P)) is contained in x-hull(y-hull(P)). The x-hull or y-hull of a polygon always contains the polygon. Thus P is contained in y-hull(P) and x-hull(P) is in x-hull(y-hull(P)) and consequently y-hull(P) is in x-hull(y-hull(P)). We now show the opposite direction, i.e. if a point p is in x-hull(y-hull(P)) then p is in union(x-hull(P), y-hull(P)). Let a point p be in x-hull(y-hull(P)) then p is either in y-hull(P), in which case we are done, or p is in the set difference between x-hull(y-hull(P)) and y-hull(P). The point p is outsjde P and visible from $+\infty$ on the x-axis, but not visible from $+\infty$ on the y-axis. Thus p is in x-hull(P), see Figure 5.2. ■

We have shown how to apply the results of Chapter 4 for the design of simple algorithms to determine hulls of rectilinear polygons. In this thesis we focus on computational geometry problems for single rectilinear polygons. Several authors have recently studied the definition and construction of the rectilinear hull of a set of points, and of a collection of rectilinear polygons [NLLW83, MF82a, OSW83]. A discussion of

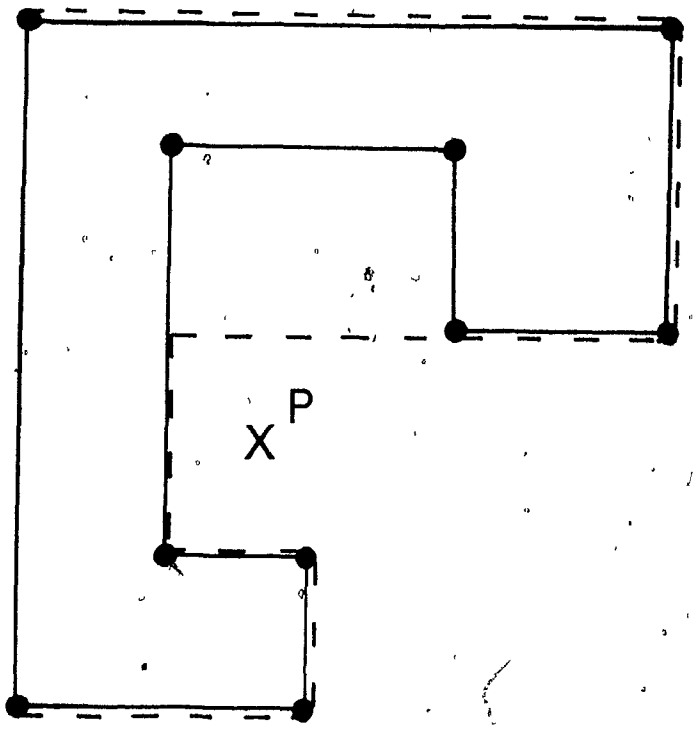


Figure 5.2

Point p is in x-hull(P) but not in y-hull(P).

the resulting problems and some algorithms are surveyed in [OSW83a] The rectilinear geometry of collections of rectilinear objects is beyond the scope of this thesis

5.2. Movement of Robots in a Rectilinear Environment

Let a floor plan of a rectilinear building (traditional art-gallery) be stored as an n-vertex rectilinear polygon P. For two points located anywhere on the plane, we ask whether they can be connected by a rectilinear path which does not intersect any edge of P. This problem is equivalent to asking whether both two points lie in the same connected component with respect to P, i.e whether the points are either both inside or both outside P. The problem has received considerable attention from both

mathematicians as well as computer scientists Jordan has solved this question by noting that a polygon partitions the plane into two connected components the inside of P and its exterior. For simple closed curves this fact is known as the Jordan Curve Theorem We will use his result as expressed for polygons

Lemma 5.2. *A point p is inside P if and only if for every horizontal half-line originating at p the number of intersections with P is odd*

Proof: The proof follows directly from the Jordan Curve Theorem and the Lemma has been stated elsewhere e.g. in [Sh77] ■

When counting the number of intersections some precaution has to be taken if the line h intersects P at a vertex or if h coincides with an edge e_i of P. If h intersects P at a vertex, we count the intersection twice, in case that both edges, incident to the vertex, lie on the same side of h, otherwise, we count the intersection once. If h and any edge e_i coincide then we count this as two intersections Thus a point can be tested for inclusion in a polygon in O(n) time If the point location query is to be performed repeatedly then it might be advantageous to invest preprocessing time in order to obtain faster query times This approach has been examined by a many researchers, for example see [Sh77, LeP79, LiT77, LiT77a, Pr81, Ki83]

Throughout this chapter we are interested in movements of robots and thus our model of computation is different from the models used in the above point location algorithms. We assume that a camera is attached to the robot enabling the robot to see. From a given location the robot can determine in constant time, the facing edges and its labels as defined in Chapter 1 For short we say that the robot can see to its left and to its right Furthermore, we assume that the robot has a sensor to prevent it from hitting walls. No global knowledge of the polygon will be required by the robot. If the robot is outside the polygon P, it may happen that in one of the four rectilinear directions no edge is visible, in which case we say that ∞ *is seen,* or that this *facing edge is* ∞. Note that if the robot is inside the rectilinear hull of P, at most one of the

four facing edges is ∞. We now present an easy test for determining whether a robot R is inside or outside a polygon P

Lemma 5.3. *Let (w,e) be the pair of labels for the west, east edges as seen by the robot R.*

(a) R is inside P if (w,e) mod 4=(3,1),

(b) R is outside P if (w,e) mod 4=(1,3)

Proof: R is inside P if and only if the east-edge is a down-edge and the west-edge is an up-edge R is outside P otherwise By Theorem 2.1 we obtain that $|w-e|=2$. If R sees the pair (3,1) as (w,e), then R is inside P, since 1-edges are down-edges and 3-edges are up-edges By Property 2 2 the net-turn between parallel edges of equal orientation is 0. We obtain that for all pairs (w,e) for which (w,e) mod 4=(3,1), the robot is inside P. In a similar way (b) can be shown ■

We will now assume that R is located at a point p inside the rectilinear hull of P, but exterior to P Let p_i, p_j, $j>i+1$, be two vertices of P. If the edge p_ip_j is an edge of the rectilinear hull of P then p_ip_j is called a *lid*. The corresponding polygonal chain $p_i, \ldots, p_j$ is called a *pocket* We study the question of how to determine the shortest rectilinear path from a point p in the interior of a pocket to the corresponding lid. The path may not intersect any edges of P In other words, we ask if the robot is working inside the maze defined by the exterior of P, what is the shortest path to leave the maze without "going" through walls? The underlying distance metric used here is the *Manhattan Metric*, i e for points $p=(p_x, p_y)$ and $q=(q_x, q_y)$ the Manhattan distance is defined as $d(p,q)=|p_x-q_x| + |p_y-q_y|$ Note that in this metric the shortest path between any two points with distinct x-coordinates and y-coordinates is non-unique Referring to the notation in Figure 5 3, both p, p#q, q and p, q#p, q are shortest

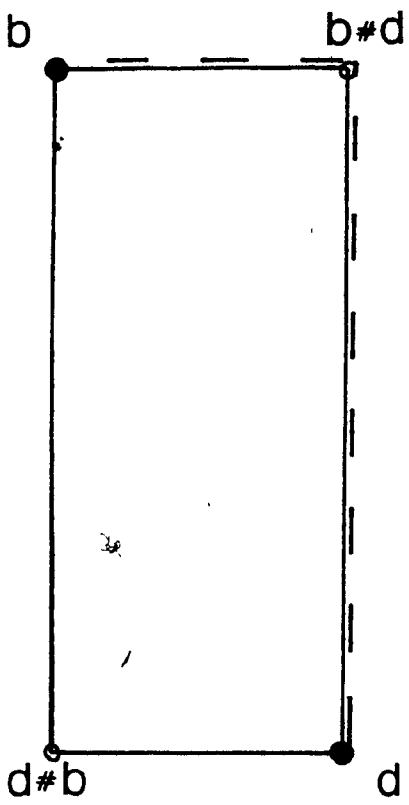


Figure 5.3

In the "Manhattan"-metric the shortest path from p to q is realized by p,q#p, q or p,p#q,q It is therefore not unique

paths

Before solving this question we examine an easier problem Let a robot R be located at a point p, inside the rectilinear hull, but in the exterior of a polygon P. We assume that somebody erects a horizontal wall from the east-edge to the west-edge through or close to point p Such a wall is an external line-segment for P. By Theorem 2 2 any such wall decomposes the outside of P into a bounded and an unbounded region. We say that R is *locked in*, if R is inside the bounded region and R is *free* otherwise By a similar construction a vertical wall could be erected.

Problem If a wall is so constructed determine the movement of R such that R is free.

The solution will be given as an instruction of the form "go up", "go down", "go left", or "go right", the semantics of which is apparent. Let (w,e) and (n,s) denote the label-pair of the (west-edge, east-edge) and (north-edge, south-edge), respectively.

Instruction set:

In case the wall is horizontal do: if $w < e$ "go down", otherwise "go up".

In case the wall is vertical do: if $n < s$ "go left", otherwise "go right".

Lemma 5.4. *Whenever a wall is constructed, using the instructions the robot will correctly move to free itself.*

Proof: A wall represents a segment of a rectilinear cut through the location of R at p. We can therefore apply Theorem 2.1 yielding that the label difference between the west-edge and the east-edge is always two in absolute value. As the segment joining the west-edge and the east-edge is an external line-segment the west-edge is an down-edge and the east-edge a up-edge. We have to determine if the bounded region is located above or below the wall. Theorem 2.2 in Chapter 2 deals with this situation. We get that if $w>e$, in particular if $w-e=2$, then the bounded region is below the wall. The correct move is therefore 'go up', which is indeed what the instruction says. In case that $w<e$, in particular if $w-e=-2$, the bounded region is, by Theorem 2.2, above the wall. Therefore the correct movement is to walk 'go down'. ■

Lemma 5.5. *The shortest rectilinear path is not unique.*

Proof: The proof follows from the example given in Figure 5.3. ■

We solve the shortest rectilinear path problem as described above. The robot moves only in one of the four rectilinear directions. A change of direction is from a horizontal movement to a vertical movement and vice versa.

Algorithm 5.3: Shortest Rectilinear Path

Input: Rectilinear polygon and position of robot R.

Output: The shortest rectilinear path to free R.

```
Pick an arbitrary orientation (vertical or horizontal);
repeat
    if one of the facing edges is labeled ∞
        then
            go towards that edge and exit.
    else
        begin
            execute instruction
            move until side edges change labels or you 'hit' a wall
                then
                    change orientation
        end
until free.
```

Lemma 5.6. *Let R be located inside a convex deficiency of P. Then at most one of the facing edges is ∞.*

Proof: The result follows directly from the fact that the rectilinear hull is the union of the x-hull and y-hull. ■

Lemma 5.7. *The algorithm Shortest Rectilinear Path correctly finds the shortest rectilinear path from p to the exterior of the rectilinear hull.*

Proof: If the robot R sees ∞ then R moves in the direction towards ∞. By the previous Lemma 5.6 the path is, in this case, unique and thus clearly minimal. We assume now that R is in its initial position and does not see ∞. Thus R cannot reach the lid without changing the direction of movability at least once. The shortest rectilinear path is therefore non-unique and R can choose an arbitrary starting orientation, i.e. either horizontal or vertical. Now let the current position of R be p_{m-1}. We now assume that $p_1,...,p_{m-1}$ is a shortest rectilinear path, which can be extended to a shortest path from p to the lid. We will show how to extend this path to

a shortest path form p to the lid. For this, we construct a wall perpendicular to $p_{m-2}p_{m-1}$ and intersecting p_{m-1}. The endpoints of the walls are located on the edges facing R at p_{m-1}. By Theorem 2.2, this wall splits the outside of P into two regions one of them is bounded, while the other is unbounded. As the path from p_1 to p_{m-1} is minimal, the robot will not go back into the same direction form which it came, i.e it will not enter the bounded region. If at p_{m-1} the facing edge-labels do not change then the robot continues walking in the same direction since it remains in the unbounded region. If, however, the facing edge-labels change then we can interpret this situation as having two parallel walls, one of them joining the previous facing edges, the other one joining the new facing edges. Both walls enclose bounded regions one located directly above the wall, the other directly below the wall. Thus R is forced to change its orientation. By Lemma 5.4, R will choose the right direction orientation and thus a unique extension of $p_1,\ldots,p_{m-1}$ to a minimal path from p_1 to p_m extendible to the lid is found. The situation when R encounters ('hits') a wall can be seen in exactly the same way as a change of label. ■

In this chapter, we have discussed an application of the labeling-scheme in the area of robotics. The solution obtained was expressed as a simple set of instructions. We have used the labeling-scheme to solve a problem, whenever winding properties and visibility aspects were relevant to its solution. A related movability problem has been studied by L. Majocchi and G. R. Sechi [MS84]. The task is to free a mouse from a rectilinear maze by a sequence of rectilinear moves. In their model the mouse has only local perception obtained by using sensors. The nose and its whiskers are used for detecting lateral and frontal pressure. Their objective is to find a rectilinear path that avoids entering 'blind alleys' and loops. Their primary interest is in the area of problem solving and learning. This, as they call it, 'Mouse in a Maze'-problem, appears to be a fruitful topic in this context. They study the problem not under a computational complexity point of view, but under the aspect of learning from previous experiences,

and thus they do not include a complexity analysis.

Chapter 6
Decomposition Techniques for Rectilinear Polygons

6.1. Guard Placement Problems

The remainder of this thesis is devoted to the discussion of some decomposition problems occurring in rectilinear geometry. In the introductory chapter, pattern recognition and shape analysis were mentioned as possible areas of application of decompositions techniques. Our primary interest in these techniques is their computational geometry aspect, where often a problem is solved efficiently by decomposing it into well defined smaller subproblems, which are subsequently merged to obtain a solution for the entire problem. As we will show, this task occurs when solving guard placement problems in traditional art-galleries, i.e. art-galleries whose floor plan can be described by rectilinear polygons. This problem, as studied by [KKK83], is a variant of the (general) art-gallery problem due to Victor Klee, mentioned in the introductory chapter. In its general form, the question can be posed as follows: how many guards are always sufficient and sometimes necessary to see the inside of an n-vertex simple polygon P, where guards are assumed to be located at stationary points inside P. Guards "see" in the sense of internal visibility as defined previously.

V. Chvátal [Ch75] has shown that the answer for an arbitrary simple n-vertex polygon is $\left\lfloor \frac{n}{3} \right\rfloor$. Later Fisk [Fi78] gave a simpler argument for the sufficiency based on the fact that the graph obtained by triangulating a simple polygon is three colorable. Any triangulation can be seen as a partitioning of the polygon into convex pieces. Each triangle contains vertices of each color. Therefore by placing guards at all vertices having the same color, it is ensured that the entire polygon is seen. Fisk completes his

argument by picking the color occurring least frequently.

In a somewhat similar way Kahn et al. [KKK83] showed that $\left\lfloor \frac{n}{4} \right\rfloor$ guards are always sufficient and for some examples also necessary to guard any rectilinear polygon. We refer to this result as the Rectilinear Art Gallery Theorem The idea pursued in [KKK83] is to construct a four-colorable graph with the property that placing guards at all vertices of any chosen color will solve the guard placement problem. Then by picking the least frequently occurring color, the $\left\lfloor \frac{n}{4} \right\rfloor$ bound on the sufficiency of the number of guards is obtained. The four-colorable graph is readily obtained from a (convex) quadrilaterization of the rectilinear polygon. We give the construction: insert a pair of intersecting diagonals into all convex quadrilaterals as illustrated in Figure 6.1. Within each quadrilateral, each vertex is connected to all other vertices, implying that at least four colors are needed. The sufficiency of four colors follows by a simple inductive argument. Notice that the convexity of the quadrilaterals is crucial to ensure that an entire quadrilateral is seen if a guard is placed on any of its vertices. We pointed out previously that not every simple polygon admits a decomposition into convex quadrilaterals. The hard part of their [KKK83] rather lengthy proof is therefore to show the existence of a quadrilaterization for any rectilinear polygon. Quoting from their paper [KKK83 p. 197]: "In many places in our paper our proofs depend on certain configurations having particular properties which appear to follow obviously from the relevant definitions. .... However, as is often true in geometrical problems, despite their "obvious truth" is seems both time-consuming and tricky to provide rigorous proofs of these assertions."

We found another aspect of the quadrilaterization problem rather intriguing: Whereas it is easy to triangulate a simple polygon 'by hand', it is much harder to find a convex quadrilaterization of rectilinear polygons. Where is the problem? Solving the

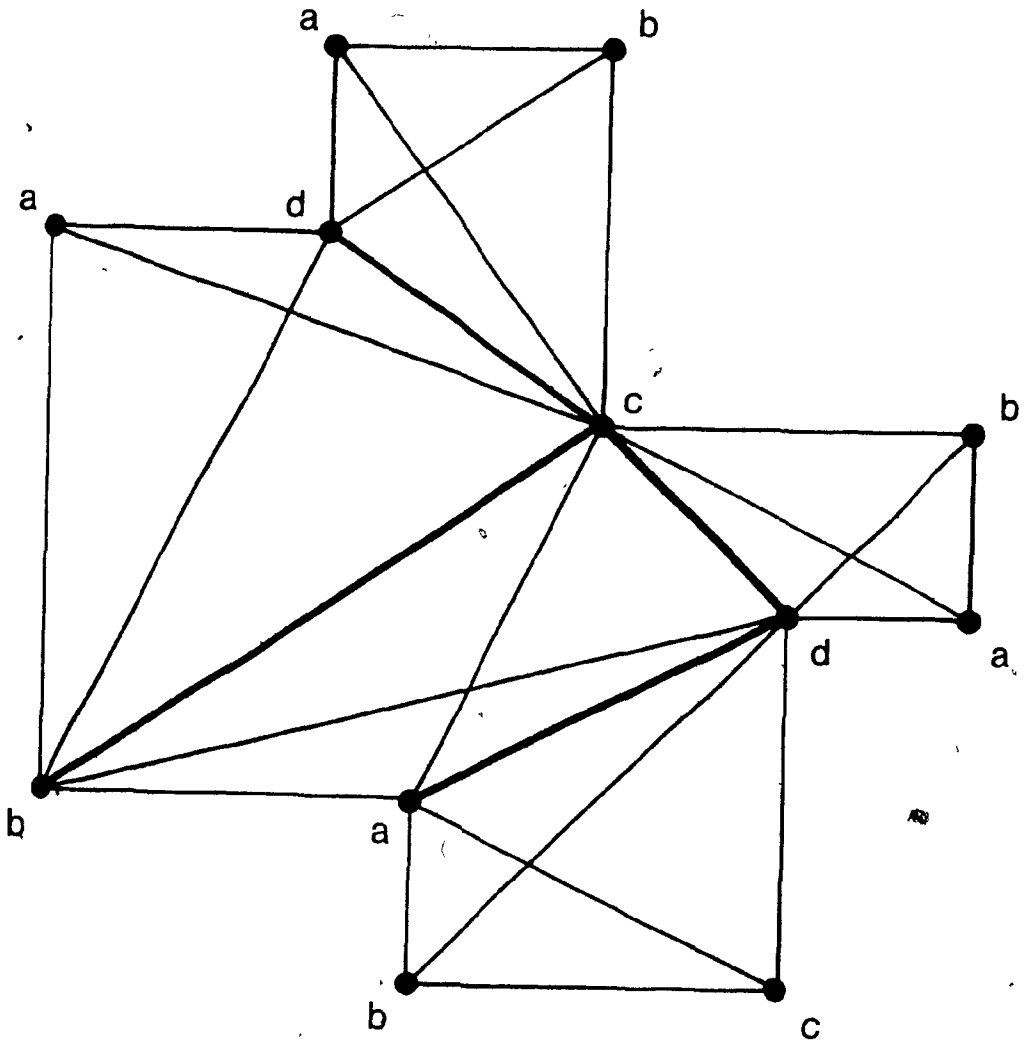


Figure 6.1

A four-coloring of a graph obtained by inserting a pair of intersecting diagonals in each convex quadrilateral.

problem manually, even for rectilinear polygons with a small number of vertices, we often end up with triangles or concave quadrilaterals, in which case we might have to backtrack, possibly even to the first diagonal inserted. O'Rourke [OR83] compares quadrilaterization to a manual guard placement pointing out: "It is quite difficult to find a convex quadrilaterization by hand, whereas it is a simple procedure to place guards by hand and eye on even quite large rectilinear polygons".

While Kahn et al. mainly proved the existence of a convex quadrilaterization for rectilinear polygons, we are primarily interested in giving efficient algorithms for solving this problem. As a by-product of our discussion we will obtain an existence proof. Before presenting our algorithms, we will give an overview of their proof. They first introduce the concept of reducibility: "A (finite) rectilinear region R is reducible if whenever every smaller finite rectilinear region is convexly quadrilaterizable then so is R". By 'smaller' they mean, in this case, fewer vertices. Next they give several reductions for rectilinear polygons These reductions are being performed by recognizing certain configurations inside the polygon. Their proof is completed inductively by showing the existence of at least one such configuration in any given rectilinear polygon with more than four vertices.

6.2. Quadrilaterization of Rectilinear Star-Shaped Polygons

In this section, a linear-time algorithm is presented for decomposing rectilinear star-shaped polygons (RSP) into convex quadrilaterals. Recall that a polygon P is star-shaped if there exists at least one point in P which can see the entire polygon. Arbitrary star-shaped polygons can be triangulated in linear time [ScL80]. The obvious way of quadrilaterizing RSPs, namely to triangulate the polygon and then to remove every other edge, fails. The reason for this is that such a decomposition may contain concave quadrilaterals, as illustrated in Figure 6.2a A correct quadrilaterization of the polygon in Figure 6.2a, is illustrated in Figure 6.2b. Our approach is to partition the polygon P into two simpler components, i.e. pyramids, both admitting quadrilaterization.

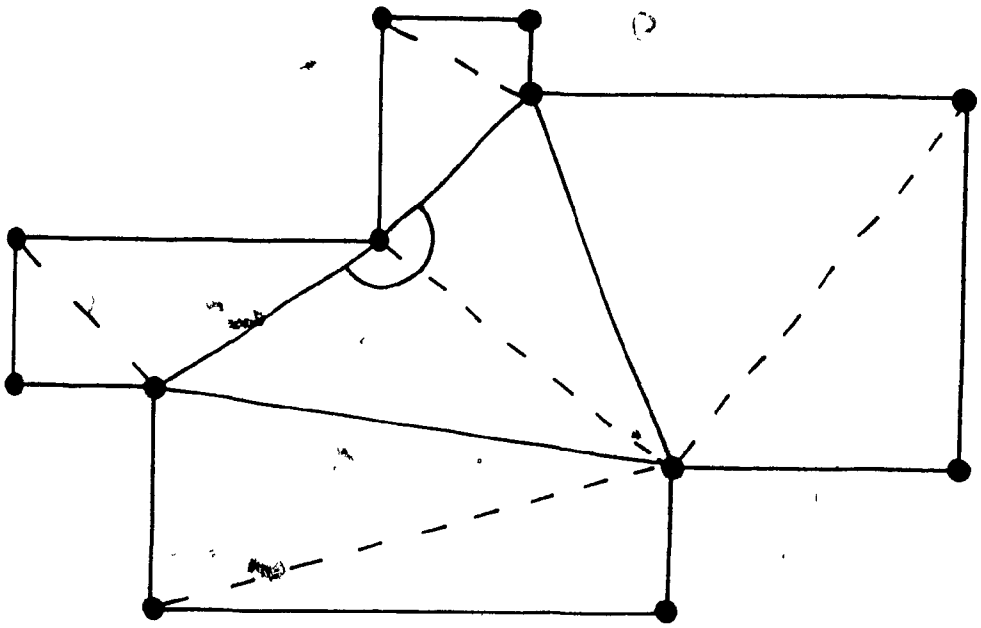


Figure 6.2a

Only if we allow swapping of diagonals, a convex quadrilaterization can be obtained from the given triangulation

6.2.1. Partitioning Rectilinear Star-Shaped Polygons into Pyramids

First we recall some properties of rectilinear star-shaped polygons and introduce some notation. Denote the vertices of the extreme edges as in Figure 6.3, also let y(p) denote the y-coordinate of point p. We have defined a stair as a polygonal chain monotone in both the x and y-direction. Furthermore, we have shown that the extreme edges of any rectilinear star-shaped polygon are connected by stairs. If we extend the horizontal and vertical edges at a reflex vertex towards the interior of the polygon, we get the shaded area in Figure 6.4. Schachter [Sc78] calls this area the *inner cone*. A reflex vertex v is *up-looking*, if the inner cone is not below the horizontal line through v, else v is *down-looking*. In Figure 6.4 (a) v is an up-looking reflex vertex, whereas in (b)

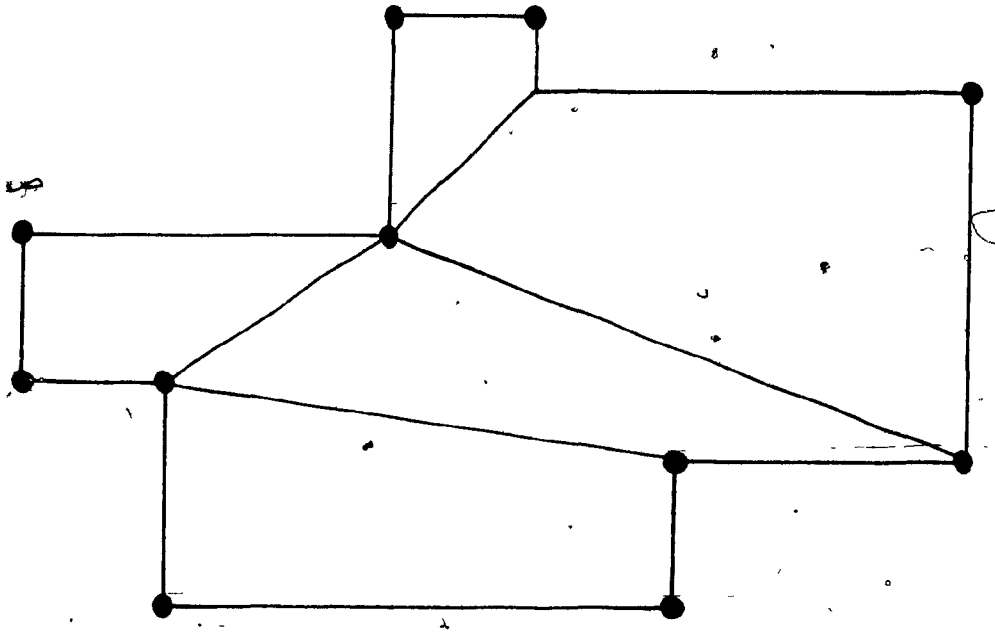


Figure 6.2b

A convex quadrilateralization of the polygon in Figure 6.2a.

v is a down-looking reflex vertex.

We defined a pyramid as a rectilinearly convex histogram and showed that a pyramid has two stairs. Now we illustrate how to decompose an RSP into two pyramids. First we find its kernel; this is straightforward once the extreme points are known, see Chapter 3 2.2. The kernel contains at least one point x. Then we construct a horizontal line through x. This line intersects the polygon P at two points say p and q. Because P is star-shaped p and q are located on the x_{min}, x_{max} edges. With these two *Steiner* points p and q we can split the polygon into two pyramids (see Figure 6.5). The two pyramids are (p, ..., t_ℓ, t_r, ..., q) and (q, .., b_r, b_ℓ, ..., p). In Chapter 3.2.2 we presented a procedure to determine the kernel of a rectilinear polygon in linear time. To compute the intersection points p, q is then trivial, so that the entire procedure can be executed in linear time.

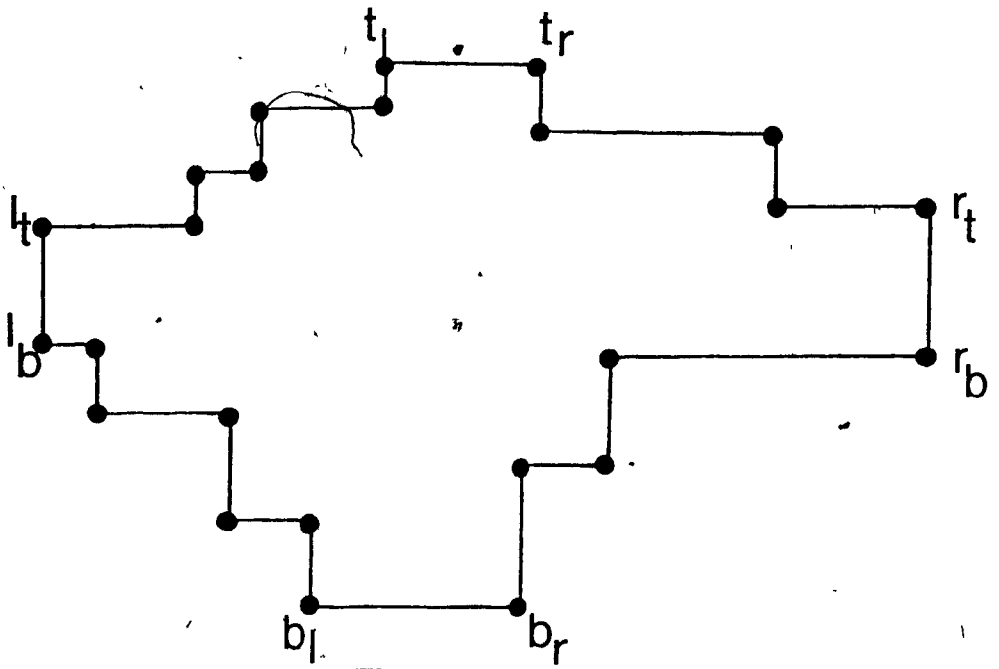


Figure 6.3

A rectilinear star-shaped polygon.

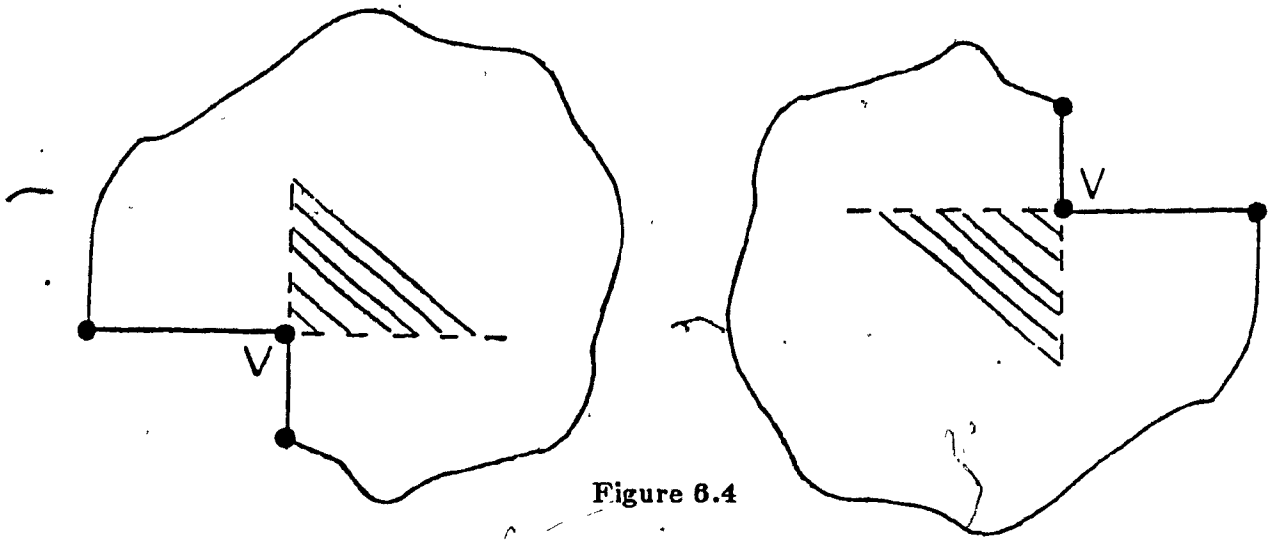


Figure 6.4

(a) An up-looking reflex vertex

(b) A down-looking reflex vertex

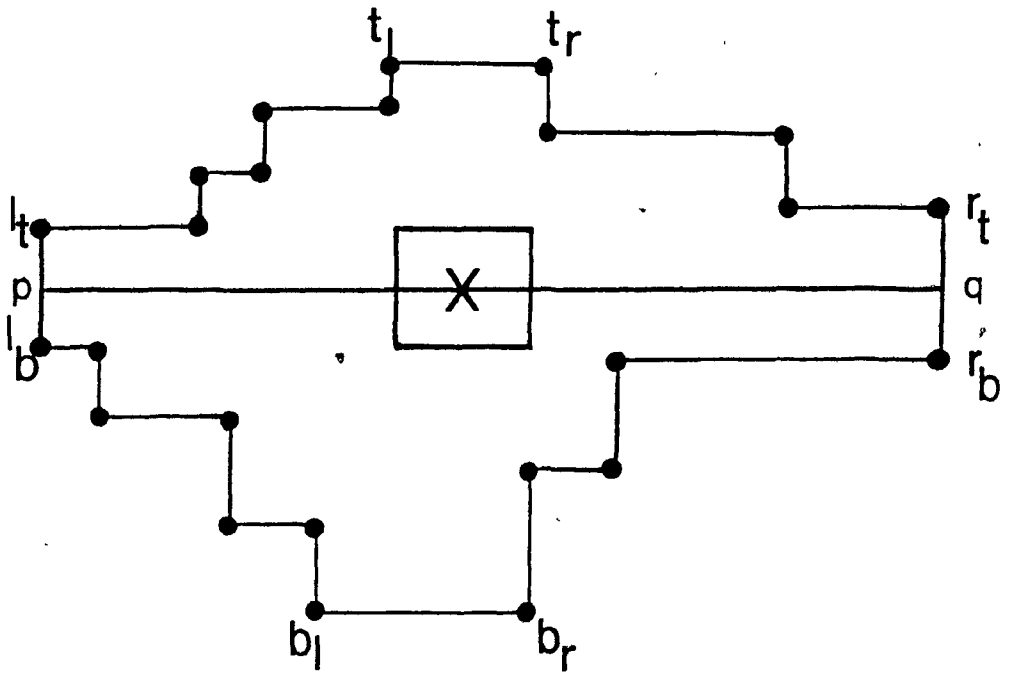


Figure 6.5

Partitioning a rectilinear star-shaped polygon into pyramids.

6.2.2. Quadrilaterization of Pyramids

In this section we show how to partition a pyramid into convex quadrilaterals. Without loss of generality we assume that the pyramid is oriented in such a manner that pq is the top edge. Let L = $\ell_1, \ldots, \ell_n$ and R = $r_1, \ldots, r_m$ be two lists of vertices with n,m elements, respectively.

Algorithm 6.1: Pyramid Quadrilaterization

Input : A pyramid P.

Output : A convex quadrilaterization of P.

Step 1:
Starting at the y_{min} edge rs construct two lists L, R:
L := all reflex vertices on the left stair
such that $y(\ell_1) < \ldots < y(\ell_n)$;
R := All reflex vertices on the right stair
such that $y(r_1) < \ldots < y(r_m)$,

Step 2:
i:=1; j:=1; (* initialize *)
while (i ≤ n and j≤m) **do**
begin
join ℓ_i **and** r_j,
if $y(\ell_i) \leq y(r_j)$
then i:=i+1
else j:=j+1
end

Step 3:
if i>n
then join all points $r_j, \ldots, r_m$ to p;
if j>m
then join all points $\ell_i, \ldots, \ell_n$ to q

Remark 6.1. *In Algorithm 6.1, if* $y(r_j) < y(\ell_i) < y(\ell_{i+1}) < \ldots \ldots < y(\ell_k) \leq y(r_{j+1})$ *then all points* $\ell_i, \ldots, \ell_k$ *are joined to* r_{j+1}, *as illustrated in Figure 6.6.*

Lemma 6.1. *Pyramids can be quadrilaterized in linear time.*

Proof: If both stairs are empty, the polygon is already a convex quadrilateral and the

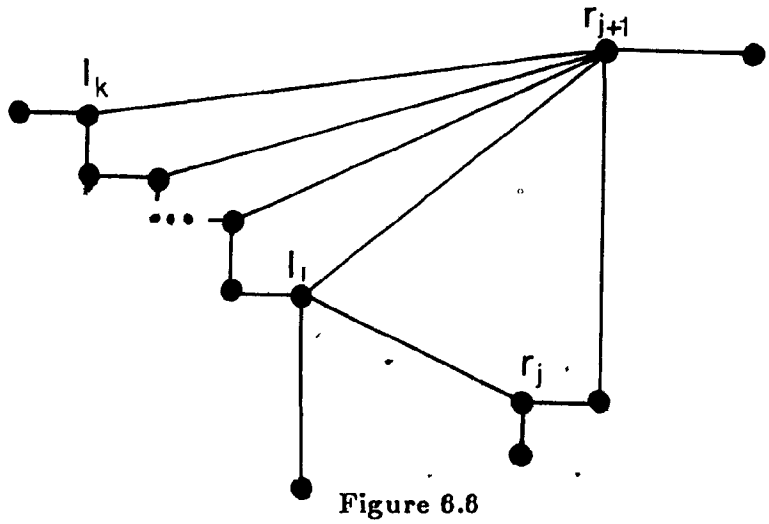


Figure 6.6

All reflex vertices between ℓ_1 and ℓ_k are joined to r_{j+1}.

algorithm stops. Let the polygon have at least one non-empty stair. The algorithm creates polygons, say $Q_1, \ldots, Q_d$.

We have to show that:

(i) $Q_1, \ldots, Q_d$ are quadrilaterals.
(ii) $Q_1, \ldots, Q_d$ are convex.
(iii) No two polygons Q_i, Q_j intersect (properly).
(iv) The union of all $Q_1, \ldots, Q_d$ is equal to the polygon.
(v) The algorithm runs in linear time.

(i) From the algorithm it is clear that every pair of consecutive reflex vertices in the left (right) stair is joined to a common vertex in the right (left) stair, therefore the components are quadrilaterals

(ii) Let v be a reflex vertex on the left stair and refer to Figure 6.7. The concavity at v is broken by any diagonal located in region B. The algorithm produces a diagonal vw such that

(a) w is located on the other stair; therefore vw is not located in region A.

(b) $y(v) \leq y(w)$; therefore vw is not located in region C.

All other reflex vertices are handled in a similar way. Since all other angles in the quadrilaterals are either $\frac{\pi}{2}$ or less, the quadrilaterals are convex.

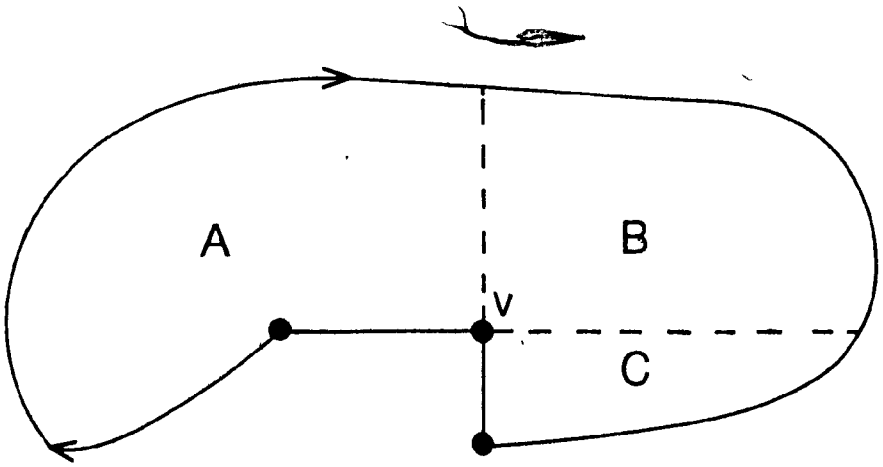


Figure 6.7

The concavity of v is broken by inserting any diagonal located in region B.

(iii) Since the vertices are scanned in order by increasing y-coordinates, it follows trivially that no two quadrilaterals intersect.

(iv) Each reflex vertex is contained in at least two quadrilaterals. Each convex vertex (except r, s (p, q) the bottom (top) vertices) is located between two reflex vertices and is therefore contained in the respective quadrilateral. The vertices r,s (p,q) are included in the lowest (highest) quadrilateral. Therefore the union over all quadrilaterals is equal to the entire polygon.

(v) The running-time of Step 1 is proportional to the sum of the number of elements in both lists and is therefore O(n). For Step 1 no sorting is needed as the stairs are monotone. Adding an edge in Step 2 and 3 takes constant time and is executed O(n) times. Therefore the total run-time is O(n). ■

Remark 6.2. *Let v be the vertex (on the left stair) joined to p and let r be the vertex (on the right stair) joined to q. If y(v)> y(r) the algorithm joins reflex vertices to q. Otherwise y(v) ≤ y(r) and the algorithm joins reflex vertices to p. This follows immediately from the algorithm.*

Remark 6.3. *We now describe a generalization of a pyramid which will be useful in the next section for merging decomposed pyramids. Let P be a pyramid with pq and sr as top and bottom edges, respectively. Let LS(r,p) and RS(q,s) denote the left and right stairs, respectively, of P. In a pyramid any horizontal edge $p_i p_j$ is such that $y(p_i) = y(p_j)$. We can generalize these pyramids to worn-down pyramids by allowing $y(p_i) \leq y(p_j)$ in LS(r,p) and letting $y(p_j) \leq y(p_i)$ in RS(q,s) for any top edge $p_i p_j$ in P. We also allow the base-edge of the pyramid, i.e. the y_{min}-edge, to be non-horizontal, provided that both its endpoints remain below the lowest reflex vertex in P. See Figure 6.8 for illustration. To obtain a concise notation let us call the resulting polygonal chains worn-down stairs. It is easy to see that with Algorithm 6.1 these worn-down pyramids can also be decomposed into convex quadrilaterals since the interior angles that were previously $\frac{\pi}{2}$ are still less than π and are therefore convex. It follows that worn-down pyramids are rectilinearly convex, edge-visible polygons with an alternating sequence of vertical edges and possibly slanted edges.*

6.2.3. Algorithm and Proof of Correctness

In this section we present an algorithm that solves the problem of decomposing a rectilinear star-shaped polygon P into convex quadrilaterals. This result was the first known algorithm to quadrilaterize any class of rectilinear polygons. We state it in its formulation as given in [SaT81].

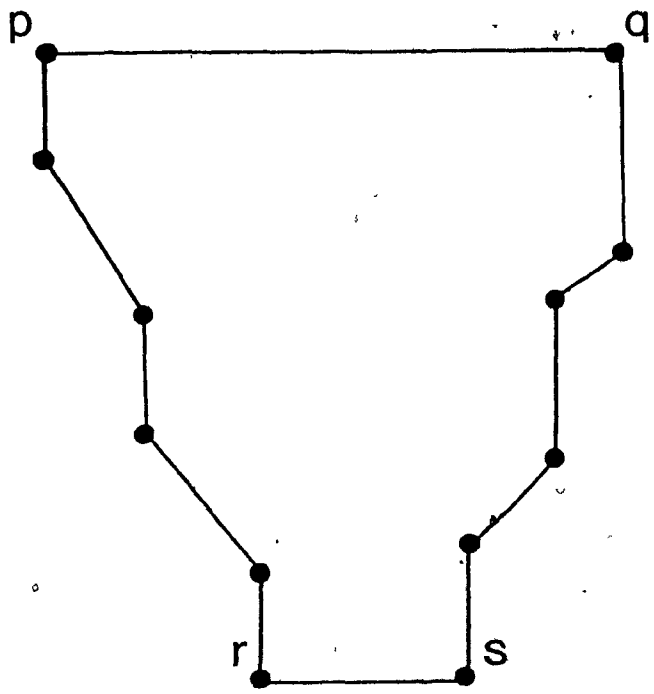


Figure 6.8

A worn-down pyramid.

Algorithm 6.2: Quadrilaterization of Star-Shaped Rectilinear Polygons

Input: A rectilinear star-shaped polygon P.

Output: A convex quadrilaterization of P.

Step 1:
Decompose the polygon P into two pyramids.

Step 2:
Decompose the pyramids obtained in Step 1 into convex quadrilaterals.

Step 3:
Merge the decompositions (the details are given below).

Theorem 6.1. *The Algorithm 6.2 partitions a given rectilinear n-vertex star-shaped polygon into convex quadrilaterals in O(n) time.*

Proof: It was shown previously that the partitioning step into pyramid, Step 1, can be done in O(n) time. In Step 2 we apply Algorithm 6.1 to both polygons resulting from Step 1. Using Lemma 6.1 this takes linear time. According to Step 3 we have to merge the two decompositions obtained in Step 2. As the polygon P is star-shaped there are only two cases to be considered: (a) The intervals $[y(\ell_t), y(\ell_b)]$ and $[y(r_t), y(r_b)]$ are such that neither is contained in the other, and (b) One of the intervals $[y(\ell_t), y(\ell_b)]$, $[y(r_t), y(r_b)]$ is contained in other.

(a) Without loss of generality let $y(\ell_t) > y(r_t)$ and $y(\ell_b) > y(r_b)$, and refer to Figure 6.9. We have noticed previously that in this case the top polygon-decomposition contains edges from p_i to p for some values of i. Similarly the bottom polygon-decomposition contains edges from q_j to q for some j.

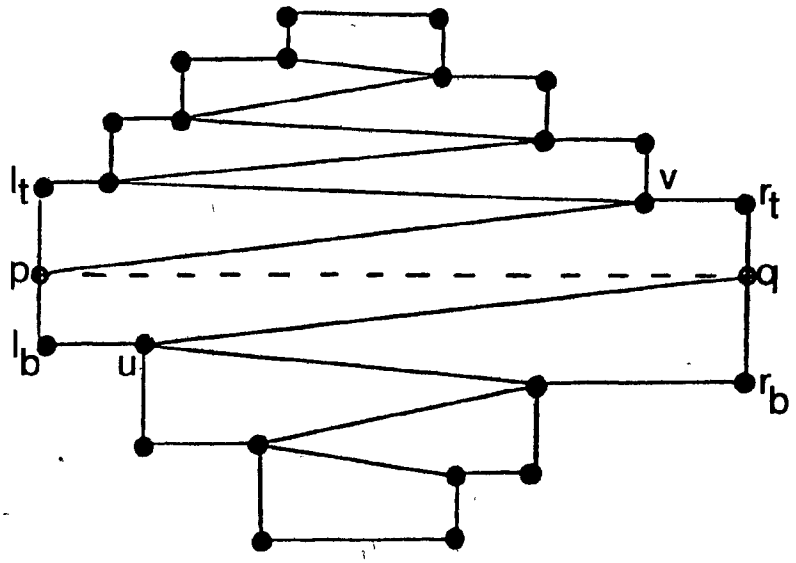


Figure 6.9

The quadrilateralizations of the top and bottom pyramids.

Now replace all edges $p_i p$ in the top polygon by $p_i \ell_b$ and all edges $q_j q$ in the bottom polygon by $q_j r_t$.

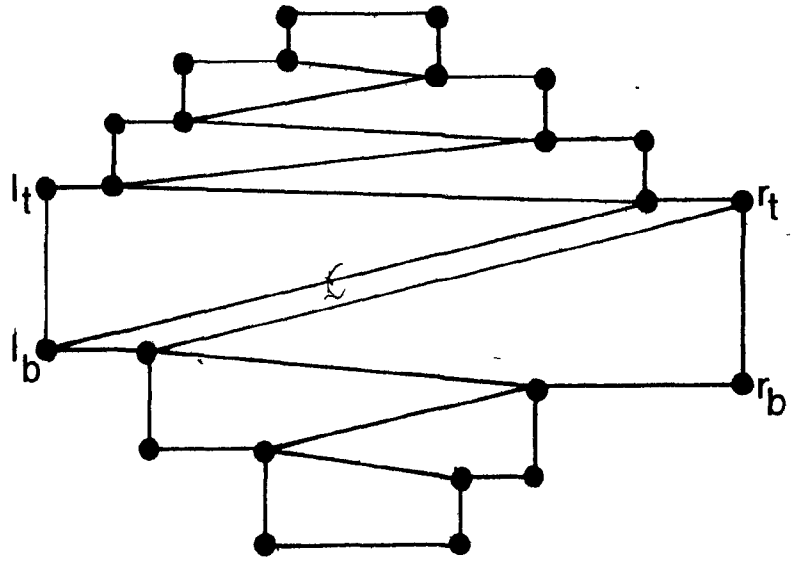


Figure 6.10

A quadrilateralization after merging the partial quadrilateralization of Figure 6.9.

By doing this we replace in each of the concerned quadrilaterals, located above and below pq, exactly one vertex, thus obtaining a new quadrilateral. As ℓ_b is below p and r_t is above q, these quadrilaterals remain convex and cannot intersect. Furthermore, the hexagon formed by deleting pq, namely (p, v, r_t, q, u, ℓ_b), clearly becomes a convex quadrilateral, since it contains two parallel edges $\ell_b u$ and $v r_t$. This procedure applied to the polygon of Figure 6.9 yields the decomposition of Figure 6.10.

(b) Without loss of generality let $[y(\ell_t), y(\ell_b)]$ be contained in $[y(r_t), y(r_b)]$ and refer to Figure 6.11. In this case $y(\ell_t) \leq y(r_t)$ and $y(\ell_b) > y(r_b)$, and with the Remark

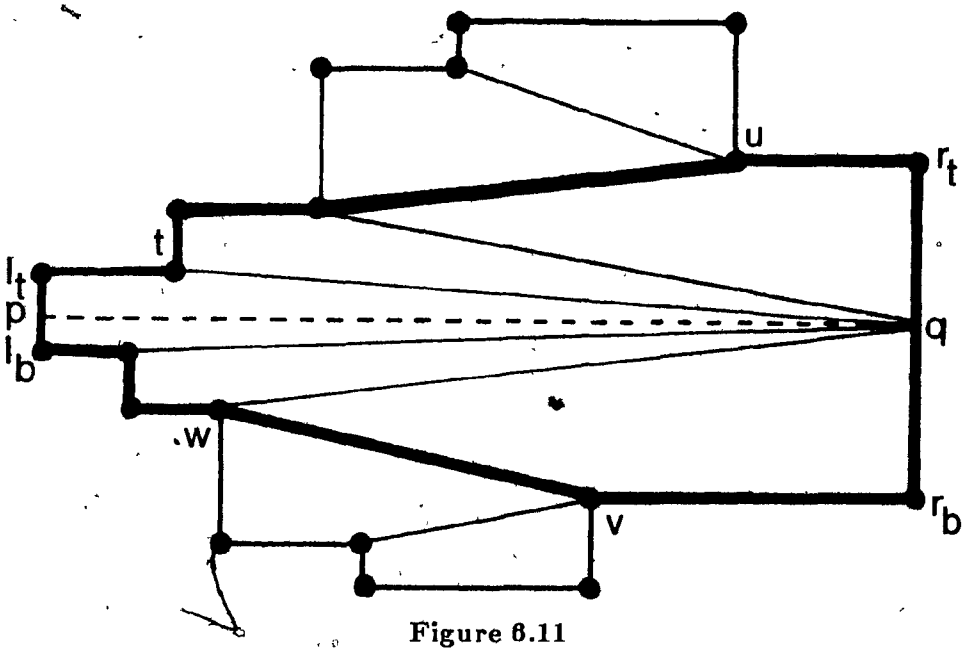


Figure 6.11

A worn-down pyramid P^*, outlined in bold, is created.

6.2. we see that in each pyramid one or more reflex vertices are joined to q. In the top polygon, let t be the highest vertex joined to q, and let u be the lowest vertex joined to t (u is not equal to q). In the bottom pyramid, let w be the lowest vertex joined to q, and let v be the highest vertex joined to w (v is not equal to q). Delete all decomposition-edges to q. Furthermore delete p, q and pq. Then consider the polygon P^*=(u, r_t, r_b, v, w, ..., t). As v is not located above w, and u is not located below t, P^* is a worn-down pyramid. Rotate P^* by $\frac{3\pi}{2}$ (clockwise) and quadrilaterize it. (In practice one would implement Algorithm 6.2 such that the rotation is avoided). The bottom-to-top approach of Algorithm 6.2 would become a left-to-right (or right-to-left) approach.) The new decomposition edges inside P^* cannot interfere with the decompositions outside P^*. Applying this procedure to the decomposition of Figure 6.11 yields that

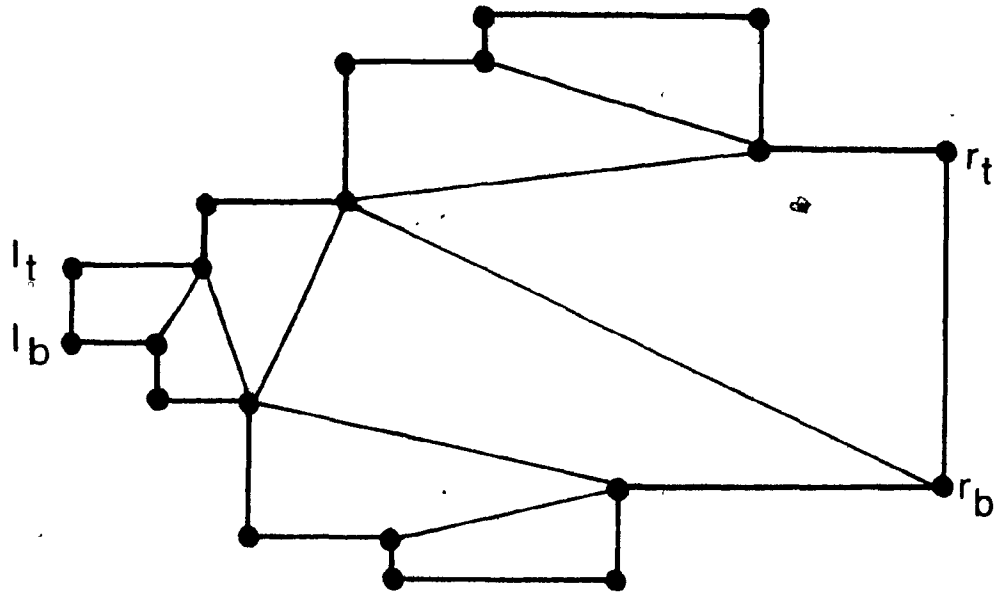


Figure 6.12

After merging the quadrilateralizations of Figure 6.11.

shown in Figure 6.12. The merging step is, therefore, either straightforward or applies Lemma 6.1 to P^*. In both cases the run-time is O(n). We can conclude therefore that the total time required to decompose a rectilinear star-shaped polygon into convex quadrilaterals is O(n) ■

6.3. Quadrilaterization of Monotone Rectilinear Polygons

In this section we extend the results of Section 6.1 and 6.2 by developing a linear-time algorithm for finding a convex quadrilaterization of a monotone rectilinear polygon P. The following algorithm results from merging algorithms designed by [El82] and the author. Recall from Section 3.2.3 that any monotone rectilinear polygon is monotone in the x-direction or the y-direction. W.l.o.g we assume this direction to be the y-direction. The two resulting extremal chains are referred to as the left chain and right chain, respectively. An edge pq of a simple polygon is called a *top-segment* if by

replacing pq by pq#p and qq#p one vertical edge and one top-edge are created (see Figure 6.13). This generalizes the notion of a top-edge in a rectilinear polygon. In a similar fashion, we can define *bottom-segments*. We will refer to bottom-segments and top-segments as *segments* of the polygon. Note that we consider top-edges and bottom-edges as top-segment, bottom-segments, respectively, however, a vertical edge is not a segment.

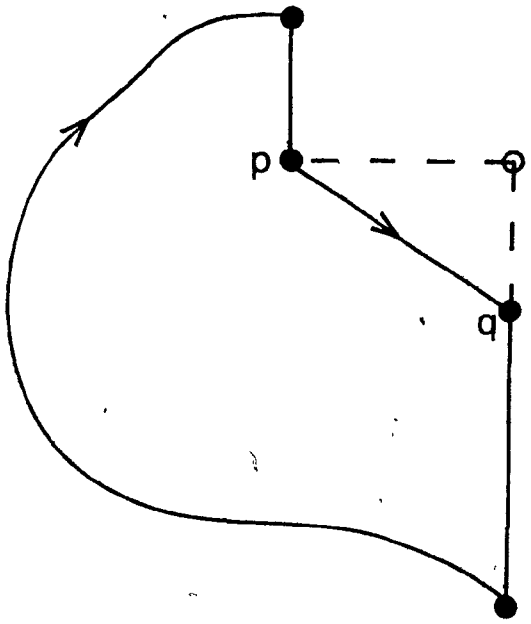


Figure 6.13

A top-segment pq; pq#p is a top-edge and q#pq is a vertical edge.

6.3.1. Algorithm and Proof of Correctness

We will first sketch the algorithm. The algorithm creates a list, $e_1,...,e_n$, of all horizontal edges in P, sorted by decreasing y-coordinates. The edges in this list are examined one by one starting with the edge of maximum y-value. In case that a top-edge is encountered, it is pushed onto a stack, S. In case that a bottom-edge e_i is encountered depending on the relative position of the top of stack edge and e_i one or two diagonals are inserted. These diagonals partition P into two or three polygons,

respectively. One such polygon will be shown to be a worn-down pyramid. For this pyramid a quadrilaterization is obtained by applying Algorithm 6.1. The remaining polygon will be shown to be quadrilaterizable. The top_of_stack-edge is either an edge of P or a diagonal inserted into P at a previous stage. We will store the items in S as pairs of the form [d, f] The first field d, is either an edge or a diagonal. The second field f, takes a value in {L, R, B}, depending on whether the endpoints of the segment belong to the left, right, or both chains respectively The field-value of an edge e_i is denoted by f_i. Let d, d_j be two diagonals with rightmost endpoints located on the right chain. Then d extends *further inside* P than d_j if its leftmost endpoint is further to the left than the leftmost endpoint of d_j. Similarly, for diagonals whose leftmost endpoints are located on the left chain. We are now able to state the algorithm.

Algorithm 6.3: Quadrilaterization of Monotone Rectilinear Polygons

Input: A monotone rectilinear polygon P.

Output: A quadrilaterization of P

```
sort the horizontal edges of P by y-coordinates,
let e_1, ..., e_n be the resulting sorted edge-list,
Initialize the stack S := [1, B];
(* The top edge of the first pyramid is pushed *)
(* its field-value indicates that it belongs to both chains *)
i:=2,
while i <> n or S <> empty do
   begin
      if i=n then
          call pyramid decomposition algorithm for stack content and e_i, exit.
      if e_i is a top-segment then
          (* case (a) *)
          begin
              (* augment the pyramid contained in S *)
              S := [i, respective field-value],
              i := i+1
          end
      else
          if e_i is a bottom-segment then
              begin
                  [e, f] := POP(S),
```

```
(* the top of stack is popped *)
if f=b or {e and e_i are on the same chain } then
    (* case (b) *)
    begin
        join e and e_i by a diagonal d;
        (* A convex quadrilateral is cut off *);
        if e is further inside P than e_i
            or e has field value B then
            begin
                S := [d, respective field-value];
                (* pyramids are wearing down *)
                i := i + 1
            end
    end
else
    (* case (c) *)
    (* e, e_i are on opposite chains *)
    begin
        join e and e_i to form a convex quadrilateral;
        (* Two diagonals, called the higher and the lower, are inserted*)
        S := [the higher diagonal, respective field-value];
        (* A pyramid is cut off from the remaining polygon *)
        i := i+1,
        S := [the lower diagonal, field-value B]
    end
end (* if e_i is a bottom segment *)
end (* while loop *).
```

Theorem 6.2. *Algorithm 6.2 quadrilaterizes monotone rectilinear polygons in linear time.*

Proof: Since Algorithm 6.3 examines each edge at most a constant number of times, the linearity of the time-complexity follows. We show the correctness of the algorithm. We first define the stack property, called property (X). Then we show that after each iteration through the main loop of the above algorithm, the stack property is maintained. The stack property (X) is defined as,

(i) The stack S contains the left and right chain of a worn-down pyramid and no other edges.

(ii) No two segments in S contain points with the same y-value.

(iii) If e_i is the next edge (or segment) examined in the while-loop of Algorithm 6.3, then e_i is connected by a vertical line to the lowest edge in S located on the

same chain as e_i.

We will motivate the definition of the stack property (X). If (X) holds then if at some time during the execution of the algorithm, both endpoints of the two chains contained in the stack are joined, a worn-down pyramid is completed. By Remark 6.3 these worn-down pyramids are quadrilaterizable. A call to Algorithm 6.2 has the effect of quadrilaterizing this pyramid, thereby emptying the stack contents. The last edge examined by Algorithm 6.3, is the edge with maximum y-coordinate. Since this edge joins the endpoints of the monotone chains stored in S, by the above, Algorithm 6.3 will always be able to terminate with an empty stack. The main step of the proof is to show that the stack property holds after each iteration of the main while-loop. We give an inductive argument on the edge number, i, in the sorted list of horizontal edges in P.

Initially, when $i=1$, the edge e_1, with maximum y-coordinate, is pushed onto S. It should be clear that in this case the stack property (X) holds. Now let for $i-1$, S satisfy (X). We have to show that after e_i has been examined, (X) is still satisfied. For this we distinguish between the cases: *case (a): e_i is a top-edge; case (b) and (c): e_i is a bottom-edge.* The case enumeration corresponds to the algorithm description. Without loss of generality let us assume that the current edge e_i is located on the right chain.

(a) Edge $e_i = p_{i-1}p_i$ is a top-edge. Consider the lowest vertex, p_k, in S which is located on the right chain. By stack property (iii), p_{i-1} and p_k are connected by a vertical line. The algorithm will push the edge e_i onto S. As e_i is a top-edge and the edges are examined in sorted order, properties (i) and (ii) hold. The next horizontal edge on the right chain is e_{i+2}. In a rectilinear polygon horizontal and vertical edges are alternating thus property (iii) also remains true.

(b) Edge e_i is a bottom-edge. Let the top_of_stack segment be pq. If pq is a horizontal edge then clearly both p and q have minimum y-value in S. Otherwise, by (iii), p and q are below any other vertex currently stored in S. In case (c), we will discuss the case that pq and e_i are located on opposite chains, i. e. both endpoints p

and q are on the left chain. All other placements of p and q onto the chains are illustrated in Figure 6.14. As p and q are on opposite chains, by the initial remark, the stack was empty before pq was pushed. Thus after popping pq, S is again empty. The diagonal pp_i is inserted into P to form a convex quadrilateral p, q, p_{i-1}, p_i. The algorithm subsequently pushes pp_i and thus, similar, to the initialization Step, (i=1), the stack property (X) holds. In other words after pushing the segment a new pyramid is initialized.

We now examine the case that both p and q are on the right chain. By induction, property (iii) holds and thus p_i, p., q, p_{i-1} form a convex quadrilateral. The algorithm, in this case, correctly inserts the diagonal pp_i cutting of a convex quadrilateral. If this diagonal is a top-segment, a fact that the algorithm detects by verifying that p_i is to the right of p, then the stack is augmented by a push-operation PUSH(pp_i) thereby satisfying (i). Otherwise, in the case that pp_i is a bottom-segment, the algorithm does not push any segment or edge. Furthermore, for the next execution of the main while-loop, e_i is merely replaced by pp_i. It remains to show that for both situations properties (ii) and (iii) hold.

By induction, property (ii) holds and thus no point in S located on a segment or vertical edge on the left chain has a y-coordinate between that of p and q. The diagonal (edge) with minimum distance in y-coordinate to e_i is pq, thus no segment other than pp_i contains a point with y-coordinate between that of p and p_i. Property (ii) is thus satisfied either in case that the segment pp_i is inserted into S or that e_i is replaced by pp_i. In a rectilinear polygon, horizontal and vertical edges alternate. In the case that pp_i is pushed, e_{i+1} satisfies property (iii). Otherwise no edge is pushed, and thus in the main loop, pp_i replaces e_i. This by induction, property (iii) must still remain true.

(c) We now examine the case that both p and q are located on the left chain,

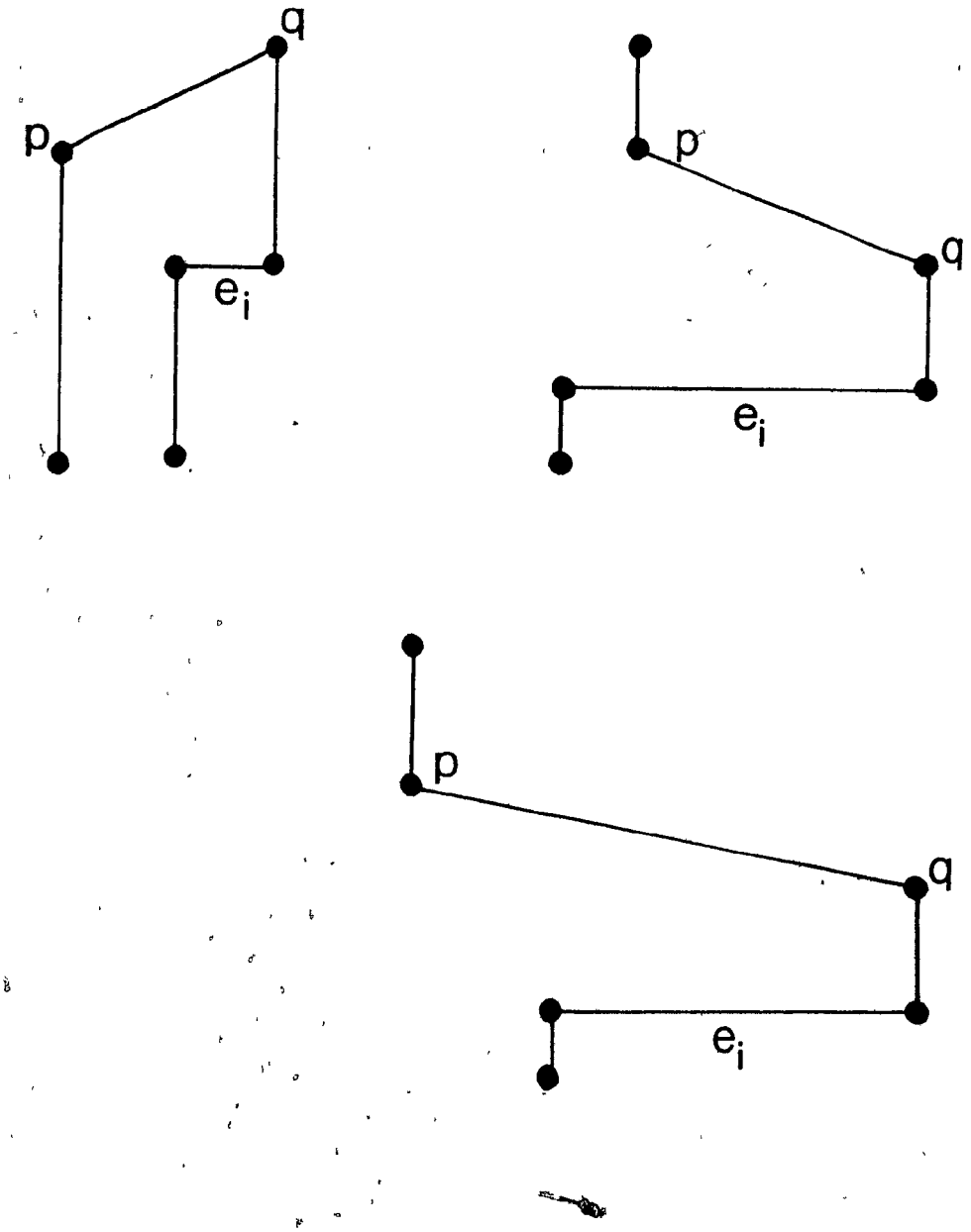


Figure 6.14

Placement of p,q when q is on the right chain.

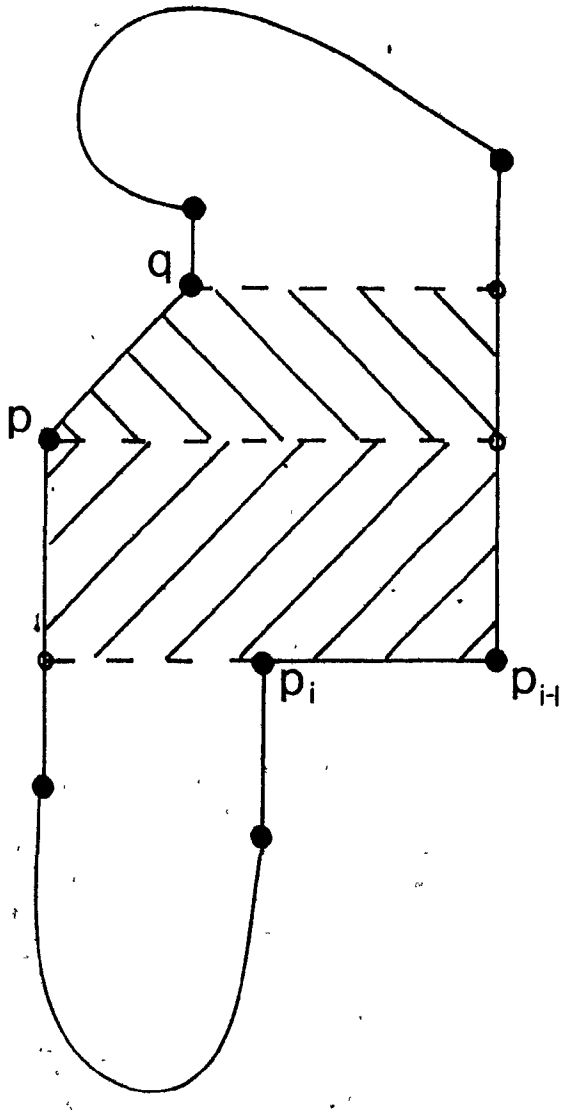


Figure 6.15

Both p, q are on the left chain. The shaded areas are empty.

(Figure 6.15). By induction, property (ii), the quadrilateral p, p_{i-1}#p, p_{i-1}#q, q is empty. (In case that pq is a horizontal edge the quadrilateral collapses and is therefore trivially empty). As pq is the top_of_stack segment also p_{i-1}#p, p_{i-1}, p#p_{i-1},p is empty. Since pq is a top-segment the vertices p_{i-1}, p_i, p, q form a convex quadrilateral. By inserting the segment qp_{i-1} the algorithms completes a worn-down pyramid stored in S. This worn-down pyramid is subsequently quadrilaterized by calling the pyramid quadrilaterization algorithm. As q, p_{i-1}#q, p_{i-1}, p#p_i is empty, pp_i is a top-segment of a new worn-down pyramid to be stored in S. The validity of Properties (i) and (ii) follows. Both p and p_i are linked to the unexamined chains of P by vertical edges thus (iii) holds. ■

The following remark is important for Section 6.4.

Remark 6.4. *Recall that the horizontal edges of P are sorted according to the y-coordinates. Let the resulting sorted list of edges be* e_1, ..., e_n. *Let* e_i, e_{i+1} *be two bottom-edges consecutive in the list and adjacent on the same chain. Algorithm 6.3 creates convex quadrilaterals, one of which contains the reflex-vertices of both* e_i, *and* e_{i+1}.

Proof: As e_i is a bottom-edge, a diagonal d joining e_i and some vertex p_i is inserted by Algorithm 6.3, depending on whether cases (b) or (c) apply, respectively. According to the description of Algorithm 6.3 this diagonal d is popped as soon the next bottom edge, e_{i+1}, is considered. As e_i and e_{i+1} are on the same chain, d and the edge e_{i+1} are joined by a diagonal to form a convex quadrilateral. Thus the reflex vertices of e_i and e_{i+1} are located in the same quadrilateral. ■

6.3.2. Polygons with the Alternating Rectilinear Property

We will define a class of monotone polygons, not necessarily rectilinear, which admit quadrilaterization. A polygon with the *alternating rectilinear property* is defined as a monotone polygon with these properties:

(1) Every even edge $e_2, \ldots, e_n$ is vertical,

(2) Let p, q be two points located on the boundary of P s.t. p, q are visible from each other along a horizontal line-segment. If p is located on a non-rectilinear segment then q is on a vertical edge of P.

Lemma 6.2. *Monotone polygons with the alternating rectilinear property can be quadrilaterized in linear time.*

Proof: We show that the Algorithm 6.3 can be employed, with only minor modification, to quadrilaterize monotone polygons with the alternating rectilinear property. Let $m \geq 1$ denote the number of bottom-segments in P. We give a proof by induction on m. In the case that $m=1$, P is a worn-down pyramid, since by Property (2) both endpoints of the bottom-segment are below any reflex vertex and all other edges belong to the stairs of a worn-down pyramid. We know that such a pyramid is quadrilaterizable. The algorithm performs the quadrilaterization by stacking all edges until the bottom-segment is encountered then the call to the Algorithm 6.2 completes the procedure in this case.

Let us now assume that any monotone polygon with the alternating rectilinear property containing $m-1 \geq 1$ bottom-segments is quadrilaterizable. The highest bottom-segment in a monotone polygon with the alternating rectilinear property and with m bottom-segments is denoted by e_1. Note that the definition of a quadrilaterizable monotone polygon allows us to speak of a highest bottom-segment, as the bottom-segments have non-overlapping y-coordinates. The discussion is analogous to cases (b) and (c) in Theorem 6.2. We only point out that in case (c), where a top-segment and a bottom-segment must be joined to form a convex quadrilateral, it is crucial that the segments have non-overlapping y-coordinates. Otherwise, as illustrated in Figure 6.16, as concave quadrilateral may be created. The property (2) forbids such

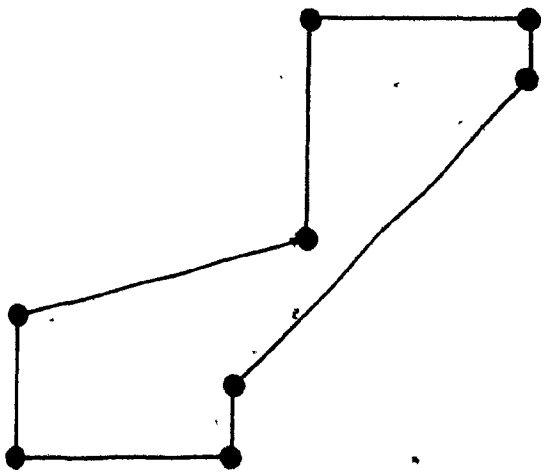


Figure 6.18

All non-rectilinear segments must have distinct y-values otherwise a convex quadrilateralization may not be achievable.

a situation. ■

6.4. Partitioning Rectilinear Polygons into Monotone Polygons

In this section, we describe an algorithm to partition a rectilinear n-vertex polygon into monotone polygons admitting quadrilaterization. The run-time of the algorithm will be shown to be O(n log n). Recall that an edge e is *visible* from a point x if there exists a point y on e, s.t. x sees y. An edge e is *strongly* visible from a point x if for all y on e, x sees y. An edge s is *strongly visible from an edge* t if there exists a point x on t from which s is strongly visible. An edge s is *completely* visible from an edge t if for all x on t the edge s is strongly visible from x. Complete visibility is symmetrical.

Lemma 6.3. *[Avis, Toussaint] Let qp be any edge of a simple polygon, and let x be a point in P. If both q and p see x, then x is visible from the entire edge qp [AT81a].*

Lemma 6.4. *Let pq be any bottom-peak in a rectilinear polygon P. The edge with minimum distance under all horizontal edges located above pq and visible from either p or q, is called ab. Then,*

(i) If edge ab is a bottom-edge then edge qp is strongly visible from one of the endpoints of ab.

(ii) If ab is a top-edge then ab and qp are completely visible from each other.

Proof: (i) We first discuss the case that ab is a bottom-edge. By assumption ab is visible from p or q and ab is above pq. Assume that the vertex from which ab is visible is p. Then exactly one point, an endpoint of ab, is visible from p. We assume that this endpoint is a, otherwise a symmetric argument holds. By Lemma 6.3, it suffices to show that both p and q are visible from a. Now consider the triangle T = {a,p,q} and refer to Figure 6.17. If T lies inside P then p sees a. Otherwise, qa is intersected by a polygonal edge and as pq is a bottom-peak this edge is not adjacent to pq. Therefore there exists a vertex p_i inside T. As P is simple T is intersected by a horizontal edge with y-coordinate between those of qp and ab. The lowest such edge is visible from p, contradicting the choice of ab.

(b) Let ab be a top-edge being visible from p. Then there exists a point x on ab that is visible from p. To show that ab and pq are completely visible from each other, it suffices to prove that *(i) q is visible from x and (ii) for all points y on ab, p sees y.*

(i) can be shown in the same way as (a), we therefore omit this part. For (ii), we show that for all points y on ab, p sees y. Assume that y is a point on ab that is not visible from p. The line segment py is thus intersected by at least one polygonal edge. If there is exactly one such edge, then this edge is adjacent to ab and by the Jordan Curve Theorem ab cannot be a top-edge. If more than one edges intersect py, then there exists, similar to (a), a horizontal edge intersecting the triangle T={y,p,p}. This is

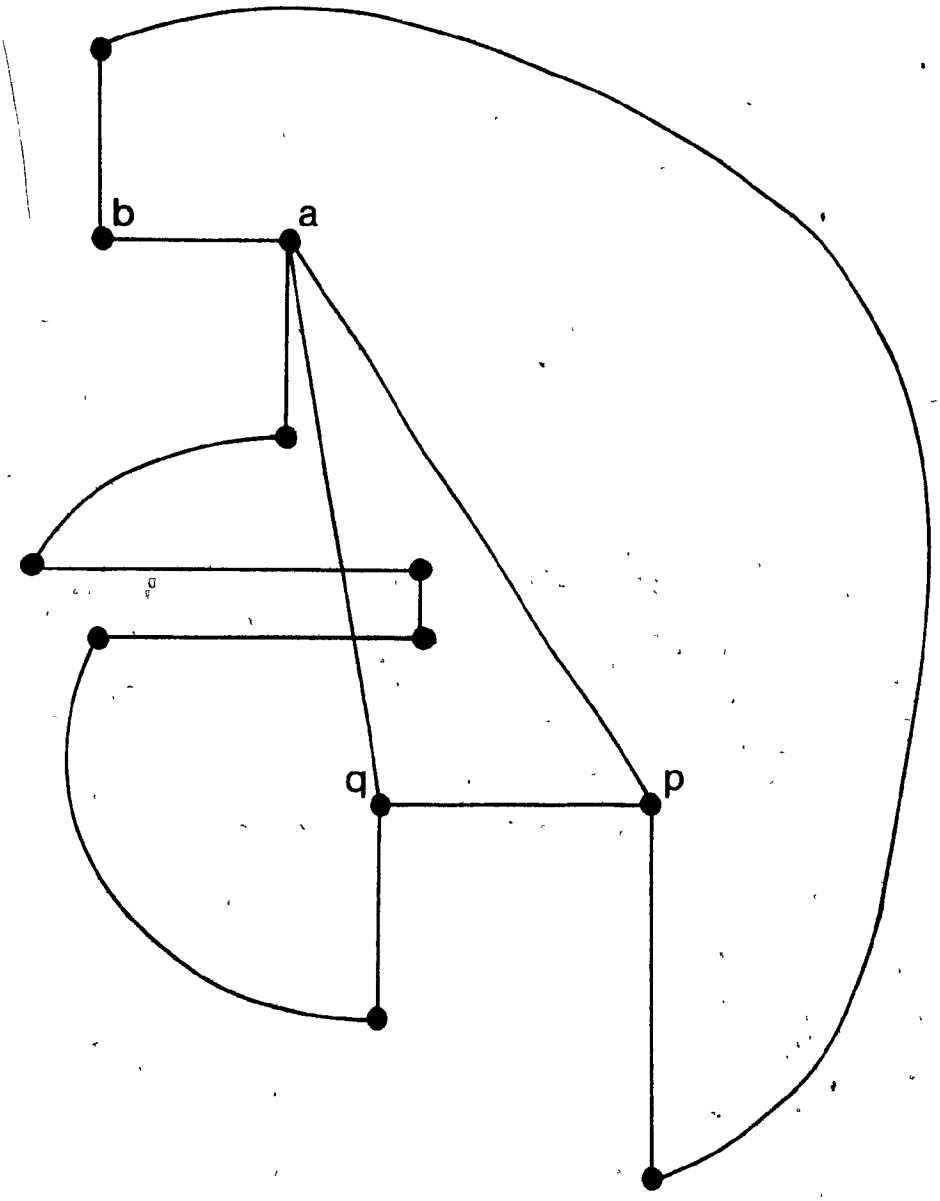


Figure 6.17

Let ab be an edge with minimum distance in y-coordinate and visible from p. If p is visible from a then pq is completely visible from a, otherwise a contradiction occurs.

illustrated in Figure 6.18. Again the lowest such edge must be visible from p, contradicting the choice of ab. Similarly, it can be shown that ab is strongly visible from p. The result then follows by applying Lemma 6.3. ■

We conclude that if ab is a bottom-edge, then the edge pq is strongly visible from exactly one endpoint of ab. Otherwise, if ab is a top-edge, then the edges ab and pq are completely visible from each other.

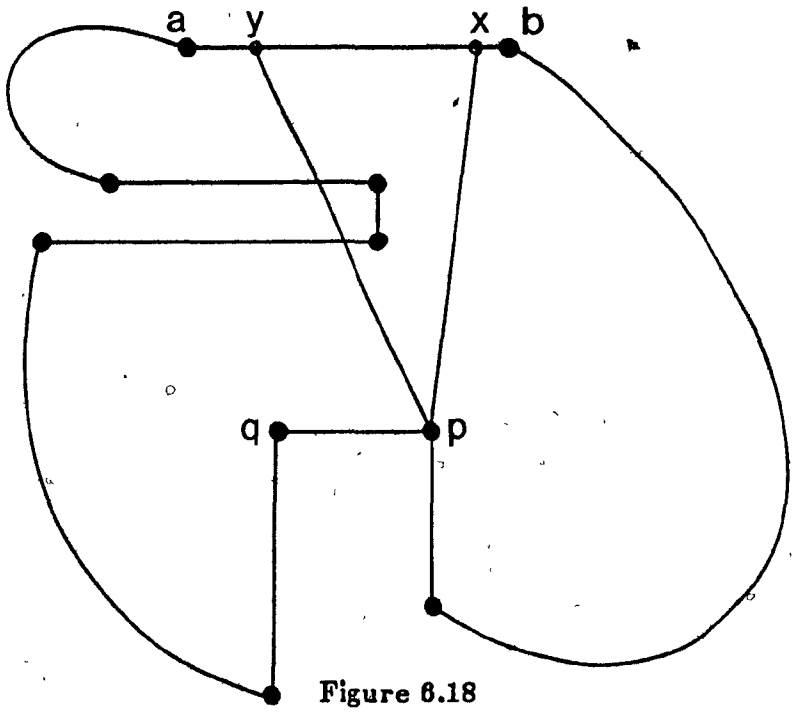


Figure 6.18

Let ab be a top-edge with minimum distance in y-coordinate and visible from q. If ab and pq are not completely visible from each other, a contradiction occurs.

Remark 6.5. *Let the assumptions be as in Lemma 6.4. Then, if ab is a bottom-edge, ab lies either completely to the left of q or completely to the right of p and the endpoint c of the adjacent edge facing pq, is lower than pq (see Figure 6.19).*

Remark 6.6. *Let pq be a bottom-peak and ab an edge with minimum distance under all horizontal edges visible from pq. Then ab is visible from one of the endpoints of pq.*

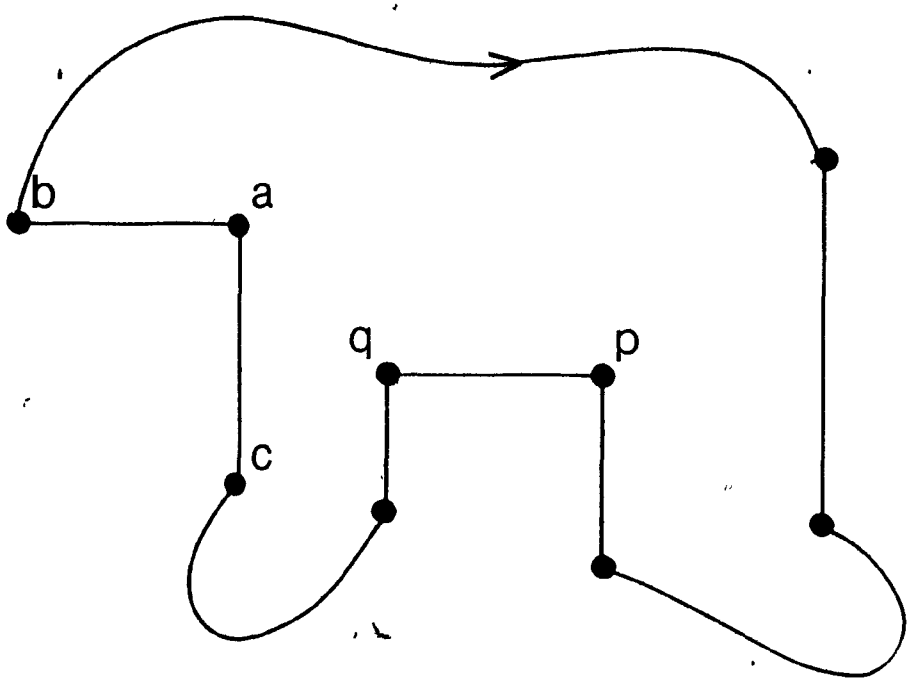


Figure 6.19

The bottom edge ab with minimum distance in y-direction visible from pq lies either completely to the left of q or completely to the right of p

Corollary 6.1. *Under the assumptions of Lemma 6.4, the rectangles (a#q,a,x,y) and (a,x,y,a#q) are empty (Figures 6.20 (a) and 6.20 (b)).*

6.4.1. Algorithm and Proof of Correctness

We are now able to state the algorithm for decomposing a rectilinear polygon P into quadrilaterizable monotone polygons. The algorithm uses a modification of the regularization technique developed by Lee and Preparata [LeP76]. Let the horizontal edges of P be sorted by y-coordinates s.t. the edges are labeled e_1, ..., e_m, where e_1 is the highest edge, and e_m is the lowest edge, and e_i is higher than e_{i+1}. We sketch the

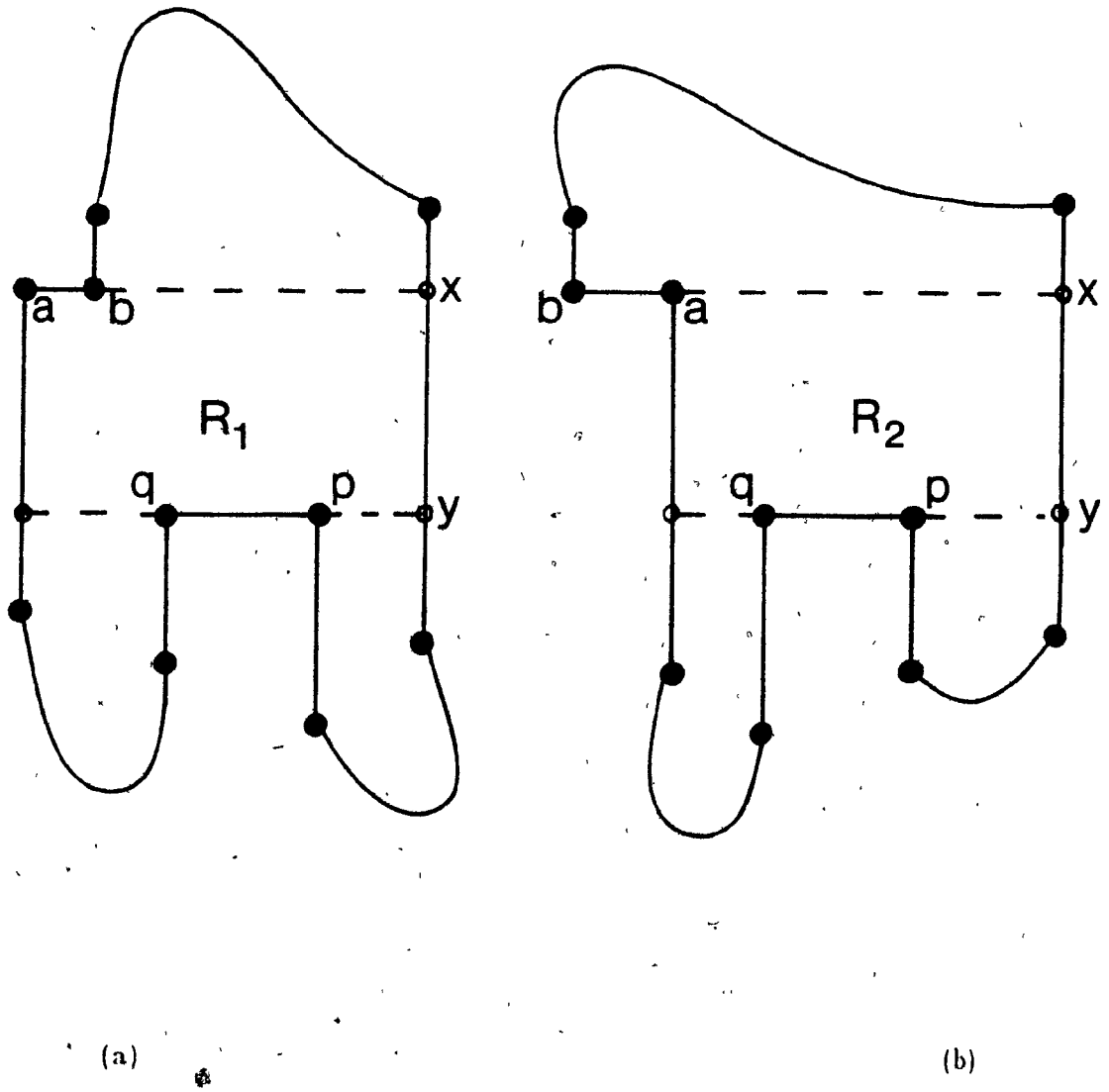


Figure 6.20

Edge ab is the closest edge visible from pq. The interiors of the rectangular regions R_1, R_2 are empty.

idea behind the algorithm. The horizontal list of edges is scanned in the same order from the highest to the lowest edge. When a peak e_i is encountered, the edge e with minimum distance in y-coordinate, from which e_i is visible, is found. If e and e_i are completely visible from each other, then e and e_i are joined by connecting their endpoints to form a convex quadrilateral. If e and e_i are not completely visible from each other, but e_i and e are strongly visible, then the endpoints of e_i and e having the minimum Euclidean distance from each other are joined. We will show that this eliminates all peaks and leaves quadrilaterizable polygons.

Algorithm 6.4: Overview of Partitioning into Monotone Polygons

Input: A rectilinear polygon P with n vertices

Output: A partitioning into monotone polygons admitting quadrilaterization

Step 1:
Sort the horizontal edges by y-coordinates, s.t. if e_1, ..., e_m is the ordered edge-sequence, then $e_1 = y_{max}$, $e_m = y_{min}$, and e_i is higher than e_{i+1}.

Step 2:
for each bottom (top)-peak e_i **do**
Find among all horizontal edges above (below) e_i and visible from e_i the one with minimum distance in y-coordinate; call this edge e
if e is a top (bottom) edge
then join the endpoints of e_i and e to form a convex quadrilateral.
else
join the endpoint of e, that is visible from e_i, to the closest (Euclidean distance) endpoint of e_i.

To achieve the desired efficiency, we will implement Step 2 using a modification of Lee's and Preparata's regularization technique.

After Step 1 has been executed, the horizontal edges are sorted by y-coordinates. The data structure used for the decomposition algorithm is a balanced tree (AVL or 2-3 tree), where nodes are records containing a vertical and a horizontal edge (or pointers to them) plus a field with values true or false. Let g_o be a vertical line at $-\infty$. Let H(j), V(j) be two field containing one horizontal and vertical edges, respectively. Let

F(j) be a field in record j, then F(j) is true if H(j) is a top-peak and no horizontal edge lower than H(j) and visible from H(j) has been encountered by the algorithm. F(j) is false, otherwise. A node(j) is therefore a triple [V(j), H(j), F(j)]. The in-order traversal of the tree T always yields a sequence node(0), ..., node(k) thought of as consecutive array-elements. Between the vertical edges V(j) and V(j+1) the minimum ordinate edge is stored in H(j). Each horizontal edge e_i is adjacent to two vertical edges, the left one, $e'(e_i)$, the right one, $e''(e_i)$. A bottom-peak e_i is encountered, if $In(e_i) = 0$; for this an insertion into the tree is made. If e_i is inserted between H(J) and H(j+1) then e_j is assigned the value of max(H(j), H(J+1)). The reader may initially ignore the third field, F(J), and consider bottom-peak and top-peak elimination in two separate passes through the polygon The additional field has been introduced to gain efficiency in that only one pass through the polygon is required. As we are scanning the polygon from top to bottom, top-peaks are encountered before they can be resolved. Therefore any unresolved top-peak is flagged (true) until its closest edge visible from it and below it is encountered. Then the appropriate diagonal(s) is (are) inserted and the field is reset to false.

Algorithm 6.5: Details of Partitioning into Monotone Polygons

Input: A sorted list of horizontal edges $e_1,...,e_m$ of polygon P as produced by Step 1 in Algorithm 6.4.

Output: A partitioning of P into monotone polygons, admitting quadrilaterization.

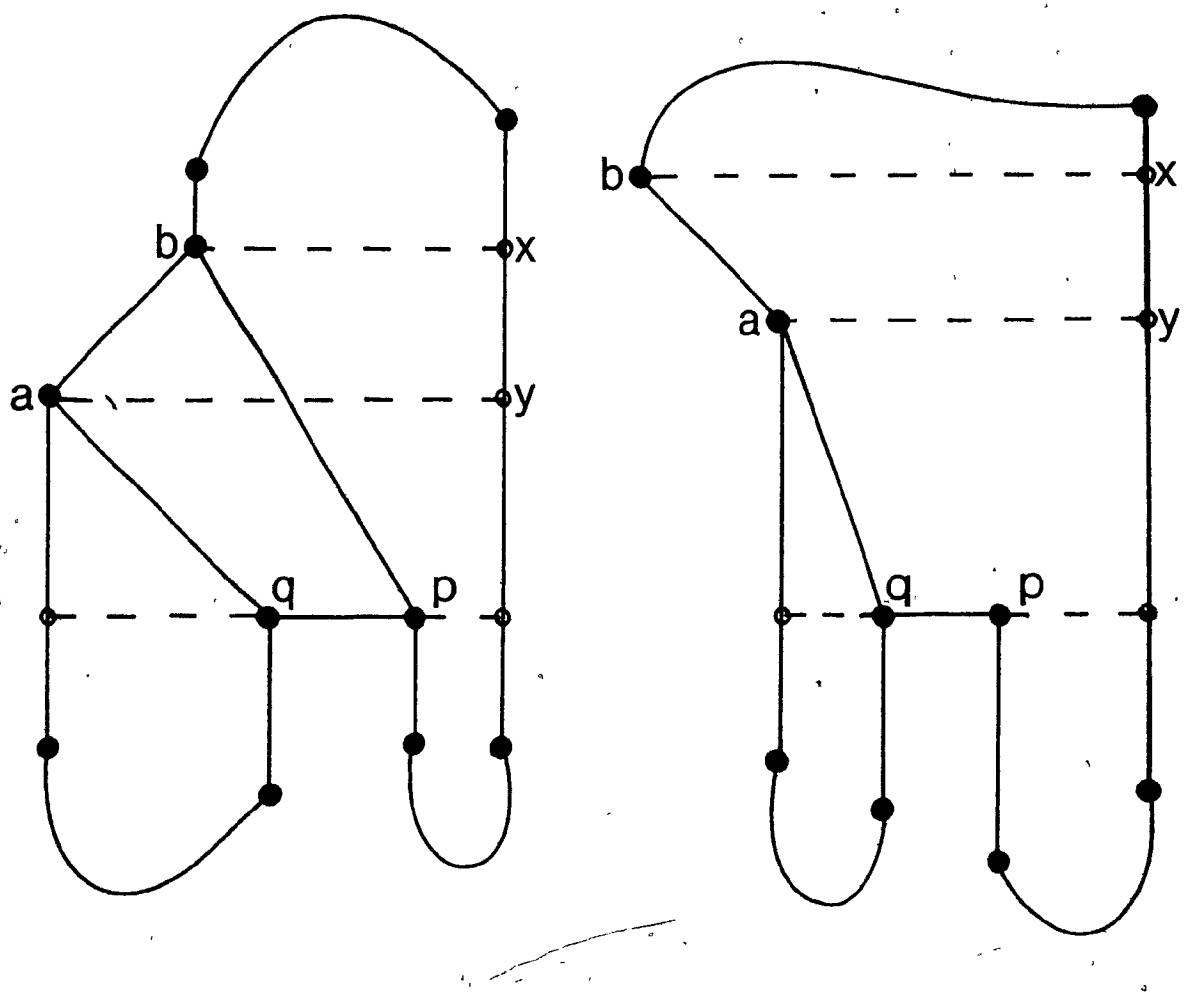
```
Initially
        H(0) := H(1) := H(2) := e_1;
        V(0) := g_0;
        V(1) := e'(e_1);
        V(2) := e''(e_1);

Let T be an initially empty AVL-tree (or 2-3 tree), insert [V(0),H(0),f], [V(1),H(1),f],
[V(2),H(2),f] into the tree T using the x-coordinates V(i), i=0,1,2 as the key.

i := 2;
while i < m do
   begin
```

```
j := smallest integer so that e_i is not to the left of V(j);
e := nil;
if e_i is a bottom-peak
   then e := H(j)
else
  if F(j) and V(j)=e'(e_i)
     then e := H(j)
  else
    if F(j+1) and V(j+1)=e''(e_i)
       then e := H(j+1),
if e <> nil
  then F(j) := F(j+1) := false;
if e_i is a top (bottom) edge
   and e is a bottom (top) edge
   then
      join the endpoints of e_i and e to form a convex quadrilateral
else
      join the endpoint of e closest to e_i to the closest endpoint of e_i;
if In(e_i) = 2 or Out(e_i)=0
   then
      begin
         H(j-1) := e_i;
         H(j+2) := e_i;
         Delete node(j) and node(j+1) in T;
         if e_i is a top-peak
            then
               begin
                  F(j-1) := true;
                  F(j+2) := true
               end
      end
else
   if In(e_i) = 1
      then
        begin
           H(j) := e_i;
           H(j-1) := e_i;
           V(j) := outgoing edge of e_i
        end
   else
      if In(e_i) = 0
         then
            begin
               INSERT [e'(e_i),e_i,false] and [e''(e_i),e_i,false]
               between node(j) and node(j+1) in T
            end;
i := i + 1
end.
```

Lemma 6.5. *Algorithm 6.4 decomposes, in O(n log n) time, a simple, n-vertex rectilinear polygon into monotone polygons admitting quadrilaterization.*

Proof: For ease of notation, we will assume for the remaining part of this section that *monotonicity* stands for monotonicity in the y-direction. Similarly, *closest* stands for minimum distance in y-direction. Let P be a polygon with the following property, (X): *P is rectilinear except for the edges located above the highest horizontal peak in P. These edges are monotone chains with the alternating rectilinear property.* We show by induction on the number, k, of horizontal peaks in P that Algorithm 6.4 decomposes P into monotone polygons that are quadrilaterizable. If k=0 then P is a monotone polygon with the alternating rectilinear property and thus P is quadrilaterizable. We assume that all polygons with k−1 horizontal peaks satisfying property (X) can be partitioned into monotone polygons admitting quadrilaterization, if Algorithm 6.4 is applied. Now, let P be a polygon satisfying property (X) containing k horizontal peaks. Let e_i=pq be the highest such peak. W.l.o.g. assume that e_i is a bottom-peak. A similar argument can be made otherwise. Algorithm 6.4 first determines the closest edge, e_j, visible from e_i. Two cases arise depending on whether e_j=ab is a top or bottom-segment, respectively. Refer to Figure 6.21 (a), (b). The point x,y are obtained by intersecting horizontal lines through both a and b with the edge opposite to ab. The alternating rectilinear property ensures that both x and y are on the same vertical edge. Analogously to Corollary 6.1 we can show that the areas R=(b,x,y,y#p,a#q,a) are empty.

Case (a): *The segment ab is a top-segment,* see Figure 6.21a. As ab is a top-segment and pq a bottom-peak, the algorithm inserts two diagonals aq and pb. Since the region R is empty, none of the two diagonals intersect any edge in P. Both p and q are located below ab thus a,b,p,q form a convex quadrilateral. By inserting the two diagonals forming the quadrilateral, Algorithm 6.4 creates two polygons P_1:=(b,...,p) and P_2:=(q,...,a). By the choice of e_i to be the highest peak, all horizontal peaks in



(a)
ab is a top-segment

(b)
ab is a bottom segment

Figure 6.21

Partitioning of a rectilinear polygon into monotone polygons admitting quadrilaterization.

these polygons are below e_i, thus P_1 and P_2 preserve property (X). Since these polygons contain fewer horizontal peaks the induction hypothesis applies and a decomposition into monotone polygons admitting quadrilaterization is obtained. This case is

analogous to one of the partitioning steps in [KKK].

Case (b): *The segment ab is a bottom-segment,* see Figure 6.21b. By an argument, similar to that of Remark 6.5, ab lies either entirely to the left of q, or entirely to the right if p. The algorithm inserts the diagonal qa or pb, depending on whether ab is to the left of q or to the right of p, respectively. Assume for this discussion that ab is to the left of q. By inserting the diagonal qa, P is partioned into two polygons P_1={a,...,q} and P_2={q,...,a}. Both P_1 and P_2 have less than k horizontal peaks. As the region R defined above is empty, polygon P_2 satisfies property (X). The other polygon P_1 does not immediately satisfy property (X), since not every other edge in P_1 is vertical. However, by a simple transformation this can be realized. This transformation converts P_1=(a,...,q) into P^*:=(a#q,a,...p) a polygon satisfying property (X). P^* differs from P_1 in that the vertex q is replaced by a#q. P^* contains fewer than k horizontal peaks. By applying the induction hypothesis to P^*, P is partitioned into monotone pieces admitting quadrilaterization. It remains to show that not only P^* but also P_1 admits quadrilaterization. The Steiner-point a#q is a concave vertex located on a bottom-edge. All peaks in P^* are below pa#q. Thus the partitioning step of P^* into monotone components does not introduce any diagonals with endpoint a#q. Therefore p,a#q,a are adjacent vertices on a monotone chain. No vertex on the right chain is located between p and a, therefore by Remark 6.4, these vertices will form a convex quadrilateral p,a#q,a,z in a quadrilaterization of P^* as produced by Algorithm 6.3. From this quadrilaterization, a quadrilaterization of P_1 is found, by replacing this quadrilateral by p,q,a,z.

Complexity-analysis: Step 1, the sorting of the horizontal edges, can be done in O(n log n) time. The algorithm visits, while executing Step 2, each edge exactly once. As there are O(n) edges and the tree depth is worst-case O(log n) each update, insert, or delete takes O(log n) time and the total complexity is O(n log n) worst case. ■

6.5. Quadrilaterization of Simple Rectilinear Polygons

We are now able to combine the results derived in Sections 6.2 and 6.3, to solve the quadrilaterization problem for simple rectilinear polygons.

Theorem 6.3. *A simple, n-vertex rectilinear polygon can be quadrilaterized in O(n log n) time.*

Proof: In Lemma 6.5 it was shown that a partitioning of rectilinear polygons into monotone polygons admitting convex quadrilaterization can be performed in O(n log n) time. By Theorem 6.2 and Lemma 6.2 these polygons can be quadrilaterized in linear time. The total complexity for decomposing an arbitrary simple rectilinear polygon into convex quadrilaterals is therefore O(n log n). ■

In this chapter we have presented several efficient algorithms for solving the quadrilaterization problem for rectilinear polygons. Since arbitrary simple polygons do not necessarily admit quadrilaterization, this partitioning problem is of particular relevance to rectilinear geometry. As was shown by Kahn, Klawe and Kleitman it has direct applications while solving the Rectilinear Art Gallery Problem [KKK83]. Recently two more proofs of the Rectilinear Art Gallery Theorem have been presented by O'Rourke [OR83] and Mannila, Wood [MaW84]. It is interesting to observe that all three proofs are different in nature. Kahn et al. use an argument via quadrilaterization. O'Rourke's proof is direct except for one graph-theoretical argument. Mannila and Wood's proof is purely geometrical.

6.6. Minimum Weight Quadrilaterization of Rectilinear Polygons

We complete our discussion of the quadrilaterization problem by examining the problem of finding the minimum weight quadrilaterization for rectilinear polygons. In some applications a decomposition that minimizes the total length of the diagonals used to form the decomposition is useful. In a joint work with M. Keil, we presented an $O(n^4)$ time algorithm for finding the *minimum edge length quadrilaterization* of a rectilinear polygon [KS84]. The result is an extension of Klincsek's approach for finding the minimum weight triangulation of a simple polygon [Kl80]. Note that a brute force approach would require examining in worst case an exponential number of possible quadrilaterizations for a given rectilinear polygon.

The main tool used for solving minimum edge length polygonal decomposition problems is dynamic programming [Ke83, LPRS82] and we make use of it here. We let W(i,j) be the weight of the minimum weight quadrilaterization of the subpolygon $P_{i,j}$ of P cut off from p_i to p_j (i.e. $p_i, p_{i+1}, \ldots, p_j$). The idea is to build up minimum weight quadrilaterizations of larger and larger subpolygons and solve the problem using dynamic programming. Let Q(i,j) store the vertices of the quadrilaterization that contains the diagonal p_ip_j in the minimum weight quadrilaterization of $P_{i,j}$. Define $d_{i,j}$ to be the Euclidean distance from p_i to p_j if p_i and p_j are visible from each other and ∞, otherwise.

Algorithm 6.6: Minimum Weight Quadrilaterization

Input: A rectilinear polygon P.

Output: The weight of the minimum weight quadrilaterization and the quadrilaterization itself are stored in W(1,n) and Q respectively.

```
for k := 1 to n-1 do
   for i := 1 to n-k do
      begin
         j := i + k
         begin
            if k = 1
               then W(i,j) := 0
            else
            if k is even
               then W(i,j) := +oo
            else
               begin
                  W(i,j) := d_ij + min(W(i,ℓ) + W(ℓ,m) + W(m,j))
                  (* where the minimum is taken over all ℓ and m such that
                     i<ℓ<m<j, and p_i, p_ℓ, p_m, p_j form
                     a convex quadrilateral *)
                  Q(i,j) := (ℓ,m)
                  (* indices of vertices exhibiting the minimum are stored in Q *)
               end
         end
      end
```

To print out all the diagonals of the minimum weight quadrilaterization call the procedure PRINT(1,n) defined as follows:

```
Procedure PRINT(i,j)
  begin
     if p_i p_j is not an edge of P
       then output segment p_i p_j
     where Q(i,j) = (ℓ,m) do
        begin
           call PRINT(i,ℓ);
           call PRINT(ℓ,m);
           call PRINT (m,j)
        end
  end.
```

Theorem 6.3. *Algorithm 6.6 solves the minimum weight quadrilaterization problem for rectilinear polygons in $O(n^4)$ time.*

Proof: Algorithm 6.6 is a straight-forward application of dynamic programming, we therefore omit the proof of correctness. *Complexity Analysis:* The values of $d_{i,j}$ can be computed in a preprocessing step in $O(n^2)$ time. There are $O(n^2)$ values of W(i,j) to be calculated. Each of these requires examining $O(n^2)$ pairs of candidates for p_l and p_m. The total run time of the algorithm is therefore $O(n^4)$. ■

In this chapter we have presented several algorithms to solve the quadrilaterization problem for rectilinear polygons. This partitioning problem is of particular relevance in the context of rectilinear geometry, since rectilinear polygons, in contrast to arbitrary simple polygons, always admit quadrilaterization. Once a quadrilaterization of a rectilinear polygon is available, a solution to the rectilinear guard placement problem is obtained. Complexity results obtained for triangulation and quadrilaterization problems are comparable, except for the respective minimum weight partitioning problems. In $O(n^3)$ a minimum weight triangulation can be obtained, whereas quadrilaterization takes $O(n^4)$ time.

Chapter 7
Concluding Remarks

The main objective of this thesis has been to use the inherent structure of rectilinear polygons in the design of geometric algorithms. A particular emphasis was placed on gaining simplicity and efficiency, as exemplified in the algorithms for detecting whether a given rectilinear polygon is star-shaped, monotone or edge-visible. Furthermore, a unifying approach to many algorithms in rectilinear computational geometry has been achieved through the labeling-scheme. The labeling scheme consists of a pre-processing step for many algorithms. Based on the structural information extracted in this step, subsequent processing becomes simple and conceptually clear. The benefits of unification became apparent when solving visibility problems (hidden-line elimination, shortest rectilinear path and rectilinear convex hull).

While this thesis has restricted itself to describing the labeling scheme for rectilinear polygons, the scheme is not restricted to this domain. The labeling scheme has been generalized to arbitrary simple polygons, and algorithms using the tool are currently under investigation by the author.

The thesis has presented solutions to several guard placement problems for a variety of classes of rectilinear polygons. Since these solutions were first presented [SaT81], [Sa82], a number of other solutions to this and related problems have appeared in the literature, eg. [OR82], [EOW83], [Wo84]. The entire thesis has concerned itself with analyzing algorithms for processing one object modeled by rectilinear polygons, at a time. Some work has been done on designing algorithms for processing more than one rectilinear object, see for example [ELOW83], [EOW83a], [NiP82], or [Wo84]. In general we observe a growing interest in problems in rectilinear geometry.

While solving some problems in rectilinear computational geometry, many problems still remain open. These include:

(1) Is $\Omega(n \log n)$ lower bound for quadrilateralization of simple rectilinear polygons?

(2) We have established a necessary condition for a given rectilinear polygon to be simple, which is based on the labeling scheme. Can a sufficient condition for simplicity be established, which can similarly be tested in linear time? Or is $\Omega(n \log n)$ lower bound for this problem?

Several aspects of the guard problem remain unsolved:

(3) The algorithm for placing guards in an n-vertex rectilinear polygon as presented in Chapter 6 always places $\left\lfloor \frac{n}{4} \right\rfloor$ guards, even if less guards would suffice for a given polygon. To the author's knowledge, no polynomial-time algorithm is known for computing the minimum number of guards necessary for a given rectilinear polygon, nor for placing these guards.

(4) $\left\lfloor \frac{n}{4} \right\rfloor$ guards are sometimes necessary and always sufficient to guard any rectilinear polygon. For simple polygons $\left\lfloor \frac{n}{3} \right\rfloor$ guards are always sufficient and sometimes necessary. Can one characterize the largest class of simple polygons for which $\left\lfloor \frac{n}{4} \right\rfloor$ guards are always sufficient?

(5) Given a convex quadrilateralization of a rectilinear polygon, a triangulation can always be obtained by inserting one diagonal in each convex quadrilateral. By just removing diagonals from an arbitrary triangulation of a rectilinear polygon, it is not always possible to form a convex quadrilateralization. Does there exist an independent characterization of triangulations, for which this would always be possible?

References

[AHU74] Aho, A.V., J.E. Hopcroft and J.D. Ullman, *The Design and Analysis of Computer Algorithms,* Addison-Wesley, Reading, MA, 1974.

[AA83] Asano, T. and T. Asano, "Minimum partition of polygonal regions into trapezoids", *Proc. 24th Annual IEEE Symposium on Foundations of Computer Science,* Tucson, AZ, November 1983, pp. 233-241.

[Av79] Avis D., "On the complexity of finding the convex hull of a set of points", Tech. Rept. SOCS 79.2, School of Computer Science, McGill University, Montréal, February 1979.

[AT81] Avis, D. and G.T. Toussaint, "An efficient algorithm for decomposing a polygon into star-shaped polygons", *Pattern Recognition,* Vol. 13, No. 6, 1981, pp 395-398.

[AT81a] Avis, D. and G.T. Toussaint, "An optimal algorithm for determining the visibility of a polygon from an edge", *IEEE Transactions on Computers,* Vol. C-30, No. 12, December 1981, pp. 910-914.

[BhE81] Bhattacharya, B.K. and H. ElGindy, "A new linear convex hull algorithm for simple polygons", Tech. Rept. No. SOCS 81.29, McGill University, Montréal, November 1981.

[Ca76] Carmo, Manfredo P. do, *Differential Geometry of Curves and Surfaces,* Prentice Hall, Eaglewood Cliffs, 1976, pp. 30-41, 390-404.

[ChD79] Chazelle, B. and D. Dobkin, "Decomposing a polygon into its convex parts", *Proc. 11th Annual ACM Symposium on Theory of Computing,* Atlanta, GA, April 1979, pp. 38-48.

[Ch80] Chazelle B., "Computational geometry and convexity", Ph D. thesis, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Penn., July 1980.

[Ch82] Chazelle, B. "A theorem on polygon cutting with applications", *Proc. 23rd Annual IEEE Symposium on Foundations of Computer Science,* Chicago, 1982, pp. 339-349.

[ChI83] Chazelle, B. and J. Incerpi, "Triangulating a polygon by divide-and-conquer, *Proc. 21st Allerton Conference on Communication, Control and Computing,* Monticello, IL., October 1983, pp. 447-456.

[Ch75] Chvátal V., "A combinatorial theorem in plane geometry", *Journal of Combinatorial Theory, Ser. B,* Vol. 18, No. 1, February 1975, pp. 39-41.

[DNW84] Dean, J., K. Ng, and M Waldvogel, "Implementation of rectilinear computational geometry algorithm", unpublished manuscript, School of Computer Science, Carleton University, Ottawa, May 1984.

[EOW83] Edelsbrunner, H., J. O'Rourke, and E. Welzl, "Stationing guards in rectilinear art galleries", Technical Report 83-2, Johns Hopkins University, Baltimore, MD., March 1983.

[ELOW83] Edelsbrunner, H., J. van Leeuwen, T. Ottman, D. Wood, "Computing the connected components of simple rectilinear geometrical objects in d-space", Technical Report, CS-83-17, University of Waterloo, Waterloo, June 1983.

[Em80] Emde, van Boas, P. "On the $\Omega(n \log n)$ lower bound for convex hull and maximal vector determination", *Information Processing Letters,* Vol. 10, No. 3, April 1980, pp. 132-136.

[El82] ElGindy, H., Personal communication.

[ElA81] ElGindy, H. and D. Avis, "A linear algorithm for computing the visibility polygon from a point", *Journal of Algorithms,* Vol. 2, No. 2, June 1981, pp. 186-197.

[EAT83] ElGindy, H., D. Avis and G.T. Toussaint, "Applications of a two-dimensional hidden-line algorithm to other geometric problems", *Computing,* Vol. 31, No. 3, 1983, pp. 191-202.

[Fi78] Fisk, S., "A short proof of Chvátal's watchman theorem," *Journal of Combinatorial Theory, Ser. B,* Vol. 24, No. 3, June 1978, p. 374.

[FeP75] Feng, H. and T. Pavlidis, "Decomposition of polygons into simpler components: feature generation for syntactical pattern recognition", *IEEE Transactions on Computers,* C-24, No. 6, June 1975, pp. 636-650.

[GJPT78] Garey, M.R., D.S. Johnson, F.P. Preparata and R.E. Tarjan, "Triangulating a simple polygon", *Information Processing Letters,* Vol. 7, No. 4, June 1978, pp. 175-179.

[Gr72] Graham R., "An efficient algorithm for determining the convex hull of a planar set", *Information Processing Letters,* Vol. 1, June 1972, pp. 132-133.

[GrY83] Graham R. and F.F. Yao, "Finding the convex hull of a simple polygon", *Journal of Algorithms,* Vol. 4, No. 4, December 1983, pp. 324-331.

[Ho76] Honsberger, R., *Mathematical Gems,* Mathematical Association of America, 1976, pp. 104-110.

[HoU79] Hopcroft, J.E. and J.D. Ullman, *"Introduction to Automata Theory, Languages, and Computation",* Addison-Wesley, Reading, MA., 1979.

[In82] Inoue, S. "An implementation of quadrilateralization algorithms", Master's Project, McGill University, Montreal, October 1982.

[KKK83] Kahn, J., M. Klawe and D. Kleitman, "Traditional galleries require fewer watchmen", *SIAM Journal of Algebraic and Discrete Methods*, Vol. 4, No. 2, June 1983, pp. 194-206.

[Ke83] Keil, J.M., "Decomposing polygons into simpler components", Ph.D. thesis, Department of Computer Science, University of Toronto, Toronto, 1983.

[KS84] Keil, J.M. and J.-R. Sack, "Minimum decompositions of objects", Tech. Report SCS-TR-42, School of Computer Science Carleton University, Ottawa, March 1984, to appear in *Computational Geometry*, Ed. G.T. Toussaint, North Holland, Oxford-New York, 1984.

[Ki83] Kirkpatrick, D.G., "Optimal search in planar subdivisions", *SIAM Journal on Computing*, Vol. 12, No. 1, February 1983, pp 28-35.

[KiS82] Kirkpatrick, D.G., and R. Seidel, "The ultimate planar convex hull algorithm?", *Proc 20th Allerton Conference on Communication, Control and Computing*, Monticello, IL., October 1982, pp. 35-12.

[Kl80] Klincsek, G T, "Minimal Triangulations of Polygonal Domains", *Annals of Discrete Math.*, Vol. 9, Combinatorics 79, Part 2, 1980, pp. 121-123.

[Kli78] Klingenberg, W., *A Course in Differential Geometry*, Springer-Verlag, Berlin-Heidelberg-New York, 1978, pp. 21-29.

[LeP77] Lee, D.T. and F.P. Preparata, "Location of a point in a planar subdivision and its applications", *SIAM Journal on Computing*, Vol. 6, No. 3, September 1977, pp. 591-606.

[LeP79] Lee, D.T. and F.P. Preparata, "An optimal algorithm for finding the kernel of a polygon", *Journal of the ACM*, Vol. 26, No. 3, July 1979, pp. 415-121.

[Le83] Lee, D.T., "Visibility of a simple polygon", *Computer Vision, Graphics and Image Processing*, Vol. 22, No. 2, May 1983, pp. 207-221.

[Le83a] Lee, D.T., "On finding the convex hull of a simple polygon", *International Journal of Computing and Information Science*, Vol. 2, No. 2, April 1983, pp. 87-98.

[LPRS82] Lingas, A.; R.Y. Pinter; R.L. Rivest and A. Shamir, "Minimum edge length partitioning of rectilinear polygons", *Proc. 20th Allerton Conference on Communication, Control and Computing*, Monticello, IL., October 1982, pp. 53-63

[LiT77] Lipton, R.J. and R.E. Tarjan, "A separator theorem for planar graphs", Waterloo Conference on Theoretical Computer Science, Waterloo, 1977, pp. 1-10.

[LiT77a] Lipton, R.J. and R.E. Tarjan, "Applications of a planar separator theorem", *Proc. 18th Annual IEEE Symposium on Foundations of Computer Science*, Providence, RI, November 1977, pp. 162-170.

[MS84] Majocchi, L. and G.R. Sechi, "The problem of a mouse in a maze", *ACM Sigart Newsletter*, No. 87, January 1984, pp. 39-44.

[McA79] McCallum D. and D.Avis, "A linear algorithm for finding the convex hull of a simple polygon", *Information Processing Letters*, Vol. 9, No. 5, December 1979, pp. 201-206.

[MaW84] Mannila, H. and D. Wood, "A simple proof of the rectilinear art gallery theorem", Tech. Rept. C-1984-16, Department of Computer Science, University of Helsinki, Finnland, 1984.

[MF82] Montuno, M.F. and A. Fournier, "Detecting intersections among star polygons", Tech. Rept. CSRG-146, University of Toronto, Toronto, September 1982.

[MF82a] Montuno, D.Y. and A. Fournier, "Finding the x-y convex hull of a set of x-y polygons", Tech. Rept. CSRG-148, University of Toronto, Toronto, November 1982.

[NeS79] Newman, W.M., Sproull, R.F., *Principles of Interactive Computer Graphics*, McGraw Hill Inc., New York, 1979.

[NLLW83] Nicholl, T.M., D.T. Lee, Y.Z. Liao and C.K. Wong, "Constructing the X-Y Convex Hull of a Set of X-Y Polygon", *BIT*, Vol. 23, 1983, pp. 456-471.

[NiP82] Nievergelt, J. and F.P. Preparata, "Plane-sweep algorithms for intersecting geometric figures", *Communication of the ACM*, Vol 25, October 1982, pp. 739-747.

[OR82] O'Rourke, J., "Galleries need fewer mobile guards: A variation on Chvátal's theorem", Technical Report 82-11, Johns Hopkins University, Baltimore, MD., October 1982.

[OR82a] O'Rourke, J. "The signature of a curve", Tech. Rept. 82-9, Johns Hopkins University, Baltimore, MD., August 1982.

[OR83] O'Rourke, J., "An alternate proof of the rectilinear art gallery theorem", Tech. Rept. 83-3, Johns Hopkins University, Baltimore, MD., March 1983.

[OSW83] Ottmann, Th., E. Soisalon-Soininen and D. Wood, "Rectilinear Convex Hull Partitioning of Sets of Rectilinear Polygons", Computer Science Tech. Rept. CS-83-19, University of Waterloo, Waterloo, June 1983.

[OSW83a] Ottmann, Th., E. Soisalon-Soininen and D. Wood, "On the Definition and Computation of Rectilinear Convex Hulls", Computer Science Tech. Rept. CS-83-07, University of Waterloo, Waterloo, October 1983.

[OWW82] Ottmann, Th., P. Widmayer and D. Wood, "A worst-case efficient algorithm for hidden-line elimination", Computer Science Tech. Rept. CS-82-33, University of Waterloo, Waterloo, October 1982.

[Pa77] Pavlidis, J., *Structural Pattern Recognition*, Springer Series in Electrophysics, Vol. 1, Springer-Verlag, Berlin-Heidelberg-New York, 1977.

[Pr77] Preparata, F.P., "Medial axis of a convex polygon", *Steps into Computational Geometry*, F.P. Preparata, Ed., Coordinated Science Lab., University of Illinois, Urbana, IL., Tech. Rept. R 760, 1977, pp. 6-9.

[Pr81] Preparata, F.P., "A new approach to planar point location", *SIAM Journal on Computing*, Vol. 10, No. 3, 1981, pp. 473-482.

[PrS81] Preparata, F.P. and K.J. Supowit, "Testing a simple polygon for monotonicity", *Information Processing Letters*, Vol. 12, No. 4, August 1981, pp. 161-164.

[Ra82] Rappaport, D., "Eliminating hidden-lines from monotone slabs", *Proc. 20th Allerton Conference on Communication, Control and Computing*, Monticello, IL., October 1982, pp. 43-52.

[RaT83] Rappaport, D. and G.T. Toussaint, "A simple linear-time algorithm for hidden-line elimination in star-shaped polygons", Tech. Rept. No. SOCS 83.23, McGill University, Montreal, November 1983.

[SaT81] Sack, J.-R. and G.T. Toussaint, "A linear-time algorithm for decomposing rectilinear star-shaped polygons into convex quadrilaterals", *Proc. 19th Allerton Conference on Communication, Control and Computing*, Monticello, IL., October 1981, pp. 21-30.

[Sa82] Sack, J.-R. "An O(n log n) algorithm for decomposing simple rectilinear polygons into convex quadrilaterals", *Proc. 20th Allerton Conference on Communication, Control and Computing*, Monticello, IL., October 1982, pp. 64-74.

[Sa83] Sack, J.-R., "A simple hidden-line algorithm for rectilinear polygons", *Proc. 21st Allerton Conference on Communication, Control and Computing*, Monticello, IL., October 1983, pp. 437-446.

[Sc78] Schachter, B., "Decomposition of polygons into convex sets", *IEEE Transactions on Computers*, C-27, No. 11, November 1978, pp. 1078-1082.

[ScL80] Schoone, A.A. and J. van Leeuwen, "Triangulating a star-shaped polygon", University of Utrecht, Utrecht, Holland, Tech. Rept. RUU-CS-80-3, April 1980.

[Sh77] Shamos, M.I., " Computational Geometry", Ph.D. thesis, Department of Computer Science, Yale University, New Haven, CT., May 1978.

[Sk70] Sklansky, J., "Recognition of convex blobs", *Pattern Recognition*, Vol. 2, No. 1, January 1970, pp. 3-10.

[SSS74] Sutherland, I.E.; R.F. Sproul; and R.A. Schumacker, "A characterization of ten hidden-surface algorithms", *Computing Surveys*, Vol. 6, No. 1, March 1974, pp. 1-56.

[To80] Toussaint, G.T., "Pattern recognition and geometrical complexity", *Proc. Fifth International Conf. on Pattern Recognition*, Vol. 2, Miami Beach, December 1980, pp. 1324-1347.

[ToE81] Toussaint, G.T. and H. El Gindy, "Traditional Galleries are star-shaped if every two paintings are visible from some common point", Tech. Rept. No. SOCS 81.10, School of Computer Science, McGill University, Montreal, 1981.

[ToA82] Toussaint, G.T. and D. Avis, "On a convex hull algorithm for polygons and its applications to triangulation problems", *Pattern Recognition*, Vol. 15, No. 1, 1982, pp. 23-29.

[To83] Toussaint, G.T., "A new linear algorithm for triangulating monotone polygons", Tech. Rept. No. SOCS-83.9, McGill University, Montréal, June 1983.

[ToS84] Toussaint, G.T. and J.-R. Sack, "Some new results on moving polygons in the plane", *Proc. Robotic Intelligence and Productivity Conference,* Detroit, November 1983, pp. 158-164.

[Wo84] Wood, D., "An isothetic view on computational geometry", to appear in *Computational Geometry,* Ed. G.T.Toussaint, North Holland, Oxford-New York, 1984.

[Ya79] Yao, A. "A lower bound to finding convex hulls", *Journal of the ACM,* Vol. 28, No. 4, October 1981, pp. 780-787.

[Ya80] Yao, F.F., "On the priority approach to hidden-surface algorithms", *Proc. 21st Annual IEEE Symposium on Foundations of Computer Science,* Clarkson University, Potsdam, NY, October 1980, pp. 301-307.